औद्योगिक प्रशिक्षण के लिए राष्ट्रीय संस्थान
National Institute for Industrial Training

[MERN Full Stack development Project]

Submitted By: Somenath Paul

Submitted To: Avik Ghosh

Submitted On: 12-12-2025

INDEX

Serial No.	Topic Name	Page No.
1	Student Profile	3
2	Acknowledgement	4
3	Introduction	5-6
4	Objectives	7-22
5	Hardware And Software Requirements	23-24
6	Future Scopes	25-26
7	Advantages & Disadvantages	27-29
8	Coding	30-188
9	Snapshots	189-199
10	Conclusion	200
11	Bibliography	201

STUDENTS PROFILE

Name – Somenath Paul

College – Techno India Batanagar

Course – Computer Science & Engineering

Semester – 8th sem (4rd year)

Year Of Passing – 2026

Phone no. – 9933681667

E-mail –somenathpaulbusiness10@gmail.com

ACKNOWLEDGEMENT

I am immensely grateful to everyone who contributed to the successful completion of my project on the Employee Management System.

First and foremost, I extend my sincere gratitude to my Training Mentor, Avik Ghosh Sir, for his unwavering support, expert guidance, and constructive feedback throughout the project. His technical insights and timely suggestions helped me overcome design and implementation challenges and significantly improved the quality of this work.

I am also deeply thankful to my faculty members and other mentors for their encouragement and assistance. Their readiness to share knowledge, provide resources, and review progress at key stages was instrumental in shaping the direction of the project.

My heartfelt thanks go to my friends and teammates for their collaboration, troubleshooting help, and moral support during development and testing. Their practical suggestions and peer reviews helped identify issues early and refine the product.

Finally, I would like to thank my family for their constant encouragement and belief in my abilities, which motivated me to complete this project with dedication and commitment.

Thank you all for being part of this journey.

INTRODUCTION

In the modern digital landscape, organizational efficiency heavily depends on how effectively employee data, tasks, and workflows are managed. Traditional manual methods are no longer sustainable for businesses that require real-time monitoring, streamlined operations, and reliable data accessibility. To address these needs, this documentation presents the development of a full-stack Employee Management System built using the MERN stack (MongoDB, Express.js, React, Node.js). This system is designed to provide a secure, scalable, and user-friendly platform for managing employee records, task assignments, attendance tracking, departmental structures, and administrative operations.

This project demonstrates the integration of powerful frontend and backend technologies to create a cohesive management system tailored for modern business environments.

Technologies Learned and Their Usage

React (Frontend)

React was used to build a fast, interactive, and modular user interface. Through component-based architecture, dynamic routing, and efficient state management, React ensured seamless navigation and real-time updates across the system. Key features built with React include:

- Employee registration and profile management
- Task assignment and task tracking UI
- Dashboard visualizations using Recharts
- Interactive modals, animations (Framer Motion), and alerts (SweetAlert2)

TailwindCSS

TailwindCSS provided a utility-first design approach, enabling rapid UI development with clean, responsive, and customizable styling. It simplified layout creation and made the interface visually consistent across all pages.

JavaScript (ES6+)

Modern JavaScript features were crucial for handling logic on both the client and server side. On the frontend, JavaScript managed form validations, dynamic components, and asynchronous API calls using Axios. On the backend, it handled request processing, authentication logic, and database operations.

Node.js & Express.js (Backend)

Node.js served as the backend runtime, while Express.js powered the server-side application logic. Key functionalities implemented include:

- RESTful API development
- Authentication and authorization using JWT
- Secure password hashing with bcryptjs
- File uploads using multer
- Robust error handling and modular routing

Express enabled rapid API development while maintaining readable and scalable code architecture.

MongoDB & Mongoose

MongoDB was used as the primary NoSQL database for storing employee records, attendance logs, task details, and admin data. Mongoose simplified schema creation, data validation, and optimized querying. MongoDB's flexibility allowed easy expansion of data models as the project scaled.

Additional Libraries & Tools

- Axios for secure HTTP communication between client and server
- Framer Motion for smooth UI animations
- Recharts for data visualization in dashboards
- Moment.js for formatting timestamps and managing dates
- Vite as a fast development server and build tool for the frontend
- Nodemon for automatic server restarts during backend development

Versatility of the MERN Technologies

Using the MERN stack enabled seamless integration between frontend and backend. React handled dynamic UI updates efficiently, while Node.js+Express delivered robust server-side logic. MongoDB ensured scalability for large datasets such as attendance logs, employee documents, and daily task updates. Combined, these technologies delivered a highly responsive, scalable, and maintainable Employee Management System suitable for both small teams and large organizations.

The skills acquired through this project — including API development, secure authentication, component-based UI design, responsive styling, and database modeling — directly contribute to professional full-stack development capabilities.

OBJECTIVES

The Employee Management System (EMS) built using the MERN stack is designed to streamline and automate the core operational activities within an organization. The primary objective is to replace manual processes with a centralized, efficient, and secure digital platform that manages employee information, tasks, attendance, departments, and administrative workflows.

This section outlines the objectives, functional scope, and key components of the system.

1. Centralized Employee Data Management

Provide a unified platform to store, update, and access employee profiles, including personal details, contact information, job roles, departments, and employment status. The goal is to eliminate data duplication and ensure organizational accuracy.

2. Efficient Task Assignment and Tracking

Enable administrators or managers to:

- Assign tasks to employees
- Set deadlines
- Add task descriptions and priority levels
- Monitor task progress in real-time

This enhances accountability and ensures operational transparency.

3. Attendance and Activity Monitoring

Introduce reliable mechanisms to record:

- Daily attendance
- Login/logout timestamps
- Work status updates

This supports performance evaluation and departmental insights.

4. Secure Authentication and Role-Based Access

Implement JWT-based authentication with bcrypt-secured passwords to ensure:

- Only authorized personnel can access sensitive data
- Role-based privileges control system interactions (admin vs. employee)

5. Automated and Real-Time Updates

Use React + Axios for instant UI updates, allowing employees and admins to view changes in real-time without page reloads.

6. Data Integrity and Scalability

With MongoDB as the backbone, the system ensures:

- Structured, high-volume data handling
- Flexibility to scale with organizational growth
- Consistency across all modules (employees, tasks, attendance, departments)

Functionality Overview

User Authentication

- Login system using JWT tokens
- Secure route protection on frontend
- Role-based dashboards: Admin and Employee views

Employee Data Management

- Add, edit, delete, and view employee records
- Maintain department, designation, and employment history
- Upload employee documents/photos via Multer

Task Management

- Admin assigns tasks to employees
- Employees can view tasks, update progress, and mark completion
- Deadlines and priority-based sorting

Attendance Tracking

- Track login and logout times
- Daily attendance logs stored in MongoDB
- Monthly attendance report generation

Search and Filter System

- Advanced search for employees by name, department, role
- Filtering for tasks by status, deadline, priority

Dashboard and Analytics

Using Recharts, the system provides:

- Employee count per department
- Task completion statistics
- Attendance summaries

Key Components

Frontend Interface

- Responsive UI built with React and TailwindCSS
- Framer Motion animations
- Protected routes using React Router
- SweetAlert2 notifications for actions (success/error prompts)

Backend Logic

- RESTful API built using Express.js
- Secure authentication (JWT)
- Data modeling with Mongoose
- Multer-based upload handling
- Middleware for validation and error-handling

Database Management

- Employee collection
- Task collection
- Attendance logs
- Admin collection
- Department collection

Optimized indexes ensure efficient data queries, even at scale..

Benefits

1. Streamlined Operations

Automates repetitive administrative processes such as maintaining records, assigning tasks, and managing attendance.

2. Real-Time Transparency

Live dashboards keep both employees and management informed.

3. High Accuracy

Reduces manual errors in employee details, task tracking, and attendance.

4. Accessibility

System accessible across devices, ensuring flexible usage for employees and admins.

Implementation Strategies

Effective implementation of an Employee Management System requires structured planning, optimized workflows, and strict adherence to software development best practices. The system must ensure data accuracy, operational efficiency, performance consistency, and long-term maintainability. The following strategies outline how the EMS is designed, developed, and maintained to ensure reliability and scalability in an organizational environment.

---

1. Ensuring Data Accuracy and Integrity

Accurate data is the backbone of any management system. To maintain precision across all modules—employees, tasks, attendance, departments—the following strategies are applied:

Data Validation Rules

- Backend validation using Mongoose schema rules (required fields, type checks, unique constraints).
- Frontend validation using controlled inputs and real-time error prompts via React.
- Prevent incorrect data entry by enforcing field-level restrictions (email format, date validation, numeric input checks).

Real-Time Consistency Checks

- Use of Axios to fetch updated employee/task data after every CRUD operation.
- Instant UI updates without reload using React state, ensuring consistency across sessions.

Duplicate Prevention

- Unique indexing in MongoDB for fields like email, employee ID, department ID.
- Backend checks before data insertion to avoid redundant entries.

---

2. Enhancing Operational Efficiency

The EMS aims to minimize manual intervention and automate key workflows.

Automated Processes

- Task assignment notifications generated automatically for employees.
- Task status updates reflected instantly in admin dashboards.
- Attendance logs generated on login/logout actions without manual entries.

Performance Optimization

- Pagination and filtering for employee lists, tasks, and attendance logs.
- Efficient querying with MongoDB indexes to reduce response times.

- Caching frequently requested data where applicable.

Workflow Streamlining

- Admins can manage employees, tasks, and departments from centralized dashboards.
- Employees use a simplified interface to update tasks and view assignments.

---

3. Improving User Experience

UX plays a crucial role in ensuring system adoption and productivity.

Responsive UI

- TailwindCSS ensures mobile-first design.
- Components adjust automatically based on screen size and orientation.

Interactivity and Feedback

- SweetAlert2 used for confirmations, warnings, and success prompts.
- Framer Motion animations enhance transitions, improving usability.
- Dynamic dashboards created with Recharts for clear visual insights.

Accessibility

- Support for keyboard navigation.
- Clear typography and consistent color schemes.
- Error/success messages designed with high contrast.

---

4. Facilitating Informed Decision-Making

Decision-making depends on accurate analytics and meaningful insights.

Reporting Tools

- Attendance summary charts (daily/monthly/yearly).
- Task completion statistics per employee and department.
- Employee distribution graphs by department.

Log & History Tracking

- Activity logs for updates, tasks, attendance, and profile changes.
- Timestamp management with Moment.js for readable formats.

Admin Insights

- Admin dashboard consolidates employee count, pending tasks, and performance metrics.

5. Ensuring Compliance and Security

Security is mandatory for any system managing sensitive employee data.

Authentication & Authorization

- JWT used for secure token-based authentication.
- Role-based access: Admin vs Employee routes protected on both frontend and backend.
- Passwords hashed using bcryptjs before database storage.

Data Protection

- HTTPS recommended for production deployment.
- CORS configuration to prevent unauthorized API usage.
- Environment variables managed using dotenv to secure credentials.

Compliance

- Adherence to modern data protection standards (similar to GDPR principles):
  - Limited access to sensitive data
  - Secure storage
  - Clear permission-based access

6. Supporting Long-Term Sustainability

The EMS is designed to remain reliable as usage scales.

Scalable Architecture

- Modular backend with separate routers for employees, tasks, attendance, departments.
- Component-based frontend structure for easy maintenance.

Code Maintainability

- Consistent folder structure for controllers, models, routes, and services.
- Reusable UI components for forms, cards, tables, modals.

Deployment Ready

- Frontend build optimized using Vite.
- Backend prepared for hosting on platforms like Render, Railway, or VPS environments.
- MongoDB Atlas recommended for cloud database scalability.

7. Implementation Best Practices

To minimize bugs, improve reliability, and strengthen the system:

Regular Testing

- API testing using Postman.
- Frontend unit/component testing where applicable.
- Manual testing for forms, CRUD operations, and dashboards.

Logging & Error Handling

- Express middleware for centralized error handling.
- Console logging or external logging tools for backend debugging.

Backup & Recovery

- Scheduled MongoDB backups.
- Version control via Git and GitHub for development history.

Project Goals

▪ Streamline employee and task management: Automate the process of employee registration, task assignment, attendance tracking, and departmental operations, reducing manual workload for administrators.
▪ Enhance user experience: Provide an intuitive, easy-to-navigate interface for both admins and employees to manage profiles, view tasks, and track progress effortlessly.
▪ Increase organizational efficiency: Implement features that improve workflow transparency, such as real-time task updates, attendance logs, and performance dashboards.
▪ Provide scalability: Design the system to scale as the number of employees, tasks, departments, and features grows over time, without performance degradation.

Project Objectives (SMART)

▪ Reduce manual administrative work by 30% within 6 months by automating employee data management, attendance recording, and task workflows.
▪ Increase employee productivity tracking accuracy by 20% within 4 months through structured task assignment, progress updates, and dashboard analytics.
▪ Enhance user satisfaction by 15% within 1 year through a responsive UI, clean visual components, and optimized task/attendance interfaces.

▪ Introduce a mobile-responsive version within 9 months enabling employees to check tasks, attendance, and notifications from any device.

Deliverables

▪ Fully functional Employee Management System:

- A user-friendly interface for admins to manage employee profiles, assign tasks, view attendance logs, and analyze performance.
- Modules for employee registration, department management, task assignment/tracking, attendance, and document uploads.
- Reporting and analytics dashboards for viewing employee distribution, task metrics, and performance indicators.

▪ Employee Dashboard:

- View assigned tasks, update work progress, mark task completion, and track attendance history.
- Real-time notifications for new tasks, deadlines, or admin updates.

▪ Administrative Tools:

- Role-based access controls
- Centralized data management
- Attendance processing and export/download support using file-saver
- Performance reports generated using Recharts

▪ Comprehensive documentation:

- User manual for both admin and employee modules
- Installation, environment setup, and deployment guide for MERN stack backend & frontend

---

VALUATION MATRIX & CRITERIA (EMS VERSION)

Criteria	Weighting	Success Metric	Threshold
Functionality	40%	Meets all system objectives including employee management, task workflows, attendance tracking, authentication, and analytics	80% of functionality operational
Usability	30%	Interface should be easy to navigate for both employees and administrators	Above-average satisfaction based on user feedback
Performance	20%	Fast load times, optimized queries, and stable uptime	95% uptime, < 3 seconds UI load time

Criteria	Weighting	Success Metric	Threshold
Security	10%	Secure authentication, encrypted password storage, protected routes	No major vulnerabilities identified after testing

This matrix provides a structured framework to evaluate project success. By defining measurable criteria, the system can be assessed objectively for reliability, performance, and user satisfaction.

VALUATION & ANALYSIS METHODS FOR THE EMPLOYEE MANAGEMENT SYSTEM

Evaluating the project's value involves analytical and financial methods to understand its feasibility, scalability, and long-term organizational impact.

---

1. Market Analysis for Employee Management Systems (EMS)

Total Addressable Market (TAM)

TAM represents the total potential market value for HR/employee management software globally.

Example Calculation:
- Global HR software market value: INR 4 trillion annually
- Average organizational spend per employee management system: INR 5,000 per employee

TAM = Market Size × Average Spend
TAM = INR 4 trillion × INR 5,000 = INR 20,000 billion

Serviceable Available Market (SAM)

SAM refers to the portion of the TAM that your EMS can realistically target.

Example Calculation:
- Target Segment: Small and medium-sized businesses (SMBs)
- Number of SMBs globally: 50 million
- Average spend per SMB: INR 40,000/year

SAM = Total SMBs × Average Spend
SAM = 50 million × 40,000 = INR 2 trillion

Serviceable Obtainable Market (SOM)

SOM is what your EMS can realistically capture within 3–5 years.

Example Calculation:
▪ Estimated penetration rate: 3%

SOM = SAM × Penetration Rate
SOM = INR 2 trillion × 3% = INR 60 billion

Summary of EMS Market Analysis

▪ TAM: INR 20,000 billion
▪ SAM: INR 2 trillion
▪ SOM: INR 60 billion

Additional Considerations

Competitive Landscape

Compare with existing platforms:

- Zoho People
- Keka
- GreytHR
- DeskTime

Your EMS differentiates through:

- Custom MERN-based flexibility
- Real-time dashboards
- Lightweight, fast, and customizable UI
- Low deployment cost

Growth Projections

Trends indicate growing demand for:

- Cloud-based HR systems
- Task automation
- Attendance analytics
- AI-based workforce insights

Market Entry Strategy

To capture SOM:
- Partner with small businesses and startups
- Offer competitive pricing models
- Highlight customization capability (unlike fixed commercial software)
- Provide free trials or demo versions

---

2. Cost Approach for the Employee Management System (EMS)

The Cost Approach evaluates the value of the Employee Management System based on the total cost required to develop or replace it, including design, infrastructure, deployment, and long-term maintenance. It also accounts for operational savings achieved after automation.

Steps Involved

- Estimating Development Costs

This includes all expenses for system development, UI/UX, backend services, cloud infrastructure, authentication modules, database configuration, and deployment.

Example:
If the development cost of the EMS is INR 5,000,000, and infrastructure/hosting/operations cost for the first year is INR 1,000,000, the total development cost becomes:

Total cost = INR 6,000,000

- Adjusting for Cost Savings or Improvements

Once implemented, the EMS improves operational efficiency by automating employee management, attendance tracking, task workflows, performance insights, and document handling.

Example:
If the EMS reduces HR and administrative workload by automating attendance, profiling, and task assignment — resulting in INR 500,000 savings annually — this operational efficiency must be factored in.

Practical Example

By calculating the development cost and adjusting for operational savings, stakeholders can determine whether investing in the EMS is financially beneficial. If automation significantly reduces repetitive HR tasks and improves decision-making, the project's value increases, making it a sound investment.

---

1. Overview of Valuation Methods

To assess the intrinsic value and financial feasibility of the Employee Management System, multiple valuation methodologies are applied. Each method provides a unique strategic perspective.

Integrated Valuation Methods

- Discounted Cash Flow (DCF) Method: Evaluates future cost savings and projected financial benefits from automation.
- Comparable Company Analysis (CCA): Compares system value with similar HRMS/EMS platforms used in industry.
- Market Approach: Assesses the EMS relative to the broader HR software and employee workflow management market.
- Cost Approach: Determines value based on total development and maintenance cost of the platform.

---

2. Key Valuation Drivers

The key financial and operational drivers for evaluating the value of the Employee Management System include:

- Operational Savings: Reduced HR workload, fewer manual errors, automated attendance & task workflows.
- Efficiency Gains: Faster employee onboarding, accurate data handling, improved reporting.
- System Adoption Rate: Number of departments/employees actively using the EMS.
- Market Penetration: The system's potential to be deployed across institutions or organizations.
- Retention Value: Long-term organizational dependency on accurate employee management systems.

---

3. Primary Sources of Valuation Data

1. Financial Statements

Used to estimate cost savings, return on investment, and internal financial impact.

- Expense Reports: HR operational savings after automation
- Projected Savings: Reduction in administrative overhead
- Cash Flow: Cost reduction from attendance automation, task management, and document processing

Practical Example:
Study HRMS cost reports from companies using Keka, Zoho People, GreytHR to estimate future ROI.

2. Market Research Reports

Used to understand industry benchmarks and HR digitalization growth.

- Market Size & Adoption Rate: HR management software demand globally
- User Demographics: Organizations primarily shifting to HR automation
- Competitive Landscape: Compare key HRMS platforms and their offerings

Practical Use:
Research from Statista, Gartner, Deloitte HR Tech reports.

---

3. Industry Benchmarks

- Operational Efficiency Metrics: Time saved per employee task, onboarding, attendance processing
- Financial Ratios: ROI of adopting HRMS solutions
- Valuation Multiples: Cost per employee for HR automation platforms

Practical Use:
Compare benchmarks with systems like GreytHR, Zoho People, SAP SuccessFactors.

---

4. Expert Opinion and Market Insights

- Analyst reports on HR tech transformation
- Consultant insights on workforce automation trends
- Interviews with HR managers to understand system impact

Practical Use:
Expert insights refine cost projections and adoption probabilities.

---

4. Data Collection and Analysis Process

To build a reliable valuation model for the EMS, the following data sources are analyzed:

- Financial statements for cost savings
- HR software adoption rates
- Efficiency improvements from automation
- Market data and organizational HR trends
- Operational metrics (manual HR tasks vs. automated workflow systems)

---

5. Sensitivity Analysis and Assumptions

Sensitivity analysis helps estimate risks and variations in EMS performance.

Assumptions include:

- 10% annual organizational growth requiring scalable EMS
- Consistent reduction in HR workload due to automation
- Projected feature expansion cost for new modules
- Discount rate of 12% due to market volatility in HR tech

Risk factors such as low adoption rate or system downtime are tested across scenarios.

---

6. Integration and Final Valuation

After collecting relevant data, valuation methods are combined:

- DCF: Calculates Net Present Value of operational savings
- Comparable Analysis: Uses competitor HRMS valuations
- Cost Approach: Evaluates development cost and replacement value

The final valuation is derived using a weighted average, providing a comprehensive value assessment of the EMS.

Risk Assessment and Mitigation (EMS Version)

1. Market Volatility

Adoption rate of HRMS tools may fluctuate.
Mitigation: Flexible module-based architecture to add or remove features.

2. Data Accuracy

Incorrect employee or attendance data affects operations.
Mitigation: Strict frontend/backend validation and schema enforcement.

3. Regulatory Changes

Data privacy laws may impact HR systems.
Mitigation: Implement secure data handling, encrypted storage, and controlled access.

4. Technological Disruption

New HR automation tools may alter EMS competitiveness.
Mitigation: Maintain modular, upgradable MERN architecture.

5. Operational Risks

Downtime, system bugs, or performance issues may impact user trust.
Mitigation: Regular updates, monitoring, backups, and performance testing.

Supplementary Information for Transparency

1. Calculation References

- DCF Method:
Include cash flow projections, discount rates, and assumptions for operational savings.

- Comparable Company Analysis:
List HRMS companies used for benchmarking.

- Precedent Transactions:
HR software acquisitions and their valuation multiples.

---

2. Supporting Data

- Operational impact reports
- HR software market statistics
- Growth projections for digital HR tools
- Internal performance benchmarks

---

3. Documentation

- Assumption Documentation:
Growth rate, savings estimation, adoption rate.

- Methodology Documentation:
Detailed explanation of valuation method choices.

- Risk Analysis:
Assessment based on HR industry volatility and digital adoption rate.

Example Calculations

Discounted Cash Flow (DCF) Example

1. Projected Annual Savings Through Automation:
   - Year 1: INR 100,000
   - Year 2: INR 110,000
   - Year 3: INR 120,000
   - Terminal Value (end of Year 3): INR 1,500,000
2. Discount Rate: 10%
3. Present Value:
   PV calculations remain same as original (updated for EMS context).

Comparable Company Analysis Example

Comparable HRMS Companies:
- HRMS A: EV/EBITDA = 8x
- HRMS B: EV/EBITDA = 9x
- HRMS C: EV/EBITDA = 7x

Average multiple = 8x

If EMS has estimated EBITDA of INR 500,000:
Enterprise Value = 8 × 500,000 = INR 4,000,000

Quality Control Measures for Valuation Accuracy

1. Data Validation
   Cross-checks for financial and operational data.
2. Reliable Data Sources
   Use audited reports and credible HR industry publications.
3. Standardized Models
   Use consistent DCF/CCA templates to avoid errors.
4. Scenario Analysis
   Test multiple workforce sizes and adoption scenarios.
5. Peer Review
   Independent review of valuation assumptions.
6. Regular Revisions
   Quarterly updates to valuation models based on organizational changes.
7. Transparent Documentation
   Maintain clear records of assumptions and methods.
8. Training & Continuous Improvement
   Ensure valuation team stays updated on HR tech trends.

HARDWARE AND SOFTWARE REQUIREMENTS

Server-Side (Hosting Server)

1. Processor
    - Minimum: Dual-core 2.0 GHz
    - Recommended: Quad-core 2.5 GHz or higher
2. Memory (RAM)
    - Minimum: 4 GB
    - Recommended: 8 GB or higher
3. Storage
    - Minimum: 50 GB HDD
    - Recommended: 100 GB SSD for better read/write performance
4. Network
    - Stable high-speed internet
    - Minimum: 10 Mbps | Recommended: 100 Mbps
5. Operating System
    - Linux (Ubuntu/CentOS) or Windows Server

Client-Side (User / Developer Computer)

1. Processor
    - Minimum: Dual-core 1.6 GHz
    - Recommended: Quad-core 2.0 GHz
2. Memory (RAM)
    - Minimum: 2 GB
    - Recommended: 4 GB or more
3. Storage
    - Minimum: 20 GB free
    - Recommended: 50 GB free
4. Network
    - Minimum: 2 Mbps | Recommended: 10 Mbps
5. Operating System
    - Windows 10+, macOS, or modern Linux distribution
6. Web Browser
    - Latest Chrome, Firefox, Edge, or Safari

Software Requirements

Server-Side

1. Runtime Environment
    - Node.js (v18+) and npm

2. Backend Framework
    - Express.js
3. Database Server
    - MongoDB (local or cloud — MongoDB Atlas)
4. Additional Tools
    - SSL certificate (for secure HTTPS)
    - Firewall/security tools

Development Environment

1. IDE/Text Editor
    - Visual Studio Code (recommended)
2. Version Control
    - Git + GitHub
3. Local Development Tools
    - Vite (for frontend dev), Nodemon (backend auto-reload)
4. Testing Tools
    - Postman or browser dev tools
5. Frontend Framework
    - React + TailwindCSS

System Specifications and Dependencies

1. React
    - For UI components and routing
    - Dependency: Vite, Tailwind, React Router
2. Node.js / Express
    - For server-side API and authentication
    - Dependency: bcryptjs, jsonwebtoken, multer, cors
3. MongoDB
    - For storing employee, task, attendance, and admin data
    - Dependency: Mongoose ODM
4. Axios
    - For frontend–backend communication

FUTURE SCOPE

The Employee Management System developed using React, TailwindCSS, Node.js, Express, and MongoDB provides a structured platform for managing employee records, tasks, attendance, and departmental operations. However, with continuous technological advancements and increasing organizational needs, several opportunities exist to expand the system's functionality, efficiency, and user experience.

---

1. Enhanced User Experience and Interface Design

Advanced Search and Filtering
The current system supports basic search capabilities. Future improvements can include:
- Keyword suggestions and autocomplete for faster employee and task lookup.
- Filters based on department, role, task status, and attendance patterns.
- Personalized dashboard elements based on user activity and role.

Responsive and Adaptive Design
Although the system is fully responsive, future versions may integrate Progressive Web App (PWA) features for offline access and mobile app–like behavior. Adaptive layouts can further optimize usability across various devices and screen types.

---

2. Integration with Modern Technologies

Artificial Intelligence and Machine Learning
- Smart task recommendations based on employee performance and workload.
- AI-driven chatbots to assist employees with queries regarding tasks, attendance, or system usage.
- NLP-based search to allow voice commands for finding employees, tasks, or reports.

Cloud Computing
Migrating to cloud infrastructure (AWS, Azure, or MongoDB Atlas) can offer:
- Scalable storage for employee documents and task histories.
- Continuous updates and deployment pipelines for new features.
- Enhanced reliability and built-in security protocols.

---

3. Expanded Functionalities

Employee Content & Knowledge Management
Future versions of the EMS can include:
- Employee knowledge base or internal documentation hub.
- Announcement boards for sharing updates, notices, and policies.
- Skill tracking modules for employee growth and training.

Community and Engagement Tools
- Peer feedback and rating system for task performance.
- Discussion groups or forums for inter-department communication.
- Gamification elements like badges for task completion, punctual attendance, and consistency.

---

4. Operational and Administrative Improvements

Automated Workflows
Backend processes can be improved with:
- Automated task reminders and deadline notifications.
- Auto-generation of attendance summaries for payroll purposes.
- Automated alerts for probation completion, onboarding steps, or document expiry.

Advanced Reporting and Analytics
Administrators may benefit from:
- Detailed insights on task completion rates, performance metrics, and attendance trends.
- Department-wise productivity and workload distribution charts.
- Customizable dashboards with real-time employee activity updates.

---

5. Security and Compliance

Enhanced Security Measures
To protect sensitive employee data, future enhancements may include:
- Multi-Factor Authentication (MFA) for admin and employee logins.
- Improved encryption for stored documents and API communication.
- Session monitoring and automatic logout for inactivity.

Compliance with Data Protection Regulations
To meet global data security standards, the EMS can adopt:
- Consent management and data access logs.
- Data anonymization for analytics and backups.
- Compliance frameworks similar to GDPR for secure employee data handling.

ADVANTAGES AND DISADVANTAGES OF THE TOUR AND TRAVEL WEBSITE

Advantages

1. Efficient Employee & Task Management

▪ Streamlines processes such as employee registration, task assignment, attendance tracking, and department management.
▪ Simplifies monitoring daily operations, deadlines, and work progress.
▪ Provides real-time updates on task completion, attendance logs, and employee activity.

2. Improved User Experience

▪ Offers a clean and user-friendly interface for employees and admins.
▪ Enables employees to view tasks, update status, check attendance, and manage personal profiles.
▪ Ensures accessibility across devices with responsive design and intuitive navigation.

3. Time and Cost Savings

▪ Reduces manual HR workload by automating attendance, task workflows, and record keeping.
▪ Minimizes paperwork through digital storage of documents, reports, and profiles.
▪ Lowers administrative costs by centralizing all employee-related functions in one system.

4. Data Accuracy and Integrity

▪ Maintains accurate and consistent records for employees, attendance, and tasks.
▪ Reduces errors through validation, controlled access, and standardized data formats.
▪ Stores data in a structured MongoDB database, ensuring easy retrieval and long-term stability.

5. Enhanced Reporting and Analytics

▪ Generates detailed reports on attendance trends, task performance, and departmental productivity.
▪ Provides insights to improve workload distribution and decision-making.
▪ Enables data-driven HR and administrative planning.

6. Scalability

- Easily scalable to accommodate growing employee counts, additional departments, and new modules.
- Supports future enhancements like payroll, performance reviews, and automated notifications.
- Can integrate external tools such as cloud storage or communication services.

7. Security

- Protects user data through JWT authentication, encrypted passwords (bcrypt), and secure APIs.
- Ensures safe access via role-based permissions for admins and employees.
- Regular updates and patches strengthen system integrity and reduce vulnerabilities.

Disadvantages

1. Initial Setup and Implementation Costs

- Requires investment in hosting servers, backend infrastructure, and deployment tools.
- Involves costs for UI development, database configuration, and API integration.
- Training admins and users to work with the system may add to the initial expenses.

2. Dependency on Technology

- Server downtime or technical issues can disrupt access to employee data or task updates.
- Requires continuous system maintenance for smooth operations.
- Dependence on stable internet connectivity for real-time updates.

3. Data Security and Privacy Concerns

- Sensitive employee information may be at risk if security standards are not followed.
- Compliance with data protection regulations requires additional planning and resources.
- Handling personal data, documents, and attendance logs demands strong security protocols.

4. Complexity of Migration

- Moving from manual HR processes to a digital EMS may be challenging.
- Migrating existing records into the new database requires careful validation.
- Data inconsistencies during migration can affect operational trust.

5. Customization and Integration Challenges

- Customizing the EMS to match organizational requirements may require technical expertise.
- Integration with payroll, biometrics, or third-party APIs may involve compatibility issues.
- Adding advanced modules increases development time and cost.

6. Learning Curve for Users and Staff

- Employees and administrators may need time to understand new workflows.
- Continuous support and training may be necessary to address early usability challenges.
- Resistance to switching from traditional or manual methods can slow adoption.

7. Ongoing Maintenance and Support

- Requires regular updates, bug fixes, and performance improvements.
- Technical support is needed to resolve issues and assist new users.
- Maintenance costs, including server hosting and storage, must be budgeted.

CODING

Frontend using ReactJS

```
// src/pages/landing-page/LandingPage.jsx
import React, { useEffect } from "react";
import { motion } from "framer-motion";
import { useNavigate } from "react-router-dom";

export default function LandingPage() {
  const navigate = useNavigate();

  useEffect(() => {
    const timer = setTimeout(() => {
      navigate("/login");
    }, 2000);

    return () => clearTimeout(timer);
  }, [navigate]);

  return (
    <div className="bg-blue-500 w-screen h-screen flex justify-center items-center">
      <motion.div
        className="flex flex-col items-center"
        initial={{ opacity: 0, scale: 1, rotateY: 90 }}
        animate={{ opacity: 1, scale: 0.8, rotateY: 0 }}
        transition={{
          duration: 2,
          ease: "easeInOut",
          repeatType: "reverse",
        }}
      >
        <img src="./website-logo.png" alt="Logo" className="w-[120px]" />
        <h2 className="text-white text-xl font-bold">Welcome to HRly</h2>
      </motion.div>
    </div>
  );
}
```

```
// src/pages/login/Login.jsx
import React, { useState } from "react";
import PageWrapper from "../../components/PageWrapper";
import { FaRegEye, FaRegEyeSlash } from "react-icons/fa";
import { FcGoogle } from "react-icons/fc";
import Swal from "sweetalert2";
import { useNavigate } from "react-router-dom";

export default function Login() {
  const [formData, setFormData] = useState({
    email: "",
    password: "",
  });

  const [showPassword, setShowPassword] = useState(false);
  const navigate = useNavigate();
```

```
  // Handle input change
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  // Handle form submission
  const handleSubmit = async (e) => {
    e.preventDefault();

    if (!formData.email || !formData.password) {
      Swal.fire("Incomplete Details", "Please fill all required fields!", "error");
      return;
    }

    try {
      const res = await fetch("http://localhost:5000/api/login", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(formData),
      });

      const data = await res.json();

      if (!res.ok) {
        Swal.fire("Login Failed", data.msg || "Invalid credentials", "error");
        return;
      }

      localStorage.setItem("token", data.token);
      localStorage.setItem("user", JSON.stringify(data.user));
      const role = data.user?.role?.toLowerCase();
      if (role === "manager") navigate("/manager/dashboard");
      else if (role === "employee") navigate("/employee/dashboard");

      Swal.fire("Login Successful!", "Redirecting to your dashboard...", "success");
    } catch (err) {
      console.error("Login error:", err);
      Swal.fire("Server Error", "Unable to connect to server.", "error");
    }
  };

  return (
    <PageWrapper>
      <div className="w-screen min-h-screen flex justify-center items-center bg-gradient-
to-r from-blue-500 via-purple-500 to-pink-500 p-4">
        {/* Outer card: responsive sizing */}
        <div className="flex flex-col md:flex-row bg-white/70 backdrop-blur-md border-b-2
border-r-2 shadow-lg rounded-xl overflow-hidden max-w-4xl w-full">
          {/* LEFT SIDE — LOGIN FORM */}
          <div className="w-full md:w-1/2 flex flex-col justify-center items-center p-8
md:p-10 bg-white/60">
            <h1 className="text-3xl font-semibold mb-2 text-gray-800">Welcome Back</h1>
            <p className="text-sm text-gray-600 mb-6 text-center">
              Please log in to access your Employee Dashboard
            </p>

            <form onSubmit={handleSubmit} className="flex flex-col w-full max-w-md">
              {/* Email */}
              <input
                type="email"
                name="email"
                placeholder="Email"
                className="placeholder-black/50 border-2 text-black border-transparent
shadow-black/5 shadow-[0px_0px_10px_0px] hover:border-purple-500 focus:border-pink-500 bg-
white p-3 focus:pl-4 mb-3 outline-0 rounded transition-all ease-in-out duration-500 w-full"
```

```
                value={formData.email}
                onChange={handleChange}
                required
              />

              {/* Password */}
              <div className="relative mb-3 w-full">
                <input
                  type={showPassword ? "text" : "password"}
                  name="password"
                  placeholder="Password"
                  className="placeholder-black/50 border-2 text-black border-transparent
shadow-black/5 shadow-[0px_0px_10px_0px] p-3 w-full outline-0 pr-10 hover:border-purple-500
focus:border-pink-500 rounded bg-white focus:pl-4 transition-all ease-in-out duration-500"
                  value={formData.password}
                  onChange={handleChange}
                  required
                />
                <button
                  type="button"
                  aria-label={showPassword ? "Hide password" : "Show password"}
                  className="absolute right-3 top-1/2 -translate-y-1/2 text-gray-600
hover:text-black transition"
                  onClick={() => setShowPassword(!showPassword)}
                >
                  {showPassword ? <FaRegEyeSlash size={20} /> : <FaRegEye size={20} />}
                </button>
              </div>

              {/* Forget password */}
              <div className="w-full flex justify-end mb-3">
                <a
                  href="/forget-password"
                  className="text-sm text-gray-700 hover:underline"
                >
                  Forgot password?
                </a>
              </div>

              {/* Login button */}
              <button
                type="submit"
                className="bg-black text-white p-3 rounded-md hover:bg-gray-800 transition
cursor-pointer w-full"
              >
                Login
              </button>

              {/* Google login */}
              <div className="flex flex-col justify-center items-center mt-4">
                <p className="text-gray-600 text-[13px] mb-2">Or login with</p>
                <button
                  type="button"
                  className="bg-white border border-gray-300 cursor-pointer text-black p-3
w-full flex justify-center items-center gap-2 rounded-md hover:bg-gray-100 transition max-
w-md"
                >
                  <FcGoogle /> Google
                </button>
              </div>
            </form>

            {/* Register redirect */}
            <p className="text-sm text-gray-700 mt-6 text-center">
              Don’t have an account?{" "}
              <a href="/register" className="text-purple-600 underline">
```

```
              Create one
            </a>
          </p>
        </div>

        {/* RIGHT SIDE — ILLUSTRATION / TEXT */}
        <div className="w-full md:w-1/2 flex flex-col justify-center items-center text-
white p-8 md:p-10 bg-gradient-to-br from-purple-600 via-pink-500 to-orange-400">
          <h2 className="text-3xl md:text-4xl font-bold mb-4">Login Portal</h2>
          <p className="text-base md:text-lg text-center mb-6 px-4 md:px-10">
            Manage your tasks, view updates, and stay connected with your team in one
place.
          </p>
          <img
            src="./website-logo.png"
            alt="Login illustration"
            className="w-36 md:w-48 drop-shadow-lg"
          />
        </div>
      </div>
    </div>
  </PageWrapper>
 );
}
```

```
// client/src/pages/Register.jsx
import React, { useState, useContext } from "react";
import { useNavigate } from "react-router-dom";
import PageWrapper from "../../components/PageWrapper";
import { FaRegEye, FaRegEyeSlash } from "react-icons/fa";
import { FaRegImage } from "react-icons/fa6";
import AIButton from "../../components/AIButton";
import Swal from "sweetalert2";
import { ThemeContext } from "../../context/ThemeContext";
import { FaArrowRight } from "react-icons/fa6";
import { FaArrowLeft } from "react-icons/fa";

function Register() {
  const [step, setStep] = useState(1);
  const navigate = useNavigate();
  const [showPassword, setShowPassword] = useState(false);
  const { isDark } = useContext(ThemeContext);

  const [formData, setFormData] = useState({
    // Step 1: Personal Details
    name: "",
    gender: "",
    dob: "",
    email: "",
    phone: "",
    address: "",
    city: "",
    state: "",
    pincode: "",
    country: "",
    emergencyContactName: "",
    emergencyPhone: "",

    // Step 2: Employee Details
    role: "",
    employeeId: "",
    department: "",
    designation: "",
    joiningDate: "",
```

```
    employmentType: "",
    reportingManager: "",
    workEmail: "",
    workMode: "",

    // Step 3: Documents & Account Setup
    photo: null,
    signature: null,
    password: "",
    confirm_password: "",
    securityQuestion: "",
    securityAnswer: "",
    termsAccepted: false,
  });

  const handleChange = (e) => {
    const { name, value, type, checked, files } = e.target;
    setFormData({
      ...formData,
      [name]: type === "checkbox" ? checked : files ? files[0] : value,
    });
  };

  const nextStep = () => setStep((prev) => prev + 1);
  const prevStep = () => setStep((prev) => prev - 1);

  const validateStep1 = () => {
    const {
      name,
      gender,
      dob,
      email,
      phone,
      address,
      city,
      state,
      pincode,
      country,
      emergencyPhone,
    } = formData;

    if (
      !name ||
      !gender ||
      !dob ||
      !email ||
      !phone ||
      !address ||
      !city ||
      !state ||
      !pincode ||
      !country
    ) {
      Swal.fire(
        "Incomplete Details",
        "Please fill all required fields!",
        "error"
      );
      return false;
    }

    // Validate age >= 18
    const birthDate = new Date(dob);
    const today = new Date();
    const age =
      today.getFullYear() -
```

```
      birthDate.getFullYear() -
      (today < new Date(birthDate.setFullYear(today.getFullYear())) ? 1 : 0);
    if (age < 18) {
      Swal.fire(
        "Underage!",
        "You must be at least 18 years old to register.",
        "error"
      );
      return false;
    }

    // Phone validation
    const phoneRegex = /^[0-9]{10}$/;
    if (!phoneRegex.test(phone)) {
      Swal.fire("Invalid Phone", "Phone number must be 10 digits.", "error");
      return false;
    }

    // Email validation
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
      Swal.fire(
        "Invalid Email",
        "Please enter a valid email address.",
        "error"
      );
      return false;
    }

    // Emergency number validation (if entered)
    if (emergencyPhone && !phoneRegex.test(emergencyPhone)) {
      Swal.fire(
        "Invalid Emergency Contact",
        "Emergency phone must be 10 digits.",
        "error"
      );
      return false;
    }

    return true;
  };

  const validateStep2 = () => {
    const {
      role,
      employeeId,
      department,
      designation,
      joiningDate,
      employmentType,
      reportingManager,
      workMode,
    } = formData;

    if (!role) {
      Swal.fire("Select Role", "Please select your role.", "error");
      return false;
    }

    if (!employeeId) {
      Swal.fire(
        "Employee ID Missing",
        "Please enter your Employee ID.",
        "error"
      );
      return false;
```

```
    }

    // Role-based employee ID format (if you want strict checks; it won't block if role not
included)
    const rolePrefixes = {
      Employee: /^EMP\d{6}$/,
      HR: /^HRM\d{6}$/,
      Manager: /^MAG\d{6}$/,
      Admin: /^ADM\d{6}$/,
    };

    if (rolePrefixes[role] && !rolePrefixes[role].test(employeeId)) {
      Swal.fire(
        `Invalid ${role} ID`,
        `For ${role}, format must be ${role.toUpperCase()} + 6 digits (e.g. ${
          role === "Employee"
            ? "EMP123456"
            : role === "HR"
            ? "HRM654876"
            : role === "Manager"
            ? "MAG456789"
            : "ADM561894"
        })`,
        "error"
      );
      return false;
    }

    if (
      !department ||
      !designation ||
      !joiningDate ||
      !employmentType ||
      !reportingManager ||
      !workMode
    ) {
      Swal.fire(
        "Missing Info",
        "Please complete all official details!",
        "error"
      );
      return false;
    }

    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(reportingManager)) {
      Swal.fire(
        "Invalid Manager Email",
        "Enter a valid manager email address.",
        "error"
      );
      return false;
    }

    return true;
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    const {
      photo,
      signature,
      password,
      confirm_password,
      securityQuestion,
      securityAnswer,
```

```
    termsAccepted,
  } = formData;

  if (
    !photo ||
    !signature ||
    !password ||
    !confirm_password ||
    !securityQuestion ||
    !securityAnswer
  ) {
    Swal.fire(
      "Incomplete Details",
      "Please complete all required fields!",
      "error"
    );
    return;
  }

  if (!termsAccepted) {
    Swal.fire(
      "Accept Terms",
      "You must accept Terms and Conditions.",
      "error"
    );
    return;
  }

  if (password !== confirm_password) {
    Swal.fire("Password Mismatch", "Passwords do not match!", "error");
    return;
  }

  try {
    const formDataToSend = new FormData();

    Object.keys(formData).forEach((key) => {
      formDataToSend.append(key, formData[key]);
    });

    const response = await fetch("http://localhost:5000/api/register", {
      method: "POST",
      body: formDataToSend,
    });

    const data = await response.json();

    if (response.ok) {
      Swal.fire(
        "✅ Success",
        data.msg || "Registration successful!",
        "success"
      );
      navigate(`/${formData.role.toLowerCase()}/dashboard`);
    } else {
      Swal.fire("❌ Error", data.msg || "Registration failed", "error");
    }
  } catch (error) {
    console.error("Registration error:", error);
    Swal.fire("Registration Error", `Something went wrong. ${error}`, "error");
  }
};

// input styles: augmented for dark mode
const inputStyleBase =
```

```
    "border-2 border-transparent hover:border-purple-500 focus:border-pink-500 p-2
focus:pl-4 m-2 outline-0 rounded transition-all ease-in-out duration-500";
  const inputLight = "bg-gray-200 text-slate-900";
  const inputDark = "bg-slate-700 text-slate-100 border-slate-700 placeholder-slate-400";
  const inputStyle = `${inputStyleBase} ${isDark ? inputDark : inputLight}`;

  // card / container classes
  const pageBg = isDark ? "bg-slate-900" : "bg-white";
  const formBg = isDark ? "bg-slate-900 text-slate-100" : "bg-white text-slate-900";
  const leftText = isDark ? "text-white" : "text-white"; // left panel remains white text
on gradient
  const mainContainerBg = isDark ? "bg-slate-800/60 backdrop-blur-md border-slate-700" :
"bg-white/70 backdrop-blur-md border-gray-200";

  // upload card styles
  const uploadBase = "flex flex-col w-full md:w-1/2 border-2 border-dashed rounded-xl p-4
justify-center items-center text-center transition cursor-pointer";
  const uploadLight = (has) =>
    has ? "border-green-400 bg-green-50 hover:bg-green-100 text-slate-900" : "border-gray-
400 hover:bg-gray-100 text-slate-900";
  const uploadDark = (has) =>
    has ? "border-green-600 bg-green-900/30 hover:bg-green-900/20 text-slate-100" :
"border-slate-700 hover:bg-slate-800 text-slate-100";

  return (
    <PageWrapper>
      {/* prevent x overflow globally for this page */}
      <div className={`min-h-screen min-w-screen overflow-x-hidden ${pageBg}`}>
        {/* responsive container: columns on md+, stacked on small screens */}
        <div className="min-h-screen flex flex-col md:flex-row">
          {/* LEFT (visual) - fixed height on large screens, non-scrollable */}
          <aside className="w-full md:w-1/2 h-auto md:h-screen sticky top-0 flex-shrink-0
bg-gradient-to-r from-purple-500 to-pink-500 flex items-center justify-center p-8 md:p-12">
            <div className="max-w-md text-center">
              <h1 className="text-4xl md:text-5xl font-bold mb-4" style={{ color: leftText
=== "text-white" ? "white" : undefined }}>Register Here</h1>
              <p className="text-base md:text-lg text-white/90 mb-4">
                Create your account to get started with our Employee Management
                System. Fill in your details carefully to help HR onboard you
                smoothly.
              </p>
              <p className="text-base md:text-lg text-white/90 border-t pt-3">
                <a href="/login" className="hover:underline">Login</a> to your existing
account.
              </p>
            </div>
          </aside>

          {/* RIGHT (form) - scrollable only */}
          <main className="w-full md:w-1/2 h-screen overflow-y-auto">
            <div className={`w-full md:max-w-xl mx-auto p-6 md:p-8 ${isDark ? "bg-slate-900
text-slate-100" : "bg-white text-slate-900"}`}>
              <h2 className="text-2xl font-semibold mb-2 text-center">Create your
Account</h2>
              <p className="text-[13px] mb-4 text-center">
                Already have an account?{" "}
                <a href="/login" className={isDark ? "text-indigo-300 underline" : "text-
blue-700 underline"}>
                  Login
                </a>
              </p>
              <p className="text-[13px] mb-6 text-center" style={{ color: isDark ?
"#cbd5e1" : undefined }}>
                Step {step} of 3
              </p>
```

```
              <form onSubmit={handleSubmit} className="flex flex-col w-full transition-all
duration-500">
                {/* STEP 1 */}
                {step === 1 && (
                  <>
                    <input
                      type="text"
                      name="name"
                      placeholder="Full Name"
                      className={inputStyle}
                      onChange={handleChange}
                      value={formData.name}
                    />
                    <select
                      name="gender"
                      className={inputStyle}
                      onChange={handleChange}
                      value={formData.gender}
                    >
                      <option value="">Select Gender</option>
                      <option value="Male">Male</option>
                      <option value="Female">Female</option>
                      <option value="Other">Other</option>
                    </select>

                    <label className={`m-2 font-medium text-sm ${isDark ? "text-slate-200"
: "text-gray-700"}`}>Date of Birth</label>
                    <input
                      type="date"
                      name="dob"
                      className={`${inputStyle} text-black hide-date-icon ${isDark ? "text-
slate-100" : ""}`}
                      onChange={handleChange}
                      value={formData.dob}
                    />

                    <input
                      type="email"
                      name="email"
                      placeholder="Email Address"
                      className={inputStyle}
                      onChange={handleChange}
                      value={formData.email}
                    />
                    <input
                      type="text"
                      name="phone"
                      placeholder="Phone Number"
                      className={inputStyle}
                      onChange={handleChange}
                      value={formData.phone}
                    />
                    <textarea
                      name="address"
                      placeholder="Full Address"
                      className={`${inputStyle} resize-none`}
                      onChange={handleChange}
                      value={formData.address}
                    />
                    <AIButton name="Address" />

                    <div className="grid grid-cols-1 sm:grid-cols-2 gap-2">
                      <input
                        type="text"
                        name="city"
                        placeholder="City"
```

```
                    className={inputStyle + " m-0"}
                    onChange={handleChange}
                    value={formData.city}
                  />
                  <input
                    type="text"
                    name="state"
                    placeholder="State"
                    className={inputStyle + " m-0"}
                    onChange={handleChange}
                    value={formData.state}
                  />
                </div>

                <div className="grid grid-cols-1 sm:grid-cols-2 gap-2 mt-2">
                  <input
                    type="text"
                    name="pincode"
                    placeholder="Pincode"
                    className={inputStyle + " m-0"}
                    onChange={handleChange}
                    value={formData.pincode}
                  />
                  <input
                    type="text"
                    name="country"
                    placeholder="Country"
                    className={inputStyle + " m-0"}
                    onChange={handleChange}
                    value={formData.country}
                  />
                </div>

                <div className="grid grid-cols-1 sm:grid-cols-2 gap-2 mt-2">
                  <input
                    type="text"
                    name="emergencyContactName"
                    placeholder="Emergency Contact Name"
                    className={inputStyle + " m-0"}
                    onChange={handleChange}
                    value={formData.emergencyContactName}
                  />
                  <input
                    type="text"
                    name="emergencyPhone"
                    placeholder="Emergency Contact Number"
                    className={inputStyle + " m-0"}
                    onChange={handleChange}
                    value={formData.emergencyPhone}
                  />
                </div>

                <div className="flex justify-end mt-4">
                  <button
                    type="button"
                    onClick={() => { if (validateStep1()) nextStep(); }}
                    className="flex items-center justify-center gap-3 bg-black text-
white p-3 cursor-pointer rounded hover:bg-gray-800 transition"
                  >
                    Next <FaArrowRight />
                  </button>
                </div>
              </>
            )}

            {/* STEP 2 */}
```

```
                {step === 2 && (
                  <>
                    <select
                      name="role"
                      className={inputStyle}
                      onChange={handleChange}
                      value={formData.role}
                    >
                      <option value="">Select Role</option>
                      <option value="Manager">Manager</option>
                      <option value="Employee">Employee</option>
                    </select>

                    {formData.role && (
                      <input
                        type="text"
                        name="employeeId"
                        placeholder={`${formData.role} ID`}
                        className={inputStyle}
                        onChange={handleChange}
                        value={formData.employeeId}
                      />
                    )}

                    <select
                      name="department"
                      className={inputStyle}
                      onChange={handleChange}
                      value={formData.department}
                    >
                      <option value="">Select Department</option>
                      <option value="Software Development">Software Development</option>
                      <option value="Quality Assurance">Quality Assurance</option>
                      <option value="UI/UX Design">UI/UX Design</option>
                      <option value="DevOps">DevOps</option>
                      <option value="Data Science">Data Science</option>
                    </select>

                    <select
                      name="designation"
                      className={inputStyle}
                      onChange={handleChange}
                      value={formData.designation}
                    >
                      <option value="">Select Designation</option>
                      <option value="Junior Developer">Junior Developer</option>
                      <option value="Senior Developer">Senior Developer</option>
                      <option value="Team Lead">Team Lead</option>
                      <option value="Project Manager">Project Manager</option>
                      <option value="HR Executive">HR Executive</option>
                      <option value="Intern">Intern</option>
                    </select>

                    <label className={`m-2 font-medium text-sm ${isDark ? "text-slate-200"
: "text-gray-700"}`}>Date of Joining</label>
                    <input
                      type="date"
                      name="joiningDate"
                      className={`${inputStyle} hide-date-icon`}
                      onChange={handleChange}
                      value={formData.joiningDate}
                    />

                    <select
                      name="employmentType"
                      className={inputStyle}
```

```
                    onChange={handleChange}
                    value={formData.employmentType}
                  >
                    <option value="">Select Employment Type</option>
                    <option value="Full-Time">Full-Time</option>
                    <option value="Part-Time">Part-Time</option>
                    <option value="Intern">Intern</option>
                    <option value="Contract">Contract</option>
                  </select>

                  <input
                    type="email"
                    name="reportingManager"
                    placeholder="Reporting Manager Email"
                    className={inputStyle}
                    onChange={handleChange}
                    value={formData.reportingManager}
                  />
                  <input
                    type="email"
                    name="workEmail"
                    placeholder="Work Email (if provided)"
                    className={inputStyle}
                    onChange={handleChange}
                    value={formData.workEmail}
                  />

                  <select
                    name="workMode"
                    className={inputStyle}
                    onChange={handleChange}
                    value={formData.workMode}
                  >
                    <option value="">Select Work Mode</option>
                    <option value="Onsite">Onsite</option>
                    <option value="Remote">Remote</option>
                    <option value="Hybrid">Hybrid</option>
                  </select>

                  <div className="flex gap-3 mt-4">
                    <button
                      type="button"
                      onClick={prevStep}
                      className="flex items-center gap-3 justify-center bg-gray-300 text-
black w-1/2 p-3 rounded hover:bg-gray-400 transition"
                    >
                      <FaArrowLeft /> Back
                    </button>
                    <button
                      type="button"
                      onClick={() => { if (validateStep2()) nextStep(); }}
                      className="flex items-center justify-center gap-3 bg-black text-
white w-1/2 p-3 cursor-pointer rounded hover:bg-gray-800 transition"
                    >
                      Next <FaArrowRight />
                    </button>
                  </div>
                </>
              )}

              {/* STEP 3 */}
              {step === 3 && (
                <>
                  <div className="flex flex-col md:flex-row gap-4 m-2">
                    {/* PHOTO UPLOAD */}
                    <div
```

```
                        className={`${uploadBase} ${isDark ? uploadDark(!!formData.photo) :
uploadLight(!!formData.photo)}`}
                      >
                        {formData.photo ? (
                          <img
                            src={URL.createObjectURL(formData.photo)}
                            alt="Uploaded"
                            className="w-24 h-24 object-cover rounded-lg mb-2"
                          />
                        ) : (
                          <FaRegImage size={40} className="text-gray-500 mb-2" />
                        )}

                        <p className={`text-sm ${formData.photo ? "text-green-600" : isDark
? "text-slate-300" : "text-gray-600"}`}>
                          {formData.photo ? "✅ Photo uploaded successfully!" : "Upload a
file or drag and drop PNG, JPG (max 1 MB)"}
                        </p>

                        <input
                          type="file"
                          name="photo"
                          accept="image/*"
                          onChange={handleChange}
                          className="mt-2 text-xs text-blue-700 font-bold cursor-pointer"
                        />

                        {formData.photo && <p className="text-xs text-gray-600 mt-1">📄
{formData.photo.name}</p>}
                      </div>

                      {/* SIGNATURE UPLOAD */}
                      <div
                        className={`${uploadBase} ${isDark ?
uploadDark(!!formData.signature) : uploadLight(!!formData.signature)}`}
                      >
                        {formData.signature ? (
                          <img
                            src={URL.createObjectURL(formData.signature)}
                            alt="Signature"
                            className="w-24 h-16 object-contain rounded-lg mb-2"
                          />
                        ) : (
                          <FaRegImage size={40} className="text-gray-500 mb-2" />
                        )}

                        <p className={`text-sm ${formData.signature ? "text-green-600" :
isDark ? "text-slate-300" : "text-gray-600"}`}>
                          {formData.signature ? "✅ Signature uploaded successfully!" :
"Upload a file or drag and drop PNG, JPG (max 1 MB)"}
                        </p>

                        <input
                          type="file"
                          name="signature"
                          accept="image/*"
                          onChange={handleChange}
                          className="mt-2 text-xs text-blue-700 font-bold cursor-pointer"
                        />

                        {formData.signature && <p className="text-xs text-gray-600 mt-1">📄
{formData.signature.name}</p>}
                      </div>
                    </div>

                    <div className="relative m-2">
```

```
                    <input
                      type={showPassword ? "text" : "password"}
                      name="password"
                      placeholder="Create Password"
                      className={`border-2 border-transparent p-2 w-full outline-0 pr-10
hover:border-purple-500 focus:border-pink-500 rounded ${isDark ? "bg-slate-700 text-slate-
100" : "bg-gray-200 text-slate-900"} focus:pl-4 transition-all ease-in-out duration-500`}
                      onChange={handleChange}
                      value={formData.password}
                      required
                    />
                    <button
                      type="button"
                      aria-label={showPassword ? "Hide password" : "Show password"}
                      className="absolute right-3 top-1/2 -translate-y-1/2 text-gray-600
hover:text-black transition"
                      onClick={() => setShowPassword(!showPassword)}
                    >
                      {showPassword ? <FaRegEyeSlash size={20} /> : <FaRegEye size={20}
/>}
                    </button>
                  </div>

                  <input
                    type="password"
                    name="confirm_password"
                    placeholder="Confirm Password"
                    className={inputStyle}
                    onChange={handleChange}
                    value={formData.confirm_password}
                  />

                  <select
                    name="securityQuestion"
                    className={inputStyle}
                    onChange={handleChange}
                    value={formData.securityQuestion}
                  >
                    <option value="">Select Security Question</option>
                    <option value="Your first pet's name?">Your first pet's
name?</option>
                    <option value="Your mother's maiden name?">Your mother's maiden
name?</option>
                    <option value="Your favorite teacher's name?">Your favorite teacher's
name?</option>
                    <option value="Your birth city?">Your birth city?</option>
                  </select>

                  <input
                    type="text"
                    name="securityAnswer"
                    placeholder="Security Answer"
                    className={inputStyle}
                    onChange={handleChange}
                    value={formData.securityAnswer}
                  />

                  <div className="flex items-center m-2">
                    <input
                      type="checkbox"
                      name="termsAccepted"
                      checked={formData.termsAccepted}
                      onChange={handleChange}
                      className="mr-2 mt-1"
                    />
                    <label className={`text-sm ${isDark ? "text-slate-200" : ""}`}>
```

```
                      I agree to the{" "}
                      <span className="text-blue-700 underline cursor-pointer">
                        Terms and Conditions
                      </span>
                    </label>
                  </div>

                  <div className="flex gap-3 mt-4">
                    <button
                      type="button"
                      onClick={prevStep}
                      className="bg-gray-300 text-black p-2 flex-1 rounded hover:bg-gray-
400 transition"
                    >
                      Back
                    </button>
                    <button
                      type="submit"
                      className="bg-black text-white p-2 flex-1 rounded hover:bg-gray-800
transition"
                    >
                      Submit
                    </button>
                  </div>
                </>
              )}
            </form>
          </div>
        </main>
      </div>
    </div>
  </PageWrapper>
  );
}

export default Register;
```

```
// client/src/pages/showUsers/UsersList.jsx
import React, { useContext, useMemo, useState } from "react";
import { UserContext } from "../context/UserContext";

export default function UsersList() {
  const { users, loading, error, refreshUsers, getEmployees } = useContext(UserContext);
  const [query, setQuery] = useState("");
  const [showOnlyEmployees, setShowOnlyEmployees] = useState(true);

  // memoized list depending on toggle / query
  const displayed = useMemo(() => {
    if (showOnlyEmployees) return getEmployees(query || null);
    if (!query) return users;
    const q = query.toLowerCase();
    return users.filter((u) => (u.name || "").toLowerCase().includes(q) || (u.email ||
"").toLowerCase().includes(q) || (u.employeeId || "").toLowerCase().includes(q));
  }, [users, showOnlyEmployees, query, getEmployees]);

  if (loading) return <div className="p-4">Loading users…</div>;
  if (error) return <div className="p-4 text-red-600">Error loading users.</div>;

  return (
    <div className="p-4">
      <div className="mb-4 flex gap-3 items-center">
        <input
          type="text"
          value={query}
          onChange={(e) => setQuery(e.target.value)}
```

```
          placeholder="Search by name, email or ID"
          className="border p-2 rounded w-72"
        />
        <label className="flex items-center gap-2">
          <input
            type="checkbox"
            checked={showOnlyEmployees}
            onChange={() => setShowOnlyEmployees((v) => !v)}
            className="h-4 w-4"
          />
          <span>Show only employees</span>
        </label>

        <button
          onClick={() => refreshUsers()}
          className="ml-auto bg-blue-600 text-white px-3 py-1 rounded"
        >
          Refresh
        </button>
      </div>

      <div className="grid gap-3">
        {displayed.length === 0 ? (
          <div className="text-gray-500">No users found.</div>
        ) : (
          displayed.map((u) => (
            <div key={u._id || u.id || u.email} className="p-3 border rounded flex items-
center gap-4">
              <div className="w-12 h-12 bg-gray-100 rounded overflow-hidden flex-shrink-0">
                {u.photo ? (
                  <img src={u.photo.startsWith("http") ? u.photo :
`${process.env.REACT_APP_API_URL || "http://localhost:5000"}/uploads/${u.photo}`}
alt={u.name} className="w-full h-full object-cover" />
                ) : (
                  <div className="flex items-center justify-center h-full text-gray-400">No
Photo</div>
                )}
              </div>

              <div className="flex-1">
                <div className="font-medium">{u.name || "—"}</div>
                <div className="text-sm text-gray-600">{u.email}</div>
                <div className="text-xs text-gray-500">
                  {u.role} {u.employeeId ? `• ${u.employeeId}` : ""}
                </div>
              </div>

              <div>
                <button className="text-sm text-blue-600 hover:underline">View</button>
              </div>
            </div>
          ))
        )}
      </div>
    </div>
  );
}
```

---

client/src/components/dashboard-page-components/DashboardNav.jsx

```
 import React, { useEffect, useState, useContext } from "react";
import { href, NavLink, useNavigate } from "react-router-dom";
import NetworkIcon from "../NetworkStatus";
import BatteryStatus from "../BatteryStatus";
import { IoIosArrowBack, IoIosArrowForward } from "react-icons/io";
```

```jsx
import { MdSearch, MdArrowDropDown } from "react-icons/md";
import { RiNotification3Fill } from "react-icons/ri";
import { SiTicktick } from "react-icons/si";
import { FiRefreshCcw } from "react-icons/fi";
import ThemeButton from "../ThemeButton";
import { ThemeContext } from "../../context/ThemeContext";

/**
 * DashboardNav
 * - Responsive behavior:
 *   - md+ : full layout (search input, datetime, icons, country)
 *   - <md : compact layout: only NetworkIcon, small arrows, Refresh, ThemeButton, profile
pic
 *   - mobile search: tap search icon to open small overlay input
 *
 * - Search behavior:
 *   - controlled input
 *   - Enter or click triggers navigation to /search?query=...
 *
 * Keep design exactly as original — only visibility and small interactive behaviors
changed.
 */

function DashboardNav({ role = "Employee", photo = "default.jpg", name = "User",
designation = "", country = "IN" }) {
  const { isDark } = useContext(ThemeContext);
  const navigate = useNavigate();

  // date / time
  const [dateTime, setDateTime] = useState(new Date());
  useEffect(() => {
    const interval = setInterval(() => setDateTime(new Date()), 1000);
    return () => clearInterval(interval);
  }, []);

  const handleRefresh = () => window.location.reload();

  // search state & mobile search toggle
  const [searchQuery, setSearchQuery] = useState("");
  const [showMobileSearch, setShowMobileSearch] = useState(false);

  const handleSearch = (q = searchQuery) => {
    const trimmed = (q || "").trim();
    if (!trimmed) return;
    // route to a search results page (adjust path if your app uses different route)
    navigate(`/search?query=${encodeURIComponent(trimmed)}`);
    setShowMobileSearch(false);
  };

  const onKeyDownSearch = (e) => {
    if (e.key === "Enter") handleSearch();
  };

  // small-screen text color helper
  const grayTextClass = isDark ? "text-gray-300" : "text-gray-700";

  return (
    <nav
      className={`flex justify-between items-center w-full fadeClass ${
        isDark ? "bg-app text-app border-white/20" : "bg-app text-app border-black/20"
      } shadow-md px-4 py-3 md:px-8 border-b-2`}
    >
      {/* LEFT: network, role + back/forward */}
      <div className="flex items-center gap-3">
        <NetworkIcon />
```

```
        {/* role text: visible on md and up */}
        <p className={`text-lg font-semibold hidden sm:block ${grayTextClass}`}>
          {role.toUpperCase()}
        </p>

        {/* nav arrows: keep them small on all sizes, but hide on smallest if you prefer
*/}
        <div className={`flex items-center gap-2 ${grayTextClass}`}>
          <IoIosArrowBack className="cursor-pointer hidden sm:block hover:text-black"
onClick={() => navigate(-1)} />
          <IoIosArrowForward className="cursor-pointer hidden sm:block hover:text-black"
onClick={() => navigate(1)} />
        </div>
      </div>

      {/* MIDDLE: desktop search + refresh + date/time.
          - hidden on small screens (md:hidden)
          - on small screens we show a compact bar via icon (see right section)
      */}
      <div className="hidden md:flex items-center gap-4">
        {/* Search (desktop) */}
        <div className="relative">
          <MdSearch size={20} className="absolute left-3 top-1/2 -translate-y-1/2 text-
gray-500" />
          <input
            type="text"
            placeholder="Search..."
            value={searchQuery}
            onChange={(e) => setSearchQuery(e.target.value)}
            onKeyDown={onKeyDownSearch}
            className="border border-gray-300 rounded-full pl-10 pr-4 py-2 text-sm outline-
none focus:ring-1 focus:ring-gray-400"
            aria-label="Search"
          />
        </div>

        {/* Refresh */}
        <button
          onClick={handleRefresh}
          className={`flex items-center gap-2 px-3 py-2 cursor-pointer ${isDark ? "text-
gray-300 hover:bg-gray-800" : "text-gray-700 hover:bg-gray-100"} rounded-md outline-1
outline-gray-300 active:scale-95 transition-all`}
          title="Refresh"
        >
          <FiRefreshCcw className="text-lg" />
          <span>Refresh</span>
        </button>

        {/* Date & time pill */}
        <div className={`text-center ${isDark ? "text-gray-300" : "text-gray-500"}`}>
          <p>
            {dateTime.toLocaleDateString()}{" "}
            <span className={`p-2 rounded ${isDark ? "bg-blue-900" : "bg-blue-100"}`}>
              {dateTime.toLocaleTimeString()}
            </span>
          </p>
        </div>
      </div>

      {/* RIGHT: icons, profile */}
      <div className="flex items-center gap-5">
        {/* Icon set - NOTE: many icons hidden on small screens to keep layout clean */}
        <div className="hidden sm:flex items-center gap-3 text-gray-600">
          <BatteryStatus className={`${isDark ? "bg-blue-900" : "bg-blue-100"}`} />
          <NavLink to={`/${role.toLowerCase()}/notifications`}>
```

```
            <RiNotification3Fill size={22} className={`cursor-pointer ${isDark ? "text-
gray-300" : "text-gray-700"}`} />
          </NavLink>
          <ThemeButton />
          <p className={`px-2 py-1 rounded-md text-xs font-medium ${isDark ? "bg-blue-900
text-gray-300" : "bg-blue-100 text-gray-700"}`}>
            {country}
          </p>
        </div>

        {/* For smallest devices: compact controls (search icon, refresh, theme, profile)
            - search icon toggles a small overlay search input
            - these buttons are visible on small screens (sm:hidden)
        */}
        <div className="flex items-center gap-3 sm:hidden">
          {/* mobile search toggle */}
          <button
            onClick={() => setShowMobileSearch((s) => !s)}
            aria-label="Open search"
            className={`p-2 rounded ${isDark ? "text-gray-300 hover:bg-gray-800" : "text-
gray-700 hover:bg-gray-100"}`}
            title="Search"
          >
            <MdSearch size={20} />
          </button>

          {/* keep refresh on small screens */}
          <button
            onClick={handleRefresh}
            aria-label="Refresh"
            className={`p-2 rounded ${isDark ? "text-gray-300 hover:bg-gray-800" : "text-
gray-700 hover:bg-gray-100"}`}
            title="Refresh"
          >
            <FiRefreshCcw size={18} />
          </button>

          {/* theme toggle */}
          <div className="p-0">
            <ThemeButton />
          </div>
        </div>

        {/* User Info — profile picture always visible;
            name/designation only appear on >= sm
        */}
        <div className={`flex items-center gap-2 border-l border-gray-200 pl-4 ${isDark ?
"border-gray-700" : ""}`}>
          <NavLink to={`/${role.toLowerCase()}/profile`} className="flex items-center gap-1
cursor-pointer">
            <img src={`http://localhost:5000/uploads/${photo}`} alt="User" className="w-9
h-9 rounded-full object-cover border-2" />
            <MdArrowDropDown className={`text-gray-600 hidden sm:block`} />
          </NavLink>

          {/* name + designation hidden on very small, show on sm+ */}
          <div className="hidden sm:block">
            <h3 className={`text-sm font-semibold ${isDark ? "text-gray-300" : "text-gray-
700"}`}>{name}</h3>
            <p className="text-xs text-gray-500">{designation}</p>
          </div>

          <SiTicktick size={20} className="text-green-500 hidden sm:block" />
        </div>
      </div>
```

```
      {/* Mobile search overlay — appears when showMobileSearch is true (small only) */}
      {showMobileSearch && (
        <div className={`fixed inset-0 z-50 flex items-start justify-center p-4
sm:hidden`}>
          {/* background dim */}
          <div className="absolute inset-0 bg-black/40" onClick={() =>
setShowMobileSearch(false)} />
          <div className={`relative w-full max-w-md mt-12`}>
            <div className={`flex items-center gap-2 p-2 rounded-full ${isDark ? "bg-slate-
700" : "bg-white"} shadow`}>
              <MdSearch size={18} className="ml-2 text-gray-500" />
              <input
                value={searchQuery}
                onChange={(e) => setSearchQuery(e.target.value)}
                onKeyDown={onKeyDownSearch}
                placeholder="Search..."
                className={`flex-1 outline-none bg-transparent px-2 ${isDark ? "text-gray-
200" : "text-gray-800"}`}
              />
              <button
                onClick={() => handleSearch()}
                className={`px-3 py-1 rounded ${isDark ? "text-gray-200" : "text-gray-
800"}`}
              >
                Search
              </button>
              <button onClick={() => setShowMobileSearch(false)} className="p-2 text-gray-
500">
                ✕
              </button>
            </div>
          </div>
        </div>
      )}
    </nav>
  );
}

export default DashboardNav;
```

```
// client/src/components/dashboard-page-components/DashboardSidebar.jsx
import React, { useContext } from "react";
import { NavLink } from "react-router-dom";
import {
  MdDashboard,
  MdPerson,
  MdAccessTime,
  MdAssignment,
  MdMessage,
  MdSettings,
  MdNotifications,
  MdLogout,
  MdRequestPage,
} from "react-icons/md";

import { ThemeContext } from "../../context/ThemeContext";

function DashboardSidebar({ user }) {
  let navUser = user.toLowerCase();
  const items = [
    { name: "Dashboard", icon: <MdDashboard />, path: `/${navUser}/dashboard` },
    { name: "My Profile", icon: <MdPerson />, path: `/${navUser}/profile` },
    { name: "Attendance", icon: <MdAccessTime />, path: `/${navUser}/attendance` },
    { name: "Leave Requests", icon: <MdRequestPage />, path: `/${navUser}/leave-requests`
},
```

```
    { name: "Tasks / Projects", icon: <MdAssignment />, path: `/${navUser}/tasks` },
    { name: "Messages", icon: <MdMessage />, path: `/${navUser}/messages` },
    { name: "Settings", icon: <MdSettings />, path: `/${navUser}/settings` },
    { name: "Notifications", icon: <MdNotifications />, path: `/${navUser}/notifications`
},
    { name: "Logout", icon: <MdLogout />, path: `/${navUser}/logout` },
  ];

  const { isDark } = useContext(ThemeContext);

  return (
    <div
      className={`h-full w-full ${
        isDark ? "bg-app text-app border-white/20" : "bg-app text-app border-gray-300"
      } text-black flex flex-col justify-between items-start shadow-lg px-3 sm:px-5 border-
r-2`}
    >
      {/* Logo / Title - keep same look on desktop, compact on small */}
      <div className="flex items-left justify-start py-5 text-2xl font-bold border-b
border-white/20 w-full">
        {/* show short label on very small screens to save space */}
        <span className="hidden sm:inline">HRly</span>
        <span className="inline sm:hidden">H</span>
      </div>

      {/* Sidebar Items */}
      {/* on small screens we make this an icon rail: center icons and hide labels */}
      <ul className="flex flex-col gap-3 w-full mt-4">
        {items.map((item, index) => (
          <li key={index} className="w-full">
            <NavLink
              to={item.path}
              className={({ isActive }) =>
                `flex items-center gap-3 sm:px-5 py-2 rounded-md transition-all duration-
300
                  ${isDark ? "hover:bg-white/20" : ""}
                  ${
                    isActive
                      ? "bg-white/10 text-blue-600 font-semibold"
                      : "hover:bg-black/20"
                  }`
              }
            >
              {/* Icon: always visible. center on small screens */}
              <span className="text-lg flex-shrink-0 w-6 h-6 flex items-center justify-
center">
                {item.icon}
              </span>

              {/* Label: hidden on small screens (sm+) */}
              <span className="text-sm hidden sm:inline">{item.name}</span>
            </NavLink>
          </li>
        ))}
      </ul>

      {/* Footer / Version */}
      <div className="text-left text-xs p-4 border-t border-white/20 w-full">
        <span className="hidden sm:inline">© 2025 HRly</span>
        <span className="inline sm:hidden text-xs">©25</span>
      </div>
    </div>
  );
}

import { FaUserCog } from "react-icons/fa";
```

```
import { GrUserManager } from "react-icons/gr";

export function DashboardSidebarManager({ user }) {
  let navUser = user.toLowerCase();
  const emp_items = [
    { name: "Dashboard Activity", icon: <MdDashboard />, path: `/${navUser}/dashboard` },
    { name: "Employee Profiles", icon: <MdPerson />, path: `/${navUser}/employee-profiles`
},
    { name: "Attendance Summary", icon: <MdAccessTime />, path: `/${navUser}/employee-
attendance-status` },
    { name: "Leave Requests Status", icon: <MdRequestPage />, path: `/${navUser}/employee-
leave-requests` },
    { name: "Tasks Manager", icon: <MdAssignment />, path: `/${navUser}/tasks` },
    { name: "Check Messages", icon: <MdMessage />, path: `/${navUser}/messages` },
    { name: "Notifications", icon: <MdNotifications />, path: `/${navUser}/notifications`
},
  ];
  const manager_items = [
    { name: "Profile", icon: <MdPerson />, path: `/${navUser}/profile` },
    { name: "Settings", icon: <MdSettings />, path: `/${navUser}/settings` },
    { name: "Logout", icon: <MdLogout />, path: `/${navUser}/logout` },
  ];

  const { isDark } = useContext(ThemeContext);

  return (
    <div
      className={`h-full w-full ${
        isDark ? "bg-app text-app border-gray-700" : "bg-app text-app border-gray-300"
      } text-black flex flex-col justify-between items-start shadow-lg px-3 sm:px-5 border-
r-2`}
    >
      {/* Logo / Title */}
      <div className="flex items-left justify-start py-5 text-2xl font-bold w-full">
        <span className="hidden sm:inline">HRly</span>
        <span className="inline sm:hidden">H</span>
      </div>

      {/* Employee management section */}
      <div className="w-full">
        <p className={`border-b-2 ml-0 sm:ml-1 mt-2 py-2 px-2 ${isDark ? "bg-blue-900/50
text-app" : "bg-gray-100 text-app"}`}>
          <span className="hidden sm:inline">Manage Employee</span>
          <span className="inline sm:hidden"><FaUserCog /></span>
        </p>

        <ul className="flex flex-col gap-1 w-full mt-2">
          {emp_items.map((item, index) => (
            <li key={index} className="w-full">
              <NavLink
                to={item.path}
                className={({ isActive }) =>
                  `flex items-center gap-3 sm:px-5 py-2 rounded-md transition-all duration-
300
                    ${isDark ? "hover:bg-white/20" : ""}
                    ${
                      isActive
                        ? "bg-white/10 text-blue-600 font-semibold"
                        : "hover:bg-black/20"
                    }`
                }
              >
                <span className="text-lg flex-shrink-0 w-6 h-6 flex items-center justify-
center">{item.icon}</span>
                <span className="text-sm hidden sm:inline">{item.name}</span>
              </NavLink>
```

```
            </li>
          ))}
        </ul>
      </div>

      {/* Manager/profile section */}
      <div className="w-full">
        <p className={`border-b-2 ml-0 sm:ml-1 mt-3 py-2 px-2 ${isDark ? "bg-blue-900/50
text-app" : "bg-gray-100 text-app"}`}>
          <span className="hidden sm:inline">Manage Profile</span>
          <span className="inline sm:hidden"><GrUserManager /></span>
        </p>

        <ul className="flex flex-col gap-1 w-full mt-2">
          {manager_items.map((item, index) => (
            <li key={index} className="w-full">
              <NavLink
                to={item.path}
                className={({ isActive }) =>
                  `flex items-center gap-3 sm:px-5 py-2 rounded-md transition-all duration-
300
                    ${isDark ? "hover:bg-white/20" : ""}
                    ${
                      isActive
                        ? "bg-white/10 text-blue-600 font-semibold"
                        : "hover:bg-black/20"
                    }`
                }
              >
                <span className="text-lg flex-shrink-0 w-6 h-6 flex items-center justify-
center">{item.icon}</span>
                <span className="text-sm hidden sm:inline">{item.name}</span>
              </NavLink>
            </li>
          ))}
        </ul>
      </div>

      {/* Footer / Version */}
      <div className="text-left text-xs py-4 border-t border-white/20 w-full">
        <span className="hidden sm:inline">© 2025 HRly</span>
        <span className="inline sm:hidden text-xs">©25</span>
      </div>
    </div>
  );
}

export default DashboardSidebar;
```

```
// client/src/components/dashboard-ui-components/AttendanceSummary.jsx
import React, { useEffect, useMemo, useState, useContext } from "react";
import api from "../../utils/api";
import { AuthContext } from "../../context/AuthContext";
import { ThemeContext } from "../../context/ThemeContext";

function monthKeyFromDateISO(iso) {
  if (!iso) return null;
  const d = new Date(iso);
  return `${d.getFullYear()}-${String(d.getMonth() + 1).padStart(2, "0")}`; // YYYY-MM
}

export default function AttendanceSummary() {
  const { user } = useContext(AuthContext);
  const { isDark } = useContext(ThemeContext);
```

```
  const [records, setRecords] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  const monthFilter = useMemo(() => {
    const now = new Date();
    return `${now.getFullYear()}-${String(now.getMonth() + 1).padStart(2, "0")}`;
  }, []);

  useEffect(() => {
    let cancelled = false;
    const fetch = async () => {
      setLoading(true);
      setError(null);
      try {
        // fetch all attendance then filter client-side by employeeId & month
        const res = await api.get("/attendance/get");
        const all = Array.isArray(res.data) ? res.data : res.data.data || [];
        const mine = all.filter(
          (r) =>
            (r.employeeId && user?.employeeId && r.employeeId === user.employeeId) ||
            (r._id && user?._id && r._id === user._id) ||
            (r.email && user?.email && r.email === user.email)
        );
        if (!cancelled) setRecords(mine);
      } catch (err) {
        if (!cancelled) {
          setError(err);
          setRecords([]);
        }
      } finally {
        if (!cancelled) setLoading(false);
      }
    };
    fetch();
    return () => {
      cancelled = true;
    };
  }, [user]);

  const stats = useMemo(() => {
    const monthRecs = records.filter(
      (r) => monthKeyFromDateISO(r.date) === monthFilter
    );
    const present = monthRecs.filter(
      (r) => (r.status || "").toLowerCase() === "present"
    ).length;
    const absent = monthRecs.filter(
      (r) => (r.status || "").toLowerCase() === "absent"
    ).length;
    const total = monthRecs.length;
    const pct = total ? Math.round((present / total) * 100) : 0;
    return { present, absent, total, pct };
  }, [records, monthFilter]);

  // theme classes
  const containerClass = isDark
    ? "bg-slate-800 border border-slate-700 text-slate-100"
    : "bg-white border border-gray-100 text-slate-900";

  const subtleClass = isDark ? "text-slate-300" : "text-gray-500";
  const highlightClass = isDark ? "text-green-300" : "text-green-700";
  const cardPadding = "rounded-lg p-4";

  // human-readable month label
  const monthLabel = useMemo(() => {
```

```
    const [y, m] = monthFilter.split("-");
    const d = new Date(Number(y), Number(m) - 1, 1);
    return d.toLocaleString(undefined, { month: "long", year: "numeric" });
  }, [monthFilter]);

  return (
    <div className={`${cardPadding} ${containerClass}`}>
      <div className="flex items-start justify-between mb-3">
        <div>
          <h3 className="font-semibold text-sm md:text-base">Attendance summary</h3>
          <div className={`text-xs ${subtleClass}`}>{monthLabel}</div>
        </div>

        <div className={`text-xs ${subtleClass} hidden sm:block`}>Tip: 80% or above is
good</div>
      </div>

      {loading ? (
        <div className={`${subtleClass} py-6 text-sm`}>Loading...</div>
      ) : error ? (
        <div className="text-red-500 py-6 text-sm">Unable to load attendance.</div>
      ) : (
        <div className="flex flex-col sm:flex-row items-stretch sm:items-center gap-4">
          {/* Percent + caption: ensure it's compact on mobile */}
          <div className="flex items-center gap-3 sm:gap-4 w-full sm:w-auto">
            <div
              className={`flex items-center justify-center rounded-full ${
                isDark ? "bg-slate-700/50" : "bg-gray-100"
              }`}
              style={{ width: 64, height: 64 }} // consistent size across breakpoints
              aria-hidden
            >
              <span className={`${highlightClass} font-bold text-lg sm:text-xl`}>
                {stats.pct}%
              </span>
            </div>

            <div className="min-w-0">
              <div className="text-xs sm:text-sm font-medium">Present this month</div>
              <div className={`text-[11px] sm:text-xs mt-1 ${subtleClass}`}>
                Based on {stats.total} record{stats.total !== 1 ? "s" : ""}
              </div>
            </div>
          </div>

          {/* Stats: place to the right on larger screens, below on small */}
          <div className="flex gap-6 mt-2 sm:mt-0 sm:ml-auto w-full sm:w-auto justify-
between sm:justify-start">
            <div className="text-center w-1/3 sm:w-auto">
              <div className="text-lg sm:text-xl font-semibold">{stats.present}</div>
              <div className={`text-xs ${subtleClass}`}>Present</div>
            </div>

            <div className="text-center w-1/3 sm:w-auto">
              <div className="text-lg sm:text-xl font-semibold">{stats.absent}</div>
              <div className={`text-xs ${subtleClass}`}>Absent</div>
            </div>

            <div className="text-center w-1/3 sm:w-auto">
              <div className="text-lg sm:text-xl font-semibold">{stats.total}</div>
              <div className={`text-xs ${subtleClass}`}>Total</div>
            </div>
          </div>

          {/* Small tip visible only on xs devices, because desktop already shows it on
header right */}
```

```
          <div className={`mt-3 sm:mt-0 sm:ml-4 text-xs ${subtleClass} sm:hidden`}>
            Tip: 80% or above is good
          </div>
        </div>
      )}
    </div>
  );
}
```

```
// client/src/components/dashboard-ui-components/DeadlinesCard.jsx
import React, { useEffect, useState, useContext } from "react";
import api from "../../utils/api";
import { ThemeContext } from "../../context/ThemeContext";
import { Link, useNavigate } from "react-router-dom";

export default function DeadlinesCard({ maxItems = 3 }) {
  const { isDark } = useContext(ThemeContext) || { isDark: false };
  const navigate = useNavigate();

  const [upcoming, setUpcoming] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    let cancelled = false;
    const fetch = async () => {
      setLoading(true);
      setError(null);
      try {
        // expected: /manager/tasks/employee/me returns array or { data: [...] }
        const res = await api.get("/manager/tasks/employee/me");
        const tasks = Array.isArray(res.data) ? res.data : res.data?.data || [];

        const now = new Date();
        // filter upcoming within next 7 days OR overdue tasks
        const windowEnd = new Date(now.getTime() + 7 * 24 * 60 * 60 * 1000);

        const filtered = tasks
          .filter((t) => t.dueDate) // ignore tasks without due date
          .map((t) => ({ ...t, dueTs: new Date(t.dueDate).getTime() }))
          .filter((t) => !isNaN(t.dueTs)) // valid date
          .filter((t) => t.dueTs >= now.getTime() - 0) // include today and future (and you
can change to include overdue)
          .sort((a, b) => a.dueTs - b.dueTs);

        // find tasks within next 7 days; if none, show nearest upcoming / overdue
        let next = filtered.filter((t) => t.dueTs <= windowEnd.getTime());
        if (next.length === 0) {
          // fallback: nearest 3 tasks from filtered (could be overdue)
          next = filtered.slice(0, maxItems);
        } else {
          next = next.slice(0, maxItems);
        }

        if (!cancelled) setUpcoming(next);
      } catch (err) {
        console.error("deadlines fetch", err);
        if (!cancelled) {
          setUpcoming([]);
          setError(err);
        }
      } finally {
        if (!cancelled) setLoading(false);
      }
    };
```

```
    fetch();
    return () => {
      cancelled = true;
    };
  }, [maxItems]);

  const containerCls = isDark
    ? "bg-slate-800 border border-slate-700 text-slate-100"
    : "bg-white border border-gray-100 text-slate-900";
  const muted = isDark ? "text-slate-300" : "text-gray-500";

  return (
    <div className={`rounded-xl shadow-sm p-4 overflow-hidden ${containerCls}`}>
      <div className="flex items-start justify-between mb-3">
        <div>
          <h3 className="font-semibold">Upcoming Deadlines</h3>
          <div className="text-xs mt-1" style={{ color: isDark ? "#9CA3AF" : "#6B7280" }}>
            Next 7 days — max {maxItems}
          </div>
        </div>
      </div>

      {loading ? (
        <div className={`py-6 text-center ${muted}`}>Loading deadlines…</div>
      ) : error ? (
        <div className="py-6 text-center text-sm text-red-400">
          Error loading deadlines.
        </div>
      ) : upcoming.length === 0 ? (
        <div className={`py-6 text-center text-sm ${muted}`}>
          No upcoming deadlines. Check the tasks page for all tasks.
        </div>
      ) : (
        <ul className="space-y-3">
          {upcoming.map((t) => {
            const due = new Date(t.dueDate);
            const daysLeft = Math.ceil((due.getTime() - Date.now()) / (24 * 3600 * 1000));
            const isOverdue = daysLeft < 0;
            const badgeCls = isOverdue
              ? "bg-red-100 text-red-700"
              : daysLeft <= 1
              ? "bg-amber-100 text-amber-800"
              : "bg-green-100 text-green-800";

            return (
              <li
                key={t._id}
                className={`p-3 rounded-lg border ${isDark ? "border-slate-700" : "border-
gray-100"} flex items-center justify-between gap-4 hover:shadow-md transition`}
                title="Click to open task"
              >
                <div className="min-w-0">
                  <div className="flex items-center gap-2">
                    <div className="font-medium text-sm truncate">{t.title || "Untitled
task"}</div>
                    <div className="text-xs text-gray-400">•</div>
                    <div className="text-xs" style={{ color: isDark ? "#9CA3AF" : "#6B7280"
}}>
                      {t.assignedTo?.map(a => a.name).slice(0,2).join(", ") || "—"}
                    </div>
                  </div>
                  <div className="text-xs mt-1 truncate" style={{ color: isDark ? "#94A3B8"
: "#6B7280" }}>
                    {t.description ? (t.description.length > 120 ? t.description.slice(0,
117) + "..." : t.description) : <span className="text-gray-400">No description</span>}
                  </div>
```

```
                </div>

                <div className="flex flex-col items-end gap-2">
                  <div className={`text-xs px-2 py-0.5 rounded-full ${badgeCls}`}>
                    {isOverdue ? `Overdue ${Math.abs(daysLeft)}d` : daysLeft === 0 ? "Due
today" : `In ${daysLeft}d`}
                  </div>
                  <div className="text-xs" style={{ color: isDark ? "#9CA3AF" : "#6B7280"
}}>
                    {due.toLocaleDateString()}
                  </div>
                </div>
              </li>
            );
          })}
        </ul>
      )}
    </div>
  );
}
```

```
// client/src/components/dashboard-ui-components/NotificationsList.jsx
import React, { useEffect, useState, useContext } from "react";
import api from "../../utils/api";
import { HiMiniBellAlert } from "react-icons/hi2";
import { ThemeContext } from "../../context/ThemeContext"; // optional — safe if not
provided

export default function NotificationsList() {
  const { isDark } = useContext(ThemeContext) || { isDark: false };

  const [notes, setNotes] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    let cancelled = false;
    const fetch = async () => {
      setLoading(true);
      try {
        // fetch latest 5, then we'll show 2 most recent
        const res = await api.get("/notifications", { params: { page: 1, limit: 5 } });
        const list = res.data?.data || res.data || [];
        const sorted = [...list].sort((a, b) => {
          const da = a.createdAt ? new Date(a.createdAt).getTime() : 0;
          const db = b.createdAt ? new Date(b.createdAt).getTime() : 0;
          return db - da;
        });
        if (!cancelled) setNotes(sorted);
      } catch (err) {
        console.error("fetch notifications", err);
        if (!cancelled) setNotes([]);
      } finally {
        if (!cancelled) setLoading(false);
      }
    };
    fetch();
    return () => {
      cancelled = true;
    };
  }, []);

  const shown = notes.slice(0, 2); // show only 2 latest

  const containerBg = isDark ? "bg-slate-800 border-slate-700 text-slate-100" : "bg-white
border-gray-100 text-slate-900";
```

```
  const itemBg = isDark ? "bg-slate-700/40 hover:bg-slate-700/60" : "bg-gray-50 hover:bg-
white";
  const typeStyles = (type) => {
    switch ((type || "").toLowerCase()) {
      case "announcement":
        return isDark ? "bg-blue-800 text-blue-200" : "bg-blue-50 text-blue-700";
      case "blog":
        return isDark ? "bg-green-800 text-green-200" : "bg-green-50 text-green-700";
      case "notice":
        return isDark ? "bg-amber-800 text-amber-200" : "bg-amber-50 text-amber-700";
      case "update":
        return isDark ? "bg-purple-800 text-purple-200" : "bg-purple-50 text-purple-700";
      default:
        return isDark ? "bg-gray-700 text-gray-200" : "bg-gray-100 text-gray-700";
    }
  };

  return (
    <div
      className={`rounded-xl shadow-sm border p-4 overflow-hidden ${containerBg}`}
      role="region"
      aria-label="Company notifications"
    >
      <div className="flex items-center gap-3 mb-4">
        <div className={`p-2 rounded-md ${isDark ? "bg-slate-700/50" : "bg-blue-50"}`}>
          <HiMiniBellAlert className={`${isDark ? "text-slate-100" : "text-blue-600"} w-5
h-5`} />
        </div>
        <div className="flex-1">
          <div className={`font-semibold ${isDark ? "text-slate-100" : "text-gray-
800"}`}>Company notifications</div>
          <div className="text-xs mt-0.5 text-gray-400">Latest updates &
announcements</div>
        </div>
      </div>

      {loading ? (
        <div className={`py-6 text-center ${isDark ? "text-slate-300" : "text-gray-
500"}`}>Loading...</div>
      ) : shown.length === 0 ? (
        <div className={`py-6 text-center text-sm ${isDark ? "text-slate-400" : "text-gray-
500"}`}>
          No notifications available
        </div>
      ) : (
        <ul className="space-y-3">
          {shown.map((n) => (
            <li
              key={n._id}
              /* Responsive layout:
                 - small devices: column layout (image on top, content below), readable
text wrapping
                 - sm+ devices: original row layout preserved
              */
              className={`flex flex-col sm:flex-row items-start sm:items-center gap-3 p-3
rounded-lg border-1 border-slate-400 ${itemBg}`}
              title={n.title}
            >
              {/* Image / avatar area */}
              <div className="flex-shrink-0 w-full sm:w-auto flex items-center justify-
center">
                {n.image ? (
                  <img
                    src={`http://localhost:5000/notification_uploads/${n.image}`}
                    alt={n.title || "notification image"}
                    className="w-12 h-12 sm:w-14 sm:h-14 rounded object-cover border"
```

```
                  />
                ) : (
                  <div
                    className={`w-12 h-12 sm:w-14 sm:h-14 rounded flex items-center
justify-center font-semibold ${typeStyles(n.type)}`}
                  >
                    <span className="text-sm">
                      {(n.title || "N").split(" ").map(s => s[0]).slice(0,2).join("")}
                    </span>
                  </div>
                )}
              </div>

              {/* Content */}
              <div className="flex-1 min-w-0 w-full">
                <div className="flex flex-col sm:flex-row sm:items-start sm:justify-between
gap-1">
                  <div className="min-w-0">
                    {/* Title — allow wrapping on small screens, but truncate on larger */}
                    <h4 className={`text-sm font-medium ${isDark ? "text-slate-100" :
"text-gray-900"} whitespace-normal sm:whitespace-nowrap sm:truncate`}>
                      {n.title}
                    </h4>

                    {/* Message — small devices: allow 2 lines, larger: single-line
truncated */}
                    <p className={`${isDark ? "text-slate-200/90" : "text-gray-600"} mt-1
text-sm sm:text-xs whitespace-nowrap truncate`}>
                      {n.message}
                    </p>
                  </div>

                  {/* Date — on small devices show below title/message; on sm+ keep to the
right */}
                  <div className="text-xs text-gray-400 mt-2 sm:mt-0 sm:ml-3 sm:flex-
shrink-0">
                    {new Date(n.createdAt).toLocaleDateString()}
                  </div>
                </div>

                {/* Bottom row: badge + read link — stacked on small, inline on larger */}
                <div className="mt-3 flex flex-col sm:flex-row sm:items-center gap-2">
                  <span className={`text-xs px-2 py-0.5 rounded-full inline-block
${typeStyles(n.type)}`} aria-hidden>
                    {n.type || "Update"}
                  </span>

                  <div className="mt-1 sm:mt-0 sm:ml-auto">
                    <a
                      href={`/notifications/${n._id}`}
                      className={`text-xs font-medium underline ${isDark ? "text-slate-
100/90" : "text-gray-700/90"}`}
                    >
                      Read
                    </a>
                  </div>
                </div>
              </div>
            </li>
          ))}
        </ul>
      )}
    </div>
  );
}
```

```jsx
// client/src/components/dashboard-ui-components/QuickLinks.jsx
import React, { useContext } from "react";
import { NavLink } from "react-router-dom";
import { ThemeContext } from "../../context/ThemeContext";

const BUTTON_STYLES = {
  Attendance: { start: "#3b82f6", end: "#6366f1" },
  "My Tasks": { start: "#34d399", end: "#059669" },
  "My Leaves": { start: "#fbbf24", end: "#fb923c" },
  Profile: { start: "#8b5cf6", end: "#4f46e5" },
  default: { start: "#d1d5db", end: "#9ca3af" },
};

export default function QuickLinks() {
  const { isDark } = useContext(ThemeContext);

  const links = [
    { to: "/employee/attendance", label: "Attendance", desc: "Mark & view" },
    { to: "/employee/tasks", label: "My Tasks", desc: "Open tasks" },
    { to: "/employee/leave-requests", label: "My Leaves", desc: "Apply / history" },
    { to: "/employee/profile", label: "Profile", desc: "View & edit" },
  ];

  const containerClass = isDark
    ? "rounded-xl p-3 sm:p-4 bg-slate-800 border border-slate-700 text-slate-100 shadow-sm"
    : "rounded-xl p-3 sm:p-4 bg-white border border-gray-100 text-slate-900 shadow-sm";
  const subtitleClass = isDark ? "text-slate-300" : "text-gray-500";

  return (
    <aside className={containerClass}>
      <div className="flex items-center justify-between mb-3">
        <h3 className="text-lg font-semibold">Quick links</h3>
        <div className={`text-xs ${subtitleClass} hidden sm:block`}>Shortcuts</div>
      </div>

      {/* responsive grid: single column on xs, two columns on sm+ */}
      <div className="grid grid-cols-1 sm:grid-cols-2 gap-3">
        {links.map((l) => {
          const style = BUTTON_STYLES[l.label] || BUTTON_STYLES.default;

          return (
            <NavLink
              key={l.to}
              to={l.to}
              aria-label={l.label}
              className={({ isActive }) =>
                `group relative flex items-center gap-3 p-3 sm:p-3 rounded-lg transition-
all focus:outline-none focus:ring-2 focus:ring-offset-1 ${
                  isActive ? "shadow-md" : "hover:scale-[1.02]"
                }`
              }
              style={({ isActive }) => ({
                background: `linear-gradient(90deg, ${style.start} 0%, ${style.end} 100%)`,
                opacity: isActive ? 1 : 0.98,
              })}
            >
              {/* left accent bar - smaller on xs */}
              <span
                className="flex-none rounded-md"
                style={{
                  width: window.innerWidth < 640 ? 6 : 10,
                  height: window.innerWidth < 640 ? 36 : 40,
                  background: `linear-gradient(180deg, ${style.start}, ${style.end})`,
                }}
                aria-hidden
              />
```

```
              <div className="flex-1 min-w-0">
                <div className="flex items-baseline justify-between">
                  <div className="font-medium truncate text-sm sm:text-
base">{l.label}</div>
                  {/* hide small desc on xs */}
                  <div className={`text-xs ${subtitleClass} hidden
sm:block`}>{l.desc}</div>
                </div>
                <div className={`text-xs mt-1 ${isDark ? "text-slate-200/80" : "text-
white/90"} truncate sm:truncate`}>

                  <span className="hidden sm:inline">Quick access to </span>
                  <span className="font-medium">{l.label}</span>
                </div>
              </div>
            </NavLink>
          );
        })}
      </div>

      {/* small footer / hint for mobile */}
      <div className={`mt-3 text-xs ${subtitleClass} sm:hidden`}>Tap a tile to open</div>
    </aside>
  );
}
```

```
// client/src/components/dashboard-ui-components/TaskProgressChart.jsx
import React, { useEffect, useState, useContext } from "react";
import api from "../../utils/api";
import {
  ResponsiveContainer,
  BarChart,
  Bar,
  XAxis,
  YAxis,
  Tooltip,
  Legend,
  CartesianGrid,
  Cell,
} from "recharts";
import { ThemeContext } from "../../context/ThemeContext";

// Light / Dark color palette
const LIGHT_COLORS = ["#60A5FA", "#FBBF24", "#34D399", "#F87171", "#A78BFA"];
const DARK_COLORS = ["#60A5FA", "#FBBF24", "#34D399", "#F87171", "#A78BFA"]; // choose
darker palette if you want

// Draw custom triangular bar shape
const getPath = (x, y, width, height) =>
  `M${x},${y + height}
   C${x + width / 3},${y + height}
    ${x + width / 2},${y + height / 3}
    ${x + width / 2},${y}
   C${x + width / 2},${y + height / 3}
    ${x + (2 * width) / 3},${y + height}
    ${x + width},${y + height}
   Z`;

function TriangleBar(props) {
  const { fill, x, y, width, height } = props;
  if (x == null || y == null || width == null || height == null) return null;
  return <path d={getPath(x, y, width, height)} fill={fill} stroke="none" />;
}

export default function TaskProgressChart() {
```

```
  const { isDark } = useContext(ThemeContext);
  const [chartData, setChartData] = useState([]);
  const [loading, setLoading] = useState(true);

  // choose palette (you can customize dark colors)
  const palette = isDark ? DARK_COLORS : LIGHT_COLORS;

  useEffect(() => {
    let cancelled = false;

    const fetch = async () => {
      setLoading(true);
      try {
        const res = await api.get("/manager/tasks/employee/me");
        const tasks = Array.isArray(res.data) ? res.data : res.data?.data || [];

        // Count tasks grouped by status
        const counts = tasks.reduce((acc, t) => {
          const s = t.status || "Pending";
          acc[s] = (acc[s] || 0) + 1;
          return acc;
        }, {});

        // Ensure a stable ordering if you prefer (Pending, In Progress, Completed,
Cancelled)
        const order = ["Pending", "In Progress", "Completed", "Cancelled", "Blocked"];
        const keys = Object.keys(counts).sort((a, b) => {
          const ia = order.indexOf(a) === -1 ? 999 : order.indexOf(a);
          const ib = order.indexOf(b) === -1 ? 999 : order.indexOf(b);
          return ia - ib;
        });

        const formatted = keys.map((status, idx) => ({
          status,
          count: counts[status],
          color: palette[idx % palette.length],
        }));

        if (!cancelled) setChartData(formatted);
      } catch (err) {
        console.error("task chart fetch error:", err);
        if (!cancelled) setChartData([]);
      } finally {
        if (!cancelled) setLoading(false);
      }
    };

    fetch();
    return () => {
      cancelled = true;
    };
  }, [isDark]);

  const total = chartData.reduce((s, d) => s + d.count, 0);

  return (
    <div
      className={`rounded-xl shadow-sm p-4 ${
        isDark
          ? "bg-slate-800 border border-slate-700 text-slate-200"
          : "bg-white border border-gray-100 text-slate-900"
      }`}
    >
      <div className="flex items-start justify-between mb-3">
        <h3 className="font-semibold">Tasks — Status Overview</h3>
      </div>
```

```
      {loading ? (
        <div className={`w-full h-40 flex items-center justify-center ${isDark ? "text-
slate-400" : "text-gray-500"}`}>
          Loading…
        </div>
      ) : chartData.length === 0 ? (
        <div className={`w-full h-40 flex flex-col items-center justify-center gap-2
${isDark ? "text-slate-400" : "text-gray-500"}`}>
          <div className="text-sm font-medium">No tasks available</div>
          <div className="text-xs">You have no assigned tasks right now.</div>
        </div>
      ) : (
        <div className="w-full h-64">
          <ResponsiveContainer width="100%" height="100%">
            <BarChart data={chartData} margin={{ top: 10, right: 20, left: 0, bottom: 5 }}>
              <CartesianGrid strokeDasharray="3 3" stroke={isDark ? "#334155" : "#e6edf3"}
/>
              <XAxis dataKey="status" tick={{ fill: isDark ? "#cbd5e1" : "#374151" }} />
              <YAxis tick={{ fill: isDark ? "#cbd5e1" : "#374151" }} allowDecimals={false}
/>
              <Tooltip
                contentStyle={{
                  background: isDark ? "#0f1724" : "#fff",
                  border: `1px solid ${isDark ? "#1f2937" : "#e5e7eb"}`,
                  color: isDark ? "#e5e7eb" : "#111827",
                  borderRadius: 6,
                }}
              />
              <Legend verticalAlign="top" height={36} />

              <Bar
                dataKey="count"
                name="Tasks"
                shape={TriangleBar}
                label={{ position: "top", fill: isDark ? "#e2e8f0" : "#111827", fontSize:
12 }}
              >
                {chartData.map((entry, index) => (
                  <Cell key={`cell-${index}`} fill={entry.color} />
                ))}
              </Bar>
            </BarChart>
          </ResponsiveContainer>
        </div>
      )}

      {/* Total badge */}
      <div className="mt-3 flex items-center justify-center">
        <div
          className={`inline-flex items-center justify-center rounded-full px-4 py-2 text-
sm font-medium ${
            isDark ? "bg-slate-700/60 text-slate-100" : "bg-gray-100 text-gray-900"
          }`}
        >
          <span className="mr-2 text-xs opacity-80">Total</span>
          <span className="text-lg font-semibold">{total} Tasks</span>
        </div>
      </div>
    </div>
  );
}
```

```
// client/src/components/dashboard-ui-components/WelcomeBanner.jsx
import React, { useContext, useMemo } from "react";
import { motion } from "framer-motion";
```

```
import { ThemeContext } from "../../context/ThemeContext";
import { FiCalendar, FiClock, FiCheckCircle, FiTrendingUp } from "react-icons/fi";

export default function WelcomeBanner({ name }) {
  const { isDark } = useContext(ThemeContext);

  const today = useMemo(() => {
    const d = new Date();
    return d.toLocaleDateString(undefined, {
      weekday: "short",
      month: "short",
      day: "numeric",
    });
  }, []);

  const containerBg = isDark
    ? "bg-gradient-to-r from-slate-800/80 via-slate-900 to-slate-800/90 text-slate-100"
    : "bg-gradient-to-r from-blue-600 to-indigo-600 text-white";

  const glass = isDark
    ? "bg-slate-800/60 backdrop-blur-sm border border-slate-700"
    : "bg-white/10";

  const chipBg = isDark ? "bg-slate-700/60 text-gray-100" : "bg-white/20 text-white";
  const subtle = isDark ? "text-slate-300" : "text-blue-100";

  return (
    <motion.section
      initial={{ opacity: 0, y: -16 }}
      animate={{ opacity: 1, y: 0 }}
      transition={{ duration: 0.45, ease: "easeOut" }}
      className={`w-full rounded-2xl p-4 sm:p-5 md:p-6 ${containerBg} shadow-lg`}
    >
      <div className="max-w-7xl mx-auto">
        <div className="flex flex-col md:flex-row items-start md:items-center gap-4 md:gap-
6">

          {/* LEFT SIDE */}
          <div className="flex-1 min-w-0">
            {/* Responsive Title */}
            <h1 className="font-semibold leading-tight text-lg sm:text-xl md:text-2xl">
              Welcome back, <span className="capitalize">{name || "there"}</span>
            </h1>

            {/* Subtitle */}
            <p className={`mt-1 sm:mt-2 text-xs sm:text-sm ${subtle}`}>
              Quick snapshot of your day & key updates.
            </p>

            {/* Small Chips */}
            <div className="mt-3 sm:mt-4 flex flex-wrap gap-2 sm:gap-3">

              {/* Date Chip */}
              <div className={`inline-flex items-center gap-1.5 sm:gap-2 px-2 sm:px-3 py-1
rounded-full ${chipBg}`}>
                <FiCalendar className="w-3.5 h-3.5 sm:w-4 sm:h-4" />
                <span className="text-[10px] sm:text-xs font-medium">{today}</span>
              </div>

              {/* Next Meeting */}
              <div className={`inline-flex items-center gap-1.5 sm:gap-2 px-2 sm:px-3 py-1
rounded-full ${glass}`}>
                <FiClock className="w-3.5 h-3.5 sm:w-4 sm:h-4 text-indigo-300" />
                <div className="text-[10px] sm:text-xs">
                  <div className="font-medium leading-tight">Next Meeting</div>
```

```
                <div className="text-[9px] sm:text-[11px] text-slate-300">2:30 PM •
Zoom</div>
              </div>
            </div>

            {/* Tasks Due */}
            <div className={`inline-flex items-center gap-1.5 sm:gap-2 px-2 sm:px-3 py-1
rounded-full ${glass}`}>
              <FiCheckCircle className="w-3.5 h-3.5 sm:w-4 sm:h-4 text-green-300" />
              <div className="text-[10px] sm:text-xs">
                <div className="font-medium leading-tight">Tasks</div>
                <div className="text-[9px] sm:text-[11px] text-slate-300">5 due
today</div>
              </div>
            </div>
          </div>
        </div>

        {/* RIGHT SIDE MINI-STATS */}
        <div className="w-full md:w-auto flex items-center justify-start md:justify-
center">
          <div
            className={`p-3 sm:p-4 rounded-2xl ${glass} flex flex-col items-center
justify-center min-w-[120px] sm:min-w-[140px]`}
          >
            <div className="text-lg sm:text-2xl md:text-3xl font-bold">+8%</div>
            <div className="text-[10px] sm:text-xs mt-1 text-slate-300 flex items-center
gap-1.5 sm:gap-2">
              <FiTrendingUp className="w-3.5 h-3.5 sm:w-4 sm:h-4" />
              <span>Productivity</span>
            </div>
          </div>
        </div>
      </div>
    </motion.section>
  );
}
```

```
// client/src/components/logout/LogoutTab.jsx
import React, { useContext } from "react";
import { useNavigate } from "react-router-dom";
import Swal from "sweetalert2";
import { ThemeContext } from "../../context/ThemeContext";

function LogoutTab() {
  const navigate = useNavigate();
  const { isDark } = useContext(ThemeContext);

  // Theme classes
  const containerBg = isDark ? "bg-app text-gray-200" : "bg-white text-gray-800";
  const secondaryBtn = isDark
    ? "bg-slate-700 text-gray-200 hover:bg-slate-600"
    : "bg-gray-200 text-gray-700 hover:bg-gray-300";
  const dangerBtn = isDark
    ? "bg-red-600 text-white hover:bg-red-700"
    : "bg-red-500 text-white hover:bg-red-600";
  const hintText = isDark ? "text-gray-300" : "text-gray-500";

  // Handle Logout Function
  const handleLogout = () => {
    localStorage.removeItem("token");
    localStorage.removeItem("user");

    // Navigate first so UI updates, then show confirmation
    navigate("/login");
```

```
    Swal.fire("You are logged out", "You have been logged out successfully!", "success");
  };

  return (
    <div className={`h-full shadow-md p-6 flex flex-col items-center justify-center
${containerBg}`}>
      <div className={`w-full max-w-md p-6 rounded-lg`}>
        <div className="flex flex-col items-center">
          <img src="/user.gif" alt="logout" className="w-40 mb-4" />
          <p className={`text-center mb-6 ${hintText}`}>
            Oh no! You are leaving... Are you sure?
          </p>

          <div className="flex gap-4">
            <button
              onClick={handleLogout}
              className={`${dangerBtn} px-6 py-2 rounded-lg transition flex items-center
gap-2 cursor-pointer`}
            >
              Yes, Logout
            </button>

            <button
              onClick={() => navigate(-1)}
              className={`${secondaryBtn} px-6 py-2 rounded-lg transition cursor-pointer`}
            >
              Cancel
            </button>
          </div>
        </div>
      </div>
    </div>
  );
}

export default LogoutTab;
```

```
// client/src/components/notifications/NotificationsTab.jsx
import React, { useEffect, useState, useContext } from "react";
import api from "../../utils/api";
import { HiMiniBellAlert } from "react-icons/hi2";
import { ThemeContext } from "../../context/ThemeContext";

const TYPE_STYLES = {
  Announcement: { bg: "bg-blue-100", text: "text-blue-700", dot: "bg-blue-600" },
  Blog: { bg: "bg-green-100", text: "text-green-700", dot: "bg-green-600" },
  Notice: { bg: "bg-yellow-100", text: "text-yellow-800", dot: "bg-yellow-600" },
  Update: { bg: "bg-purple-100", text: "text-purple-700", dot: "bg-purple-600" },
  Other: { bg: "bg-gray-100", text: "text-gray-700", dot: "bg-gray-500" },
};

function getTypeClasses(type, isDark) {
  const base = TYPE_STYLES[type] || TYPE_STYLES.Other;
  if (!isDark) return `${base.bg} ${base.text} ${base.dot}`;
  // dark overrides for the chip (keeps color hint but darker)
  const darkMap = {
    Announcement: { bg: "bg-blue-900", text: "text-blue-100", dot: "bg-blue-400" },
    Blog: { bg: "bg-green-900", text: "text-green-100", dot: "bg-green-400" },
    Notice: { bg: "bg-yellow-900", text: "text-yellow-100", dot: "bg-yellow-400" },
    Update: { bg: "bg-purple-900", text: "text-purple-100", dot: "bg-purple-400" },
    Other: { bg: "bg-slate-800", text: "text-slate-100", dot: "bg-slate-400" },
  };
  const dark = darkMap[type] || darkMap.Other;
  return `${dark.bg} ${dark.text} ${dark.dot}`;
}
```

```
export default function NotificationsTab() {
  const [notifications, setNotifications] = useState([]);
  const [loading, setLoading] = useState(true);
  const [query, setQuery] = useState("");
  const [typeFilter, setTypeFilter] = useState("");
  const { isDark } = useContext(ThemeContext);

  // Theme classes
  const pageBg = isDark ? "bg-app text-gray-200" : "bg-white text-gray-800";
  const panelInput = isDark
    ? "border p-2 rounded w-full md:w-64 bg-slate-800 text-gray-200 border-slate-700"
    : "border p-2 rounded w-full md:w-64 bg-white text-gray-700 border-gray-300";
  const selectInput = isDark
    ? "border p-2 rounded bg-slate-800 text-gray-200 border-slate-700"
    : "border p-2 rounded bg-white text-gray-700 border-gray-300";
  const btnPrimary = isDark ? "px-3 py-2 bg-blue-700 text-white rounded hover:bg-blue-600"
: "px-3 py-2 bg-blue-600 text-white rounded hover:bg-blue-700";
  const btnGhost = isDark ? "px-3 py-2 bg-slate-800 rounded hover:bg-slate-700" : "px-3 py-
2 bg-gray-100 rounded hover:bg-gray-200";
  const cardBg = isDark ? "bg-slate-800 border-slate-700 text-gray-200" : "bg-white border-
gray-100 text-gray-800";
  const cardFooterBg = isDark ? "bg-slate-900 border-t border-slate-700" : "bg-gray-50
border-t border-gray-100";
  const headingText = isDark ? "text-gray-100" : "text-gray-800";
  const subText = isDark ? "text-gray-400" : "text-gray-500";

  const fetchList = async (params = {}) => {
    setLoading(true);
    try {
      // public endpoint for notifications
      const res = await api.get("/notifications", {
        params: { page: 1, limit: 50, ...params },
      });
      setNotifications(res.data.data || []);
    } catch (err) {
      console.error("fetch notifications:", err);
      setNotifications([]);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchList(); // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  const onSearch = () => {
    fetchList({ search: query, type: typeFilter });
  };

  const clear = () => {
    setQuery("");
    setTypeFilter("");
    fetchList();
  };

  return (
    <div className={`h-full shadow-md p-6 overflow-y-auto ${pageBg}`}>
      <div className="flex flex-col md:flex-row items-start md:items-center gap-3 mb-5">
        <div className="flex items-center gap-3">
          <HiMiniBellAlert className="text-blue-500 w-6 h-6" />
          <div>
            <h2 className={`text-xl font-semibold ${headingText}`}>Company
Notifications</h2>
```

```
            <div className={`text-sm ${subText}`}>Latest announcements, updates and posts
from your company.</div>
          </div>
        </div>

        <div className="ml-auto w-full md:w-auto flex flex-col md:flex-row gap-2 items-
center">
          <input
            value={query}
            onChange={(e) => setQuery(e.target.value)}
            placeholder="Search title or message"
            className={panelInput + " md:w-64"}
          />

          <select
            value={typeFilter}
            onChange={(e) => setTypeFilter(e.target.value)}
            className={selectInput}
          >
            <option value="">All types</option>
            <option value="Announcement">Announcement</option>
            <option value="Blog">Blog</option>
            <option value="Notice">Notice</option>
            <option value="Update">Update</option>
          </select>

          <button onClick={onSearch} className={btnPrimary}>Search</button>

          <button onClick={clear} className={btnGhost}>Clear</button>
        </div>
      </div>

      {/* Grid of notification cards */}
      {loading ? (
        <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
          {Array.from({ length: 6 }).map((_, i) => (
            <div key={i} className={`${cardBg} p-4 border rounded-lg animate-pulse h-40`}
/>
          ))}
        </div>
      ) : notifications.length === 0 ? (
        <div className={`flex items-center justify-center h-56 ${subText}`}>
          No notifications to display
        </div>
      ) : (
        <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
          {notifications.map((note) => {
            const type = note.type || "Other";
            // type chip classes combined for bg/text/dot
            const typeChip = getTypeClasses(type, isDark).split(" ");
            const chipBg = typeChip[0];
            const chipText = typeChip[1];
            const chipDot = typeChip[2];

            return (
              <article
                key={note._id}
                className={`${cardBg} border rounded-lg shadow-sm hover:shadow-md
transition overflow-hidden flex flex-col`}
                aria-labelledby={`note-${note._id}`}
              >
                <div className="flex gap-4 p-4">
                  <div className="w-28 h-20 rounded overflow-hidden flex-shrink-0 bg-gray-
50 border">
                    {note.image ? (
                      <img
```

```
                      src={`http://localhost:5000/notification_uploads/${note.image}`}
                      alt={note.title}
                      className="w-full h-full object-cover"
                    />
                  ) : (
                    <div className="w-full h-full flex items-center justify-center text-
gray-400">
                      No Image
                    </div>
                  )}
                </div>

                <div className="flex-1 min-w-0">
                  <div className="flex items-start justify-between gap-3">
                    <h3 id={`note-${note._id}`} className="font-semibold text-lg
truncate">
                      {note.title}
                    </h3>

                    <div className={`px-2 py-1 text-xs font-medium rounded-full ${chipBg}
${chipText}`}>
                      <span className={`inline-block w-2 h-2 rounded-full mr-2
${chipDot}`} />
                      {type}
                    </div>
                  </div>

                  <p className={`text-sm mt-2 ${isDark ? "text-gray-300" : "text-gray-
600"} line-clamp-3 whitespace-pre-wrap`}>
                    {note.message}
                  </p>

                  <div className="mt-3 text-xs flex justify-between items-center">
                    <div className={`${isDark ? "text-gray-400" : "text-gray-500"}`}>{new
Date(note.createdAt).toLocaleString()}</div>
                    <div className={`${isDark ? "text-gray-400" : "text-gray-500"}`}>By:
{note.createdByName || "Admin"}</div>
                  </div>
                </div>
              </div>

              <div className={`${cardFooterBg} p-3 flex items-center justify-between`}>
                <div className={`text-xs ${isDark ? "text-gray-400" : "text-gray-
500"}`}>ID: {note._id}</div>
                <div className="flex gap-2">
                  <a
                    href={`/manager/notifications/${note._id}`}
                    className={`px-3 py-1 rounded text-sm border ${isDark ? "bg-slate-800
border-slate-700 hover:bg-slate-700" : "bg-white border-gray-200 hover:bg-gray-100"}`}
                  >
                    View
                  </a>
                </div>
              </div>
            </article>
          );
        })}
      </div>
    )}
  </div>
);
}
```

```
// client/src/components/settings/SettingsTab.jsx
import React, { useState, useContext } from "react";
import { AuthContext } from "../../context/AuthContext";
```

```
import axios from "axios";
import Swal from "sweetalert2";
import { ThemeContext } from "../../context/ThemeContext";

const SettingsTab = () => {
  const { user, token } = useContext(AuthContext);
  const { isDark } = useContext(ThemeContext);

  const [formData, setFormData] = useState({
    fullName: user?.name || "",
    email: user?.email || "",
    phone: "",
    address: "",
    currentPassword: "",
    newPassword: "",
    confirmPassword: "",
    securityQuestion: "",
    securityAnswer: "",
  });

  const [loading, setLoading] = useState(false);

  // Theme classes
  const pageBg = isDark ? "bg-app text-gray-200" : "bg-white text-gray-800";
  const panelBg = isDark ? "bg-slate-800 border-slate-700" : "bg-white border-gray-200";
  const sectionTitle = isDark ? "text-gray-100" : "text-gray-800";
  const labelText = isDark ? "text-gray-300" : "text-gray-700";
  const inputBase = isDark
    ? "w-full border px-3 py-2 rounded bg-slate-700 text-gray-100 border-slate-600
focus:outline-none focus:border-indigo-400"
    : "w-full border px-3 py-2 rounded bg-white text-gray-800 border-gray-300
focus:outline-none focus:border-indigo-400";
  const inputReadonly = isDark
    ? "w-full border px-3 py-2 rounded bg-slate-700 text-gray-300 cursor-not-allowed
border-slate-600"
    : "w-full border px-3 py-2 rounded bg-gray-100 text-gray-700 cursor-not-allowed border-
gray-300";
  const selectBase = inputBase;
  const btnPrimary = isDark
    ? "bg-indigo-600 hover:bg-indigo-500 text-white px-6 py-2 rounded disabled:opacity-60"
    : "bg-blue-600 hover:bg-blue-700 text-white px-6 py-2 rounded disabled:opacity-60";

  // Handle Input Change
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((p) => ({ ...p, [name]: value }));
  };

  // Password Validation
  const validatePassword = (password) => {
    const regex =
      /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&#^])[A-Za-z\d@$!%*?&#^]{8,}$/;
    return regex.test(password);
  };

  // Handle Submit
  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);

    // Password checks only if user entered new password
    if (formData.newPassword || formData.confirmPassword) {
      if (formData.newPassword !== formData.confirmPassword) {
        Swal.fire({
          icon: "error",
          title: "Password Mismatch",
```

```
          text: "New Password and Confirm Password do not match!",
        });
        setLoading(false);
        return;
      }

      if (!validatePassword(formData.newPassword)) {
        Swal.fire({
          icon: "warning",
          title: "Weak Password",
          text: "Password must include uppercase, lowercase, number, and special character
(min 8 characters).",
        });
        setLoading(false);
        return;
      }
    }

    try {
      const res = await axios.put(
        "http://localhost:5000/api/settings/update-settings",
        {
          phone: formData.phone,
          address: formData.address,
          securityQuestion: formData.securityQuestion,
          securityAnswer: formData.securityAnswer,
          currentPassword: formData.currentPassword,
          newPassword: formData.newPassword,
          confirmPassword: formData.confirmPassword,
        },
        {
          headers: {
            Authorization: `Bearer ${token}`,
            "Content-Type": "application/json",
          },
        }
      );

      Swal.fire({
        icon: "success",
        title: "Success",
        text: res.data.msg || "Changes saved successfully!",
      });
    } catch (err) {
      console.error("Update error:", err);
      Swal.fire({
        icon: "error",
        title: "Update Failed",
        text:
          err.response?.data?.msg ||
          "Something went wrong. Please try again.",
      });
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className={`p-6 w-full h-full ${pageBg}`}>
      <div className={`w-full h-full p-6 rounded-lg ${panelBg}`}>
        <h2 className={`text-2xl font-semibold mb-3 ${sectionTitle}`}>Account Settings</h2>
        <hr className={isDark ? "border-slate-700 mb-4" : "border-gray-200 mb-4"} />

        <form onSubmit={handleSubmit} className="space-y-6 mt-4">
          {/* Profile Info */}
          <div>
```

```
            <h3 className={`text-lg font-semibold mb-3 ${sectionTitle}`}>Profile
Information</h3>
            <div className="grid grid-cols-1 sm:grid-cols-2 gap-4">
              <div>
                <label className={`block mb-1 ${labelText}`}>Full Name</label>
                <input
                  type="text"
                  name="fullName"
                  value={formData.fullName}
                  readOnly
                  className={inputReadonly}
                />
              </div>
              <div>
                <label className={`block mb-1 ${labelText}`}>Email</label>
                <input
                  type="email"
                  name="email"
                  value={formData.email}
                  readOnly
                  className={inputReadonly}
                />
              </div>
              <div>
                <label className={`block mb-1 ${labelText}`}>Phone Number</label>
                <input
                  type="text"
                  name="phone"
                  value={formData.phone}
                  onChange={handleChange}
                  className={inputBase}
                  placeholder="Enter phone number"
                />
              </div>
              <div>
                <label className={`block mb-1 ${labelText}`}>Address</label>
                <input
                  type="text"
                  name="address"
                  value={formData.address}
                  onChange={handleChange}
                  className={inputBase}
                  placeholder="Enter address"
                />
              </div>
            </div>
          </div>

          {/* Change Password */}
          <div>
            <h3 className={`text-lg font-semibold mb-3 ${sectionTitle}`}>Change
Password</h3>
            <div className="grid grid-cols-1 sm:grid-cols-3 gap-4">
              <div>
                <label className={`block mb-1 ${labelText}`}>Current Password</label>
                <input
                  type="password"
                  name="currentPassword"
                  value={formData.currentPassword}
                  onChange={handleChange}
                  className={inputBase}
                  placeholder="Enter current password"
                />
              </div>
              <div>
                <label className={`block mb-1 ${labelText}`}>New Password</label>
```

```
              <input
                type="password"
                name="newPassword"
                value={formData.newPassword}
                onChange={handleChange}
                className={inputBase}
                placeholder="Enter new password"
              />
            </div>
            <div>
              <label className={`block mb-1 ${labelText}`}>Confirm Password</label>
              <input
                type="password"
                name="confirmPassword"
                value={formData.confirmPassword}
                onChange={handleChange}
                className={inputBase}
                placeholder="Confirm new password"
              />
            </div>
          </div>
        </div>

        {/* Security Preferences */}
        <div>
          <h3 className={`text-lg font-semibold mb-3 ${sectionTitle}`}>Security
Preferences</h3>
          <div className="grid grid-cols-1 sm:grid-cols-2 gap-4">
            <div>
              <label className={`block mb-1 ${labelText}`}>Security Question</label>
              <select
                name="securityQuestion"
                onChange={handleChange}
                className={selectBase}
                value={formData.securityQuestion}
              >
                <option value="">Select Security Question</option>
                <option value="Your first pet's name?">
                  Your first pet's name?
                </option>
                <option value="Your mother's maiden name?">
                  Your mother's maiden name?
                </option>
                <option value="Your favorite teacher's name?">
                  Your favorite teacher's name?
                </option>
                <option value="Your birth city?">Your birth city?</option>
              </select>
            </div>

            <div>
              <label className={`block mb-1 ${labelText}`}>Security Answer</label>
              <input
                type="text"
                name="securityAnswer"
                value={formData.securityAnswer}
                onChange={handleChange}
                className={inputBase}
                placeholder="Enter your security answer"
              />
            </div>
          </div>
        </div>

        {/* Save Button */}
        <div>
```

```
            <button
              type="submit"
              disabled={loading}
              className={btnPrimary}
            >
              {loading ? "Saving..." : "Save Changes"}
            </button>
          </div>
        </form>
      </div>
    </div>
  );
};

export default SettingsTab;
```

```
// client/src/components/AttendanceButton.jsx
import React, { useContext, useState } from "react";
import axios from "axios";
import Swal from "sweetalert2";
import { AuthContext } from "../context/AuthContext";

const AttendanceButton = () => {
  const { user } = useContext(AuthContext); // ensure AuthContext provides user
  const [buttonText, setButtonText] = useState("Mark Attendance");
  const [disabled, setDisabled] = useState(false);
  // let text = "Mark Attendance";

  const handleMarkAttendance = async () => {
    if (!user) {
      Swal.fire("Not signed in", "Please login first.", "warning");
      return;
    }

    try {
      const token = localStorage.getItem("token");
      const res = await axios.post(
        "http://localhost:5000/api/attendance/mark",
        {
          employeeId: user.employeeId,
          name: user.name,
          role: user.role,
        },
        {
          headers: { Authorization: `Bearer ${token}` },
        }
      );

      // Success (201)
      Swal.fire({
        icon: "success",
        title: "Attendance marked",
        text: res.data.msg || "Attendance marked successfully",
        timer: 2000,
        showConfirmButton: false,
      });
      setButtonText("Attendance Done");
      setDisabled(true);
    } catch (err) {
      // If server responded with JSON (e.g., 400) then show that message
      const status = err.response?.status;
      const serverMsg = err.response?.data?.msg;

      if (status === 400 && serverMsg) {
        // Duplicate attendance / business validation
        Swal.fire({
```

```
          icon: "info",
          title: "Already marked",
          html: serverMsg,
        });
        setButtonText("Attendance Done");
        setDisabled(true);
        return;
      }

      if (status === 401) {
        // Unauthorized
        Swal.fire({
          icon: "warning",
          title: "Not authorized",
          text: serverMsg || "Please login again.",
        });
        // optionally redirect to login:
        // window.location.href = "/login";
        return;
      }

      // Generic fallback
      Swal.fire({
        icon: "error",
        title: "Something went wrong",
        text: serverMsg || err.message || "Unable to mark attendance.",
      });
    }
  };

  return (
    <button
      onClick={handleMarkAttendance}
      disabled={disabled}
      className={`px-4 py-2 rounded text-white font-medium transition ${
        disabled
          ? "bg-gray-500 cursor-not-allowed"
          : "bg-blue-600 hover:bg-blue-700 cursor-pointer"
      }`}
    >
      {buttonText}
    </button>
  );
};

export default AttendanceButton;
```

```
// client/src/components/BatteryStatus.jsx
import React, { useEffect, useState } from "react";
import { FaBatteryFull, FaBatteryHalf, FaBatteryQuarter } from "react-icons/fa";



import { useContext } from "react";
import { ThemeContext } from "../context/ThemeContext";

const BatteryStatus = () => {
  const [batteryLevel, setBatteryLevel] = useState(null);
  const { isDark } = useContext(ThemeContext);

  useEffect(() => {
    const fetchBattery = async () => {
      if (navigator.getBattery) {
        const battery = await navigator.getBattery();
        setBatteryLevel(battery.level * 100);
```

```
        battery.addEventListener("levelchange", () =>
          setBatteryLevel(battery.level * 100)
        );
      }
    };

    fetchBattery();
  }, []);

  const getColor = () => {
    if (batteryLevel === null) return "text-gray-400";
    if (batteryLevel <= 30) return "text-red-500";
    if (batteryLevel > 30 && batteryLevel <= 50) return "text-yellow-400";
    return "text-green-500";
  };

  const getIcon = () => {
    if (batteryLevel <= 30) return <FaBatteryQuarter />;
    if (batteryLevel <= 70) return <FaBatteryHalf />;
    return <FaBatteryFull />;
  };

  return (
    <div className="flex items-center space-x-1">
      <span className={`text-lg ${getColor()}`}>{getIcon()}</span>
      <span className={`text-sm font-medium ${isDark ? "text-gray-300" : "text-gray-
800"}`}>
        {batteryLevel !== null ? `${Math.round(batteryLevel)}%` : "…"}
      </span>
    </div>
  );
};
export default BatteryStatus;
```

```
// client/src/components/NetworkStatus.jsx
import React, { useEffect, useState } from "react";
import { FaWifi } from "react-icons/fa";

const NetworkIcon = () => {
  const [isOnline, setIsOnline] = useState(navigator.onLine);

  useEffect(() => {
    const handleOnline = () => setIsOnline(true);
    const handleOffline = () => setIsOnline(false);

    window.addEventListener("online", handleOnline);
    window.addEventListener("offline", handleOffline);

    return () => {
      window.removeEventListener("online", handleOnline);
      window.removeEventListener("offline", handleOffline);
    };
  }, []);
  return (
    <FaWifi
      size={20}
      className={`transition-colors duration-300 ${
        isOnline ? "text-green-500" : "text-gray-400"
      }`}
      title={isOnline ? "Online" : "Offline"}
    />
  );
};

export default NetworkIcon;
```

```jsx
// client/src/components/PageWrapper.jsx
import React from "react";
import { motion } from "framer-motion";

const PageWrapper = ({ children }) => {
  return (
    <motion.div
      initial={{ opacity: 0 }}          // Start invisible
      animate={{ opacity: 1 }}          // Fade in
      exit={{ opacity: 0 }}             // Fade out when leaving
      transition={{ duration: 0.8 }}    // duration
      style={{ width: "100%", height: "100%" }}
    >
      {children}
    </motion.div>
  );
};

export default PageWrapper;
```

```jsx
// client/src/components/ThemeButton.jsx
import React, { useContext } from "react";
import { FaMoon, FaSun } from "react-icons/fa";
import { ThemeContext } from "../context/ThemeContext";

export default function ThemeButton({ className = "" }) {
  const { isDark, toggle } = useContext(ThemeContext);

  return (
    <button
      onClick={toggle}
      aria-label={isDark ? "Switch to light mode" : "Switch to dark mode"}
      title={isDark ? "Light mode" : "Dark mode"}
      className={`p-2 rounded-md hover:scale-105 transition-transform ${className}`}
    >
      {isDark ? <FaSun className="text-yellow-400" /> : <FaMoon className="text-gray-700"
/>}
    </button>
  );
}
```

```jsx
// client/src/pages/Unauthorized.jsx
import React, { useContext } from "react";
import { NavLink, useNavigate } from "react-router-dom";
import { AuthContext } from "../context/AuthContext";
const userBlock = "/user-block.gif";

const Unauthorized = () => {
  const { user, logout } = useContext(AuthContext);
  const navigate = useNavigate();

  // safe normalized role -> path
  const rolePath = user?.role ? `/${String(user.role).toLowerCase()}/dashboard` : null;

  const handleTryAgain = () => {
    if (rolePath) {
      navigate(rolePath);
      window.location.reload();
    } else {
      // if no role (not logged in) send to login
      navigate("/login");
    }
  };

  const handleLogout = () => {
    // optional: give user a way to logout and re-login
```

```
    if (logout) logout();
    navigate("/login");
  };

  return (
    <div className="flex flex-col justify-center items-center h-screen text-center bg-
gradient-to-br from-blue-600 to-purple-600 w-screen p-6">
      <img
        src={userBlock}
        alt="Access denied"
        className="rounded-full w-20 h-20 mb-4 object-cover shadow-lg"
      />

      <h3 className="text-3xl font-bold text-white mb-2">Access Denied</h3>

      <p className="text-gray-200 font-normal text-[18px] max-w-xl">
        You do not have permission to view this page.
      </p>

      <div className="flex gap-3 mt-6">
        {rolePath ? (
          <button
            onClick={handleTryAgain}
            className="px-4 py-2 rounded bg-white/10 border border-white/30 text-white
hover:bg-white/20 transition"
          >
            Click twice to try again
          </button>
        ) : (
          <NavLink to="/login" className="px-4 py-2 rounded bg-white text-black">
            Login
          </NavLink>
        )}

        <button
          onClick={handleLogout}
          className="px-4 py-2 rounded bg-red-500 text-white hover:bg-red-600 transition"
        >
          Logout
        </button>
      </div>
    </div>
  );
};

export default Unauthorized;
```

```
// client/src/context/AttendanceContext.jsx
import React, { createContext, useState, useEffect } from "react";
import axios from "axios";

export const AttendanceContext = createContext();

export const AttendanceProvider = ({ children }) => {
  const [attendance, setAttendance] = useState([]);
  const [loading, setLoading] = useState(true);

  const fetchAttendance = async () => {
    try {
      const token = localStorage.getItem("token");
      const res = await axios.get("http://localhost:5000/api/attendance/get", {
        headers: { Authorization: `Bearer ${token}` },
      });
      setAttendance(res.data);
    } catch (err) {
      console.error("Error fetching attendance:", err);
```

```
      setAttendance([]); // ensure empty array
    } finally {
      setLoading(false);
    }
  };

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

  return (
    <AttendanceContext.Provider value={{ attendance, loading, fetchAttendance }}>
      {children}
    </AttendanceContext.Provider>
  );
};
```

```
// client/src/context/AuthContext.jsx
import React, { createContext, useState, useEffect } from "react";

export const AuthContext = createContext();

/**
 * Backward-compatible AuthProvider.
 * - Keeps the same public API: { user, token, login, logout }
 * - Adds `loading` internally to avoid flashes (not required to consume)
 * - Fetches fresh user profile when a token exists
 *
 * Notes:
 * - Uses VITE_API_URL if present, otherwise http://localhost:5000
 * - Expects GET /api/user/me (protected) to return the user object
 */
export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(() => {
    try {
      return JSON.parse(localStorage.getItem("user"));
    } catch {
      return null;
    }
  });

  const [token, setToken] = useState(() => localStorage.getItem("token"));
  const [loading, setLoading] = useState(Boolean(token && !user));

  // Helper to derive API base
  const API_BASE = import.meta.env.VITE_API_URL || "http://localhost:5000";

  useEffect(() => {
    // If there is a token but no user, fetch profile.
    // Also re-run when token changes (e.g. on login/logout).
    let cancelled = false;

    const fetchProfile = async () => {
      if (!token) {
        setLoading(false);
        return;
      }

      // If user already exists and token hasn't changed, skip fetch
      // (this keeps behaviour close to your previous implementation).
      if (user) {
        setLoading(false);
        return;
      }

      setLoading(true);
```

```
      try {
        const resp = await fetch(`${API_BASE}/api/user/me`, {
          headers: { Authorization: `Bearer ${token}` },
        });

        if (!resp.ok) {
          // token invalid/expired -> clear
          setToken(null);
          setUser(null);
          localStorage.removeItem("token");
          localStorage.removeItem("user");
          setLoading(false);
          return;
        }

        // backend returns user object (not wrapped)
        const data = await resp.json();

        // If backend returns { user: {...} } handle it too
        const profile = data?.user ? data.user : data;
        if (!cancelled) {
          setUser(profile);
          localStorage.setItem("user", JSON.stringify(profile));
          setLoading(false);
        }
      } catch (err) {
        console.error("Auth check failed", err);
        if (!cancelled) {
          // keep token as-is for now but clear user to be safe
          setUser(null);
          localStorage.removeItem("user");
          setLoading(false);
        }
      }
    };

    fetchProfile();
    return () => {
      cancelled = true;
    };
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [token]); // only re-run when token changes

  // Called by your login flow. Keep signature same.
  const login = (newToken, newUser) => {
    setToken(newToken);
    setUser(newUser);
    localStorage.setItem("token", newToken);
    localStorage.setItem("user", JSON.stringify(newUser));
  };

  const logout = () => {
    setToken(null);
    setUser(null);
    localStorage.removeItem("token");
    localStorage.removeItem("user");
  };

  return (
    <AuthContext.Provider value={{ user, token, login, logout, loading }}>
      {children}
    </AuthContext.Provider>
  );
};
```

```jsx
// client/src/context/LeaveContext.jsx
import { createContext, useState, useEffect } from "react";
import axios from "axios";

export const LeaveContext = createContext();

export const LeaveProvider = ({ children }) => {
  const [leaves, setLeaves] = useState([]);
  const [loading, setLoading] = useState(true);

  const fetchLeaves = async () => {
    try {
      const token = localStorage.getItem("token");
      const res = await axios.get("http://localhost:5000/api/leaves/show", {
        headers: { Authorization: `Bearer ${token}` },
      });
      setLeaves(res.data);
    } catch (err) {
      console.error("Error fetching leaves:", err);
    } finally {
      setLoading(false);
    }
  };

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

  return (
    <LeaveContext.Provider value={{ leaves, loading, fetchLeaves }}>
      {children}
    </LeaveContext.Provider>
  );
};
```

```jsx
// client/src/context/ThemeContext.jsx
import React, { createContext, useEffect, useState } from "react";

export const ThemeContext = createContext({
  isDark: false,
  toggle: () => {},
});

export const ThemeProvider = ({ children }) => {
  const [isDark, setIsDark] = useState(() => {
    // initial from localStorage or system preference fallback
    try {
      const saved = localStorage.getItem("theme");
      if (saved) return saved === "dark";
      return window.matchMedia && window.matchMedia("(prefers-color-scheme:
dark)").matches;
    } catch {
      return false;
    }
  });

  // Ensure html attribute matches state on mount and state change
  useEffect(() => {
    const html = document.documentElement;
    if (isDark) html.setAttribute("data-theme", "dark");
    else html.removeAttribute("data-theme");
  }, [isDark]);

  // toggle with a short transition helper:
  const toggle = () => {
    const html = document.documentElement;
```

```
    // add transition helper so CSS animates
    html.classList.add("theme-transition");
    // small setTimeout to ensure class applied before changing theme
    setTimeout(() => {
      setIsDark(prev => {
        const next = !prev;
        try { localStorage.setItem("theme", next ? "dark" : "light"); } catch {}
        return next;
      });
      // remove transition helper after duration
      setTimeout(() => html.classList.remove("theme-transition"), 420);
    }, 10);
  };

  return <ThemeContext.Provider value={{ isDark, toggle
}}>{children}</ThemeContext.Provider>;
};
```

```
// client/src/context/UserContext.jsx
import React, { createContext, useCallback, useEffect, useState } from "react";
import axios from "axios";

export const UserContext = createContext({
  users: [],
  loading: false,
  error: null,
  refreshUsers: () => Promise.resolve(),
  getEmployees: () => [],
});

export const UserProvider = ({ children }) => {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // Fetch users from backend
  const fetchUsers = useCallback(async () => {
    setLoading(true);
    setError(null);
    try {
      const token = localStorage.getItem("token");
      const res = await axios.get(
        (process.env.REACT_APP_API_URL || "http://localhost:5000") + "/api/user-data",
        {
          headers: { Authorization: `Bearer ${token}` },
        }
      );
      // Expect res.data to be an array of users or { users: [...] }
      const payload = Array.isArray(res.data) ? res.data : res.data.users || [];
      setUsers(payload);
    } catch (err) {
      console.error("User fetch error:", err);
      setError(err);
      setUsers([]);
    } finally {
      setLoading(false);
    }
  }, []);

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

  // Helper: return only employees (role === 'Employee', case-insensitive)
  const getEmployees = useCallback(
    (extraFilter = null) =>
```

```
      users.filter((u) => {
        const role = (u.role || "").toString().toLowerCase().trim();
        const isEmployee = role === "employee";
        if (!isEmployee) return false;
        if (!extraFilter) return true;
        // extraFilter can be a function or a string to match name/email
        if (typeof extraFilter === "function") return extraFilter(u);
        if (typeof extraFilter === "string") {
          const q = extraFilter.toLowerCase();
          return (
            (u.name || "").toLowerCase().includes(q) ||
            (u.email || "").toLowerCase().includes(q) ||
            (u.employeeId || "").toLowerCase().includes(q)
          );
        }
        return true;
      }),
    [users]
  );

  return (
    <UserContext.Provider
      value={{
        users,
        loading,
        error,
        refreshUsers: fetchUsers,
        getEmployees,
      }}
    >
      {children}
    </UserContext.Provider>
  );
};
```

```
// client/src/routes/AppRoutes.jsx
import React, { useContext } from "react";
import { Routes, Route, NavLink } from "react-router-dom";

import LandingPage from "../pages/landing-page/LandingPage";
import Register from "../pages/register/Register";
import Login from "../pages/login/Login";

// Import Role-based Routes
import EmpDashboard from "../user-pages/employee/dashboard/EmpDashboard";
import EmpProfile from "../user-pages/employee/profile/EmpProfile";
import EmpAttendance from "../user-pages/employee/attendance/EmpAttendance";
import EmpLeave from "../user-pages/employee/leave/EmpLeave";
import EmpTask from "../user-pages/employee/tasks/EmpTask";
import EmpMessages from "../user-pages/employee/messages/EmpMessages";
import EmpSettings from "../user-pages/employee/settings/EmpSettings";
import EmpNotifications from "../user-pages/employee/notifications/EmpNotifications";
import EmpLogout from "../user-pages/employee/EmpLogout";
// import ManagerRoutes from "./ManagerRoutes";
import ManagerDashboard from "../user-pages/manager/dashboard/ManagerDashboard";
import ManagerProfile from "../user-pages/manager/profile/ManagerProfile";
import ManagerAttendance from "../user-pages/manager/attendance/ManagerAttendance";
import ManagerLeave from "../user-pages/manager/leave-requests/ManagerLeaveStatus";
import ManagerTask from "../user-pages/manager/tasks/ManagerTask";
import EditTaskPage from "../user-pages/manager/tasks/EditTaskPage";
import ManagerMessages from "../user-pages/manager/messages/ManagerMessages";
import ManagerSettings from "../user-pages/manager/settings/ManagerSettings";
import ManagerNotifications from "../user-
pages/manager/notifications/ManagerNotifications";
import ManagerLogout from "../user-pages/manager/ManagerLogout";
```

```
import ProtectedRoute from "./ProtectedRoute";
import Unauthorized from "../components/Unauthorized";

import EmpData from "../user-pages/manager/employee_profiles/EmpData";
import EmployeeDetailPage from "../user-
pages/manager/employee_profiles/EmployeeDetailPage";

function AppRoutes() {
  return (
    <Routes>
      {/* 🔓 Public Routes */}
      <Route path="/" element={<LandingPage />} />
      <Route path="/register" element={<Register />} />
      <Route path="/login" element={<Login />} />
      <Route path="/unauthorized" element={<Unauthorized />} />

      {/* 🔒 Employee Routes */}
      <Route
        path="/employee/dashboard"
        element={
          <ProtectedRoute allowedRoles={["Employee"]}>
            <EmpDashboard />
          </ProtectedRoute>
        }
      />
      <Route
        path="/employee/profile"
        element={
          <ProtectedRoute allowedRoles={["Employee"]}>
            <EmpProfile />
          </ProtectedRoute>
        }
      />
      <Route
        path="/employee/attendance"
        element={
          <ProtectedRoute allowedRoles={["Employee"]}>
            <EmpAttendance />
          </ProtectedRoute>
        }
      />
      <Route
        path="/employee/leave-requests"
        element={
          <ProtectedRoute allowedRoles={["Employee"]}>
            <EmpLeave />
          </ProtectedRoute>
        }
      />
      <Route
        path="/employee/tasks"
        element={
          <ProtectedRoute allowedRoles={["Employee"]}>
            <EmpTask />
          </ProtectedRoute>
        }
      />
      <Route
        path="/employee/messages"
        element={
          <ProtectedRoute allowedRoles={["Employee"]}>
            <EmpMessages />
          </ProtectedRoute>
        }
      />
```

```
<Route
  path="/employee/settings"
  element={
    <ProtectedRoute allowedRoles={["Employee"]}>
      <EmpSettings />
    </ProtectedRoute>
  }
/>
<Route
  path="/employee/notifications"
  element={
    <ProtectedRoute allowedRoles={["Employee"]}>
      <EmpNotifications />
    </ProtectedRoute>
  }
/>
<Route
  path="/employee/logout"
  element={
    <ProtectedRoute allowedRoles={["Employee"]}>
      <EmpLogout />
    </ProtectedRoute>
  }
/>

{/* □□ Manager Routes */}
{/* □ Employees Profile */}
<Route
  path="/manager/employee-profiles"
  element={
    <ProtectedRoute allowedRoles={["Manager"]}>
      <EmpData />
    </ProtectedRoute>
  }
/>
{/* Manager: Employee detail */}
<Route
  path="/manager/employee-profiles/:id"
  element={
    <ProtectedRoute allowedRoles={["Manager"]}>
      <EmployeeDetailPage />
    </ProtectedRoute>
  }
/>

{/* Manager: Employee list */}
<Route
  path="/manager/employee-profiles"
  element={
    <ProtectedRoute allowedRoles={["Manager"]}>
      <EmpData />
    </ProtectedRoute>
  }
/>

{/* □ Dashboard */}
<Route
  path="/manager/dashboard"
  element={
    <ProtectedRoute allowedRoles={["Manager"]}>
      <ManagerDashboard />
    </ProtectedRoute>
  }
/>

{/* □ Profile */}
```

```
      <Route
        path="/manager/profile"
        element={
          <ProtectedRoute allowedRoles={["Manager"]}>
            <ManagerProfile />
          </ProtectedRoute>
        }
      />

      {/*  Attendance */}
      <Route
        path="/manager/employee-attendance-status"
        element={
          <ProtectedRoute allowedRoles={["Manager"]}>
            <ManagerAttendance />
          </ProtectedRoute>
        }
      />

      {/*  Leave */}
      <Route
        path="/manager/employee-leave-requests"
        element={
          <ProtectedRoute allowedRoles={["Manager"]}>
            <ManagerLeave />
          </ProtectedRoute>
        }
      />

      {/*  Tasks */}
      <Route
        path="/manager/tasks"
        element={
          <ProtectedRoute allowedRoles={["Manager"]}>
            <ManagerTask />
          </ProtectedRoute>
        }
      />

      <Route
  path="/manager/tasks/:id/edit"
  element={
    <ProtectedRoute allowedRoles={["Manager","HR","Admin"]}>
      <EditTaskPage />
    </ProtectedRoute>
  }
/>

      {/*  Messages */}
      <Route
        path="/manager/messages"
        element={
          <ProtectedRoute allowedRoles={["Manager"]}>
            <ManagerMessages />
          </ProtectedRoute>
        }
      />

      {/*  Notifications */}
      <Route
        path="/manager/notifications"
        element={
          <ProtectedRoute allowedRoles={["Manager"]}>
            <ManagerNotifications />
          </ProtectedRoute>
        }
```

```
      />

      {/*  Settings */}
      <Route
        path="/manager/settings"
        element={
          <ProtectedRoute allowedRoles={["Manager"]}>
            <ManagerSettings />
          </ProtectedRoute>
        }
      />

      {/*  Logout */}
      <Route
        path="/manager/logout"
        element={
          <ProtectedRoute allowedRoles={["Manager"]}>
            <ManagerLogout />
          </ProtectedRoute>
        }
      />
    </Routes>
  );
}

export default AppRoutes;
```

```
// client/src/routes/ProtectedRoute.jsx
import React, { useContext } from "react";
import { Navigate } from "react-router-dom";
import { AuthContext } from "../context/AuthContext";

const ProtectedRoute = ({ children, allowedRoles }) => {
  const { user } = useContext(AuthContext);

  if (!user) return <Navigate to="/login" replace />; // not logged in
  if (allowedRoles && !allowedRoles.includes(user.role)) return <Navigate
to="/unauthorized" replace />;

  return children;
};

export default ProtectedRoute;
```

```
// client/src/user-pages/employee/dashboard/EmpDashboard.jsx
import React from "react";
import EmpDashboardLayout from "../EmpDashboardLayout";
import DashboardTab from "./DashboardTab";

function EmpDashboard() {
  return (
    <>
      <EmpDashboardLayout content={<DashboardTab />} />
    </>
  );
}

export default EmpDashboard;
```

```
// client/src/user-pages/employee/dashboard/DashboardTab.jsx
import React, { useContext } from "react";
import WelcomeBanner from "../../../components/dashboard-ui-components/WelcomeBanner";
import AttendanceSummary from "../../../components/dashboard-ui-
components/AttendanceSummary";
import TaskProgressChart from "../../../components/dashboard-ui-
components/TaskProgressChart";
```

```jsx
import NotificationsList from "../../../components/dashboard-ui-
components/NotificationsList";
import DeadlinesCard from "../../../components/dashboard-ui-components/DeadlinesCard";
import QuickLinks from "../../../components/dashboard-ui-components/QuickLinks";

import { AuthContext } from "../../../context/AuthContext";
import { ThemeContext } from "../../../context/ThemeContext";

const DashboardTab = () => {
  const { user } = useContext(AuthContext);
  const { isDark } = useContext(ThemeContext);

  return (
    <div
      className={`h-full p-6 overflow-y-auto ${
        isDark ? "bg-slate-900 text-slate-100" : "bg-gray-50 text-gray-900"
      }`}
    >
      <div className="max-w-7xl mx-auto space-y-6">
        <WelcomeBanner name={user?.name} role={user?.role} />

        <div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
          <div className="lg:col-span-2 space-y-6">
            <AttendanceSummary />
            <TaskProgressChart />
            <QuickLinks />
          </div>

          <div className="space-y-6">
            <NotificationsList />
            <DeadlinesCard />
          </div>
        </div>
      </div>
    </div>
  );
};

export default DashboardTab;
```

```jsx
// client/src/user-pages/employee/attendance/EmpAttendance.jsx
import React from "react";
import EmpDashboardLayout from "../EmpDashboardLayout";
import AttendanceTab from "./AttendanceTab";

function EmpAttendance() {
  return (
    <>
      <EmpDashboardLayout content={<AttendanceTab />} />
    </>
  );
}

export default EmpAttendance;
```

```jsx
// client/src/user-pages/employee/attendance/AttendanceTab.jsx
import React, { useContext, useMemo } from "react";
import {
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  ResponsiveContainer,
```

```
} from "recharts";
import AttendanceButton from "../../../components/AttendanceButton";
import { AttendanceContext } from "../../../context/AttendanceContext";
import { AuthContext } from "../../../context/AuthContext";
import { ThemeContext } from "../../../context/ThemeContext";

const AttendanceTab = () => {
  const { user } = useContext(AuthContext);
  const { attendance, loading } = useContext(AttendanceContext);
  const { isDark } = useContext(ThemeContext); // <-- must be inside component

  if (loading)
    return (
      <p className={isDark ? "text-gray-200" : "text-gray-700"}>
        Loading attendance...
      </p>
    );

  if (!attendance || attendance.length === 0)
    return (
      <div
        className={`p-6 rounded-lg shadow ${
          isDark ? "bg-slate-800 text-gray-200" : "bg-white text-gray-800"
        }`}
      >
        <div className="flex justify-between mb-4">
          <h2
            className={`text-2xl font-semibold ${
              isDark ? "text-gray-100" : "text-gray-800"
            }`}
          >
            Attendance Overview
          </h2>
          <AttendanceButton />
        </div>
        <p className={isDark ? "text-gray-300" : "text-gray-600"}>
          No attendance records found yet.
        </p>
      </div>
    );

  // Filter current user's attendance records
  const userAttendance = attendance.filter(
    (record) => record.employeeId === user.employeeId
  );

  // Compute summary stats
  const summary = useMemo(() => {
    const result = { present: 0, absent: 0, leave: 0 };
    userAttendance.forEach((record) => {
      if (record.status === "Present") result.present++;
      else if (record.status === "Absent") result.absent++;
      else if (record.status === "Leave") result.leave++;
    });
    return result;
  }, [userAttendance]);

  // Prepare last 30 days data
  const chartData = useMemo(() => {
    const today = new Date();
    const last30Days = Array.from({ length: 30 }).map((_, i) => {
      const date = new Date(today);
      date.setDate(today.getDate() - (29 - i));
      const dateStr = date.toISOString().split("T")[0];
      const record = userAttendance.find((r) => r.date === dateStr);
```

```
      return {
        date: dateStr,
        Present: record?.status === "Present" ? 4 : 0,
        Absent: record?.status === "Absent" ? 1 : 0,
        Leave: record?.status === "Leave" ? 2 : 0,
      };
    });
    return last30Days;
  }, [userAttendance]);

  // Sort attendance log (latest first)
  const attendanceLog = [...userAttendance].sort(
    (a, b) => new Date(b.date) - new Date(a.date)
  );

  // Common bg/text classes based on theme
  const pageBg = isDark ? "bg-app text-gray-200" : "bg-white text-gray-800";
  const panelBg = isDark
    ? "bg-slate-800 border-slate-700"
    : "bg-gray-50 border-gray-100";
  const cardShadow = "shadow-sm";

  return (
    <div className={`h-full p-6 overflow-y-auto ${pageBg} ${cardShadow}`}>
      {/* Header */}
      <div
        className={`mb-6 border-b pb-3 flex items-center justify-between ${
          isDark ? "border-slate-700" : "border-gray-200"
        }`}
      >
        <h2
          className={`text-2xl font-semibold ${
            isDark ? "text-gray-100" : "text-gray-800"
          }`}
        >
          Attendance Overview
        </h2>
        <AttendanceButton />
      </div>

      {/* Summary Cards */}
      <div className="grid grid-cols-1 sm:grid-cols-3 gap-6 mb-8">
        <div
          className={`${
            isDark
              ? "bg-green-900 border-l-4 border-green-700 text-green-200"
              : "bg-green-100 border-l-4 border-green-500 text-green-700"
          } p-4 rounded-lg ${cardShadow}`}
        >
          <h3 className="text-lg font-semibold">Present</h3>
          <p className="text-2xl font-bold">{summary.present}</p>
        </div>

        <div
          className={`${
            isDark
              ? "bg-red-900 border-l-4 border-red-700 text-red-200"
              : "bg-red-100 border-l-4 border-red-500 text-red-700"
          } p-4 rounded-lg ${cardShadow}`}
        >
          <h3 className="text-lg font-semibold">Absent</h3>
          <p className="text-2xl font-bold">{summary.absent}</p>
        </div>

        <div
          className={`${
```

```
        isDark
          ? "bg-yellow-900 border-l-4 border-yellow-700 text-yellow-200"
          : "bg-yellow-100 border-l-4 border-yellow-500 text-yellow-700"
      } p-4 rounded-lg ${cardShadow}`}
    >
      <h3 className="text-lg font-semibold">On Leave</h3>
      <p className="text-2xl font-bold">{summary.leave}</p>
    </div>
  </div>

  {/* Attendance Chart */}
  <div className={`${panelBg} p-5 rounded-2xl ${cardShadow} border mb-8`}>
    <h3
      className={`text-lg font-semibold mb-4 ${
        isDark ? "text-gray-100" : "text-gray-700"
      }`}
    >
      Attendance Trend (Last 30 Days)
    </h3>
    <div className="w-full h-64">
      <ResponsiveContainer>
        <BarChart data={chartData}>
          <CartesianGrid
            strokeDasharray="3 3"
            stroke={isDark ? "#334155" : "#e5e7eb"}
          />
          <XAxis
            dataKey="date"
            tick={{ fontSize: 10, fill: isDark ? "#cbd5e1" : "#374151" }}
          />
          <YAxis
            allowDecimals={false}
            tick={{ fill: isDark ? "#cbd5e1" : "#374151" }}
          />
          <Tooltip
            wrapperStyle={{
              backgroundColor: isDark ? "#0f172a" : "#fff",
              color: isDark ? "#e2e8f0" : "#0f172a",
            }}
          />
          <Legend />
          <Bar dataKey="Present" fill="#4CAF50" />
          <Bar dataKey="Absent" fill="#F44336" />
          <Bar dataKey="Leave" fill="#FFC107" />
        </BarChart>
      </ResponsiveContainer>
    </div>
  </div>

  {/* Attendance Log */}
  <div className={`${panelBg} p-5 rounded-2xl ${cardShadow} border`}>
    <h3
      className={`text-lg font-semibold mb-4 ${
        isDark ? "text-gray-100" : "text-gray-700"
      }`}
    >
      Recent Attendance Log
    </h3>
    <div className="overflow-x-auto">
      <table className="min-w-full border-collapse">
        <thead>
          <tr
            className={`${
              isDark
                ? "bg-indigo-900 text-indigo-100"
                : "bg-indigo-100 text-gray-700"
```

```
                }`}
              >
                <th className="py-2 px-4 text-left">Date</th>
                <th className="py-2 px-4 text-left">Status</th>
                <th className="py-2 px-4 text-left">Remarks</th>
              </tr>
            </thead>
            <tbody>
              {attendanceLog.length > 0 ? (
                attendanceLog.map((record, index) => (
                  <tr
                    key={index}
                    className={`border-b transition-all ${
                      isDark
                        ? "hover:bg-slate-800 border-slate-700"
                        : "hover:bg-indigo-50 border-gray-200"
                    }`}
                  >
                    <td className="py-2 px-4">{record.date}</td>
                    <td
                      className={`py-2 px-4 font-semibold ${
                        record.status === "Present"
                          ? isDark
                            ? "text-green-300"
                            : "text-green-600"
                          : record.status === "Absent"
                          ? isDark
                            ? "text-red-300"
                            : "text-red-600"
                          : isDark
                          ? "text-yellow-300"
                          : "text-yellow-600"
                      }`}
                    >
                      {record.status}
                    </td>
                    <td
                      className={`py-2 px-4 ${
                        isDark ? "text-gray-300" : "text-gray-600"
                      }`}
                    >
                      {record.remarks || "—"}
                    </td>
                  </tr>
                ))
              ) : (
                <tr>
                  <td
                    colSpan="3"
                    className={`py-4 text-center ${
                      isDark ? "text-gray-300" : "text-gray-500"
                    }`}
                  >
                    No records found
                  </td>
                </tr>
              )}
            </tbody>
          </table>
        </div>
      </div>
    </div>
  );
};
export default AttendanceTab;
```

```jsx
// client/src/user-pages/employee/leave/EmpLeave.jsx
import React from "react";
import EmpDashboardLayout from "../EmpDashboardLayout";
import LeaveTab from "./LeaveTab";

function EmpLeave() {
  return (
    <>
      <EmpDashboardLayout content={<LeaveTab />} />
    </>
  );
}

export default EmpLeave;
```

```jsx
// client/src/user-pages/employee/leave/LeaveTab.jsx
import React, { useState, useEffect, useContext } from "react";
import { FaPaperPlane, FaClipboardList } from "react-icons/fa";
import Swal from "sweetalert2";
import axios from "axios";
import AIButton from "../../../components/AIButton";
import { ThemeContext } from "../../../context/ThemeContext";

function LeaveTab() {
  const [formData, setFormData] = useState({
    leaveType: "",
    fromDate: "",
    toDate: "",
    reason: "",
  });
  const [leaveRequests, setLeaveRequests] = useState([]);
  const [user, setUser] = useState(null);
  const { isDark } = useContext(ThemeContext); // <-- inside component

  // Theme-based classes
  const pageBg = isDark ? "bg-app text-gray-200" : "bg-white text-gray-800";
  const panelBg = isDark ? "bg-slate-800 border-slate-700" : "bg-gray-50 border-gray-100";
  const inputBase = isDark
    ? "w-full border p-2 rounded-lg bg-slate-700 text-gray-100 border-slate-600 focus:ring-
0 focus:border-indigo-500"
    : "w-full border p-2 rounded-lg bg-white text-gray-700 border-gray-300 focus:ring-0
focus:border-indigo-400";
  const selectBase = inputBase;
  const textareaBase = inputBase;
  const headerText = isDark ? "text-gray-100" : "text-gray-800";
  const tableText = isDark ? "text-gray-200" : "text-gray-800";
  const tableHeaderBg = isDark ? "bg-indigo-900 text-indigo-100" : "bg-indigo-100 text-
gray-700";
  const rowHover = isDark ? "hover:bg-slate-800 border-slate-700" : "hover:bg-indigo-50
border-gray-200";

  // Load user details (from localStorage)
  useEffect(() => {
    const userData = JSON.parse(localStorage.getItem("user"));
    if (userData) {
      setUser(userData);
      fetchLeaves();
    }
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  // Fetch leaves from backend
  const fetchLeaves = async () => {
    try {
      const token = localStorage.getItem("token");
      const res = await axios.get("http://localhost:5000/api/leaves/show", {
```

```
        headers: { Authorization: `Bearer ${token}` },
      });
      setLeaveRequests(res.data || []);
    } catch (error) {
      console.error("Error fetching leaves:", error);
    }
  };

  // Handle input change
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((p) => ({ ...p, [name]: value }));
  };

  // Submit leave request
  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!user) {
      Swal.fire("Error", "User not logged in!", "error");
      return;
    }

    try {
      const token = localStorage.getItem("token");
      const payload = {
        employeeId: user.employeeId,
        name: user.name,
        role: user.role,
        leaveType: formData.leaveType,
        fromDate: formData.fromDate,
        toDate: formData.toDate,
        reason: formData.reason,
      };

      const res = await axios.post("http://localhost:5000/api/leaves/apply", payload, {
        headers: { Authorization: `Bearer ${token}` },
      });

      Swal.fire("Success", res.data.msg, "success");
      setFormData({ leaveType: "", fromDate: "", toDate: "", reason: "" });
      fetchLeaves(); // Refresh list
    } catch (error) {
      const msg = error.response?.data?.msg || "Failed to submit leave request!";
      Swal.fire("Error", msg, "error");
    }
  };

  return (
    <div className={`h-full shadow-md p-6 overflow-y-auto ${pageBg}`}>
      {/* Header */}
      <div className={`mb-6 border-b pb-3 flex items-center justify-between ${isDark ?
"border-slate-700" : "border-gray-200"}`}>
        <h2 className={`text-2xl font-semibold flex items-center gap-2 ${headerText}`}>
          <FaClipboardList className="text-indigo-500" /> Leave Requests
        </h2>
      </div>

      {/* Leave Form */}
      <div className={`${panelBg} border p-5 rounded-2xl shadow-sm mb-8`}>
        <form onSubmit={handleSubmit} className="grid grid-cols-1 md:grid-cols-2 gap-5">
          {/* Employee ID (readonly) */}
          <div>
            <label className={`block font-medium mb-1 ${isDark ? "text-gray-300" : "text-
gray-600"}`}>Employee ID</label>
            <input
              type="text"
```

```
              name="employeeId"
              value={user?.employeeId || ""}
              readOnly
              className={inputBase}
            />
          </div>

          {/* Leave Type */}
          <div>
            <label className={`block font-medium mb-1 ${isDark ? "text-gray-300" : "text-
gray-600"}`}>Leave Type</label>
            <select
              name="leaveType"
              value={formData.leaveType}
              onChange={handleChange}
              required
              className={selectBase}
            >
              <option value="">Select Leave Type</option>
              <option value="Sick Leave">Sick Leave</option>
              <option value="Casual Leave">Casual Leave</option>
              <option value="Earned Leave">Earned Leave</option>
            </select>
          </div>

          {/* Start Date */}
          <div>
            <label className={`block font-medium mb-1 ${isDark ? "text-gray-300" : "text-
gray-600"}`}>Start Date</label>
            <input
              type="date"
              name="fromDate"
              value={formData.fromDate}
              onChange={handleChange}
              required
              className={inputBase}
            />
          </div>

          {/* End Date */}
          <div>
            <label className={`block font-medium mb-1 ${isDark ? "text-gray-300" : "text-
gray-600"}`}>End Date</label>
            <input
              type="date"
              name="toDate"
              value={formData.toDate}
              onChange={handleChange}
              required
              className={inputBase}
            />
          </div>

          {/* Reason */}
          <div className="md:col-span-2">
            <label className={`block font-medium mb-1 ${isDark ? "text-gray-300" : "text-
gray-600"}`}>Reason</label>
            <textarea
              name="reason"
              value={formData.reason}
              onChange={handleChange}
              placeholder="Explain your reason..."
              rows="3"
              required
              className={textareaBase}
            />
```

```
          </div>

          {/* Buttons */}
          <div className="md:col-span-2 text-right flex items-center justify-end gap-3">
            <AIButton name="Reason" />
            <button
              type="submit"
              className="bg-indigo-600 text-white px-6 py-2 rounded-lg shadow hover:bg-
indigo-700 transition-all flex items-center gap-2"
            >
              <FaPaperPlane /> Submit Request
            </button>
          </div>
        </form>
      </div>

      {/* Leave History */}
      <div className={`${panelBg} border p-5 rounded-2xl shadow-sm`}>
        <h3 className={`text-lg font-semibold mb-4 ${headerText}`}>Leave Request
History</h3>

        <div className="overflow-x-auto">
          <table className={`min-w-full border-collapse ${tableText}`}>
            <thead>
              <tr className={tableHeaderBg}>
                <th className="py-2 px-4 text-left">Type</th>
                <th className="py-2 px-4 text-left">From</th>
                <th className="py-2 px-4 text-left">To</th>
                <th className="py-2 px-4 text-left">Reason</th>
                <th className="py-2 px-4 text-left">Status</th>
              </tr>
            </thead>

            <tbody>
              {leaveRequests && leaveRequests.length > 0 && user ? (
                leaveRequests
                  .filter((req) => req.employeeId === user.employeeId)
                  .map((req) => (
                    <tr key={req._id} className={`border-b transition-all ${rowHover}`}>
                      <td className="py-2 px-4">{req.leaveType}</td>
                      <td className="py-2 px-4">{req.fromDate}</td>
                      <td className="py-2 px-4">{req.toDate}</td>
                      <td className="py-2 px-4">{req.reason}</td>
                      <td
                        className={`py-2 px-4 font-semibold ${
                          req.status === "Approved"
                            ? (isDark ? "text-green-300" : "text-green-600")
                            : req.status === "Rejected"
                            ? (isDark ? "text-red-300" : "text-red-600")
                            : (isDark ? "text-yellow-300" : "text-yellow-600")
                        }`}
                      >
                        {req.status}
                      </td>
                    </tr>
                  ))
              ) : (
                <tr>
                  <td colSpan="5" className={`text-center py-4 ${isDark ? "text-gray-300" :
"text-gray-500"}`}>
                    No leave records found
                  </td>
                </tr>
              )}
            </tbody>
          </table>
```

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

```
// client/src/user-pages/employee/messages/EmpMessages.jsx
import React from "react";
import EmpDashboardLayout from "../EmpDashboardLayout";
import MessagesTab from "./MessagesTab";

function EmpMessages() {
  return (
    <>
      <EmpDashboardLayout content={<MessagesTab />} />
    </>
  );
}
export default EmpMessages;
```

```
// client/src/user-pages/employee/messages/MessagesTab.jsx
import React, { useState } from "react";
import { FaPaperPlane } from "react-icons/fa";
import { useContext } from "react";
import { ThemeContext } from "../../../context/ThemeContext";

// Dummy data (you'll replace this later with API calls)
const usersData = [
  {
    id: 1,
    name: "John Manager",
    role: "Manager",
    avatar: "https://i.pravatar.cc/150?img=3",
    lastMessage: "Please update me on your progress.",
  },
  {
    id: 2,
    name: "HR Priya",
    role: "HR",
    avatar: "https://i.pravatar.cc/150?img=5",
    lastMessage: "Don't forget to mark your attendance today.",
  },
  {
    id: 3,
    name: "Dev Ankit",
    role: "Employee",
    avatar: "https://i.pravatar.cc/150?img=10",
    lastMessage: "Thanks! I'll check the code now.",
  },
];

const initialMessages = {
  1: [
    { fromSelf: false, text: "Hey, how's the dashboard task going?" },
    { fromSelf: true, text: "Almost done! Just finishing the charts." },
  ],
  2: [{ fromSelf: false, text: "Please submit your leave form ASAP." }],
  3: [
    { fromSelf: true, text: "Can you help with the API integration?" },
    { fromSelf: false, text: "Sure, share your code repo." },
  ],
};

export default function MessagesTab() {
  const { isDark } = useContext(ThemeContext);
```

```
const [users, setUsers] = useState(usersData);
const [selectedUser, setSelectedUser] = useState(usersData[0]);
const [messages, setMessages] = useState(initialMessages);
const [newMessage, setNewMessage] = useState("");

// theme classes
const wrapperBg = isDark
  ? "bg-slate-900 text-gray-200"
  : "bg-gray-50 text-gray-800";
const sidebarBg = isDark
  ? "bg-slate-800 border-slate-700"
  : "bg-white border-r border-gray-200";
const sidebarHeader = isDark
  ? "text-gray-100 border-b border-slate-700"
  : "text-gray-800 border-b border-gray-200";
const userRowHover = isDark ? "hover:bg-slate-700" : "hover:bg-gray-100";
const selectedRow = isDark ? "bg-slate-700" : "bg-gray-200";
const chatHeaderBg = isDark
  ? "bg-slate-800 border-b border-slate-700"
  : "bg-white border-b border-gray-200";
const chatAreaBg = isDark ? "bg-slate-900" : "bg-gray-100";
const incomingBubble = isDark
  ? "bg-slate-800 text-gray-200"
  : "bg-white text-gray-800";
const outgoingBubble = "bg-blue-500 text-white";
const inputBg = isDark
  ? "bg-slate-800 text-gray-200 border-slate-700"
  : "bg-white text-gray-800 border-gray-300";
const sendBtn =
  "ml-3 bg-blue-500 hover:bg-blue-600 text-white p-2 rounded-full";

// Send message (in future, connect to backend POST /api/messages)
const handleSend = () => {
  if (newMessage.trim() === "") return;
  const userId = selectedUser.id;

  setMessages((prev) => ({
    ...prev,
    [userId]: [...(prev[userId] || []), { fromSelf: true, text: newMessage }],
  }));
  setNewMessage("");
};

return (
  <div className={`flex h-full shadow-md overflow-hidden ${wrapperBg}`}>
    {/* Sidebar - User List */}
    <div className={`w-1/3 ${sidebarBg} overflow-y-auto`}>
      <h3 className={`text-xl font-semibold p-6 ${sidebarHeader}`}>
        Messages
      </h3>
      {users.map((u) => (
        <div
          key={u.id}
          onClick={() => setSelectedUser(u)}
          className={`flex items-center p-3 cursor-pointer ${userRowHover} ${
            selectedUser?.id === u.id ? selectedRow : ""
          }`}
        >
          <img
            src={u.avatar}
            alt={u.name}
            className="w-10 h-10 rounded-full mr-3"
          />
          <div>
            <p
```

```
            className={`font-medium ${
              isDark ? "text-gray-100" : "text-gray-800"
            }`}
          >
            {u.name}
          </p>
          <p
            className={`text-sm ${
              isDark ? "text-gray-400" : "text-gray-500"
            } truncate w-40`}
          >
            {u.lastMessage}
          </p>
        </div>
      </div>
    ))}
  </div>

  {/* Chat Window */}
  <div className="flex-1 flex flex-col">
    {/* Chat Header */}
    <div className={`flex items-center gap-3 p-4 ${chatHeaderBg}`}>
      <img
        src={selectedUser.avatar}
        alt={selectedUser.name}
        className="w-10 h-10 rounded-full"
      />
      <div>
        <p
          className={`font-semibold ${
            isDark ? "text-gray-100" : "text-gray-800"
          }`}
        >
          {selectedUser.name}
        </p>
        <p
          className={`text-sm ${
            isDark ? "text-gray-400" : "text-gray-500"
          }`}
        >
          {selectedUser.role}
        </p>
      </div>
    </div>

    {/* Chat Messages */}
    <div className={`flex-1 overflow-y-auto p-4 space-y-3 ${chatAreaBg}`}>
      {messages[selectedUser.id]?.map((msg, index) => (
        <div
          key={index}
          className={`flex ${
            msg.fromSelf ? "justify-end" : "justify-start"
          }`}
        >
          <div
            className={`px-4 py-2 rounded-2xl max-w-xs ${
              msg.fromSelf
                ? outgoingBubble + " rounded-br-none"
                : incomingBubble + " rounded-bl-none"
            }`}
          >
            {msg.text}
          </div>
        </div>
      ))}
    </div>
```

```
        {/* Message Input */}
        <div
          className={`p-3 ${
            isDark
              ? "bg-slate-800 border-t border-slate-700"
              : "bg-white border-t border-gray-200"
          } flex items-center`}
        >
          <input
            type="text"
            placeholder="Type a message..."
            value={newMessage}
            onChange={(e) => setNewMessage(e.target.value)}
            className={`flex-1 rounded-full px-4 py-2 outline-none focus:ring-2
${inputBg}`}
          />
          <button onClick={handleSend} className={sendBtn}>
            <FaPaperPlane />
          </button>
        </div>
      </div>
    </div>
  );
}
```

```
// client/src/user-pages/employee/notifications/EmpNotifications.jsx
import React from "react";
import NotificationsTab from "../../../components/notifications/NotificationsTab";
import EmpDashboardLayout from "../EmpDashboardLayout";

function EmpNotifications() {
  return (
    <>
      <EmpDashboardLayout content={<NotificationsTab />} />
    </>
  );
}

export default EmpNotifications;
```

```
// client/src/user-pages/employee/profile/EmpProfile.jsx
import React from "react";
import EmpDashboardLayout from "../EmpDashboardLayout";
import ProfileTab from "./ProfileTab";

function EmpDashboard() {
  return (
    <>
      <EmpDashboardLayout content={<ProfileTab />} />
    </>
  );
}

export default EmpDashboard;
```

```
// client/src/user-pages/employee/profile/ProfileTab.jsx
import React, { useContext } from "react";
import { FaUserEdit } from "react-icons/fa";
import { NavLink } from "react-router-dom";
import { AuthContext } from "../../../context/AuthContext";
import { ThemeContext } from "../../../context/ThemeContext";

function ProfileTab() {
  const { isDark } = useContext(ThemeContext);
  const { user } = useContext(AuthContext);
```

```
  let navUser = user.role.toLowerCase();

  return (
    <div
      className={`h-full shadow-md overflow-y-auto ${
        isDark ? "bg-app" : "bg-white"
      }`}
    >
      {/* Header */}
      <div
        className={`flex flex-col md:flex-row items-center justify-between mb-6 border-b-2
border-b-black/20 p-8 ${
          isDark ? "bg-blue-900/30" : "bg-blue-50"
        }`}
      >
        <div className="flex items-center space-x-4">
          <img
            src={`http://localhost:5000/uploads/${user.photo}`}
            alt="Profile"
            className="w-24 h-24 rounded-full shadow-md border-2 border-indigo-500"
          />
          <div>
            <h2
              className={`text-2xl font-semibold text-gray-800 ${
                isDark ? "text-app" : "text-app"
              }`}
            >
              {user.name}
              <span className="text-sm text-gray-400"> ({user.role})</span>
            </h2>
            <p className="text-gray-500">{user.designation}</p>
            <p className="text-sm text-gray-400">{user.department}</p>
          </div>
        </div>

        <NavLink to={`/${navUser}/settings`}>
          <button
            href="/"
            className="flex items-center mt-4 md:mt-0 bg-indigo-600 text-white px-4 py-2
rounded-lg shadow hover:bg-indigo-700 transition-all"
          >
            <FaUserEdit className="mr-2" /> Edit Profile
          </button>
        </NavLink>
      </div>

      {/* Profile Info Grid */}
      <div className="grid grid-cols-1 md:grid-cols-2 gap-6 px-8">
        {/* Basic Info */}
        <div
          className={` rounded-2xl p-5 shadow-sm ${
            isDark ? "text-gray-200 bg-white/10" : "text-gray-700 bg-gray-50"
          }`}
        >
          <h3 className="text-lg font-semibold mb-4">Basic Information</h3>
          <ul
            className={`space-y-2 ${
              isDark ? "text-gray-300" : "text-gray-600"
            }`}
          >
            <li>
              <strong>Email:</strong> {user.email}
            </li>
            <li>
              <strong>Phone:</strong> {user.phone}
            </li>
```

```
            <li>
              <strong>Gender:</strong> {user.gender}
            </li>
            <li>
              <strong>Date of Joining:</strong>{" "}
              {new Date(user.joiningDate).toLocaleDateString()}
            </li>
          </ul>
        </div>

        {/* Address Info */}
        <div
          className={`rounded-2xl p-5 shadow-sm ${
            isDark ? "text-gray-200 bg-white/10" : "text-gray-700 bg-gray-50"
          }`}
        >
          <h3 className="text-lg font-semibold mb-4">Address Information</h3>
          <p
            className={`space-y-2 ${
              isDark ? "text-gray-300" : "text-gray-600"
            }`}
          >
            {user.address} <br /> City- {user.city}
            <br /> State- {user.state}
            <br /> Pincode- {user.pincode}
            <br /> Country- {user.country}
          </p>
        </div>
      </div>
    </div>
  );
}

export default ProfileTab;
```

```
// client/src/user-pages/employee/settings/EmpSettings.jsx
import React from "react";
import EmpDashboardLayout from "../EmpDashboardLayout";
import SettingsTab from "../../../components/settings/SettingsTab";

function EmpSettings() {
  return (
    <>
      <EmpDashboardLayout content={<SettingsTab />} />
    </>
  );
}

export default EmpSettings;
```

```
// client/src/user-pages/employee/tasks/EmpTask.jsx
import React from "react";
import EmpDashboardLayout from "../EmpDashboardLayout";
import TaskTab from "./TaskTab";

function EmpTask() {
  return (
    <>
      <EmpDashboardLayout content={<TaskTab />} />
    </>
  );
}

export default EmpTask;
```

```jsx
// client/src/user-pages/employee/tasks/TaskTab.jsx
import React, { useEffect, useState, useContext } from "react";
import api from "../../../utils/api";
import { AuthContext } from "../../../context/AuthContext";
import { ThemeContext } from "../../../context/ThemeContext";
import Swal from "sweetalert2";

export default function TaskTab() {
  const { user } = useContext(AuthContext);
  const { isDark } = useContext(ThemeContext);

  const [tasks, setTasks] = useState([]);
  const [loading, setLoading] = useState(true);
  const [actionLoading, setActionLoading] = useState({}); // { [taskId]: true }

  const fetchMyTasks = async () => {
    setLoading(true);
    try {
      const res = await api.get("/manager/tasks/employee/me"); // route exposed
      const list = Array.isArray(res.data) ? res.data : res.data?.data || [];
      setTasks(list);
    } catch (err) {
      console.error("fetch my tasks error:", err);
      setTasks([]);
      Swal.fire(
        "Error",
        err.response?.data?.msg || "Unable to fetch tasks",
        "error"
      );
    } finally {
      setLoading(false);
    }
  };

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

  // centralised status change handler with confirmations and UI feedback
  const changeMyTaskStatus = async (id, newStatus, options = {}) => {
    const { requireConfirm = false, confirmText = "" } = options;

    if (requireConfirm) {
      const ok = await Swal.fire({
        title: `Confirm ${newStatus}?`,
        text:
          confirmText ||
          `Are you sure you want to mark this task as "${newStatus}"?`,
        icon: "question",
        showCancelButton: true,
        confirmButtonText: newStatus,
      });
      if (!ok.isConfirmed) return;
    }

    setActionLoading((s) => ({ ...s, [id]: true }));

    try {
      const res = await api.patch(`/manager/tasks/${id}/status`, {
        status: newStatus,
      });

      // Try to read updated task from response
      const updated =
        res.data && (res.data.updatedTask || res.data.task || res.data);
      if (updated && (updated._id || updated.id)) {
```

```
        setTasks((prev) =>
          prev.map((t) =>
            t._id === id || t.id === id ? { ...t, ...updated } : t
          )
        );
      } else {
        // fallback: refetch
        await fetchMyTasks();
      }

      Swal.fire("Updated", `Task marked ${newStatus}`, "success");
    } catch (err) {
      console.error("changeMyTaskStatus error:", err);
      const msg =
        err.response?.data?.msg ||
        err.response?.data?.error ||
        "Failed to update task status";
      Swal.fire("Error", msg, "error");
    } finally {
      setActionLoading((s) => ({ ...s, [id]: false }));
    }
  };

  // theme classes
  const pageBg = isDark
    ? "bg-slate-900 text-gray-200"
    : "bg-gray-50 text-gray-900";
  const cardBg = isDark
    ? "bg-slate-800 border border-slate-700"
    : "bg-white border border-gray-100";
  const subtleText = isDark ? "text-gray-300" : "text-gray-600";
  const labelText = isDark ? "text-gray-100" : "text-gray-900";

  const statusStyles = {
    Completed: {
      badge: isDark
        ? "bg-green-900/30 text-green-300"
        : "bg-green-100 text-green-800",
      row: isDark ? "ring-1 ring-green-700/30" : "",
    },
    "In Progress": {
      badge: isDark
        ? "bg-yellow-900/25 text-yellow-300"
        : "bg-yellow-100 text-yellow-800",
      row: isDark ? "ring-1 ring-yellow-700/20" : "",
    },
    Pending: {
      badge: isDark
        ? "bg-slate-700/40 text-gray-200"
        : "bg-gray-100 text-gray-800",
      row: "",
    },
    Blocked: {
      badge: isDark ? "bg-red-900/25 text-red-300" : "bg-red-100 text-red-800",
      row: isDark ? "ring-1 ring-red-700/20" : "",
    },
    Other: {
      badge: isDark
        ? "bg-slate-700/30 text-gray-200"
        : "bg-gray-100 text-gray-800",
      row: "",
    },
  };

  if (loading)
    return (
```

```
      <div className={`p-6 min-h-screen ${pageBg}`}>
        <div className="max-w-6xl mx-auto">
          <div className={subtleText}>Loading tasks...</div>
        </div>
      </div>
    );

  return (
    <div className={`p-6 min-h-full ${pageBg}`}>
      <div className="max-w-6xl mx-auto">
        <h2 className="text-xl font-semibold mb-4">My Tasks</h2>

        {tasks.length === 0 ? (
          <div className={`${cardBg} rounded p-4`}>
            <div className={subtleText}>No tasks assigned.</div>
          </div>
        ) : (
          // responsive grid: 1 col sm, 2 col md, 3 col lg
          <div className="grid grid-cols-1 sm:grid-cols-1 md:grid-cols-2 lg:grid-cols-3
gap-4">
            {tasks.map((t) => {
              const tid = t._id || t.id;
              const isBusy = !!actionLoading[tid];
              const statusRaw = String(t.status || "Pending");
              const statusKey = statusStyles[statusRaw] ? statusRaw : "Other";
              const status = statusStyles[statusKey];

              return (
                <article
                  key={tid}
                  className={`${cardBg} rounded-lg p-4 flex flex-col justify-between gap-3
${status.row}`}
                >
                  <header>
                    <div className="flex items-start justify-between gap-2">
                      <div>
                        <h3
                          className="text-lg font-medium"
                          style={{ color: isDark ? undefined : undefined }}
                        >
                          {t.title}
                        </h3>
                        <div
                          className="text-xs mt-1"
                          style={{ color: isDark ? undefined : undefined }}
                        >
                          <span className={subtleText}>Due:</span>{" "}
                          <span className={labelText}>
                            {t.dueDate
                              ? new Date(t.dueDate).toLocaleDateString()
                              : "—"}
                          </span>
                        </div>
                      </div>

                      <div>
                        <span
                          className={`inline-flex items-center px-2 py-1 text-xs font-
medium rounded ${status.badge}`}
                        >
                          {statusKey}
                        </span>
                      </div>
                    </div>
                  </header>
```

```
<div className="flex-1">
  <p
    className="text-sm mt-2"
    style={{ color: isDark ? undefined : undefined }}
  >
    {t.description || "No description provided."}
  </p>

  <div className="mt-3 text-xs">
    <div className={subtleText}>Assignees:</div>
    {Array.isArray(t.assignedTo) &&
    t.assignedTo.length > 0 ? (
      <ul className="mt-1 space-y-1">
        {t.assignedTo.map((a) => (
          <li key={a.employeeId} className="text-sm">
            <span className={labelText}>{a.name || "—"}</span>{" "}
            <span className="text-xs text-gray-400">
              • {a.employeeId}
            </span>
          </li>
        ))}
      </ul>
    ) : (
      <div className="text-sm mt-1 text-gray-400">
        No assignees
      </div>
    )}
  </div>
</div>

<footer className="flex items-center justify-between gap-3 mt-3">
  <div className="text-xs text-gray-400">
    Priority:{" "}
    <span className={labelText}>
      {t.priority || "Medium"}
    </span>
  </div>

  <div className="flex items-center gap-2">
    {/* Start */}
    {statusKey === "Pending" && (
      <button
        onClick={() => changeMyTaskStatus(tid, "In Progress")}
        disabled={isBusy}
        className={`px-3 py-1 rounded text-sm ${
          isBusy
            ? "bg-yellow-200 text-yellow-800/60 cursor-not-allowed"
            : "bg-yellow-100 text-yellow-800"
        }`}
      >
        {isBusy ? "Working..." : "Start"}
      </button>
    )}

    {/* Complete */}
    {statusKey === "In Progress" && (
      <button
        onClick={() =>
          changeMyTaskStatus(tid, "Completed", {
            requireConfirm: true,
            confirmText: "Mark this task as completed?",
          })
        }
        disabled={isBusy}
        className={`px-3 py-1 rounded text-sm ${
          isBusy
```

```
                            ? "bg-green-200 text-green-800/60 cursor-not-allowed"
                            : "bg-green-100 text-green-800"
                        }`}
                      >
                        {isBusy ? "Saving..." : "Complete"}
                      </button>
                    )}

                    {/* If completed or others, show disabled view */}
                    {statusKey === "Completed" && (
                      <div className="px-3 py-1 rounded text-sm bg-green-500 text-white">
                        Done ✓
                      </div>
                    )}
                  </div>
                </footer>
              </article>
            );
          })}
        </div>
      )}
    </div>
  </div>
  );
}
```

```
// client/src/user-pages/employee/EmpDashboardLayout.jsx
import React, { useContext } from "react";
import DashboardNav from "../../components/dashboard-page-components/DashboardNav";
import DashboardSidebar from "../../components/dashboard-page-components/DashboardSidebar";
import { AuthContext } from "../../context/AuthContext";

function EmpDashboardLayout({ content }) {
  const { user } = useContext(AuthContext);
  return (
    <div className="w-screen h-screen flex flex-col bg-blue-500">
      <DashboardNav
        role={user?.role || "Employee"}
        photo={user?.photo || ""}
        name={user?.name || "User"}
        designation={user?.designation || "Employee"}
        country={user?.country || "Unknown"}
      />
      <div className="flex-grow flex justify-center items-center">
        <div className="w-1/6 h-full bg-amber-300">
          <DashboardSidebar user={user.role} />
        </div>
        <div className="w-5/6 h-[664px] bg-gray-100">{content}</div>
      </div>
    </div>
  );
}
export default EmpDashboardLayout;
```

```
// client/src/user-pages/employee/EmpLogout.jsx
import React from "react";
import EmpDashboardLayout from "./EmpDashboardLayout";
import LogoutTab from "../../components/logout/LogoutTab";

function EmpLogout() {
  return (
      <EmpDashboardLayout content={<LogoutTab />} />
  );
}
export default EmpLogout;
```

```jsx
// client/src/user-pages/manager/dashboard/ManagerDashboard.jsxF
import React from "react";
import ManagerDashboardLayout from "../ManagerDashboardLayout";
import DashboardTab from "./DashboardTab";

function ManagerDashboard() {
  return (
    <>
      <ManagerDashboardLayout content={<DashboardTab />} />
    </>
  );
}
export default ManagerDashboard;
```

```jsx
// client/src/user-pages/manager/dashboard/DashboardTab.jsx
import React, { useContext } from "react";
import WelcomeBanner from "../../../components/dashboard-ui-components/WelcomeBanner";
import AttendanceSummary from "../../../components/dashboard-ui-
components/AttendanceSummary";
import TaskProgressChart from "../../../components/dashboard-ui-
components/TaskProgressChart";
import NotificationsList from "../../../components/dashboard-ui-
components/NotificationsList";
import DeadlinesCard from "../../../components/dashboard-ui-components/DeadlinesCard";
import QuickLinks from "../../../components/dashboard-ui-components/QuickLinks";

import { AuthContext } from "../../../context/AuthContext";
import { ThemeContext } from "../../../context/ThemeContext";

const DashboardTab = () => {
  const { user } = useContext(AuthContext);
  const { isDark } = useContext(ThemeContext);
  return (
    <div
      className={`h-full rounded-lg shadow-md p-5 overflow-y-auto ${
        isDark ? "bg-app text-app" : "bg-app text-app"
      }`}
    >
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
        <div className="col-span-1 md:col-span-2 lg:col-span-3">
          <WelcomeBanner name={user.name} />
        </div>
        <AttendanceSummary />
        <DeadlinesCard />
        <TaskProgressChart />
        <NotificationsList />
        <QuickLinks />
      </div>
    </div>
  );
};

export default DashboardTab;
```

```jsx
// client/src/user-pages/manager/attendance/ManagerAttendance.jsx
import React from "react";
import ManagerDashboardLayout from "../ManagerDashboardLayout";
import AttendanceTab from "./AttendanceTab";

function ManagerAttendance() {
  return (
      <ManagerDashboardLayout content={<AttendanceTab />} />
  );
}
export default ManagerAttendance;
```

```jsx
// client/src/user-pages/manager/attendance/AttendanceTab.jsx
import React, { useEffect, useMemo, useState, useContext } from "react";
import api from "../../../utils/api";
import {
  ResponsiveContainer,
  BarChart,
  Bar,
  XAxis,
  YAxis,
  Legend,
  Tooltip as ReTooltip,
  CartesianGrid,
} from "recharts";
import { saveAs } from "file-saver";
import { ThemeContext } from "../../../context/ThemeContext";

// COLORS
const STATUS_COLORS = {
  Present: "#16a34a",
  Absent: "#ef4444",
  Leave: "#f59e0b",
  Holiday: "#3b82f6",
  Other: "#9ca3af",
};

const ATTENDANCE_THRESHOLD = 0.8;

// Month formatter
function monthKeyFromDateISO(iso) {
  if (!iso) return null;
  const d = new Date(iso);
  return `${d.getFullYear()}-${String(d.getMonth() + 1).padStart(2, "0")}`;
}

// Percent formatter
function formatPercent(n) {
  return `${Math.round(n * 100)}%`;
}

// Navigate month helpers
function prevMonth(m) {
  const [y, mm] = m.split("-").map(Number);
  const d = new Date(y, mm - 2, 1);
  return `${d.getFullYear()}-${String(d.getMonth() + 1).padStart(2, "0")}`;
}
function nextMonth(m) {
  const [y, mm] = m.split("-").map(Number);
  const d = new Date(y, mm, 1);
  return `${d.getFullYear()}-${String(d.getMonth() + 1).padStart(2, "0")}`;
}

export default function AttendanceTab() {
  const { isDark } = useContext(ThemeContext);

  const [records, setRecords] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  const [monthFilter, setMonthFilter] = useState(() => {
    const n = new Date();
    return `${n.getFullYear()}-${String(n.getMonth() + 1).padStart(2, "0")}`;
  });
  const [employeeQuery, setEmployeeQuery] = useState("");
  const [selectedEmployee, setSelectedEmployee] = useState(null);

  // Theme classes
```

```
  const pageBg = isDark ? "bg-app text-gray-200" : "bg-white text-gray-800";
  const panelBg = isDark
    ? "bg-slate-800/80 border-slate-700"
    : "bg-white border-gray-100";
  const inputBase = isDark
    ? "border p-2 rounded bg-slate-700 text-gray-200 border-slate-600"
    : "border p-2 rounded bg-white text-gray-800 border-gray-300";
  const smallBtn = isDark
    ? "px-2 py-1 bg-slate-700 text-gray-200 rounded hover:bg-slate-600"
    : "px-2 py-1 bg-gray-200 text-gray-800 rounded hover:bg-gray-300";
  const exportBtn = isDark
    ? "px-3 py-1 bg-green-600 text-white rounded hover:bg-green-500"
    : "px-3 py-1 bg-green-600 text-white rounded hover:bg-green-700";
  const tableHeadBg = isDark
    ? "bg-slate-800 text-slate-100"
    : "bg-gray-50 text-gray-700";
  const rowLowBg = isDark ? "bg-rose-900/30" : "bg-red-50";
  const badgeLow = isDark
    ? "bg-rose-800 text-rose-100"
    : "bg-red-200 text-red-800";
  const badgeOk = isDark
    ? "bg-green-700 text-green-100"
    : "bg-green-100 text-green-800";
  const emptyText = isDark ? "text-gray-400" : "text-gray-500";
  const cardBg = isDark
    ? "bg-slate-800/90 border-slate-700"
    : "bg-white border-gray-100";

  // FETCH ATTENDANCE
  useEffect(() => {
    let cancelled = false;
    async function load() {
      setLoading(true);
      try {
        const res = await api.get("attendance/get");
        const all = Array.isArray(res.data) ? res.data : res.data.data || [];
        const employeesOnly = all.filter(
          (r) => (r.role || "").toLowerCase() === "employee"
        );
        if (!cancelled) setRecords(employeesOnly);
      } catch (err) {
        setError(err);
        setRecords([]);
      } finally {
        if (!cancelled) setLoading(false);
      }
    }
    load();
    return () => (cancelled = true);
  }, []);

  // GROUP BY EMPLOYEE
  const employees = useMemo(() => {
    const map = new Map();
    for (const r of records) {
      const id = r.employeeId;
      if (!map.has(id)) {
        map.set(id, { employeeId: id, name: r.name, records: [] });
      }
      map.get(id).records.push(r);
    }
    return Array.from(map.values());
  }, [records]);

  // PER EMPLOYEE STATS FOR SELECTED MONTH
  const employeeStats = useMemo(() => {
```

```
    const stats = employees.map((emp) => {
      const filtered = emp.records.filter(
        (rec) => monthKeyFromDateISO(rec.date) === monthFilter
      );

      const counts = { Present: 0, Absent: 0, Leave: 0, Holiday: 0, Other: 0 };
      filtered.forEach((x) => {
        const s = x.status || "Other";
        if (counts[s] !== undefined) counts[s]++;
        else counts.Other++;
      });

      const total = filtered.length;
      const presentPct = total ? counts.Present / total : 0;

      return {
        employeeId: emp.employeeId,
        name: emp.name,
        counts,
        total,
        presentPct,
      };
    });

    const q = employeeQuery.trim().toLowerCase();
    if (q)
      return stats.filter(
        (s) =>
          s.name.toLowerCase().includes(q) ||
          s.employeeId.toLowerCase().includes(q)
      );

    return stats;
  }, [employees, monthFilter, employeeQuery]);

  // TOP 3 EMPLOYEES FOR BAR CHART
  const barData = useMemo(() => {
    return [...employeeStats]
      .sort((a, b) => b.presentPct - a.presentPct)
      .slice(0, 3) // ONLY TOP 3
      .map((s) => ({
        name: s.name.length > 12 ? s.name.slice(0, 10) + "..." : s.name,
        pct: Math.round(s.presentPct * 100),
      }));
  }, [employeeStats]);

  // EXPORT CSV
  const exportCSV = () => {
    const header = ["employeeId", "name", "date", "status", "remarks"];
    const rows = records.map((r) => [
      r.employeeId,
      r.name,
      r.date,
      r.status,
      (r.remarks || "").replace(/\n/g, " "),
    ]);

    const csv = [header, ...rows]
      .map((r) => r.map((c) => `"${String(c).replace(/"/g, '""')}"`).join(","))
      .join("\n");

    saveAs(
      new Blob([csv], { type: "text/csv" }),
      `attendance_${monthFilter}.csv`
    );
  };
```

```
if (loading)
  return <div className={`p-6 ${pageBg}`}>Loading attendance…</div>;
if (error)
  return (
    <div className={`p-6 ${pageBg} ${emptyText}`}>
      Error loading attendance.
    </div>
  );

return (
  <div className={`p-6 h-full overflow-hidden ${pageBg}`}>
    {/* HEADER */}
    <div className="flex flex-col md:flex-row md:items-center mb-4 gap-3">
      <h2
        className={`text-xl font-semibold ${
          isDark ? "text-gray-100" : "text-gray-900"
        }`}
      >
        Attendance
      </h2>

      <div className="ml-auto flex flex-wrap gap-3 items-center">
        {/* MONTH PICKER + ARROWS */}
        <div className="flex items-center gap-2">
          <button
            onClick={() => setMonthFilter(prevMonth(monthFilter))}
            className={smallBtn}
          >
            ←
          </button>

          <input
            type="month"
            value={monthFilter}
            onChange={(e) => setMonthFilter(e.target.value)}
            className={inputBase}
          />

          <button
            onClick={() => setMonthFilter(nextMonth(monthFilter))}
            className={smallBtn}
          >
            →
          </button>
        </div>

        {/* SEARCH */}
        <input
          placeholder="Search name or ID"
          value={employeeQuery}
          onChange={(e) => setEmployeeQuery(e.target.value)}
          className={inputBase}
        />

        <button onClick={exportCSV} className={exportBtn}>
          Export CSV
        </button>
      </div>
    </div>

    {/* MAIN GRID WITH SCROLL */}
    <div className="grid grid-cols-1 lg:grid-cols-3 gap-6 h-[80vh]">
      {/* LEFT SIDE TABLE */}
      <div
        className={`col-span-2 ${panelBg} rounded shadow p-4 overflow-y-auto`}
```

```
>
  <table className="w-full text-sm border-collapse">
    <thead className={`${tableHeadBg} sticky top-0`}>
      <tr>
        <th className="p-2 text-left">#</th>
        <th className="p-2 text-left">Name</th>
        <th className="p-2 text-left">ID</th>
        <th className="p-2 text-left">Present</th>
        <th className="p-2 text-left">Absent</th>
        <th className="p-2 text-left">Leave</th>
        <th className="p-2 text-left">Total</th>
        <th className="p-2 text-left">Attendance</th>
        <th className="p-2 text-left">Action</th>
      </tr>
    </thead>

    <tbody>
      {employeeStats.map((s, idx) => {
        const low = s.presentPct < ATTENDANCE_THRESHOLD;
        return (
          <tr
            key={s.employeeId}
            className={`border-b ${low ? rowLowBg : ""}`}
          >
            <td className="p-2">{idx + 1}</td>
            <td className="p-2">{s.name}</td>
            <td className="p-2">{s.employeeId}</td>
            <td className="p-2">{s.counts.Present}</td>
            <td className="p-2">{s.counts.Absent}</td>
            <td className="p-2">{s.counts.Leave}</td>
            <td className="p-2">{s.total}</td>
            <td className="p-2">
              <span
                className={`px-2 py-1 rounded text-sm ${
                  low ? badgeLow : badgeOk
                }`}
              >
                {formatPercent(s.presentPct)}
              </span>
            </td>
            <td className="p-2">
              <button
                onClick={() => setSelectedEmployee(s.employeeId)}
                className={
                  isDark
                    ? "text-indigo-300 hover:underline"
                    : "text-blue-600 hover:underline"
                }
              >
                Details
              </button>
            </td>
          </tr>
        );
      })}

      {employeeStats.length === 0 && (
        <tr>
          <td colSpan="9" className={`p-4 text-center ${emptyText}`}>
            No attendance records.
          </td>
        </tr>
      )}
    </tbody>
  </table>
</div>
```

```
        {/* RIGHT ANALYTICS PANEL - SCROLL FIXED */}
        <div
          className={`${cardBg} rounded shadow p-4 overflow-y-auto relative z-10 min-w-
[300px]`}
        >
          <h3
            className={`font-semibold mb-3 ${
              isDark ? "text-gray-100" : "text-gray-800"
            }`}
          >
            Analytics (Top 3)
          </h3>

          {/* BAR CHART (Top 3) */}
          <div className="h-48 mb-6">
            <ResponsiveContainer width="100%" height="100%">
              <BarChart data={barData}>
                <CartesianGrid
                  strokeDasharray="3 3"
                  stroke={isDark ? "#1f2937" : "#e5e7eb"}
                />
                <XAxis
                  dataKey="name"
                  tick={{ fill: isDark ? "#cbd5e1" : "#374151" }}
                />
                <YAxis
                  unit="%"
                  tick={{ fill: isDark ? "#cbd5e1" : "#374151" }}
                />
                <ReTooltip
                  wrapperStyle={{
                    backgroundColor: isDark ? "#0b1220" : "#fff",
                    color: isDark ? "#e2e8f0" : "#0f172a",
                  }}
                />
                <Legend
                  wrapperStyle={{ color: isDark ? "#e2e8f0" : "#0f172a" }}
                />
                <Bar
                  dataKey="pct"
                  name="Attendance (%)"
                  fill={isDark ? "#60a5fa" : "#2563eb"}
                />
              </BarChart>
            </ResponsiveContainer>
          </div>

          {/* SELECTED EMPLOYEE DETAILS */}
          <div>
            <h4 className={`font-medium mb-2 ${isDark ? "text-gray-100" : ""}`}>
              Selected Employee
            </h4>
            {!selectedEmployee && (
              <div className={emptyText}>Click “Details” from the table.</div>
            )}

            {selectedEmployee &&
              (() => {
                const e = employeeStats.find(
                  (x) => x.employeeId === selectedEmployee
                );
                if (!e)
                  return <div className={emptyText}>No data available.</div>;

                return (
```

```
                  <div className="text-sm space-y-1">
                    <div>
                      <strong>Name:</strong> {e.name}
                    </div>
                    <div>
                      <strong>ID:</strong> {e.employeeId}
                    </div>
                    <div>
                      <strong>Present:</strong> {e.counts.Present}
                    </div>
                    <div>
                      <strong>Absent:</strong> {e.counts.Absent}
                    </div>
                    <div>
                      <strong>Leave:</strong> {e.counts.Leave}
                    </div>
                    <div>
                      <strong>Attendance:</strong>{" "}
                      <span
                        className={`px-2 py-1 rounded ${
                          e.presentPct < ATTENDANCE_THRESHOLD
                            ? badgeLow
                            : badgeOk
                        }`}
                      >
                        {formatPercent(e.presentPct)}
                      </span>
                    </div>
                  </div>
                );
              })()}
            </div>
          </div>
        </div>
      </div>
    );
}
```

```
// client/src/user-pages/manager/employee_profiles/EmpData.jsx
import React from "react";
import ManagerDashboardLayout from "../ManagerDashboardLayout";
import EmpDataTab from "./EmpDataTab";

function EmpData() {
  return (
    <>
      <ManagerDashboardLayout content={<EmpDataTab />} />
    </>
  );
}
export default EmpData;
```

```
// client/src/user-pages/manager/employee_profiles/EmpDataTab.jsx
import React, { useContext, useEffect, useMemo, useState } from "react";
import { useNavigate } from "react-router-dom";
import api from "../../../utils/api";
import { AuthContext } from "../../../context/AuthContext";
import { ThemeContext } from "../../../context/ThemeContext";

export default function EmpDataTab() {
  const { token } = useContext(AuthContext);
  const { isDark } = useContext(ThemeContext);
  const navigate = useNavigate();

  const [employees, setEmployees] = useState([]);
  const [meta, setMeta] = useState({ page: 1, limit: 25, total: 0, pages: 1 });
```

```
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  const [query, setQuery] = useState("");
  const [department, setDepartment] = useState("");
  const [showOnlyEmployees, setShowOnlyEmployees] = useState(true);

  const [selected, setSelected] = useState(null); // employee id for modal

  const fetchEmployees = async (opts = {}) => {
    setLoading(true);
    setError(null);
    try {
      const params = {
        page: opts.page ?? meta.page,
        limit: opts.limit ?? meta.limit,
        search: opts.search ?? query ?? undefined,
        department: opts.department ?? department ?? undefined,
      };

      Object.keys(params).forEach(
        (k) => params[k] === undefined && delete params[k]
      );

      const res = await api.get("/manager/employees", { params });

      const resMeta = res.data?.meta || {
        page: params.page || 1,
        limit: params.limit || 25,
        total: (res.data?.data || []).length,
        pages: 1,
      };
      const resData = res.data?.data || [];

      setEmployees(resData);
      setMeta({
        page: resMeta.page || 1,
        limit: resMeta.limit || params.limit || 25,
        total: resMeta.total || resData.length,
        pages:
          resMeta.pages ||
          Math.max(
            1,
            Math.ceil(
              (resMeta.total || resData.length) /
                (resMeta.limit || params.limit || 25)
            )
          ),
      });
    } catch (err) {
      console.error("fetchEmployees error", err);
      setError(err);
      setEmployees([]);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    if (!token && localStorage.getItem("token")) {
      // api interceptor should already pick it up
    }
    fetchEmployees({ page: 1 });
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [token]);
```

```
  const displayed = useMemo(() => {
    if (!employees) return [];
    if (showOnlyEmployees)
      return employees.filter(
        (u) => (u.role || "").toLowerCase() === "employee"
      );
    if (!query) return employees;
    const q = query.toLowerCase();
    return employees.filter(
      (u) =>
        (u.name || "").toLowerCase().includes(q) ||
        (u.email || "").toLowerCase().includes(q) ||
        (u.employeeId || "").toLowerCase().includes(q)
    );
  }, [employees, showOnlyEmployees, query]);

  // Theme classes
  const pageBg = isDark ? "bg-app text-gray-200" : "bg-white text-gray-800";
  const containerBg = isDark
    ? "bg-slate-800 border-slate-700"
    : "bg-white border-gray-100";
  const inputBase = isDark
    ? "border rounded p-2 bg-slate-700 text-gray-200 border-slate-600"
    : "border rounded p-2 bg-white text-gray-800 border-gray-300";
  const selectBase = inputBase;
  const checkboxBase = isDark
    ? "h-4 w-4 accent-indigo-500"
    : "h-4 w-4 accent-indigo-600";
  const btnPrimary = isDark
    ? "ml-auto bg-indigo-600 text-white px-3 py-1 rounded hover:bg-indigo-500"
    : "ml-auto bg-blue-600 text-white px-3 py-1 rounded hover:bg-blue-700";
  const tableHeadBg = isDark
    ? "bg-slate-800 text-slate-100"
    : "bg-gray-100 text-gray-700";
  const rowBorder = isDark ? "border-slate-700" : "border-gray-200";
  const cellText = isDark ? "text-gray-200" : "text-gray-700";
  const emptyText = isDark ? "text-gray-400" : "text-gray-500";

  if (loading) return <div className={`p-6 ${pageBg}`}>Loading employees…</div>;
  if (error)
    return (
      <div className={`p-6 ${isDark ? "text-red-400" : "text-red-600"}`}>
        Error loading employees. {error.message || ""}
      </div>
    );

  return (
    <div className={`p-4 ${pageBg} h-full`}>
      <div className="mb-4 flex gap-3 items-center">
        <input
          type="text"
          value={query}
          onChange={(e) => setQuery(e.target.value)}
          placeholder="Search by name, email or ID"
          className={`${inputBase} w-72`}
        />

        <select
          value={department}
          onChange={(e) => setDepartment(e.target.value)}
          className={`${selectBase}`}
        >
          <option value="">All Departments</option>
          <option value="Software Development">Software Development</option>
          <option value="Quality Assurance">Quality Assurance</option>
          <option value="UI/UX Design">UI/UX Design</option>
```

```
            <option value="DevOps">DevOps</option>
            <option value="Data Science">Data Science</option>
          </select>

          <label className={`flex items-center gap-2 ${cellText}`}>
            <input
              type="checkbox"
              checked={showOnlyEmployees}
              onChange={() => setShowOnlyEmployees((v) => !v)}
              className={checkboxBase}
            />
            <span className={cellText}>Show only employees</span>
          </label>

          <button
            onClick={() => fetchEmployees({ page: 1, search: query, department })}
            className={btnPrimary}
          >
            Search / Refresh
          </button>
        </div>

        <div className={`${containerBg} rounded shadow overflow-hidden`}>
          <table className="w-full text-sm">
            <thead className={`${tableHeadBg}`}>
              <tr>
                <th className="p-3 text-left">Photo</th>
                <th className="p-3 text-left">Name</th>
                <th className="p-3 text-left">Email</th>
                <th className="p-3 text-left">Role / ID</th>
                <th className="p-3 text-left">Department</th>
                <th className="p-3 text-left">Designation</th>
                <th className="p-3 text-left">Action</th>
              </tr>
            </thead>
            <tbody>
              {displayed.length === 0 ? (
                <tr>
                  <td colSpan="7" className={`p-4 text-center ${emptyText}`}>
                    No employees found.
                  </td>
                </tr>
              ) : (
                displayed.map((u) => (
                  <tr
                    key={u._id || u.id || u.email}
                    className={`border-b ${rowBorder}`}
                  >
                    <td className="p-3">
                      <div className="w-12 h-12 rounded overflow-hidden bg-gray-100">
                        {u.photo ? (
                          <img
                            src={
                              u.photo.startsWith("http")
                                ? u.photo
                                : `http://localhost:5000/uploads/${u.photo}`
                            }
                            alt={u.name}
                            className="w-full h-full object-cover"
                          />
                        ) : (
                          <div className="flex items-center justify-center h-full text-gray-
400">
                            –
                          </div>
                        )}
```

```
                  </div>
                </td>
                <td className={`p-3 font-medium ${cellText}`}>{u.name}</td>
                <td className={`p-3 ${cellText}`}>{u.email}</td>
                <td className={`p-3 ${cellText}`}>
                  {u.role} {u.employeeId ? `• ${u.employeeId}` : ""}
                </td>
                <td className={`p-3 ${cellText}`}>{u.department}</td>
                <td className={`p-3 ${cellText}`}>{u.designation}</td>
                <td className="p-3">
                  <button
                    onClick={() =>
                      navigate(`/manager/employee-profiles/${u._id || u.id}`)
                    }
                    className={`text-sm ${
                      isDark
                        ? "text-indigo-300 hover:underline"
                        : "text-blue-600 hover:underline"
                    }`}
                  >
                    View
                  </button>
                </td>
              </tr>
            ))
          )}
        </tbody>
      </table>
    </div>

    {/* Pagination controls (simple) */}
    <div className="mt-3 flex items-center gap-3">
      <button
        onClick={() => {
          if (meta.page > 1) {
            fetchEmployees({ page: meta.page - 1 });
          }
        }}
        className={`${inputBase} px-3 py-1`}
        disabled={meta.page <= 1}
      >
        Prev
      </button>

      <div className={cellText}>
        Page {meta.page} of {meta.pages}
      </div>

      <button
        onClick={() => {
          if (meta.page < meta.pages) fetchEmployees({ page: meta.page + 1 });
        }}
        className={`${inputBase} px-3 py-1`}
        disabled={meta.page >= meta.pages}
      >
        Next
      </button>
    </div>

    {/* Detail modal */}
    {selected && (
      <EmployeeDetailModal id={selected} onClose={() => setSelected(null)} />
    )}
  </div>
  );
}
```

```
// client/src/user-pages/manager/employee_profiles/EmployeeDetailPage.jsx
import React, { useEffect, useState, useContext } from "react";
import { useParams, useNavigate } from "react-router-dom";
import api from "../../../utils/api";
import Swal from "sweetalert2";
import { ThemeContext } from "../../../context/ThemeContext";

export default function EmployeeDetailPage() {
  const { id } = useParams();
  const navigate = useNavigate();
  const { isDark } = useContext(ThemeContext);

  const [employee, setEmployee] = useState(null);
  const [loading, setLoading] = useState(true);
  const [editing, setEditing] = useState(false);
  const [form, setForm] = useState({});
  const [saving, setSaving] = useState(false);

  useEffect(() => {
    let cancelled = false;

    const normalize = (payload) => {
      const p = payload || {};
      return p.data || p.user || p;
    };

    const fetchEmployee = async () => {
      setLoading(true);
      try {
        const res = await api.get(`/manager/employees/${id}`);
        const obj = normalize(res.data);
        if (!cancelled) {
          setEmployee(obj);
          setForm({
            name: obj.name || "",
            gender: obj.gender || "",
            dob: obj.dob ? new Date(obj.dob).toISOString().slice(0, 10) : "",
            email: obj.email || "",
            phone: obj.phone || "",
            address: obj.address || "",
            city: obj.city || "",
            state: obj.state || "",
            pincode: obj.pincode || "",
            country: obj.country || "",
            emergencyContactName: obj.emergencyContactName || "",
            emergencyPhone: obj.emergencyPhone || "",
            role: obj.role || "",
            employeeId: obj.employeeId || "",
            department: obj.department || "",
            designation: obj.designation || "",
            joiningDate: obj.joiningDate ? new Date(obj.joiningDate).toISOString().slice(0,
10) : "",
            employmentType: obj.employmentType || "",
            reportingManager: obj.reportingManager || "",
            workEmail: obj.workEmail || "",
            workMode: obj.workMode || "",
            photo: obj.photo || "",
            signature: obj.signature || "",
          });
        }
      } catch (err) {
        console.error("fetch employee error", err);
        Swal.fire("Error", err?.response?.data?.msg || "Unable to fetch employee",
"error");
      } finally {
```

```
      if (!cancelled) setLoading(false);
    }
  };

  fetchEmployee();
  return () => {
    cancelled = true;
  };
}, [id]);

const handleChange = (e) => {
  const { name, value } = e.target;
  setForm((p) => ({ ...p, [name]: value }));
};

const handleSave = async () => {
  if (!form.name || !form.email) {
    Swal.fire("Validation", "Name and Email are required.", "warning");
    return;
  }

  setSaving(true);
  try {
    const payload = {
      name: form.name,
      gender: form.gender,
      dob: form.dob || null,
      email: form.email,
      phone: form.phone,
      address: form.address,
      city: form.city,
      state: form.state,
      pincode: form.pincode,
      country: form.country,
      emergencyContactName: form.emergencyContactName,
      emergencyPhone: form.emergencyPhone,
      role: form.role,
      employeeId: form.employeeId,
      department: form.department,
      designation: form.designation,
      joiningDate: form.joiningDate || null,
      employmentType: form.employmentType,
      reportingManager: form.reportingManager,
      workEmail: form.workEmail,
      workMode: form.workMode,
    };

    const res = await api.put(`/manager/employees/${id}`, payload);

    const updated = res.data?.data || res.data?.user || res.data || res;

    setEmployee(updated);
    setEditing(false);
    Swal.fire("Saved", "Employee updated successfully.", "success");
  } catch (err) {
    console.error("update error", err);
    const msg =
      err?.response?.data?.msg ||
      err?.response?.data?.message ||
      err.message ||
      "Failed to update";
    Swal.fire("Error", msg, "error");
  } finally {
    setSaving(false);
  }
};
```

```
  if (loading) return <div className="p-6">Loading employee...</div>;

  if (!employee)
    return (
      <div className="p-6">
        <div className="text-red-600">Employee not found.</div>
        <button onClick={() => navigate(-1)} className="mt-4 px-3 py-1 bg-gray-200
rounded">
          Back
        </button>
      </div>
    );

  // Theme classes
  const outerBg = isDark
    ? "bg-gradient-to-b from-slate-900 via-slate-800 to-slate-900"
    : "bg-gradient-to-b from-[#0f172a] via-[#1e3a8a] to-[#7c3aed]";
  const panelBg = isDark ? "bg-slate-800/90 text-gray-100" : "bg-white/95 text-slate-900";
  const headingText = isDark ? "text-gray-100" : "text-slate-800";
  const labelText = isDark ? "text-gray-300" : "text-gray-500";
  const inputBase = isDark
    ? "w-full border p-2 rounded bg-slate-700 text-gray-100 border-slate-600"
    : "w-full border p-2 rounded bg-white text-gray-900 border-gray-300";
  const textareaBase = inputBase;
  const buttonPrimary = isDark
    ? "px-3 py-2 bg-green-600 text-white rounded-md hover:bg-green-500"
    : "px-3 py-2 bg-green-600 text-white rounded-md hover:bg-green-700";
  const buttonSecondary = isDark
    ? "px-3 py-2 bg-slate-700 text-gray-200 rounded-md hover:bg-slate-600"
    : "px-3 py-2 bg-gray-200 text-gray-800 rounded-md hover:bg-gray-300";
  const metaText = isDark ? "text-gray-300" : "text-gray-600";

  return (
    <div className={`min-h-screen min-w-[99.5vw] w-full py-10 ${outerBg}`}>
      <div className="max-w-5xl mx-auto p-6 md:p-8 rounded-2xl">
        <div className={`${panelBg} rounded-2xl border-1-1 border-b-1 p-6 md:p-8`}>
          <div className="flex items-center justify-between mb-6">
            <h1 className={`text-2xl font-bold ${headingText}`}>Employee Profile</h1>
            <div className="flex items-center gap-3">
              {!editing ? (
                <button onClick={() => setEditing(true)} className="px-3 py-2 bg-blue-600
text-white rounded-md">
                  Edit
                </button>
              ) : (
                <>
                  <button onClick={handleSave} disabled={saving} className={buttonPrimary}>
                    {saving ? "Saving..." : "Save"}
                  </button>
                  <button
                    onClick={() => {
                      setEditing(false);
                      setForm({ ...form });
                    }}
                    className={buttonSecondary}
                  >
                    Cancel
                  </button>
                </>
              )}
              <button onClick={() => navigate(-1)} className={buttonSecondary}>
                Back
              </button>
            </div>
          </div>
```

```
          <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
            {/* Left: photo & basic info */}
            <div className="col-span-1 flex flex-col items-center gap-4">
              <div className="w-40 h-40 rounded-lg overflow-hidden bg-gray-100">
                {employee.photo ? (
                  <img
                    src={employee.photo.startsWith("http") ? employee.photo :
`http://localhost:5000/uploads/${employee.photo}`}
                    alt={employee.name}
                    className="w-full h-full object-cover"
                  />
                ) : (
                  <div className="w-full h-full flex items-center justify-center text-gray-
400">No Photo</div>
                )}
              </div>

              <div className="text-center">
                <div className="text-lg font-semibold">{employee.name}</div>
                <div className={`text-sm ${metaText}`}>
                  {employee.role} {employee.employeeId ? `• ${employee.employeeId}` : ""}
                </div>
              </div>

              <div className="w-full">
                <div className={`text-sm ${labelText} mb-1`}>Contact</div>
                {editing ? (
                  <input name="phone" value={form.phone} onChange={handleChange}
className={inputBase} />
                ) : (
                  <div className="text-sm">{employee.phone || "—"}</div>
                )}

                <div className={`text-sm ${labelText} mt-3 mb-1`}>Email</div>
                {editing ? (
                  <input name="email" value={form.email} onChange={handleChange}
className={inputBase} />
                ) : (
                  <div className="text-sm">{employee.email || "—"}</div>
                )}
              </div>
            </div>

            {/* Middle: personal details */}
            <div className="col-span-1 md:col-span-2">
              <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
                <div>
                  <label className={`text-xs ${labelText}`}>Full Name</label>
                  {editing ? (
                    <input name="name" value={form.name} onChange={handleChange}
className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.name}</div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Gender</label>
                  {editing ? (
                    <select name="gender" value={form.gender} onChange={handleChange}
className={inputBase}>
                      <option value="">Select</option>
                      <option value="Male">Male</option>
                      <option value="Female">Female</option>
                      <option value="Other">Other</option>
```

```
                    </select>
                  ) : (
                    <div className="py-2">{employee.gender || "—"}</div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Date of Birth</label>
                  {editing ? (
                    <input type="date" name="dob" value={form.dob} onChange={handleChange}
                    <div className="py-2">{employee.dob ? new
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Emergency Contact</label>
                  {editing ? (
                    <>
                      <input name="emergencyContactName" value={form.emergencyContactName}
                      <input name="emergencyPhone" value={form.emergencyPhone}
                    </>
                  ) : (
                    <div>{employee.emergencyContactName || "—"}</div>
                    <div>{employee.emergencyPhone
                    </div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Address</label>
                  {editing ? (
                    <textarea name="address" value={form.address} onChange={handleChange}
                  ) : (
                    <div>{employee.address || "—
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>City</label>
                  {editing ? (
                    <input name="city" value={form.city} onChange={handleChange}
                  ) : (
                    <div>{employee.city || "—"}</div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>State / Pincode</label>
                  <div>
                    {editing ? (
                      <>
                        <input name="state" value={form.state} onChange={handleChange}
                        <input name="pincode" value={form.pincode} onChange={handleChange}
                      </>
```

```
                    ) : (
                      <div className="py-2">
                        {(employee.state || "—") + (employee.pincode ? ` •
${employee.pincode}` : "")}
                      </div>
                    )}
                  </div>
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Country</label>
                  {editing ? (
                    <input name="country" value={form.country} onChange={handleChange}
className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.country || "—"}</div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Work Email</label>
                  {editing ? (
                    <input name="workEmail" value={form.workEmail} onChange={handleChange}
className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.workEmail || "—"}</div>
                  )}
                </div>
              </div>

              <div className="my-6 border-t" />

              <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
                <div>
                  <label className={`text-xs ${labelText}`}>Department</label>
                  {editing ? (
                    <input name="department" value={form.department}
onChange={handleChange} className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.department || "—"}</div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Designation</label>
                  {editing ? (
                    <input name="designation" value={form.designation}
onChange={handleChange} className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.designation || "—"}</div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Employee ID</label>
                  {editing ? (
                    <input name="employeeId" value={form.employeeId}
onChange={handleChange} className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.employeeId || "—"}</div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Joining Date</label>
                  {editing ? (
```

```
                    <input type="date" name="joiningDate" value={form.joiningDate}
onChange={handleChange} className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.joiningDate ? new
Date(employee.joiningDate).toLocaleDateString() : "—"}</div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Employment Type</label>
                  {editing ? (
                    <input name="employmentType" value={form.employmentType}
onChange={handleChange} className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.employmentType || "—"}</div>
                  )}
                </div>

                <div>
                  <label className={`text-xs ${labelText}`}>Work Mode</label>
                  {editing ? (
                    <input name="workMode" value={form.workMode} onChange={handleChange}
className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.workMode || "—"}</div>
                  )}
                </div>

                <div className="md:col-span-2">
                  <label className={`text-xs ${labelText}`}>Reporting Manager</label>
                  {editing ? (
                    <input name="reportingManager" value={form.reportingManager}
onChange={handleChange} className={inputBase} />
                  ) : (
                    <div className="py-2">{employee.reportingManager || "—"}</div>
                  )}
                </div>
              </div>
            </div>

            {/* Signature */}
            <div className="mt-6">
              <div className="text-sm font-medium mb-2">Signature</div>
              <div className="w-80 h-28 bg-gray-50 border rounded overflow-hidden">
                {employee.signature ? (
                  <img
                    src={
                      employee.signature.startsWith("http")
                        ? employee.signature
                        : `http://localhost:5000/uploads/${employee.signature}`
                    }
                    alt="signature"
                    className="w-full h-full object-contain"
                  />
                ) : (
                  <div className="flex items-center justify-center h-full text-gray-400">No
Signature</div>
                )}
              </div>
            </div>
          </div>
        </div>
      </div>
    );
}
```

```jsx
// client/src/user-pages/manager/leave-requests/ManagerLeaveStatus.jsx
import React from "react";
import ManagerDashboardLayout from "../ManagerDashboardLayout";
import LeaveRequestsTab from "./LeaveRequestsTab";

function ManagerLeaveStatus() {
  return (
    <>
      <ManagerDashboardLayout content={<LeaveRequestsTab />} />
    </>
  );
}
export default ManagerLeaveStatus;
```

```jsx
// client/src/user-pages/manager/leave-requests/LeaveRequestsTab.jsx
import React, { useEffect, useMemo, useState, useContext } from "react";
import api from "../../../utils/api";
import Swal from "sweetalert2";
import {
  ResponsiveContainer,
  PieChart,
  Pie,
  Cell,
  Tooltip as ReTooltip,
  Legend,
} from "recharts";
import { ThemeContext } from "../../../context/ThemeContext";

const STATUS_COLORS = {
  Pending: "#f59e0b",
  Verified: "#3b82f6",
  Approved: "#16a34a",
  Rejected: "#ef4444",
  Expired: "#9ca3af",
  Other: "#6b7280",
};

export default function LeaveRequestsTab() {
  const { isDark } = useContext(ThemeContext);

  const [leaves, setLeaves] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // UI state
  const [selected, setSelected] = useState(null);
  const [statusFilter, setStatusFilter] = useState("All");
  const [query, setQuery] = useState("");
  const [sortOrder, setSortOrder] = useState("newest"); // newest | oldest

  // Theme classes
  const pageBg = isDark ? "bg-slate-900 text-gray-200" : "bg-white text-gray-900";
  const panelBg = isDark ? "bg-slate-800 border-slate-700" : "bg-white border-gray-100";
  const inputBase = isDark
    ? "border p-2 rounded bg-slate-700 text-gray-200 border-slate-600"
    : "border p-2 rounded bg-white text-gray-800 border-gray-300";
  const selectBase = inputBase;
  const btnReset = isDark ? "px-3 py-1 bg-slate-700 text-gray-200 rounded hover:bg-slate-
600" : "px-3 py-1 bg-gray-100 rounded hover:bg-gray-200";
  const btnPrimary = isDark ? "px-2 py-1 bg-blue-600 text-white rounded hover:bg-blue-500"
: "px-2 py-1 bg-blue-600 text-white rounded hover:bg-blue-700";
  const tableHeadBg = isDark ? "bg-slate-800 text-slate-100" : "bg-gray-50 text-gray-700";
  const rowHover = isDark ? "hover:bg-slate-700" : "hover:bg-gray-50";
  const badgeMapDark = {
    Pending: "bg-yellow-800 text-yellow-100",
```

```
    Verified: "bg-blue-900 text-blue-100",
    Approved: "bg-green-900 text-green-100",
    Rejected: "bg-rose-900 text-rose-100",
    Expired: "bg-slate-700 text-slate-100",
    Other: "bg-slate-700 text-slate-100",
  };
  const badgeMapLight = {
    Pending: "bg-yellow-100 text-yellow-800",
    Verified: "bg-blue-100 text-blue-800",
    Approved: "bg-green-100 text-green-800",
    Rejected: "bg-red-100 text-red-800",
    Expired: "bg-gray-200 text-gray-800",
    Other: "bg-gray-100 text-gray-700",
  };
  const emptyText = isDark ? "text-gray-400" : "text-gray-500";
  const analyticsCard = isDark ? "bg-slate-800 border-slate-700 text-gray-200" : "bg-white
border-gray-100 text-gray-800";
  const modalBg = isDark ? "bg-slate-800 text-gray-200" : "bg-white text-gray-900";

  // fetch with fallback path (getLeaves or leaves)
  const fetchLeaves = async () => {
    try {
      setLoading(true);
      setError(null);

      // try existing endpoint first, fallback to manager/leaves
      let res;
      try {
        res = await api.get("/manager/getLeaves");
      } catch (err) {
        res = await api.get("/manager/leaves");
      }

      // res.data may be array or { data: [...] }
      const data = Array.isArray(res.data) ? res.data : res.data.data || [];
      setLeaves(data);
    } catch (err) {
      console.error("fetchLeaves error", err);
      setError(err);
      setLeaves([]);
    } finally {
      setLoading(false);
    }
  };

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

  // Helper: isExpired
  const isExpired = (toDate) => {
    if (!toDate) return false;
    try {
      const t = new Date(toDate);
      const today = new Date();
      // compare date-only
      const tOnly = new Date(t.getFullYear(), t.getMonth(), t.getDate());
      const todayOnly = new Date(today.getFullYear(), today.getMonth(), today.getDate());
      return tOnly < todayOnly;
    } catch {
      return false;
    }
  };

  // Derived filtered list
  const filtered = useMemo(() => {
```

```
    if (!leaves || leaves.length === 0) return [];

    const q = (query || "").trim().toLowerCase();

    const out = leaves
      .map((l) => {
        const expired = isExpired(l.toDate);
        // show 'Expired' visually if expired and still Pending
        const effectiveStatus = expired && l.status === "Pending" ? "Expired" : l.status ||
"Other";
        return { ...l, effectiveStatus, expired: !!expired };
      })
      .filter((l) => {
        // status filter
        if (statusFilter !== "All" && (l.effectiveStatus || "").toLowerCase() !==
statusFilter.toLowerCase()) {
          return false;
        }
        // text search (name, employeeId, leaveType, reason)
        if (!q) return true;
        return (
          (l.name || "").toLowerCase().includes(q) ||
          (l.employeeId || "").toLowerCase().includes(q) ||
          (l.leaveType || "").toLowerCase().includes(q) ||
          (l.reason || "").toLowerCase().includes(q)
        );
      });

    out.sort((a, b) => {
      if (sortOrder === "newest") return new Date(b.appliedAt || b.createdAt || 0) - new
Date(a.appliedAt || a.createdAt || 0);
      return new Date(a.appliedAt || a.createdAt || 0) - new Date(b.appliedAt ||
b.createdAt || 0);
    });

    return out;
  }, [leaves, statusFilter, query, sortOrder]);

  // Pie chart data (counts by effectiveStatus)
  const pieData = useMemo(() => {
    const counts = {};
    for (const l of leaves) {
      const expired = isExpired(l.toDate);
      const st = expired && l.status === "Pending" ? "Expired" : l.status || "Other";
      counts[st] = (counts[st] || 0) + 1;
    }
    return Object.keys(counts).map((k) => ({ name: k, value: counts[k] }));
  }, [leaves]);

  // Action handlers
  const updateStatus = async (id, endpoint, label) => {
    const ok = await Swal.fire({
      title: `Confirm ${label}?`,
      text: `Are you sure you want to ${label.toLowerCase()} this leave?`,
      icon: "question",
      showCancelButton: true,
      confirmButtonText: label,
    });

    if (!ok.isConfirmed) return;

    try {
      // using manager endpoints you defined: /manager/leaves/:id/verify etc.
      await api.put(`/manager/leaves/${id}/${endpoint}`);
      Swal.fire("Success", `Leave ${label.toLowerCase()} successfully`, "success");
      fetchLeaves();
```

```
    } catch (err) {
      console.error("updateStatus error", err);
      Swal.fire("Error", err.response?.data?.msg || "Something went wrong", "error");
    }
  };

  if (loading) return <div className={`p-6 ${pageBg}`}>Loading leaves…</div>;
  if (error) return <div className={`p-6 ${pageBg} ${isDark ? "text-red-400" : "text-red-
600"}`}>Error loading leave requests.</div>;

  return (
    <div className={`p-6 ${pageBg} min-h-full`}>
      <div className="flex items-center justify-between mb-4">
        <h2 className={`text-xl font-semibold ${isDark ? "text-gray-100" : "text-gray-
900"}`}>Employee Leave Requests</h2>

        <div className="flex gap-3 items-center">
          <input
            type="text"
            placeholder="Search by name, id, type, reason..."
            value={query}
            onChange={(e) => setQuery(e.target.value)}
            className={`${inputBase} w-72`}
          />

          <select
            value={statusFilter}
            onChange={(e) => setStatusFilter(e.target.value)}
            className={selectBase}
          >
            <option value="All">All Statuses</option>
            <option value="Pending">Pending</option>
            <option value="Verified">Verified</option>
            <option value="Approved">Approved</option>
            <option value="Rejected">Rejected</option>
            <option value="Expired">Expired</option>
          </select>

          <select
            value={sortOrder}
            onChange={(e) => setSortOrder(e.target.value)}
            className={selectBase}
          >
            <option value="newest">Newest first</option>
            <option value="oldest">Oldest first</option>
          </select>

          <button
            onClick={() => { setQuery(""); setStatusFilter("All"); fetchLeaves(); }}
            className={btnReset}
          >
            Reset
          </button>
        </div>
      </div>

      <div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
        {/* Left: table (spans 2 columns on large screens) */}
        <div className={`lg:col-span-2 ${panelBg} rounded shadow p-4`}>
          <div className="mb-3 flex items-center justify-between">
            <div className={`text-sm ${isDark ? "text-gray-300" : "text-gray-
600"}`}>Showing {filtered.length} results</div>
            <div className={`text-sm ${isDark ? "text-gray-400" : "text-gray-
500"}`}>Refresh to get latest</div>
          </div>
```

```
          <div className="overflow-auto max-h-[68vh]">
            <table className="w-full text-sm border-collapse">
              <thead className={`${tableHeadBg} sticky top-0`}>
                <tr>
                  <th className="p-2 text-left">Name</th>
                  <th className="p-2 text-left">Employee ID</th>
                  <th className="p-2 text-left">Type</th>
                  <th className="p-2 text-left">From</th>
                  <th className="p-2 text-left">To</th>
                  <th className="p-2 text-left">Status</th>
                  <th className="p-2 text-left">Action</th>
                </tr>
              </thead>
              <tbody>
                {filtered.length === 0 && (
                  <tr>
                    <td colSpan="7" className={`p-4 text-center ${emptyText}`}>
                      No leave requests found.
                    </td>
                  </tr>
                )}

                {filtered.map((l) => {
                  const expired = isExpired(l.toDate);
                  const statusForBadge = expired && l.status === "Pending" ? "Expired" :
l.status || "Other";

                  const badgeClassMap = isDark ? badgeMapDark : badgeMapLight;

                  return (
                    <tr key={l._id} className={`border-b ${rowHover}`}>
                      <td className="p-2">{l.name}</td>
                      <td className="p-2">{l.employeeId}</td>
                      <td className="p-2">{l.leaveType}</td>
                      <td className="p-2">{l.fromDate}</td>
                      <td className="p-2">{l.toDate}</td>
                      <td className="p-2">
                        <span className={`px-2 py-1 text-xs rounded
${badgeClassMap[statusForBadge] || (isDark ? "bg-slate-700 text-slate-100" : "bg-gray-100
text-gray-700")}`}>
                          {statusForBadge}
                        </span>
                      </td>

                      <td className="p-2 flex gap-2">
                        <button
                          className={btnPrimary}
                          onClick={() => setSelected(l)}
                        >
                          Details
                        </button>

                        {/* VERIFY button only available when Pending and not expired */}
                        {l.status === "Pending" && !expired && (
                          <button
                            className={isDark ? "px-2 py-1 bg-blue-500 text-white text-xs
rounded hover:bg-blue-400" : "px-2 py-1 bg-blue-500 text-white text-xs rounded hover:bg-
blue-600"}
                            onClick={() => updateStatus(l._id, "verify", "Verify")}
                          >
                            Verify
                          </button>
                        )}

                        {/* APPROVE/REJECT only after verification */}
                        {l.status === "Verified" && (
```

```
                        <>
                          <button
                            className={isDark ? "px-2 py-1 bg-green-500 text-white text-
xs rounded hover:bg-green-400" : "px-2 py-1 bg-green-500 text-white text-xs rounded
hover:bg-green-600"}
                            onClick={() => updateStatus(l._id, "approve", "Approve")}
                          >
                            Approve
                          </button>

                          <button
                            className={isDark ? "px-2 py-1 bg-red-500 text-white text-xs
rounded hover:bg-red-400" : "px-2 py-1 bg-red-500 text-white text-xs rounded hover:bg-red-
600"}
                            onClick={() => updateStatus(l._id, "reject", "Reject")}
                          >
                            Reject
                          </button>
                        </>
                      )}
                    </td>
                  </tr>
                );
              })}
            </tbody>
          </table>
        </div>
      </div>

      {/* Right: Analytics */}
      <div className={`${analyticsCard} rounded shadow p-4`}>
        <h3 className="font-semibold mb-3">Analytics</h3>

        <div style={{ width: "100%", height: 240 }}>
          <ResponsiveContainer width="100%" height="100%">
            <PieChart>
              <Pie
                data={pieData}
                dataKey="value"
                nameKey="name"
                cx="50%"
                cy="50%"
                outerRadius={80}
                label={(entry) => `${entry.name} (${entry.value})`}
              >
                {pieData.map((entry) => (
                  <Cell key={entry.name} fill={STATUS_COLORS[entry.name] ||
STATUS_COLORS.Other} />
                ))}
              </Pie>
              <ReTooltip wrapperStyle={{ backgroundColor: isDark ? "#0b1220" : "#fff",
color: isDark ? "#e2e8f0" : "#0f172a" }} />
              <Legend verticalAlign="bottom" height={36} wrapperStyle={{ color: isDark ?
"#e2e8f0" : "#0f172a" }} />
            </PieChart>
          </ResponsiveContainer>
        </div>

        <div className="mt-4 space-y-2 text-sm">
          <div className="font-medium">Status summary</div>
          <div className="grid grid-cols-2 gap-2">
            {pieData.map((p) => (
              <div key={p.name} className="flex items-center gap-2">
                <span className="w-3 h-3 rounded" style={{ background:
STATUS_COLORS[p.name] || STATUS_COLORS.Other }}></span>
                <div>
```

```
                  <div className="text-sm">{p.name}</div>
                  <div className={`text-xs ${isDark ? "text-gray-400" : "text-gray-
500"}`}>{p.value} request(s)</div>
                </div>
              </div>
            ))}
          </div>
        </div>

        <div className="mt-4">
          <button onClick={() => fetchLeaves()} className={isDark ? "px-3 py-1 bg-blue-
600 text-white rounded hover:bg-blue-500" : "px-3 py-1 bg-blue-600 text-white rounded
hover:bg-blue-700"}>
            Refresh
          </button>
        </div>
      </div>
    </div>

    {/* DETAILS MODAL */}
    {selected && (
      <div className="fixed inset-0 bg-black/40 flex items-center justify-center z-50 p-
4">
        <div className={`${modalBg} rounded-lg p-6 max-w-lg w-full shadow-lg overflow-
auto max-h-[80vh]`}>
          <h3 className="text-lg font-semibold mb-3">Leave Details</h3>

          <div className="space-y-2 text-sm">
            <div><strong>Name:</strong> {selected.name}</div>
            <div><strong>Employee ID:</strong> {selected.employeeId}</div>
            <div><strong>Type:</strong> {selected.leaveType}</div>
            <div><strong>From:</strong> {selected.fromDate}</div>
            <div><strong>To:</strong> {selected.toDate}</div>
            <div><strong>Status:</strong> {selected.status}</div>
            <div><strong>Reason:</strong></div>
            <p className={isDark ? "text-gray-200 bg-slate-700 p-2 rounded whitespace-
pre-wrap" : "text-gray-800 bg-gray-100 p-2 rounded whitespace-pre-
wrap"}>{selected.reason}</p>
            <div className={`text-xs ${isDark ? "text-gray-400" : "text-gray-
500"}`}>Applied: {selected.appliedAt ? new Date(selected.appliedAt).toLocaleString() : "—
"}</div>
          </div>

          <div className="mt-4 flex justify-end gap-3">
            <button onClick={() => setSelected(null)} className={isDark ? "px-3 py-1 bg-
slate-700 text-gray-200 rounded hover:bg-slate-600" : "px-3 py-1 bg-gray-200 text-gray-800
rounded hover:bg-gray-300"}>Close</button>
          </div>
        </div>
      </div>
    )}
  </div>
  );
}
```

```
// client/src/user-pages/manager/messages/ManagerMessages.jsx
import React from "react";
import ManagerDashboardLayout from "../ManagerDashboardLayout";
import MessagesTab from "./MessagesTab";

function ManagerMessages() {
  return (
      <ManagerDashboardLayout content={<MessagesTab />} />
  );
}
export default ManagerMessages;
```

```jsx
// client/src/user-pages/manager/messages/MessagesTab.jsx
import React, { useState, useContext } from "react";
import { FaPaperPlane } from "react-icons/fa";
import { ThemeContext } from "../../../context/ThemeContext";

const initialMessages = {
  1: [
    { fromSelf: false, text: "Hey, how’s the dashboard task going?" },
    { fromSelf: true, text: "Almost done! Just finishing the charts." },
  ],
  2: [{ fromSelf: false, text: "Please submit your leave form ASAP." }],
  3: [
    { fromSelf: true, text: "Can you help with the API integration?" },
    { fromSelf: false, text: "Sure, share your code repo." },
  ],
};

export default function MessagesTab() {
  const { isDark } = useContext(ThemeContext);

  const [users] = useState(usersData);
  const [selectedUser, setSelectedUser] = useState(usersData[0]);
  const [messages, setMessages] = useState(initialMessages);
  const [newMessage, setNewMessage] = useState("");

  // theme classes
  const wrapperBg = isDark ? "bg-app text-gray-200" : "bg-gray-50 text-gray-800";
  const sidebarBg = isDark ? "bg-slate-800 border-slate-700" : "bg-white border-r border-
gray-200";
  const sidebarHeader = isDark ? "text-gray-100 border-b border-slate-700" : "text-gray-800
border-b border-gray-200";
  const userRowHover = isDark ? "hover:bg-slate-700" : "hover:bg-gray-100";
  const selectedRow = isDark ? "bg-slate-700" : "bg-gray-200";
  const chatHeaderBg = isDark ? "bg-slate-800 border-b border-slate-700" : "bg-white
border-b border-gray-200";
  const chatAreaBg = isDark ? "bg-slate-900" : "bg-gray-100";
  const incomingBubble = isDark ? "bg-slate-800 text-gray-200" : "bg-white text-gray-800";
  const outgoingBubble = "bg-blue-500 text-white";
  const inputBg = isDark ? "bg-slate-800 text-gray-200 border-slate-700" : "bg-white text-
gray-800 border-gray-300";
  const sendBtn = "ml-3 bg-blue-500 hover:bg-blue-600 text-white p-2 rounded-full";

  const handleSend = () => {
    if (!selectedUser) return;
    if (newMessage.trim() === "") return;
    const userId = selectedUser.id;

    setMessages((prev) => ({
      ...prev,
      [userId]: [...(prev[userId] || []), { fromSelf: true, text: newMessage }],
    }));
    setNewMessage("");
  };

  return (
    <div className={`flex h-full shadow-md overflow-hidden ${wrapperBg}`}>
      {/* Sidebar - User List */}
      <div className={`w-1/3 ${sidebarBg} overflow-y-auto`}>
        <h3 className={`text-xl font-semibold p-6 ${sidebarHeader}`}>Messages</h3>
        {users.map((u) => (
          <button
            key={u.id}
            onClick={() => setSelectedUser(u)}
            className={`w-full text-left flex items-center p-3 cursor-pointer
${userRowHover} ${
```

```
              selectedUser?.id === u.id ? selectedRow : ""
            }`}
          >
            <img src={u.avatar} alt={u.name} className="w-10 h-10 rounded-full mr-3" />
            <div className="truncate">
              <p className={`font-medium truncate ${isDark ? "text-gray-100" : "text-gray-
800"}`}>{u.name}</p>
              <p className={`text-sm truncate ${isDark ? "text-gray-400" : "text-gray-500"}
w-40`}>
                {u.lastMessage}
              </p>
            </div>
          </button>
        ))}
      </div>

      {/* Chat Window */}
      <div className="flex-1 flex flex-col">
        {/* Chat Header */}
        <div className={`flex items-center gap-3 p-4 ${chatHeaderBg}`}>
          <img
            src={selectedUser?.avatar}
            alt={selectedUser?.name}
            className="w-10 h-10 rounded-full"
          />
          <div>
            <p className={`font-semibold ${isDark ? "text-gray-100" : "text-gray-800"}`}>
              {selectedUser?.name || "Select a user"}
            </p>
            <p className={`text-sm ${isDark ? "text-gray-400" : "text-gray-500"}`}>
              {selectedUser?.role || ""}
            </p>
          </div>
        </div>

        {/* Chat Messages */}
        <div className={`flex-1 overflow-y-auto p-4 space-y-3 ${chatAreaBg}`}>
          {(selectedUser && messages[selectedUser.id]) ? (
            messages[selectedUser.id].map((msg, index) => (
              <div key={index} className={`flex ${msg.fromSelf ? "justify-end" : "justify-
start"}`}>
                <div
                  className={`px-4 py-2 rounded-2xl max-w-xs break-words ${
                    msg.fromSelf ? `${outgoingBubble} rounded-br-none` : `${incomingBubble}
rounded-bl-none`
                  }`}
                >
                  {msg.text}
                </div>
              </div>
            ))
          ) : (
            <div className={isDark ? "text-gray-400" : "text-gray-600"}>No conversation
selected.</div>
          )}
        </div>

        {/* Message Input */}
        <div className={`p-3 ${isDark ? "bg-slate-800 border-t border-slate-700" : "bg-
white border-t border-gray-200"} flex items-center`}>
          <input
            type="text"
            placeholder="Type a message..."
            value={newMessage}
            onChange={(e) => setNewMessage(e.target.value)}
```

```
            className={`flex-1 rounded-full px-4 py-2 outline-none focus:ring-2
${inputBg}`}
            onKeyDown={(e) => { if (e.key === "Enter") handleSend(); }}
            aria-label="Type a message"
          />
          <button onClick={handleSend} className={sendBtn} aria-label="Send message">
            <FaPaperPlane />
          </button>
        </div>
      </div>
    </div>
  );
}
```

```
// client/src/user-pages/manager/notifications/ManagerNotifications.jsx
import React from "react";
import ManagerNotificationsTab from "./ManagerNotificationsTab";
import ManagerDashboardLayout from "../ManagerDashboardLayout";

function ManagerNotifications() {
  return (
    <>
      <ManagerDashboardLayout content={<ManagerNotificationsTab />} />
    </>
  );
}

export default ManagerNotifications;
```

```
// client/src/user-pages/manager/notifications/ManagerNotificationsTab.jsx
import React, { useEffect, useMemo, useState, useContext } from "react";
import api from "../../../utils/api";
import Swal from "sweetalert2";
import CreateNotificationModal from "./CreateNotificationModal";
import { HiMiniBellAlert } from "react-icons/hi2";
import { IoIosRemoveCircleOutline } from "react-icons/io";
import { ThemeContext } from "../../../context/ThemeContext";

const TYPE_COLORS = {
  Announcement: { bg: "bg-blue-50", text: "text-blue-700", dot: "bg-blue-500" },
  Blog: { bg: "bg-green-50", text: "text-green-700", dot: "bg-green-500" },
  Notice: { bg: "bg-yellow-50", text: "text-yellow-800", dot: "bg-yellow-500" },
  Update: { bg: "bg-purple-50", text: "text-purple-700", dot: "bg-purple-500" },
  Other: { bg: "bg-gray-50", text: "text-gray-700", dot: "bg-gray-500" },
};

export default function ManagerNotificationsTab() {
  const { isDark } = useContext(ThemeContext);

  const [data, setData] = useState([]);
  const [meta, setMeta] = useState({});
  const [loading, setLoading] = useState(true);
  const [openCreate, setOpenCreate] = useState(false);

  // filters / query
  const [query, setQuery] = useState("");
  const [typeFilter, setTypeFilter] = useState(""); // "", Announcement, Blog, Notice,
Update
  const [dateFrom, setDateFrom] = useState("");
  const [dateTo, setDateTo] = useState("");

  // paging (optional)
  const [page, setPage] = useState(1);
  const [limit] = useState(24);

  // Theme classes
```

```
  const pageBg = isDark ? "bg-app text-gray-200" : "bg-white text-gray-900";
  const panelBg = isDark ? "bg-slate-800/80 border-slate-700 text-gray-200" : "bg-white
border-gray-100 text-gray-900";
  const inputBase = isDark
    ? "border p-2 rounded bg-slate-700 text-gray-200 border-slate-600"
    : "border p-2 rounded bg-white text-gray-800 border-gray-300";
  const selectBase = inputBase;
  const smallBtn = isDark ? "px-3 py-2 bg-slate-700 text-gray-200 rounded hover:bg-slate-
600" : "px-3 py-2 bg-gray-100 rounded hover:bg-gray-200";
  const primaryBtn = isDark ? "px-4 py-2 text-white rounded shadow" : "px-4 py-2 text-white
rounded shadow";
  const cardBg = isDark ? "bg-slate-800 border-slate-700 text-gray-200" : "bg-white border-
gray-100 text-gray-900";
  const skeletonBg = isDark ? "animate-pulse bg-slate-700/60 h-40 rounded" : "animate-pulse
bg-white h-40 rounded border";
  const badgeContainer = (t) =>
    isDark
      ? `px-2 py-1 text-xs font-medium rounded-full ${t.bg.replace("-50", "-
900").replace("bg-gray-50", "bg-slate-700")} ${t.text.replace("700", "100")}`
      : `px-2 py-1 text-xs font-medium rounded-full ${t.bg} ${t.text}`;

  const fetchList = async (opts = {}) => {
    setLoading(true);
    try {
      const params = {
        page: opts.page ?? page,
        limit: opts.limit ?? limit,
      };
      if ((opts.search ?? query).trim()) params.search = opts.search ?? query;
      if ((opts.type ?? typeFilter).trim()) params.type = opts.type ?? typeFilter;
      if ((opts.from ?? dateFrom).trim()) params.from = opts.from ?? dateFrom;
      if ((opts.to ?? dateTo).trim()) params.to = opts.to ?? dateTo;

      const res = await api.get("/manager/notifications", { params });
      setData(res.data.data || []);
      setMeta(res.data.meta || {});
      setPage(params.page);
    } catch (err) {
      console.error("fetch notifications", err);
      setData([]);
      setMeta({});
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchList({ page: 1 });
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  const remove = async (id) => {
    const ok = await Swal.fire({
      title: "Delete?",
      text: "Delete this notification permanently?",
      icon: "warning",
      showCancelButton: true,
      confirmButtonText: "Delete",
    });
    if (!ok.isConfirmed) return;
    try {
      await api.delete(`/manager/notifications/${id}`);
      Swal.fire("Deleted", "Notification deleted", "success");
      fetchList({ page: 1 });
    } catch (err) {
      Swal.fire("Error", err.response?.data?.msg || "Failed to delete", "error");
```

```
    }
  };

  const clearFilters = () => {
    setQuery("");
    setTypeFilter("");
    setDateFrom("");
    setDateTo("");
    setPage(1);
    fetchList({ page: 1, search: "", type: "", from: "", to: "" });
  };

  const filteredCount = useMemo(() => data.length, [data]);

  return (
    <div className={`p-6 ${pageBg} min-h-full`}>
      <div className="flex flex-col md:flex-row md:items-center gap-4 mb-6">
        <div className="flex items-center gap-3">
          <HiMiniBellAlert className="text-blue-500 w-7 h-7" />
          <div>
            <h2 className={`text-xl font-semibold ${isDark ? "text-gray-100" : "text-gray-
900"}`}>Create / Manage Notifications</h2>
            <div className={`text-sm ${isDark ? "text-gray-400" : "text-gray-500"}`}>
              Create announcements, blog posts, notices and updates (Manager / HR / Admin).
            </div>
          </div>
        </div>

        {/* Filters & Actions - grouped and aligned */}
        <div className="ml-auto w-full md:w-auto">
          {/* Top row: search + type + primary actions */}
          <div className={`grid grid-cols-1 md:grid-cols-4 gap-2 items-center`}>
            <input
              placeholder="Search title or message"
              value={query}
              onChange={(e) => setQuery(e.target.value)}
              className={`${inputBase} md:col-span-2`}
            />

            <select
              value={typeFilter}
              onChange={(e) => setTypeFilter(e.target.value)}
              className={selectBase}
            >
              <option value="">All types</option>
              <option value="Announcement">Announcement</option>
              <option value="Blog">Blog</option>
              <option value="Notice">Notice</option>
              <option value="Update">Update</option>
            </select>

            <div className="flex gap-2 justify-end">
              <button
                onClick={() => fetchList({ page: 1 })}
                className={smallBtn}
                aria-label="Search notifications"
              >
                Search
              </button>

              <button
                onClick={clearFilters}
                className={`px-3 py-2 ${isDark ? "bg-slate-700 text-gray-200" : "bg-white
border"} rounded text-sm flex items-center gap-2`}
                aria-label="Clear filters"
                title="Clear filters"
```

```
              >
                <IoIosRemoveCircleOutline /> Clear
              </button>
            </div>
          </div>

          {/* Date range row with apply aligned right */}
          <div className="mt-2 grid grid-cols-1 md:grid-cols-3 gap-2 items-center">
            <input
              type="date"
              value={dateFrom}
              onChange={(e) => setDateFrom(e.target.value)}
              className={inputBase}
              placeholder="From date"
            />
            <input
              type="date"
              value={dateTo}
              onChange={(e) => setDateTo(e.target.value)}
              className={inputBase}
              placeholder="To date"
            />
            <div className="flex gap-2 justify-end">
              <button
                onClick={() => fetchList({ page: 1 })}
                className={`${primaryBtn} bg-blue-600 hover:bg-blue-700 cursor-pointer`}
              >
                Apply Date
              </button>
            </div>
          </div>
        </div>
      </div>

      {/* Summary + New button */}
      <div className="flex items-center justify-between mb-4 gap-3">
        <div className={`${isDark ? "text-gray-300" : "text-gray-600"} text-sm`}>
          Showing <strong>{filteredCount}</strong> notifications
          {meta.total ? <span> • total {meta.total}</span> : null}
        </div>
        <div className="flex items-center gap-2">
          <button
            onClick={() => setOpenCreate(true)}
            className={`${primaryBtn} bg-green-600 hover:bg-green-700 cursor-pointer`}
          >
            New Notification
          </button>
        </div>
      </div>

      {/* Grid */}
      <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
        {loading ? (
          // Loading skeletons
          Array.from({ length: 6 }).map((_, i) => (
            <div key={i} className={skeletonBg} />
          ))
        ) : data.length === 0 ? (
          <div className={`col-span-full p-8 text-center ${isDark ? "text-gray-400" :
"text-gray-500"} ${cardBg} rounded shadow`}>
            No notifications found. Create one using "New Notification".
          </div>
        ) : (
          data.map((n) => {
            const t = TYPE_COLORS[n.type] || TYPE_COLORS.Other;
            return (
```

```
              <article
                key={n._id}
                className={`${cardBg} border rounded-lg shadow-sm hover:shadow-md
transition flex flex-col overflow-hidden`}
                aria-labelledby={`note-${n._id}`}
              >
                <div className="flex gap-4 p-4">
                  <div className="w-24 h-24 rounded overflow-hidden flex-shrink-0 bg-gray-
50 border">
                    {n.image ? (
                      <img
                        src={`http://localhost:5000/notification_uploads/${n.image}`}
                        alt={n.title}
                        className="w-full h-full object-cover"
                      />
                    ) : (
                      <div className="w-full h-full flex items-center justify-center text-
gray-300">
                        No Image
                      </div>
                    )}
                  </div>

                  <div className="flex-1 min-w-0">
                    <div className="flex items-start justify-between gap-3">
                      <h3 id={`note-${n._id}`} className="font-semibold text-lg truncate">
                        {n.title}
                      </h3>

                      <div className={badgeContainer(t)}>
                        <span className={`inline-block w-2 h-2 rounded-full mr-2`} style={{
backgroundColor: t.dot }} />
                        {n.type || "Other"}
                      </div>
                    </div>

                    <p className={`text-sm mt-2 ${isDark ? "text-gray-300" : "text-gray-
600"} line-clamp-3 whitespace-pre-wrap`}>
                      {n.message}
                    </p>

                    <div className={`mt-3 flex items-center justify-between text-xs
${isDark ? "text-gray-400" : "text-gray-400"}`}>
                      <div>{new Date(n.createdAt).toLocaleString()}</div>
                      <div className="text-right">
                        <div className={`text-xs ${isDark ? "text-gray-300" : "text-gray-
500"}`}>By: {n.createdByName || "Manager"}</div>
                      </div>
                    </div>
                  </div>
                </div>

                <div className={`${isDark ? "bg-slate-900/70 border-t border-slate-700" :
"bg-gray-50 border-t"} p-3 flex items-center justify-between gap-2`}>
                  <div className="flex gap-2">
                    <button
                      onClick={() => window.open(`/manager/notifications/${n._id}`,
"_blank")}
                      className={`${isDark ? "px-3 py-1 bg-slate-800 border border-slate-
700 text-gray-200" : "px-3 py-1 bg-white border text-sm"} rounded hover:bg-gray-100`}
                    >
                      View
                    </button>

                    <button
                      onClick={() => {
```

```
                      navigator.clipboard?.writeText(window.location.origin +
`/manager/notifications/${n._id}`);
                      Swal.fire("Copied", "Link copied to clipboard", "success");
                    }}
                    className={`${isDark ? "px-3 py-1 bg-slate-800 border border-slate-
700 text-gray-200" : "px-3 py-1 bg-white border text-sm"} rounded hover:bg-gray-100`}
                  >
                    Share
                  </button>
                </div>

                <div className="flex gap-2">
                  <button
                    onClick={() => remove(n._id)}
                    className="px-3 py-1 bg-red-700 text-white rounded text-sm hover:bg-
red-800"
                    aria-label={`Delete notification ${n.title}`}
                  >
                    Delete
                  </button>
                </div>
              </div>
            </article>
          );
        })
      )}
    </div>

    {/* Pagination controls (simple) */}
    {meta.pages && meta.pages > 1 && (
      <div className="mt-6 flex items-center justify-center gap-3">
        <button
          className={`${isDark ? "px-3 py-1 bg-slate-700 text-gray-200 rounded" : "px-3
py-1 border rounded"}`}
          onClick={() => fetchList({ page: Math.max(1, (page || 1) - 1) })}
          disabled={(page || 1) <= 1}
        >
          Prev
        </button>
        <div className={isDark ? "text-gray-300" : ""}>Page {page || 1} of
{meta.pages}</div>
        <button
          className={`${isDark ? "px-3 py-1 bg-slate-700 text-gray-200 rounded" : "px-3
py-1 border rounded"}`}
          onClick={() => fetchList({ page: Math.min(meta.pages, (page || 1) + 1) })}
          disabled={(page || 1) >= meta.pages}
        >
          Next
        </button>
      </div>
    )}

    <CreateNotificationModal
      open={openCreate}
      onClose={() => {
        setOpenCreate(false);
        fetchList({ page: 1 });
      }}
    />
  </div>
 );
}
```

```
// client/src/user-pages/manager/notifications/CreateNotificationModal.jsx
import React, { useState, useContext, useEffect } from "react";
import api from "../../../utils/api";
```

```
import Swal from "sweetalert2";
import { ThemeContext } from "../../../context/ThemeContext";

export default function CreateNotificationModal({ open, onClose }) {
  const { isDark } = useContext(ThemeContext);

  const [title, setTitle] = useState("");
  const [message, setMessage] = useState("");
  const [type, setType] = useState("Announcement");
  const [image, setImage] = useState(null);
  const [submitting, setSubmitting] = useState(false);

  // reset fields when modal opens/closes
  useEffect(() => {
    if (open) {
      setTitle("");
      setMessage("");
      setType("Announcement");
      setImage(null);
      setSubmitting(false);
    }
  }, [open]);

  if (!open) return null;

  const handleSubmit = async () => {
    if (!title.trim() || !message.trim()) {
      return Swal.fire("Validation", "Title & message required", "warning");
    }

    try {
      setSubmitting(true);
      const fd = new FormData();
      fd.append("title", title);
      fd.append("message", message);
      fd.append("type", type);
      if (image) fd.append("image", image);

      const res = await api.post("/manager/notifications", fd, {
        headers: { "Content-Type": "multipart/form-data" },
      });

      Swal.fire("Created", "Notification created.", "success");
      onClose && onClose(res.data);
    } catch (err) {
      console.error("create notification error", err);
      Swal.fire("Error", err.response?.data?.msg || "Failed to create", "error");
    } finally {
      setSubmitting(false);
    }
  };

  // theme classes
  const overlayBg = "fixed inset-0 z-50 flex items-center justify-center";
  const backdrop = isDark ? "bg-black/60" : "bg-black/40";
  const panelBg = isDark ? "bg-slate-800 text-gray-200 border border-slate-700" : "bg-white
text-gray-900";
  const inputBase = isDark
    ? "w-full border p-2 rounded bg-slate-700 text-gray-100 border-slate-600"
    : "w-full border p-2 rounded bg-white text-gray-900 border-gray-300";
  const textareaBase = inputBase;
  const selectBase = inputBase;
  const fileText = isDark ? "text-sm text-gray-300" : "text-sm text-gray-700";
  const btnCancel = isDark ? "px-3 py-1 bg-slate-700 text-gray-200 rounded hover:bg-slate-
600" : "px-3 py-1 bg-gray-200 text-gray-800 rounded hover:bg-gray-300";
```

```
  const btnPrimary = isDark ? "px-3 py-1 bg-blue-600 text-white rounded hover:bg-blue-500
disabled:opacity-60" : "px-3 py-1 bg-blue-600 text-white rounded hover:bg-blue-700
disabled:opacity-60";
  const closeBtn = isDark ? "text-gray-300 hover:text-gray-100" : "text-gray-600
hover:text-gray-800";

  return (
    <div className={`${overlayBg} ${backdrop} p-4`}>
      <div className={`w-full max-w-xl rounded-lg p-6 ${panelBg} shadow-lg`}>
        <div className="flex justify-between items-center mb-4">
          <h3 className="text-lg font-semibold">New Notification</h3>
          <button onClick={onClose} className={`text-sm ${closeBtn}`}>Close</button>
        </div>

        <div className="space-y-3">
          <div>
            <label className={`text-sm ${isDark ? "text-gray-300" : "text-gray-
700"}`}>Title</label>
            <input
              className={inputBase}
              value={title}
              onChange={(e) => setTitle(e.target.value)}
              placeholder="Enter title"
            />
          </div>

          <div>
            <label className={`text-sm ${isDark ? "text-gray-300" : "text-gray-
700"}`}>Message</label>
            <textarea
              className={textareaBase}
              rows={4}
              value={message}
              onChange={(e) => setMessage(e.target.value)}
              placeholder="Write the message..."
            />
          </div>

          <div className="flex flex-col md:flex-row gap-3">
            <div className="flex-1">
              <label className={`text-sm ${isDark ? "text-gray-300" : "text-gray-
700"}`}>Type</label>
              <select className={selectBase} value={type} onChange={(e) =>
setType(e.target.value)}>
                <option>Announcement</option>
                <option>Blog</option>
                <option>Notice</option>
                <option>Update</option>
              </select>
            </div>

            <div className="flex-1">
              <label className={`text-sm ${isDark ? "text-gray-300" : "text-gray-
700"}`}>Image (optional)</label>
              <div className="flex items-center gap-3">
                <input
                  id="notification-image"
                  type="file"
                  accept="image/*"
                  onChange={(e) => setImage(e.target.files?.[0] ?? null)}
                  className={`text-xs p-2 rounded cursor-pointer ${isDark ? "bg-gray-900 "
: "bg-gray-300 "}`}
                />
                <div className={fileText}>{image ? image.name : "No file selected"}</div>
              </div>
            </div>
```

```
          </div>

          <div className="flex justify-end gap-2 pt-2">
            <button onClick={onClose} className={btnCancel} type="button">Cancel</button>
            <button onClick={handleSubmit} disabled={submitting} className={btnPrimary}
type="button">
              {submitting ? "Saving..." : "Create"}
            </button>
          </div>
        </div>
      </div>
    </div>
  );
}
```

```
// client/src/user-pages/manager/profile/ManagerProfile.jsx
import React from "react";
import ManagerDashboardLayout from "../ManagerDashboardLayout";
import ProfileTab from "./ProfileTab";

function ManagerDashboard() {
  return (
    <>
      <ManagerDashboardLayout content={<ProfileTab />} />
    </>
  );
}

export default ManagerDashboard;
```

```
// client/src/user-pages/manager/profile/ProfileTab.jsx
import React, { useContext } from "react";
import { FaUserEdit } from "react-icons/fa";
import { NavLink } from "react-router-dom";
import { AuthContext } from "../../../context/AuthContext";
import { ThemeContext } from "../../../context/ThemeContext";

function ProfileTab() {
  const { user } = useContext(AuthContext);
  const { isDark } = useContext(ThemeContext);

  // Guard: if user not loaded yet
  if (!user) {
    return (
      <div className={`h-full rounded-lg shadow-md p-6 overflow-y-auto ${isDark ? "bg-
slate-900 text-gray-200" : "bg-white text-gray-900"}`}>
        <div className={isDark ? "text-gray-400" : "text-gray-600"}>Loading profile…</div>
      </div>
    );
  }

  const navUser = (user.role || "user").toLowerCase();

  // Theme classes
  const pageBg = isDark ? "bg-slate-900 text-gray-200" : "bg-white text-gray-900";
  const panelBg = isDark ? "bg-slate-800/80 border-slate-700" : "bg-gray-50";
  const cardBg = isDark ? "bg-slate-800 border-slate-700 text-gray-200" : "bg-gray-50 text-
gray-900";
  const headerText = isDark ? "text-gray-100" : "text-gray-800";
  const metaText = isDark ? "text-gray-300" : "text-gray-500";
  const btnPrimary = isDark ? "flex items-center mt-4 md:mt-0 bg-indigo-600 text-white px-4
py-2 rounded-lg shadow hover:bg-indigo-500 transition-all" : "flex items-center mt-4 md:mt-
0 bg-indigo-600 text-white px-4 py-2 rounded-lg shadow hover:bg-indigo-700 transition-all";
  const imgWrapperBg = isDark ? "bg-slate-700" : "bg-gray-100";
  const borderClass = isDark ? "border border-slate-700" : "border";
```

```
  const joiningDateText = user.joiningDate ? (() => {
    try {
      return new Date(user.joiningDate).toLocaleDateString();
    } catch {
      return user.joiningDate;
    }
  })() : "—";

  return (
    <div className={`h-full shadow-md p-6 overflow-y-auto ${pageBg}`}>
      {/* Header */}
      <div className={`flex flex-col md:flex-row items-center justify-between mb-6 border-b
pb-4 ${isDark ? "border-slate-700" : "border-gray-200"}`}>
        <div className="flex items-center space-x-4">
          <div className={`w-24 h-24 rounded-full shadow-md ${imgWrapperBg} ${borderClass}
overflow-hidden flex items-center justify-center`}>
            {user.photo ? (
              <img
                src={user.photo.startsWith("http") ? user.photo :
`http://localhost:5000/uploads/${user.photo}`}
                alt="Profile"
                className="w-full h-full object-cover"
              />
            ) : (
              <div className={isDark ? "text-gray-400" : "text-gray-400"}>No Photo</div>
            )}
          </div>

          <div>
            <h2 className={`text-2xl font-semibold ${headerText}`}>
              {user.name}
              <span className={`text-sm ${metaText}`}> ({user.role})</span>
            </h2>
            <p className={metaText}>{user.designation || "—"}</p>
            <p className="text-sm" style={{ color: isDark ? "#9CA3AF" : "#6B7280"
}}>{user.department || "—"}</p>
          </div>
        </div>

        <NavLink to={`/${navUser}/settings`}>
          <button className={btnPrimary}>
            <FaUserEdit className="mr-2" /> Edit Profile
          </button>
        </NavLink>
      </div>

      {/* Profile Info Grid */}
      <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
        {/* Basic Info */}
        <div className={`${cardBg} rounded-2xl p-5 shadow-sm ${borderClass}`}>
          <h3 className={`text-lg font-semibold ${headerText} mb-4`}>
            Basic Information
          </h3>
          <ul className="space-y-2" style={{ color: isDark ? "#E5E7EB" : "#374151" }}>
            <li>
              <strong className={headerText}>Email:</strong> <span className={metaText}>
{user.email || "—"}</span>
            </li>
            <li>
              <strong className={headerText}>Phone:</strong> <span className={metaText}>
{user.phone || "—"}</span>
            </li>
            <li>
              <strong className={headerText}>Gender:</strong> <span className={metaText}>
{user.gender || "—"}</span>
            </li>
```

```
            <li>
              <strong className={headerText}>Date of Joining:</strong> <span
className={metaText}> {joiningDateText}</span>
            </li>
          </ul>
        </div>

        {/* Address Info */}
        <div className={`${cardBg} rounded-2xl p-5 shadow-sm ${borderClass}`}>
          <h3 className={`text-lg font-semibold ${headerText} mb-4`}>
            Address Information
          </h3>
          <p className={metaText}>
            {user.address || "—"}<br />
            <strong className={headerText}>City:</strong> {user.city || "—"}<br />
            <strong className={headerText}>State:</strong> {user.state || "—"}<br />
            <strong className={headerText}>Pincode:</strong> {user.pincode || "—"}<br />
            <strong className={headerText}>Country:</strong> {user.country || "—"}
          </p>
        </div>
      </div>
    </div>
  );
}

export default ProfileTab;
```

```
// client/src/user-pages/manager/settings/ManagerSettings.jsx
import React from "react";
import ManagerDashboardLayout from "../ManagerDashboardLayout";
import SettingsTab from "../../../components/settings/SettingsTab";

function ManagerSettings() {
  return (
    <>
      <ManagerDashboardLayout content={<SettingsTab />} />
    </>
  );
}

export default ManagerSettings;
```

```
// client/src/user-pages/manager/tasks/ManagerTask.jsx
import React from "react";
import ManagerDashboardLayout from "../ManagerDashboardLayout";
import TaskTab from "./TaskTab";

function ManagerTask() {
  return (
    <>
      <ManagerDashboardLayout content={<TaskTab />} />
    </>
  );
}

export default ManagerTask;
```

```
// client/src/user-pages/manager/tasks/TaskTab.jsx
import React, { useEffect, useState, useContext } from "react";
import { useNavigate } from "react-router-dom";
import api from "../../../utils/api";
import AssignTaskModal from "./AssignTaskModal";
import { AuthContext } from "../../../context/AuthContext";
import Swal from "sweetalert2";
import { ThemeContext } from "../../../context/ThemeContext";
import { FiCheckCircle, FiClock, FiAlertCircle, FiXCircle } from "react-icons/fi";
```

```
import { TbCpu } from "react-icons/tb";

export default function TaskTab() {
  const { user } = useContext(AuthContext);
  const { isDark } = useContext(ThemeContext);
  const navigate = useNavigate();

  const [tasks, setTasks] = useState([]);
  const [meta, setMeta] = useState({ page: 1, limit: 25, total: 0, pages: 1 });
  const [loading, setLoading] = useState(true);
  const [open, setOpen] = useState(false);
  const [query, setQuery] = useState("");

  const fetchTasks = async (params = {}) => {
    setLoading(true);
    try {
      const res = await api.get("/manager/tasks", {
        params: { page: 1, limit: 100, search: query, ...params },
      });
      setTasks(res.data.data || []);
      setMeta(res.data.meta || {});
    } catch (err) {
      console.error("fetch tasks:", err);
      setTasks([]);
      Swal.fire("Error", err.response?.data?.msg || "Failed to load tasks", "error");
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchTasks();
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  const handleCreated = (task) => {
    fetchTasks();
  };

  const handleDelete = async (taskId) => {
    const result = await Swal.fire({
      title: "Delete task?",
      text: "This action cannot be undone.",
      icon: "warning",
      showCancelButton: true,
      confirmButtonText: "Delete",
      confirmButtonColor: "#d33",
      cancelButtonText: "Cancel",
    });

    if (!result.isConfirmed) return;

    try {
      await api.delete(`/manager/tasks/${taskId}`);
      await Swal.fire({ title: "Deleted", text: "Task deleted.", icon: "success", timer:
1200, showConfirmButton: false });
      fetchTasks();
    } catch (err) {
      console.error("delete task error", err);
      Swal.fire("Error", err.response?.data?.msg || "Failed to delete task", "error");
    }
  };

  // Status mapping for badge text, icon and colors
  const STATUS_MAP = {
    "Completed": {
```

```
      label: "Completed",
      icon: <FiCheckCircle />,
      badge: "bg-green-100 text-green-800",
      row: isDark ? "bg-green-900/20" : "bg-green-50",
    },
    "In Progress": {
      label: "In Progress",
      icon: <FiClock />,
      badge: "bg-yellow-100 text-yellow-800",
      row: isDark ? "bg-yellow-900/15" : "bg-yellow-50",
    },
    "Pending": {
      label: "Pending",
      icon: <FiClock />,
      badge: "bg-gray-100 text-gray-800",
      row: isDark ? "bg-slate-700/20" : "bg-white",
    },
    "Blocked": {
      label: "Blocked",
      icon: <FiXCircle />,
      badge: "bg-red-100 text-red-800",
      row: isDark ? "bg-red-900/15" : "bg-red-50",
    },
    "Computer": {
      label: "Computer",
      icon: <TbCpu />,
      badge: "bg-indigo-100 text-indigo-800",
      row: isDark ? "bg-indigo-900/12" : "bg-indigo-50",
    },
    "Other": {
      label: "Other",
      icon: <FiAlertCircle />,
      badge: "bg-gray-100 text-gray-800",
      row: isDark ? "bg-slate-700/10" : "bg-white",
    },
  };

  const containerBg = isDark ? "bg-slate-900 text-gray-200" : "bg-gray-50 text-gray-900";
  const panelBg = isDark ? "bg-slate-800 border border-slate-700 text-gray-200" : "bg-white
border border-gray-100";
  const inputBase = isDark
    ? "border p-2 rounded bg-slate-700 border-slate-600 text-gray-100"
    : "border p-2 rounded bg-white border-gray-300 text-gray-900";
  const btnPrimary = isDark
    ? "px-3 py-2 bg-blue-600 text-white rounded hover:bg-blue-500"
    : "px-3 py-2 bg-blue-600 text-white rounded hover:bg-blue-700";
  const btnSecondary = isDark
    ? "px-3 py-2 bg-slate-700 text-gray-100 rounded hover:bg-slate-600"
    : "px-3 py-2 bg-gray-200 text-gray-800 rounded hover:bg-gray-300";

  return (
    <div className={`p-6 min-h-full ${containerBg}`}>
      <div className="max-w-6xl mx-auto">
        <div className="flex flex-col md:flex-row items-start md:items-center justify-
between mb-4 gap-3">
          <h2 className="text-xl font-semibold">Task Management</h2>

          <div className="flex gap-2 items-center w-full md:w-auto">
            <input
              placeholder="Search title/employee"
              value={query}
              onChange={(e) => setQuery(e.target.value)}
              className={`${inputBase} flex-1 md:flex-none`}
              onKeyDown={(e) => {
                if (e.key === "Enter") fetchTasks();
              }}
```

```
            />
            <button onClick={() => fetchTasks()} className={`${btnSecondary} ml-0 md:ml-
2`}>
              Search
            </button>
            <button onClick={() => setOpen(true)} className={`${btnPrimary} ml-auto md:ml-
2`}>
              Assign Task
            </button>
          </div>
        </div>

        <div className={`${panelBg} rounded shadow p-4`}>
          {loading ? (
            <div className={isDark ? "text-gray-300" : "text-gray-600"}>Loading...</div>
          ) : (
            <div className="overflow-auto max-h-[60vh]">
              <table className="w-full text-sm">
                <thead className={`${isDark ? "bg-slate-800" : "bg-gray-50"} sticky top-
0`}>
                  <tr>
                    <th className="p-2 text-left w-12">#</th>
                    <th className="p-2 text-left">Title</th>
                    <th className="p-2 text-left">Assignees</th>
                    <th className="p-2 text-left">Due</th>
                    <th className="p-2 text-left">Priority</th>
                    <th className="p-2 text-left">Status</th>
                    <th className="p-2 text-left">Actions</th>
                  </tr>
                </thead>

                <tbody>
                  {tasks.map((t, idx) => {
                    const statusKey = (t.status || "Other");
                    const map = STATUS_MAP[statusKey] || STATUS_MAP["Other"];
                    return (
                      <tr
                        key={t._id}
                        className={`border-b transition-all ${map.row}`}
                      >
                        <td className="p-2 align-top">{idx + 1}</td>

                        <td className="p-2 font-medium align-top">{t.title}</td>

                        <td className="p-2 align-top">
                          {Array.isArray(t.assignedTo) && t.assignedTo.length > 0 ? (
                            t.assignedTo.map((a) => (
                              <div key={a.employeeId} className="text-xs">
                                {a.name || "—"} <span className="text-gray-400">•
{a.employeeId}</span>
                              </div>
                            ))
                          ) : (
                            <div className="text-xs text-gray-400">No assignees</div>
                          )}
                        </td>

                        <td className="p-2 align-top">{t.dueDate ? new
Date(t.dueDate).toLocaleDateString() : "—"}</td>

                        <td className="p-2 align-top">{t.priority}</td>

                        <td className="p-2 align-top">
                          <div className={`inline-flex items-center gap-2 px-2 py-1 text-xs
font-medium rounded ${map.badge}`}>
                            <span className="text-sm">{map.icon}</span>
```

```
                          <span>{map.label}</span>
                        </div>
                      </td>

                      <td className="p-2 align-top">
                        <div className="flex gap-2">
                          <button
                            onClick={() => navigate(`/manager/tasks/${t._id}/edit`)}
                            className={isDark ? "px-2 py-1 text-xs bg-slate-700 text-
gray-100 rounded hover:bg-slate-600" : "px-2 py-1 text-xs bg-gray-100 rounded hover:bg-
gray-200"}
                          >
                            View / Edit
                          </button>

                          <button
                            onClick={() => handleDelete(t._id)}
                            className="px-2 py-1 text-xs bg-red-100 text-red-700 rounded
hover:bg-red-200"
                          >
                            Delete
                          </button>
                        </div>
                      </td>
                    </tr>
                  );
                })}

                {tasks.length === 0 && (
                  <tr>
                    <td colSpan="7" className="p-4 text-center text-gray-500">No
tasks.</td>
                  </tr>
                )}
              </tbody>
            </table>
          </div>
        )}
      </div>

      <AssignTaskModal open={open} onClose={() => setOpen(false)}
onCreated={handleCreated} />
    </div>
  </div>
  );
}
```

```
// client/src/user-pages/manager/tasks/AssignTaskModal.jsx
import React, { useEffect, useState, useContext } from "react";
import api from "../../../utils/api";
import Swal from "sweetalert2";
import { AuthContext } from "../../../context/AuthContext";
import { ThemeContext } from "../../../context/ThemeContext";

/**
  Props:
    open, onClose, onCreated (callback)
*/
export default function AssignTaskModal({ open, onClose, onCreated }) {
  const { user } = useContext(AuthContext);
  const { isDark } = useContext(ThemeContext);

  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [dueDate, setDueDate] = useState("");
  const [priority, setPriority] = useState("Medium");
```

```
  const [tags, setTags] = useState("");
  const [search, setSearch] = useState("");
  const [results, setResults] = useState([]); // search results for employees
  const [selected, setSelected] = useState([]); // array of { employeeId, name }
  const [loadingResults, setLoadingResults] = useState(false);

  // reset when closed
  useEffect(() => {
    if (!open) {
      setTitle("");
      setDescription("");
      setDueDate("");
      setPriority("Medium");
      setTags("");
      setSearch("");
      setResults([]);
      setSelected([]);
    }
  }, [open]);

  // debounce search -> api call
  useEffect(() => {
    if (!search || !search.trim()) {
      setResults([]);
      setLoadingResults(false);
      return;
    }

    setLoadingResults(true);
    const t = setTimeout(async () => {
      try {
        const q = search.trim();
        const res = await api.get("/manager/employees", {
          params: { search: q, limit: 12 },
        });
        const list = (res.data?.data || []).map((u) => ({
          employeeId: u.employeeId,
          name: u.name,
        }));
        // filter out already selected
        const filtered = list.filter(
          (l) => !selected.some((s) => s.employeeId === l.employeeId)
        );
        setResults(filtered);
      } catch (err) {
        console.error("searchEmployees error", err);
        setResults([]);
      } finally {
        setLoadingResults(false);
      }
    }, 300);

    return () => clearTimeout(t);
  }, [search, selected]);

  const toggleSelect = (emp) => {
    const exists = selected.find((s) => s.employeeId === emp.employeeId);
    if (exists) {
      setSelected((sel) => sel.filter((s) => s.employeeId !== emp.employeeId));
      // put back to results so user can re-add quickly
      setResults((r) => [emp, ...r]);
    } else {
      setSelected((sel) => [...sel, emp]);
      // remove from results UI
      setResults((r) => r.filter((x) => x.employeeId !== emp.employeeId));
    }
```

```
  };

  const handleCreate = async () => {
    if (!title.trim() || selected.length === 0) {
      Swal.fire(
        "Validation",
        "Please provide title and at least one assignee",
        "warning"
      );
      return;
    }

    try {
      const payload = {
        title: title.trim(),
        description,
        assignedTo: selected.map((s) => ({
          employeeId: s.employeeId,
          name: s.name,
        })),
        dueDate: dueDate || null,
        priority,
        tags: tags
          ? tags.split(",").map((t) => t.trim()).filter(Boolean)
          : [],
      };

      const res = await api.post("/manager/tasks", payload);
      Swal.fire("Created", "Task created successfully", "success");
      onCreated && onCreated(res.data);
      onClose && onClose();
    } catch (err) {
      console.error("create task error", err);
      Swal.fire(
        "Error",
        err.response?.data?.msg || "Failed to create task",
        "error"
      );
    }
  };

  if (!open) return null;

  // theme classes
  const overlay = "fixed inset-0 z-50 flex items-center justify-center p-4";
  const backdrop = isDark ? "bg-black/70" : "bg-black/40";
  const panel = isDark
    ? "bg-slate-900 text-gray-200 border border-slate-700"
    : "bg-white text-gray-900 border border-gray-100";
  const inputBase = isDark
    ? "w-full border p-2 rounded bg-slate-800 border-slate-700 text-gray-100
placeholder:text-gray-400"
    : "w-full border p-2 rounded bg-white border-gray-300 text-gray-900 placeholder:text-
gray-500";
  const labelBase = isDark ? "text-sm block text-gray-200" : "text-sm block text-gray-700";
  const chipBase = isDark
    ? "px-3 py-1 rounded-full bg-slate-800 border border-slate-700 text-gray-200 flex
items-center gap-2"
    : "px-3 py-1 rounded-full bg-blue-50 border border-blue-100 text-blue-800 flex items-
center gap-2";
  const addBtn = isDark
    ? "px-2 py-1 text-xs rounded bg-green-700 text-white hover:bg-green-600"
    : "px-2 py-1 text-xs rounded bg-green-100 text-green-800 hover:bg-green-200";
  const removeBtn = isDark
    ? "ml-2 text-xs text-red-300 hover:text-red-100"
    : "ml-2 text-xs text-red-600 hover:text-red-800";
```

```
  const resultRow = isDark
    ? "flex items-center justify-between p-2 rounded hover:bg-slate-800/60 border border-
slate-700"
    : "flex items-center justify-between p-2 rounded hover:bg-gray-50 border border-gray-
100";

  return (
    <div className={`${overlay} ${backdrop}`}>
      <div className={`w-full max-w-2xl rounded-lg shadow-lg p-6 ${panel} overflow-auto
max-h-[90vh]`}>
        <div className="flex items-center justify-between mb-4">
          <h3 className="text-lg font-semibold">Assign Task</h3>
          <button onClick={onClose} className={isDark ? "text-gray-300 hover:text-gray-100"
: "text-gray-600 hover:text-gray-900"}>
            Close
          </button>
        </div>

        <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
          <div className="md:col-span-2">
            <label className={labelBase}>Title</label>
            <input
              className={inputBase}
              value={title}
              onChange={(e) => setTitle(e.target.value)}
              placeholder="Task title"
            />
          </div>

          <div className="md:col-span-2">
            <label className={labelBase}>Description</label>
            <textarea
              className={`${inputBase} min-h-[90px]`}
              value={description}
              onChange={(e) => setDescription(e.target.value)}
            />
          </div>

          <div>
            <label className={labelBase}>Due date</label>
            <input
              type="date"
              className={inputBase}
              value={dueDate}
              onChange={(e) => setDueDate(e.target.value)}
            />
          </div>

          <div>
            <label className={labelBase}>Priority</label>
            <select
              className={inputBase}
              value={priority}
              onChange={(e) => setPriority(e.target.value)}
            >
              <option>Low</option>
              <option>Medium</option>
              <option>High</option>
              <option>Critical</option>
            </select>
          </div>

          <div className="md:col-span-2">
            <label className={labelBase}>Tags (comma separated)</label>
            <input
              className={inputBase}
```

```
              value={tags}
              onChange={(e) => setTags(e.target.value)}
              placeholder="e.g. frontend, urgent"
            />
          </div>

          {/* Employee search & selection */}
          <div className="md:col-span-2">
            <label className={labelBase}>Search employee to assign</label>

            {/* Search input + hint */}
            <div className="mt-1 flex flex-col sm:flex-row sm:items-center gap-2">
              <input
                className={`${inputBase} flex-1`}
                value={search}
                onChange={(e) => setSearch(e.target.value)}
                placeholder="Search by name or ID"
                aria-label="Search employees"
              />
              <div className="flex gap-2">
                <button
                  type="button"
                  onClick={() => {
                    // quick clear
                    setSearch("");
                    setResults([]);
                  }}
                  className={isDark ? "px-3 py-2 cursor-pointer bg-slate-700 rounded text-
gray-200" : "px-3 py-1 bg-gray-100 rounded text-gray-800"}
                >
                  Clear
                </button>
              </div>
            </div>

            {/* Selected chips */}
            <div className="mt-3 flex gap-2 flex-wrap">
              {selected.length === 0 ? (
                <div className={isDark ? "text-gray-400 text-sm" : "text-gray-500 text-
sm"}>No assignees yet</div>
              ) : (
                selected.map((s) => (
                  <div key={s.employeeId} className={chipBase}>
                    <span className="text-sm truncate">{s.name} • {s.employeeId}</span>
                    <button onClick={() => toggleSelect(s)} className={removeBtn} aria-
label={`Remove ${s.name}`}>
                      ×
                    </button>
                  </div>
                ))
              )}
            </div>

            {/* Results dropdown */}
            <div className="mt-3">
              {loadingResults && (
                <div className={isDark ? "text-gray-300 text-sm" : "text-gray-600 text-
sm"}>Searching…</div>
              )}

              {!loadingResults && results.length === 0 && search.trim() !== "" && (
                <div className={isDark ? "text-gray-400 text-sm" : "text-gray-500 text-
sm"}>No matches</div>
              )}

              {!loadingResults && results.length > 0 && (
```

```
                <div className="mt-2 grid gap-2 max-h-48 overflow-auto">
                  {results.map((r) => (
                    <div key={r.employeeId} className={resultRow}>
                      <div>
                        <div className={isDark ? "font-medium text-gray-100" : "font-medium
text-gray-900"}>{r.name}</div>
                        <div className={isDark ? "text-xs text-gray-400" : "text-xs text-
gray-500"}>{r.employeeId}</div>
                      </div>
                      <div>
                        <button
                          onClick={() => toggleSelect(r)}
                          className={addBtn}
                          aria-label={`Add ${r.name}`}
                        >
                          Add
                        </button>
                      </div>
                    </div>
                  ))}
                </div>
              )}
            </div>
          </div>
        </div>

        {/* Action buttons */}
        <div className="mt-4 flex justify-end gap-3">
          <button onClick={onClose} className={isDark ? "px-3 py-1 bg-slate-700 rounded
text-gray-200" : "px-3 py-1 bg-gray-200 rounded text-gray-800"}>
            Cancel
          </button>
          <button onClick={handleCreate} className="px-3 py-1 bg-blue-600 text-white
rounded hover:bg-blue-700">
            Create Task
          </button>
        </div>
      </div>
    </div>
  );
}
```

```
// client/src/user-pages/manager/tasks/EditTaskPage.jsx
import React, { useEffect, useState, useContext } from "react";
import { useParams, useNavigate } from "react-router-dom";
import api from "../../../utils/api";
import Swal from "sweetalert2";
import { ThemeContext } from "../../../context/ThemeContext";

export default function EditTaskPage() {
  const { id } = useParams(); // this is task _id
  const navigate = useNavigate();
  const { isDark } = useContext(ThemeContext);

  const [task, setTask] = useState(null);
  const [form, setForm] = useState({
    title: "",
    description: "",
    dueDate: "",
    priority: "Medium",
    tags: "",
    assignedTo: [], // [{ employeeId, name }]
  });
  const [loading, setLoading] = useState(true);
  const [saving, setSaving] = useState(false);
```

```
  useEffect(() => {
    let mounted = true;
    const fetchTask = async () => {
      try {
        setLoading(true);
        const res = await api.get(`/manager/tasks/${id}`);
        if (!mounted) return;

        const data = res.data;
        setTask(data);
        setForm({
          title: data.title || "",
          description: data.description || "",
          dueDate: data.dueDate ? data.dueDate.slice(0, 10) : "",
          priority: data.priority || "Medium",
          tags: Array.isArray(data.tags)
            ? data.tags.join(", ")
            : data.tags || "",
          assignedTo: data.assignedTo || [],
        });
      } catch (err) {
        console.error("fetchTask", err);
        Swal.fire(
          "Error",
          err.response?.data?.msg || "Unable to fetch task",
          "error"
        );
      } finally {
        if (mounted) setLoading(false);
      }
    };
    fetchTask();
    return () => {
      mounted = false;
    };
  }, [id]);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setForm((p) => ({ ...p, [name]: value }));
  };

  const handleSave = async () => {
    if (!form.title.trim()) {
      await Swal.fire("Validation", "Title required", "warning");
      return;
    }

    setSaving(true);
    try {
      const payload = {
        title: form.title.trim(),
        description: form.description,
        dueDate: form.dueDate || null,
        priority: form.priority,
        tags: form.tags
          ? form.tags.split(",").map((t) => t.trim()).filter(Boolean)
          : [],
        assignedTo: form.assignedTo, // keep as-is (no UI editing yet)
      };

      await api.put(`/manager/tasks/${id}`, payload);

      await Swal.fire({
        title: "Saved",
        text: "Task updated",
```

```
        icon: "success",
        timer: 1200,
        showConfirmButton: false,
        allowOutsideClick: false,
      });

      navigate(-1);
    } catch (err) {
      console.error("update task", err);
      await Swal.fire(
        "Error",
        err.response?.data?.msg || "Failed to update",
        "error"
      );
    } finally {
      setSaving(false);
    }
  };

  // shared classes
  const pageBg = isDark
    ? "bg-gradient-to-b from-slate-900 via-slate-950 to-slate-900 text-gray-100"
    : "bg-gradient-to-b from-gray-50 to-gray-300 text-gray-900";

  const cardBg = isDark
    ? "bg-slate-900/80 border border-slate-700"
    : "bg-white shadow border border-gray-100";

  const inputBase = isDark
    ? "w-full border p-2 rounded bg-slate-800 border-slate-700 text-gray-100
placeholder:text-gray-400"
    : "w-full border p-2 rounded bg-white border-gray-300 text-gray-900 placeholder:text-
gray-400";

  const labelClass = isDark ? "text-sm block text-gray-200" : "text-sm block text-gray-
700";

  const secondaryBtn = isDark
    ? "px-3 py-1 bg-slate-700 text-gray-100 rounded hover:bg-slate-600"
    : "px-3 py-1 bg-gray-200 text-gray-800 rounded hover:bg-gray-300";

  const primaryBtn = (disabled) =>
    disabled
      ? "px-3 py-1 bg-gray-400 text-gray-800 rounded cursor-not-allowed"
      : "px-3 py-1 bg-blue-600 text-white rounded hover:bg-blue-700";

  if (loading) {
    return (
      <div className={`p-6 min-h-screen flex items-center justify-center ${pageBg}`}>
        <div className={isDark ? "text-gray-300" : "text-gray-600"}>
          Loading...
        </div>
      </div>
    );
  }

  return (
    <div
      className={`min-h-screen min-w-screen flex items-center justify-center p-4 md:p-6
${pageBg}`}
    >
      <div className="w-full max-w-4xl flex flex-col">
        {/* Header */}
        <div className="flex flex-col md:flex-row md:items-center justify-between gap-3 mb-
4">
          <h1 className="text-2xl font-semibold">Edit Task</h1>
```

```
      <div className="flex gap-2">
        <button
          onClick={() => navigate(-1)}
          className={secondaryBtn}
          type="button"
        >
          Back
        </button>
        <button
          onClick={handleSave}
          disabled={saving}
          className={primaryBtn(saving)}
          type="button"
        >
          {saving ? "Saving..." : "Save"}
        </button>
      </div>
    </div>

    {/* Card */}
    <div className={`${cardBg} rounded-lg p-6 grid gap-4`}>
      <div>
        <label className={labelClass}>Title</label>
        <input
          name="title"
          value={form.title}
          onChange={handleChange}
          className={inputBase}
          placeholder="Task title"
        />
      </div>

      <div>
        <label className={labelClass}>Description</label>
        <textarea
          name="description"
          value={form.description}
          onChange={handleChange}
          className={`${inputBase} min-h-[100px]`}
          placeholder="Describe the task..."
        />
      </div>

      <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
        <div>
          <label className={labelClass}>Due Date</label>
          <input
            type="date"
            name="dueDate"
            value={form.dueDate}
            onChange={handleChange}
            className={inputBase}
          />
        </div>

        <div>
          <label className={labelClass}>Priority</label>
          <select
            name="priority"
            value={form.priority}
            onChange={handleChange}
            className={inputBase}
          >
            <option>Low</option>
            <option>Medium</option>
            <option>High</option>
```

```
                <option>Critical</option>
              </select>
            </div>
          </div>

          <div>
            <label className={labelClass}>Tags (comma separated)</label>
            <input
              name="tags"
              value={form.tags}
              onChange={handleChange}
              className={inputBase}
              placeholder="e.g. frontend, urgent"
            />
          </div>

          {/* Assignees (read-only for now) */}
          <div>
            <label className={labelClass}>Assignees</label>
            <div
              className={
                isDark
                  ? "border border-slate-700 p-2 rounded bg-slate-900/60"
                  : "border border-gray-300 p-2 rounded bg-gray-50"
              }
            >
              {form.assignedTo.length === 0 ? (
                <div className={isDark ? "text-gray-400 text-sm" : "text-gray-500 text-
sm"}>
                  No assignees
                </div>
              ) : (
                form.assignedTo.map((a) => (
                  <div key={a.employeeId} className="text-sm">
                    {a.name} • {a.employeeId}
                  </div>
                ))
              )}
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}
```

```
// client/src/user-pages/manager/ManagerDashboard.jsx
import React from "react";
import { Routes, Route } from "react-router-dom";
import EmpDataTab from "./employee_profiles/EmpDataTab";
import EmployeeDetailPage from "./employee_profiles/EmployeeDetailPage";

export default function ManagerDashboard() {
  return (
    <div className="p-6">
      <div className="flex items-center justify-between mb-6">
        <h1 className="text-2xl font-bold">Manager Dashboard</h1>
      </div>
      <Routes>
        <Route path="/" element={<div>Select an option or click Employee Details</div>} />
        <Route path="employee-profiles" element={<EmpDataTab />} />
        <Route path="employee-profiles/:id" element={<EmployeeDetailPage />} />
      </Routes>
    </div>
  );
}
```

```jsx
// client/src/user-pages/manager/ManagerDashboardLayout.jsx
import React, { useContext } from "react";
import DashboardNav from "../../components/dashboard-page-components/DashboardNav";
import {DashboardSidebarManager} from "../../components/dashboard-page-
components/DashboardSidebar";
import { AuthContext } from "../../context/AuthContext";

function ManagerDashboardLayout({ content }) {
  const { user } = useContext(AuthContext);
  return (
    <div className="w-screen h-screen flex flex-col bg-blue-500">
      <DashboardNav
        role={user?.role || "Manager"}
        photo={user?.photo || ""}
        name={user?.name || "User"}
        designation={user?.designation || "Manager"}
        country={user?.country || "Unknown"}
      />
      <div className="flex-grow flex justify-center items-center">
        <div className="w-1/6 h-full bg-amber-300">
          <DashboardSidebarManager user={user.role} />
        </div>
        <div className="w-5/6 h-[664px] bg-gray-100">{content}</div>
      </div>
    </div>
  );
}

export default ManagerDashboardLayout;
```

```jsx
// client/src/user-pages/manager/ManagerLogout.jsx
import React from 'react'
import ManagerDashboardLayout from "./ManagerDashboardLayout";
import LogoutTab from '../../components/logout/LogoutTab'

function ManagerLogout() {
  return (
    <>
      <ManagerDashboardLayout
      content={<LogoutTab />}
      />
    </>
  )
}

export default ManagerLogout
```

```jsx
// client/src/utils/api.js
import axios from "axios";

const api = axios.create({
  baseURL: "http://localhost:5000/api",
  headers: { "Content-Type": "application/json" },
});

api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("token");
    if (token) config.headers.Authorization = `Bearer ${token}`;
    return config;
  },
  (error) => Promise.reject(error)
);

api.interceptors.response.use(
```

```
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem("token");
      localStorage.removeItem("user");
      if (window.location.pathname !== "/login") window.location.href = "/login";
    }
    return Promise.reject(error);
  }
);

export default api;
```

```
// client/src/App.jsx
import "./App.css";
import { BrowserRouter } from "react-router-dom";
import AppRoutes from "./routes/AppRoutes";
import { useEffect } from "react";
import { AuthProvider } from "./context/AuthContext";

function App() {
  // Auto logout on token expiry
  useEffect(() => {
    const token = localStorage.getItem("token");
    if (token) {
      try {
        const payload = JSON.parse(atob(token.split(".")[1]));
        const isExpired = payload.exp * 1000 < Date.now();

        if (isExpired) {
          localStorage.removeItem("token");
          localStorage.removeItem("user");
          window.location.href = "/login";
        }
      } catch (error) {
        console.error("Invalid token:", error);
        localStorage.removeItem("token");
        localStorage.removeItem("user");
        window.location.href = "/login";
      }
    }
  }, []);

  return (
    <AuthProvider>
      <BrowserRouter>
        <AppRoutes />
      </BrowserRouter>
    </AuthProvider>
  );
}

export default App;
```

```
/* index.css */
@import "tailwindcss";

:root {
  font-family: system-ui, Avenir, Helvetica, Arial, sans-serif;
  line-height: 1.5;
  font-weight: 400;

  color-scheme: light dark;
  color: rgba(255, 255, 255, 0.87);
  background-color: #242424;
```

```
  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

body {
  margin: 0;
  display: flex;
  place-items: center;
  font-family: system-ui;
}

@layer components {
  .hide-date-icon::-webkit-calendar-picker-indicator {
    background-color: rgb(75, 75, 75);
    border-radius: 3px;
  }
}

/* ==== Custom Scrollbar Style ==== */
::-webkit-scrollbar {
  width: 8px;                /* Width of vertical scrollbar */
  height: 8px;               /* Height of horizontal scrollbar */
}

::-webkit-scrollbar-track {
  background: #f1f1f1;       /* Track color */
  border-radius: 8px;
}

::-webkit-scrollbar-thumb {
  background: linear-gradient(180deg, #6366f1, #3b82f6); /* Indigo → Blue gradient */
  border-radius: 8px;
}

::-webkit-scrollbar-thumb:hover {
  background: linear-gradient(180deg, #4f46e5, #2563eb); /* Darker on hover */
}

/* Optional: Smooth scrolling everywhere */
html {
  scroll-behavior: smooth;
}

/* src/index.css (or src/theme.css) ---------------------------------- */

/* 1) Theme variables - light is default */
:root {
  --bg: #ffffff;
  --card-bg: #f8fafc;
  --muted: #6b7280;
  --text: #111827;
  --primary: #4f46e5; /* indigo */
  --border: #e5e7eb;
}

/* 2) Dark theme variables (applied when data-theme="dark" on html) */
:root[data-theme="dark"] {
  --bg: #0b1220;       /* deep background */
  --card-bg: #0f1724;  /* slightly lighter card */
  --muted: #9ca3af;
  --text: #e6edf3;
  --primary: #7c3aed;  /* brighter indigo */
  --border: #1f2937;
}
```

```css
/* 3) Helper utility classes you can use in React's className */
.bg-app { background-color: var(--bg); }
.text-app { color: var(--text); }
.card { background-color: var(--card-bg); border: 1px solid var(--border); color: var(--
text); }

/* small helpers */
.text-muted { color: var(--muted); }
.btn-primary {
  background-color: var(--primary);
  color: white;
  padding: 0.5rem 0.75rem;
  border-radius: 0.375rem;
}

/* 4) Smooth transition only when toggling theme.
   We'll add a short-lived class on <html> so transitions don't affect initial paint. */
.theme-transition, .theme-transition * {
  transition: background-color 360ms ease, color 360ms ease, border-color 360ms ease, box-
shadow 360ms ease;
}

/* 5) Optional small fade animation for components */
.theme-fade { animation: fadeIn 360ms ease; }
@keyframes fadeIn { from { opacity: 0.92 } to { opacity: 1 } }

/* 6) Make sure basic page uses variables */
html, body, #root {
  height: 100%;
}
body {
  margin: 0;
  background: var(--bg);
  color: var(--text);
  transition: none; /* disable default; controlled via .theme-transition */
  font-family: Inter, system-ui, -apple-system, "Segoe UI", Roboto, "Helvetica Neue",
Arial;
}

/* ==== DARK MODE SCROLLBAR (only when data-theme="dark") ==== */
:root[data-theme="dark"] ::-webkit-scrollbar-track {
  background: #0f1724;       /* matches --card-bg */
}

:root[data-theme="dark"] ::-webkit-scrollbar-thumb {
  background: linear-gradient(180deg, #4f46e5, #312e81); /* deeper indigo gradient */
}

:root[data-theme="dark"] ::-webkit-scrollbar-thumb:hover {
  background: linear-gradient(180deg, #4338ca, #1e1b4b); /* darker hover */
}
```

```jsx
// client/src/main.jsx
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import "./index.css";
import App from "./App.jsx";
import { AuthProvider } from "./context/AuthContext";
import { ThemeProvider } from "./context/ThemeContext";
import { AttendanceProvider } from "./context/AttendanceContext.jsx";
import { LeaveProvider } from "./context/LeaveContext";

createRoot(document.getElementById("root")).render(
  <StrictMode>
    <AuthProvider>
```

```
      <AttendanceProvider>
        <LeaveProvider>
          <ThemeProvider>
            <App />
          </ThemeProvider>
        </LeaveProvider>
      </AttendanceProvider>
    </AuthProvider>
  </StrictMode>
);
```

```
<!-- client/index.html -->
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/png" href="/website-logo.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>HRly: Employee Management Portal | Easy and Secure Access</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

// package.json

```
{
  "name": "client",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "@tailwindcss/vite": "^4.1.16",
    "axios": "^1.13.1",
    "file-saver": "^2.0.5",
    "framer-motion": "^12.23.24",
    "react": "^19.1.1",
    "react-dom": "^19.1.1",
    "react-icons": "^5.5.0",
    "react-router-dom": "^7.9.4",
    "recharts": "^3.3.0",
    "sweetalert2": "^11.26.3",
    "tailwindcss": "^4.1.16",
    "usehooks-ts": "^3.1.1"
  },
  "devDependencies": {
    "@eslint/js": "^9.36.0",
    "@types/react": "^19.1.16",
    "@types/react-dom": "^19.1.9",
    "@vitejs/plugin-react": "^5.0.4",
    "eslint": "^9.36.0",
    "eslint-plugin-react-hooks": "^5.2.0",
    "eslint-plugin-react-refresh": "^0.4.22",
    "globals": "^16.4.0",
    "vite": "^7.1.7"
  }
}
```

Backend with NodeJS, MongoDB

```
// server/config/db.js
const mongoose = require("mongoose");

const connectDB = async () => {
  try {
    const uri = process.env.MONGO_URI;
    await mongoose.connect(uri, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log("✅ MongoDB connected to", mongoose.connection.name);
  } catch (err) {
    console.error("❌ MongoDB connection error:", err.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

```
// server/config/multer.js
const multer = require("multer");
const path = require("path");
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "uploads/"); // create uploads/ at server root
  },
  filename: function (req, file, cb) {
    const ext = path.extname(file.originalname);
    cb(null, file.fieldname + "-" + Date.now() + ext);
  }
});

const upload = multer({ storage });

module.exports = upload;
```

```
// server/controllers/manager/notificationController.js
const Notification = require("../../models/notification");
const path = require("path");

exports.createNotification = async (req, res) => {
  try {
    const { title, message, type, visibleTo } = req.body;
    if (!title || !message) return res.status(400).json({ msg: "Title and message are
required." });

    const payload = {
      title, message, type: type || "Announcement", visibleTo: visibleTo || "All",
      createdBy: req.user.userId,
      createdByName: req.user.name || req.user.role || "Manager"
    };

    if (req.file) payload.image = req.file.filename;

    const n = new Notification(payload);
    await n.save();
    res.status(201).json(n);
  } catch (err) {
    console.error("createNotification error:", err);
    res.status(500).json({ msg: "Server error", error: err.message });
  }
};

exports.listNotifications = async (req, res) => {
```

```
  try {
    // optional query params: ?limit=20&page=1&type=Announcement&search=diwali
    const page = parseInt(req.query.page, 10) || 1;
    const limit = parseInt(req.query.limit, 10) || 50;
    const skip = (page - 1) * limit;

    const filters = {};
    if (req.query.type) filters.type = req.query.type;
    if (req.query.search) {
      const q = req.query.search.trim();
      filters.$or = [{ title: { $regex: q, $options: "i" } }, { message: { $regex: q,
$options: "i" } }];
    }

    const [total, data] = await Promise.all([
      Notification.countDocuments(filters),
      Notification.find(filters).sort({ createdAt: -1 }).skip(skip).limit(limit)
    ]);

    res.json({ meta: { total, page, limit, pages: Math.ceil(total/limit) || 1 }, data });
  } catch (err) {
    console.error("listNotifications error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};

exports.getNotification = async (req, res) => {
  try {
    const doc = await Notification.findById(req.params.id);
    if (!doc) return res.status(404).json({ msg: "Notification not found" });
    res.json(doc);
  } catch (err) {
    console.error("getNotification error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};

exports.deleteNotification = async (req, res) => {
  try {
    const doc = await Notification.findByIdAndDelete(req.params.id);
    if (!doc) return res.status(404).json({ msg: "Not found" });
    res.json({ msg: "Deleted", id: req.params.id });
  } catch (err) {
    console.error("deleteNotification error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};
```

```
// server/controllers/manager/taskController.js
const Task = require("../../models/task");
const User = require("../../models/users"); // for optional lookups

// POST /api/manager/tasks
exports.createTask = async (req, res) => {
  try {
    const { title, description, assignedTo = [], dueDate, priority, tags = [] } = req.body;
    if (!title || !Array.isArray(assignedTo) || assignedTo.length === 0) {
      return res.status(400).json({ msg: "Title and at least one assignee are required."
});
    }

    const createdBy = {
      userId: req.user?.userId,
      name: req.user?.name || undefined,
      role: req.user?.role || undefined,
    };
```

```
    const task = new Task({
      title,
      description,
      assignedTo,
      dueDate: dueDate || null,
      priority: priority || "Medium",
      tags: Array.isArray(tags) ? tags : [],
      createdBy,
    });

    await task.save();
    res.status(201).json(task);
  } catch (err) {
    console.error("createTask error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};

// GET /api/manager/tasks  (paginated + search)
exports.listTasks = async (req, res) => {
  try {
    const page = parseInt(req.query.page, 10) || 1;
    const limit = parseInt(req.query.limit, 10) || 50;
    const skip = (page - 1) * limit;

    const filters = {};
    if (req.query.search) {
      const q = req.query.search.trim();
      filters.$or = [
        { title: { $regex: q, $options: "i" } },
        { description: { $regex: q, $options: "i" } },
        { "assignedTo.employeeId": { $regex: q, $options: "i" } },
        { "assignedTo.name": { $regex: q, $options: "i" } },
      ];
    }
    if (req.query.status) filters.status = req.query.status;
    if (req.query.priority) filters.priority = req.query.priority;

    const [total, data] = await Promise.all([
      Task.countDocuments(filters),
      Task.find(filters).sort({ createdAt: -1 }).skip(skip).limit(limit),
    ]);

    res.json({
      meta: { total, page, limit, pages: Math.max(1, Math.ceil(total / limit)) },
      data,
    });
  } catch (err) {
    console.error("listTasks error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};

// GET /api/manager/tasks/:id
exports.getTask = async (req, res) => {
  try {
    const task = await Task.findById(req.params.id);
    if (!task) return res.status(404).json({ msg: "Task not found" });
    res.json(task);
  } catch (err) {
    console.error("getTask error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};
```

```
// PUT /api/manager/tasks/:id  (manager edits)
exports.updateTask = async (req, res) => {
  try {
    const updates = (({ title, description, assignedTo, dueDate, priority, tags, status })
=> ({ title, description, assignedTo, dueDate, priority, tags, status }))(req.body);
    // Remove undefined keys
    Object.keys(updates).forEach((k) => updates[k] === undefined && delete updates[k]);

    const task = await Task.findByIdAndUpdate(req.params.id, updates, { new: true });
    if (!task) return res.status(404).json({ msg: "Task not found" });
    res.json(task);
  } catch (err) {
    console.error("updateTask error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};

// PATCH /api/manager/tasks/:id/status  (both manager and employees can update status;
employees limited to allowed statuses)
exports.patchStatus = async (req, res) => {
  try {
    const { status } = req.body;
    if (!status) return res.status(400).json({ msg: "Status required" });

    // allowed statuses (same enum as model)
    const allowed = ["Pending", "In Progress", "Completed", "Cancelled"];
    if (!allowed.includes(status)) return res.status(400).json({ msg: "Invalid status" });

    // Optionally: if user.role is 'Employee', ensure they are an assignee of this task
    const task = await Task.findById(req.params.id);
    if (!task) return res.status(404).json({ msg: "Task not found" });

    const actorRole = (req.user && req.user.role) || "";
    if (String(actorRole).toLowerCase() === "employee") {
      // check if employee is assigned
      // determine employeeId: either JWT contains employeeId or fetch from users
collection
      let employeeId = req.user.employeeId;
      if (!employeeId) {
        const u = await User.findById(req.user.userId).select("employeeId");
        employeeId = u?.employeeId;
      }
      if (!employeeId) return res.status(403).json({ msg: "Cannot determine employee
identity" });

      const isAssigned = task.assignedTo.some((a) => String(a.employeeId) ===
String(employeeId));
      if (!isAssigned) return res.status(403).json({ msg: "You are not assigned to this
task" });
    }

    task.status = status;
    await task.save();
    res.json(task);
  } catch (err) {
    console.error("patchStatus error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};

// DELETE /api/manager/tasks/:id
exports.deleteTask = async (req, res) => {
  try {
    const task = await Task.findByIdAndDelete(req.params.id);
    if (!task) return res.status(404).json({ msg: "Task not found" });
    res.json({ msg: "Task deleted" });
```

```
  } catch (err) {
    console.error("deleteTask error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};

// GET /api/manager/tasks/employee/me  -> tasks assigned to current employee
exports.listMyTasks = async (req, res) => {
  try {
    // determine employeeId: prefer token's employeeId if you included it, else read from
users collection
    let employeeId = req.user.employeeId;
    if (!employeeId) {
      const u = await User.findById(req.user.userId).select("employeeId");
      employeeId = u?.employeeId;
    }
    if (!employeeId) return res.status(400).json({ msg: "Employee identifier missing" });

    const tasks = await Task.find({ "assignedTo.employeeId": employeeId }).sort({ dueDate:
1, createdAt: -1 });
    res.json({ meta: { total: tasks.length }, data: tasks });
  } catch (err) {
    console.error("listMyTasks error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};
```

```
// server/controllers/manager/userController.js
const User = require("../../models/users");

/**
 * GET /api/manager/employees
 * Returns list of users who have role: "Employee".
 * Supports optional query params: ?page=1&limit=25&search=xyz&department=Dev
 */
exports.listEmployees = async (req, res) => {
  try {
    // pagination
    const page = parseInt(req.query.page, 10) || 1;
    const limit = parseInt(req.query.limit, 10) || 50;
    const skip = (page - 1) * limit;

    // filters
    const filters = { role: "Employee" };

    if (req.query.department) filters.department = req.query.department;
    if (req.query.designation) filters.designation = req.query.designation;

    // search on name / email / employeeId
    if (req.query.search) {
      const q = req.query.search.trim();
      filters.$or = [
        { name: { $regex: q, $options: "i" } },
        { email: { $regex: q, $options: "i" } },
        { employeeId: { $regex: q, $options: "i" } },
      ];
    }

    // fetch total count & paginated docs
    const [total, employees] = await Promise.all([
      User.countDocuments(filters),
      User.find(filters)
        .select("-password -securityAnswer -__v") // hide sensitive fields
        .sort({ createdAt: -1 })
        .skip(skip)
        .limit(limit),
```

```
    ]);

    res.json({
      meta: { total, page, limit, pages: Math.ceil(total / limit) || 1 },
      data: employees,
    });
  } catch (err) {
    console.error("manager.listEmployees error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};

/**
 * GET /api/manager/employees/:id
 * Get single employee by id
 */
exports.getEmployeeById = async (req, res) => {
  try {
    const id = req.params.id;
    const employee = await User.findById(id).select("-password -securityAnswer -__v");
    if (!employee) return res.status(404).json({ msg: "Employee not found" });
    if ((employee.role || "").toLowerCase() !== "employee")
      return res.status(400).json({ msg: "User is not an employee" });

    res.json(employee);
  } catch (err) {
    console.error("manager.getEmployeeById error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};



/**
 * PUT /api/manager/employees/:id
 * Update an employee (manager-only endpoint).
 * Whitelisted fields only — prevents privilege escalation.
 */
exports.updateEmployee = async (req, res) => {
  try {
    const id = req.params.id;
    const payload = req.body || {};

    // allowed fields manager may change
    const allowed = [
      "name", "gender", "dob", "email", "phone", "address", "city", "state", "pincode",
"country",
      "emergencyContactName", "emergencyPhone",
      "department", "designation", "employeeId", "joiningDate", "employmentType",
      "reportingManager", "workEmail", "workMode", "role"
    ];

    const update = {};
    allowed.forEach((k) => {
      if (Object.prototype.hasOwnProperty.call(payload, k)) update[k] = payload[k];
    });

    // Basic validations
    if (update.email) {
      const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
      if (!emailRegex.test(update.email)) return res.status(400).json({ msg: "Invalid email
format" });
    }
    if (update.phone && !/^\d{10}$/.test(String(update.phone))) return
res.status(400).json({ msg: "Phone must be 10 digits" });
```

```
    if (update.emergencyPhone && update.emergencyPhone !== "" &&
!/^\d{10}$/.test(String(update.emergencyPhone))) return res.status(400).json({ msg:
"Emergency phone must be 10 digits" });

    // Parse dates
    if (update.dob) update.dob = new Date(update.dob);
    if (update.joiningDate) update.joiningDate = new Date(update.joiningDate);

    // Ensure role stays valid if provided
    if (update.role && !["Admin", "HR", "Manager", "Employee"].includes(update.role)) {
      return res.status(400).json({ msg: "Invalid role value" });
    }

    // Update and return updated doc
    const updated = await User.findByIdAndUpdate(id, { $set: update }, { new: true
}).select("-password -securityAnswer -__v");
    if (!updated) return res.status(404).json({ msg: "Employee not found" });

    return res.json(updated);
  } catch (err) {
    console.error("manager.updateEmployee error:", err);
    return res.status(500).json({ msg: "Server error" });
  }
};
```

```
// server/controllers/attendanceController.js
const Attendance = require("../models/attendance");

exports.markAttendance = async (req, res) => {
  try {
    const { employeeId, name, role } = req.body;
    const today = new Date().toISOString().split("T")[0];
    const now = new Date();
    const hour = now.getHours();
    const minute = now.getMinutes();

    // Prevent double marking
    const existing = await Attendance.findOne({ employeeId, date: today });
    if (existing) {
      return res.status(400).json({ msg: "Attendance already marked for today." });
    }

    let remarks = "On-Time";
    let status = "Present";

    // If after 11:00 AM → late
    if (hour > 11 || (hour === 11 && minute > 0)) {
      const lateMinutes = (hour - 10) * 60 + minute;
      remarks = `Late by ${lateMinutes} minutes`;
    }

    const newAttendance = new Attendance({
      employeeId,
      name,
      role,
      status,
      date: today,
      remarks,
    });

    await newAttendance.save();
    res.status(201).json({ msg: "Attendance marked successfully", newAttendance });
  } catch (err) {
    console.error(err);
    res.status(500).json({ msg: "Server error" });
  }
```

```
};

exports.getAttendance = async (req, res) => {
  try {
    const attendance = await Attendance.find();
    if (!attendance || attendance.length === 0) {
      return res.status(404).json({ msg: "No attendance records found" });
    }
    res.status(200).json(attendance);
  } catch (err) {
    console.error("Error fetching attendance:", err);
    res.status(500).json({ msg: "Server error" });
  }
};
```

```
// server/controllers/leaveController.js
const Leave = require("../models/leave");
const mongoose = require("mongoose");

// helper to ensure valid ObjectId
function isValidId(id) {
  return mongoose.Types.ObjectId.isValid(id);
}

// □ Apply for Leave
exports.applyLeave = async (req, res) => {
  try {
    const { employeeId, name, role, leaveType, fromDate, toDate, reason } = req.body;

    if (!employeeId || !name || !role || !leaveType || !fromDate || !toDate || !reason) {
      return res.status(400).json({ msg: "All fields are required." });
    }

    const overlappingLeave = await Leave.findOne({
      employeeId,
      fromDate: { $lte: toDate },
      toDate: { $gte: fromDate },
    });

    if (overlappingLeave) {
      return res.status(400).json({ msg: "You already have a leave request in this period."
});
    }

    const leave = new Leave({
      employeeId,
      name,
      role,
      leaveType,
      fromDate,
      toDate,
      reason,
    });

    await leave.save();
    res.status(201).json({ msg: "Leave applied successfully.", leave });
  } catch (error) {
    res.status(500).json({ msg: "Error applying leave", error: error.message });
  }
};

// □ Fetch leaves (works for both Admin & Employee)
exports.getLeaves = async (req, res) => {
  try {
    const leaves = await Leave.find();
    if (!leaves || leaves.length === 0) {
```

```
      return res.status(404).json({ msg: "No Leaves record found" });
    }
    res.status(200).json(leaves);
  } catch (error) {
    res.status(500).json({ msg: "Error fetching leaves", error: error.message });
  }
};

// Verify leave: set status -> "Verified"
exports.verifyLeave = async (req, res) => {
  try {
    const { id } = req.params;
    if (!isValidId(id)) return res.status(400).json({ msg: "Invalid leave id" });

    const current = await Leave.findById(id);
    if (!current) return res.status(404).json({ msg: "Leave not found" });

    // only Pending -> Verified
    if (current.status !== "Pending") {
      return res.status(400).json({ msg: `Cannot verify leave with status
'${current.status}'` });
    }

    // expired check (optional)
    const today = new Date();
    const to = new Date(current.toDate);
    if (to < today) {
      return res.status(400).json({ msg: "Cannot verify an expired leave" });
    }

    const updated = await Leave.findByIdAndUpdate(
      id,
      { status: "Verified" },
      { new: true, runValidators: true }
    );

    res.status(200).json({ msg: "Leave verified", leave: updated });
  } catch (err) {
    console.error("verifyLeave error:", err);
    res.status(500).json({ msg: "Error verifying leave", error: err.message });
  }
};

exports.approveLeave = async (req, res) => {
  try {
    const { id } = req.params;
    if (!isValidId(id)) return res.status(400).json({ msg: "Invalid leave id" });

    const current = await Leave.findById(id);
    if (!current) return res.status(404).json({ msg: "Leave not found" });

    if (current.status !== "Verified") {
      return res.status(400).json({ msg: "Only verified leaves can be approved" });
    }

    const updated = await Leave.findByIdAndUpdate(
      id,
      { status: "Approved" },
      { new: true, runValidators: true }
    );

    res.status(200).json({ msg: "Leave approved", leave: updated });
  } catch (err) {
    console.error("approveLeave error:", err);
    res.status(500).json({ msg: "Error approving leave", error: err.message });
  }
```

```
};

exports.rejectLeave = async (req, res) => {
  try {
    const { id } = req.params;
    if (!isValidId(id)) return res.status(400).json({ msg: "Invalid leave id" });

    const current = await Leave.findById(id);
    if (!current) return res.status(404).json({ msg: "Leave not found" });

    if (current.status !== "Verified") {
      return res.status(400).json({ msg: "Only verified leaves can be rejected" });
    }

    const updated = await Leave.findByIdAndUpdate(
      id,
      { status: "Rejected" },
      { new: true, runValidators: true }
    );

    res.status(200).json({ msg: "Leave rejected", leave: updated });
  } catch (err) {
    console.error("rejectLeave error:", err);
    res.status(500).json({ msg: "Error rejecting leave", error: err.message });
  }
};
```

```
// server/controllers/register.js
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const User = require("../models/users");

exports.register = async (req, res) => {
  try {
    const {
      name, gender, dob, email, phone, address, city, state, pincode, country,
      emergencyContactName, emergencyPhone,
      role, employeeId, department, designation, joiningDate, employmentType,
      reportingManager, workEmail, workMode,
      password,
      securityQuestion, securityAnswer, termsAccepted,
    } = req.body;

    if (!name || !email || !password) {
      return res.status(400).json({ success: false, msg: "Please fill all required fields"
});
    }

    let user = await User.findOne({ email });
    if (user) return res.status(400).json({ success: false, msg: "User already exists" });

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    let photo = null, signature = null;
    if (req.files) {
      if (req.files.photo && req.files.photo.length > 0) {
        photo = req.files.photo[0].filename;
      }
      if (req.files.signature && req.files.signature.length > 0) {
        signature = req.files.signature[0].filename;
      }
    }

    user = new User({
      name, gender, dob, email, phone, address, city, state, pincode, country,
```

```
      emergencyContactName, emergencyPhone,
      role, employeeId, department, designation, joiningDate, employmentType,
      reportingManager, workEmail, workMode,
      photo, signature,
      password: hashedPassword,
      securityQuestion, securityAnswer,
      termsAccepted,
    });

    await user.save();

    const payload = { userId: user._id, role: user.role };
    const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: "1d" });

    // ✅ Always send JSON back
    res.status(201).json({
      success: true,
      msg: "User registered successfully",
      token,
      user: {
        id: user._id,
        name: user.name,
        email: user.email,
        role: user.role,
        department: user.department,
      },
    });
  } catch (err) {
    console.error("Registration error:", err);
    res.status(500).json({
      success: false,
      msg: "Server error during registration",
      error: err.message,
    });
  }
};

exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;
    if (!email || !password)
      return res.status(400).json({ success: false, msg: "Missing credentials" });

    const user = await User.findOne({ email });
    if (!user)
      return res.status(400).json({ success: false, msg: "Invalid credentials" });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch)
      return res.status(400).json({ success: false, msg: "Invalid credentials" });

    const payload = { userId: user._id, role: user.role };
    const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: "1d" });

    res.json({
      success: true,
      msg: "Login successful",
      token,
      user: {
        id: user._id,
        name: user.name,
        email: user.email,
        role: user.role,
        gender: user.gender,
        photo: user.photo,
        designation: user.designation,
```

```
        country: user.country,
        department: user.department,
        phone: user.phone,
        address: user.address,
        city: user.city,
        state: user.state,
        pincode: user.pincode,
        employeeId: user.employeeId,
        joiningDate: user.joiningDate,
        employmentType: user.employmentType,
      },
    });
  } catch (err) {
    console.error("Login error:", err);
    res.status(500).json({
      success: false,
      msg: "Server error during login",
      error: err.message,
    });
  }
};
```

```
// server/controllers/settingsController.js
const User = require("../models/users");
const bcrypt = require("bcryptjs");

// ✅ Update user settings (profile + password + security)
exports.updateProfile = async (req, res) => {
  try {
    const {
      phone,
      address,
      securityQuestion,
      securityAnswer,
      currentPassword,
      newPassword,
      confirmPassword,
    } = req.body;

    const userId = req.user.userId;
    const user = await User.findById(userId);
    if (!user) return res.status(404).json({ msg: "User not found" });

    // ✅ Update editable fields
    if (phone) user.phone = phone;
    if (address) user.address = address;
    if (securityQuestion) user.securityQuestion = securityQuestion;
    if (securityAnswer) user.securityAnswer = securityAnswer;

    // ✅ Handle password change only if user entered new password
    if (newPassword && confirmPassword && currentPassword) {
      const isMatch = await bcrypt.compare(currentPassword, user.password);
      if (!isMatch)
        return res.status(400).json({ msg: "Current password is incorrect" });

      const hashedPassword = await bcrypt.hash(newPassword, 10);
      user.password = hashedPassword;
    }

    await user.save();

    res.status(200).json({
      msg: "Settings updated successfully",
      user: {
        name: user.name,
        email: user.email,
```

```
        phone: user.phone,
        address: user.address,
        securityQuestion: user.securityQuestion,
        securityAnswer: user.securityAnswer,
      },
    });
  } catch (error) {
    console.error("Error updating settings:", error);
    res.status(500).json({ msg: "Error updating settings", error: error.message });
  }
};
```

```
// server/controllers/taskController.js
const Task = require("../models/task");
const User = require("../models/users");

/**
 * GET /api/employee/tasks
 * - returns tasks assigned to the requesting user (by userId or employeeId)
 * - supports ?page & ?limit & ?search optional
 */
exports.getMyTasks = async (req, res) => {
  try {
    const userId = req.user?.userId; // from token
    if (!userId) return res.status(401).json({ msg: "Unauthorized" });

    // fetch user record to read employeeId if needed
    const user = await User.findById(userId).select("employeeId role");
    const employeeId = user?.employeeId;

    const { page = 1, limit = 50, search } = req.query;
    const skip = (page - 1) * limit;

    const filters = {
      $or: [],
    };

    // tasks where assignedTo contains this user _id
    filters.$or.push({ assignedTo: userId });

    // tasks where assignedEmployeeIds contains this user's employeeId (if present)
    if (employeeId) filters.$or.push({ assignedEmployeeIds: employeeId });

    // If manager accidentally assigned by email, also match by email (optional)
    // e.g. filters.$or.push({ "assignedEmails": user.email });

    // If search param present, add text match on title/description
    if (search && typeof search === "string") {
      const q = search.trim();
      filters.$and = [
        {
          $or: [
            { title: { $regex: q, $options: "i" } },
            { description: { $regex: q, $options: "i" } },
            { assignedEmployeeIds: { $regex: q, $options: "i" } },
          ],
        },
      ];
    }

    // If no OR entries created, return empty
    if (!filters.$or.length) {
      return res.json({ meta: { total: 0, page: 1, limit }, data: [] });
    }

    // build final query: if $and exists, include otherwise just $or
```

```
    const finalQuery = filters.$and ? { $and: filters.$and, $or: filters.$or } : { $or:
filters.$or };

    const [total, data] = await Promise.all([
      Task.countDocuments(finalQuery),
      Task.find(finalQuery)
        .sort({ createdAt: -1 })
        .skip(Number(skip))
        .limit(Number(limit))
        .lean(),
    ]);

    res.json({
      meta: { total, page: Number(page), limit: Number(limit), pages: Math.max(1,
Math.ceil(total / Number(limit))) },
      data,
    });
  } catch (err) {
    console.error("getMyTasks error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};

/**
 * PUT /api/employee/tasks/:id/progress
 * Body: { progress?: number, status?: string }
 * - Only assigned employee(s) can update
 */
exports.updateTaskProgress = async (req, res) => {
  try {
    const userId = req.user?.userId;
    if (!userId) return res.status(401).json({ msg: "Unauthorized" });

    const taskId = req.params.id;
    const { progress, status } = req.body;

    const task = await Task.findById(taskId);
    if (!task) return res.status(404).json({ msg: "Task not found" });

    // fetch current user's employeeId for comparison
    const user = await User.findById(userId).select("employeeId role");
    const employeeId = user?.employeeId;

    const isAssignedByUserId = (task.assignedTo || []).some((a) => String(a) ===
String(userId));
    const isAssignedByEmployeeId = employeeId && (task.assignedEmployeeIds || []).some((e)
=> String(e) === String(employeeId));

    if (!isAssignedByUserId && !isAssignedByEmployeeId) {
      return res.status(403).json({ msg: "Not allowed: you are not assigned to this task"
});
    }

    // validate and set
    if (typeof progress !== "undefined") {
      const p = Number(progress);
      if (Number.isNaN(p) || p < 0 || p > 100) {
        return res.status(400).json({ msg: "Invalid progress value" });
      }
      task.progress = p;
      // optionally set status automatically
      if (p === 100) task.status = "Completed";
      else if (task.status === "Assigned") task.status = "In Progress";
    }

    if (typeof status !== "undefined") {
```

```
      // prevent employee from setting invalid status (manager-only statuses can be
restricted)
      const allowed = ["Assigned", "In Progress", "Blocked", "Completed", "Cancelled"];
      if (!allowed.includes(status)) return res.status(400).json({ msg: "Invalid status"
});
      // employees should not be able to set 'Assigned' back from Completed? you can
restrict here if needed
      task.status = status;
      if (status === "Completed") task.progress = 100;
    }

    await task.save();
    return res.json({ msg: "Task updated", task });
  } catch (err) {
    console.error("updateTaskProgress error:", err);
    res.status(500).json({ msg: "Server error" });
  }
};
```

```
// server/middleware/authMiddleware.js
const jwt = require("jsonwebtoken");
const User = require("../models/users"); // optional, for deeper verification

module.exports = async function (req, res, next) {
  const authHeader = req.headers.authorization || req.header("x-auth-token");
  const token =
    authHeader && authHeader.startsWith("Bearer ")
      ? authHeader.split(" ")[1]
      : authHeader;

  if (!token) return res.status(401).json({ msg: "No token, authorization denied" });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Optionally verify user exists (recommended)
    // const user = await User.findById(decoded.id).select("-password");
    // if (!user) return res.status(404).json({ msg: "User not found" });

    req.user = decoded; // attach decoded payload to request
    next();
  } catch (err) {
    console.error("Auth error:", err);
    res.status(401).json({ msg: "Token is not valid" });
  }
};
```

```
// server/middleware/roleMiddleware.js
/**
 * Usage: role("Manager") or role("Admin","HR")
 * Accepts one or more roles (strings).
 */
module.exports = function (...allowedRoles) {
  // normalize allowed roles to lowercase trimmed
  const normalized = allowedRoles.map((r) => String(r).toLowerCase().trim());
  return (req, res, next) => {
    try {
      const payloadRole = (req.user && req.user.role) || "";
      const userRole = String(payloadRole).toLowerCase().trim();
      if (!userRole) return res.status(401).json({ msg: "Role missing in token" });

      if (!normalized.includes(userRole)) {
        return res.status(403).json({ msg: "Access denied: insufficient role" });
      }
      next();
    } catch (err) {
```

```
      console.error("roleMiddleware error:", err);
      res.status(500).json({ msg: "Server error" });
    }
  };
};
```

```
// server/middleware/upload.js
const multer = require("multer");
const path = require("path");
const fs = require("fs");

const uploadDir = path.join(__dirname, "..", "notification_uploads"); // changed folder
name
if (!fs.existsSync(uploadDir)) fs.mkdirSync(uploadDir, { recursive: true });

const storage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, uploadDir),
  filename: (req, file, cb) => {
    const ext = path.extname(file.originalname);
    const name = `${Date.now()}-${Math.random().toString(36).slice(2,8)}${ext}`;
    cb(null, name);
  }
});

const fileFilter = (req, file, cb) => {
  const allowed = ["image/png","image/jpeg","image/jpg","image/webp"];
  cb(null, allowed.includes(file.mimetype));
};

// 1 MB limit (adjust if needed)
const upload = multer({ storage, fileFilter, limits: { fileSize: 1 * 1024 * 1024 } });

module.exports = upload;
```

```
// server/models/attendance.js
const mongoose = require("mongoose");

const attendanceSchema = new mongoose.Schema({
  employeeId: { type: String, required: true },
  name: { type: String, required: true },
  role: { type: String },
  status: { type: String, enum: ["Present", "Absent"], default: "Absent" },
  date: {
    type: String,
    required: true,
    default: () => new Date().toISOString().split("T")[0], // auto YYYY-MM-DD
  },
  remarks: { type: String, default: "" },
});

module.exports = mongoose.model("Attendance", attendanceSchema, "attendance");
```

```
// server/models/leave.js
const mongoose = require("mongoose");

const leaveSchema = new mongoose.Schema({
  employeeId: { type: String, required: true },
  name: { type: String, required: true },
  role: { type: String, required: true, default: "Employee" },
  leaveType: { type: String, required: true },
  fromDate: { type: String, required: true }, // you used ISO string in DB
  toDate: { type: String, required: true },
  reason: { type: String, required: true },
  status: {
    type: String,
    enum: [
```

```
      "Pending",   // default new request
      "Verified",  // manager marked verified (pre-approve)
      "Approved",  // approved by manager
      "Rejected",  // rejected by manager
      "Expired",   // optional tag when toDate passed without action
    ],
    default: "Pending",
  },
  appliedAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model("Leave", leaveSchema, "leave_requests");
```

```
// server/models/notification.js
const mongoose = require("mongoose");

const notificationSchema = new mongoose.Schema({
  title: { type: String, required: true, trim: true },
  message: { type: String, required: true, trim: true },
  type: { type: String, enum: ["Announcement", "Blog", "Notice", "Update"], default:
"Announcement" },
  image: { type: String }, // filename or full URL
  createdBy: { type: mongoose.Schema.Types.ObjectId, ref: "users" },
  createdByName: { type: String },
  visibleTo: { type: String, enum: ["All", "Employees", "Managers"], default: "All" }, //
optional
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model("Notification", notificationSchema, "notifications");
```

```
// server/models/task.js
const mongoose = require("mongoose");

const AssigneeSchema = new mongoose.Schema({
  employeeId: { type: String, required: true },
  name: { type: String },
});

const taskSchema = new mongoose.Schema({
  title: { type: String, required: true, trim: true },
  description: { type: String, default: "" },
  assignedTo: { type: [AssigneeSchema], default: [] }, // can be multiple assignees
  createdBy: {
    userId: { type: mongoose.Schema.Types.ObjectId, ref: "users" },
    name: { type: String },
    role: { type: String },
  },
  dueDate: { type: Date },
  priority: { type: String, enum: ["Low", "Medium", "High", "Critical"], default: "Medium"
},
  tags: { type: [String], default: [] },
  status: { type: String, enum: ["Pending", "In Progress", "Completed", "Cancelled"],
default: "Pending" },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now },
});

taskSchema.pre("save", function (next) {
  this.updatedAt = Date.now();
  next();
});

taskSchema.pre("findOneAndUpdate", function (next) {
  this.set({ updatedAt: Date.now() });
  next();
```

```
});

module.exports = mongoose.model("Task", taskSchema, "tasks");
```

```
// server/models/users.js
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({
  // Step 1: Personal Details
  name: { type: String, required: true, minlength: 3 },
  gender: { type: String, enum: ["Male", "Female", "Other"] },
  dob: { type: Date },
  email: { type: String, required: true, unique: true },
  phone: { type: String },
  address: { type: String },
  city: { type: String },
  state: { type: String },
  pincode: { type: String },
  country: { type: String },
  emergencyContactName: { type: String },
  emergencyPhone: { type: String },

  // Step 2: Employee Details
  role: {
    type: String,
    enum: ["Admin", "HR", "Manager", "Employee"],
    default: "Employee",
  },
  employeeId: { type: String },
  department: { type: String },
  designation: { type: String },
  joiningDate: { type: Date },
  employmentType: {
    type: String,
    enum: ["Full-Time", "Part-Time", "Internship", "Contract"],
  },
  reportingManager: { type: String },
  workEmail: { type: String },
  workMode: { type: String, enum: ["Onsite", "Remote", "Hybrid"] },

  // Step 3: Documents & Account Setup
  photo: { type: String }, // File path or URL
  signature: { type: String }, // File path or URL
  password: { type: String, required: true },
  securityQuestion: { type: String },
  securityAnswer: { type: String },
  termsAccepted: { type: Boolean, default: false },

  // Metadata
  createdAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model("users", userSchema, "registers");
```

```
// server/routes/employee/tasks.js
const express = require("express");
const router = express.Router();
const auth = require("../../middleware/authMiddleware");
const taskController = require("../../controllers/taskController");

// GET my tasks (paginated)
router.get("/tasks", auth, taskController.getMyTasks);
// update progress/status
router.put("/tasks/:id/progress", auth, taskController.updateTaskProgress);

module.exports = router;
```

```
// server/routes/manager/leaveRoutes.js
const express = require("express");
const router = express.Router();
const leaveController = require("../../controllers/leaveController");
const auth = require("../../middleware/authMiddleware");
const ensureRole = require("../../middleware/roleMiddleware");

// Use auth + role check so only manager/hr/admin can call these routes
router.get("/getLeaves", auth, ensureRole("Manager","HR","Admin"),
leaveController.getLeaves);

// individual actions
router.put("/:id/verify", auth, ensureRole("Manager","HR","Admin"),
leaveController.verifyLeave);
router.put("/:id/approve", auth, ensureRole("Manager","HR","Admin"),
leaveController.approveLeave);
router.put("/:id/reject", auth, ensureRole("Manager","HR","Admin"),
leaveController.rejectLeave);

module.exports = router;
```

```
// server/routes/manager/notificationRoutes.js
const express = require("express");
const router = express.Router();
const auth = require("../../middleware/authMiddleware");
const role = require("../../middleware/roleMiddleware");
const upload = require("../../middleware/upload");
const controller = require("../../controllers/manager/notificationController");

// Manager/HR/Admin create, list, delete
router.post("/", auth, role("Manager","HR","Admin"), upload.single("image"),
controller.createNotification);
router.get("/", auth, role("Manager","HR","Admin"), controller.listNotifications);
router.get("/:id", auth, role("Manager","HR","Admin"), controller.getNotification);
router.delete("/:id", auth, role("Manager","HR","Admin"), controller.deleteNotification);

module.exports = router;
```

```
// server/routes/manager/taskRoutes.js
const express = require("express");
const router = express.Router();
const auth = require("../../middleware/authMiddleware");
const role = require("../../middleware/roleMiddleware"); // usage:
role("Manager","HR","Admin")
const controller = require("../../controllers/manager/taskController");

// Manager operations (create, list, get, update, delete)
router.post("/", auth, role("Manager","HR","Admin"), controller.createTask);
router.get("/", auth, role("Manager","HR","Admin"), controller.listTasks);
router.get("/:id", auth, role("Manager","HR","Admin"), controller.getTask);
router.put("/:id", auth, role("Manager","HR","Admin"), controller.updateTask);
router.delete("/:id", auth, role("Manager","HR","Admin"), controller.deleteTask);

// Status patch - allow both managers and employees (auth required)
router.patch("/:id/status", auth, controller.patchStatus);

// Employee-specific: get tasks for current employee
router.get("/employee/me", auth, controller.listMyTasks);

module.exports = router;
```

```
// server/routes/manager/userRoutes.js
const express = require("express");
const router = express.Router();
const managerController = require("../../controllers/manager/userController");
```

```
const auth = require("../../middleware/authMiddleware");
const role = require("../../middleware/roleMiddleware");

// All manager routes protected: must be authenticated and role = Manager
router.get("/employees", auth, role("Manager"), managerController.listEmployees);
// GET single
router.get("/employees/:id", auth, role("Manager"), managerController.getEmployeeById);

// PUT /api/manager/employees/:id
router.put("/employees/:id", auth, role ? role("Manager") : auth,
managerController.updateEmployee);
module.exports = router;
```

```
// server/routes/attendanceRoutes.js
const express = require("express");
const router = express.Router();
const { markAttendance, getAttendance } = require("../controllers/attendanceController");
// ✅ fix name
const authMiddleware = require("../middleware/authMiddleware");

// route to mark attendance
router.post("/mark", authMiddleware, markAttendance);
// route for get attendance data
router.get("/get", authMiddleware, getAttendance);

module.exports = router;
```

```
// server/routes/auth.js
const express = require("express");
const router = express.Router();
const { register, login } = require("../controllers/register"); // your controller file
const multer = require("multer");
const fs = require("fs");
const path = require("path");

// ensure uploads folder exists
const uploadDir = path.join(__dirname, "..", "uploads");
if (!fs.existsSync(uploadDir)) fs.mkdirSync(uploadDir, { recursive: true });

// Multer storage setup
const storage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, uploadDir),
  filename: (req, file, cb) => cb(null, Date.now() + "-" + file.originalname),
});

const upload = multer({ storage });

// register (multipart/form-data -> photo + signature + other fields)
router.post(
  "/register",
  upload.fields([
    { name: "photo", maxCount: 1 },
    { name: "signature", maxCount: 1 },
  ]),
  register
);

// login (JSON)
router.post("/login", login);

module.exports = router;
```

```
// server/routes/leaveRoutes.js
const express = require("express");
const router = express.Router();
const {
```

```
  applyLeave,
  getLeaves,
} = require("../controllers/leaveController");
const authMiddleware = require("../middleware/authMiddleware");

// Employee & Admin Routes
router.post("/apply", authMiddleware, applyLeave);
router.get("/show", authMiddleware, getLeaves);
// router.put("/update/:id", authMiddleware, updateLeaveStatus);

module.exports = router;
```

```
// server/routes/notificationPublicRoutes.js
const express = require("express");
const router = express.Router();
const auth = require("../middleware/authMiddleware");
const Notification = require("../models/notification");

router.get("/", auth, async (req, res) => {
  try {
    // optional params: page, limit, search, type
    const page = parseInt(req.query.page, 10) || 1;
    const limit = parseInt(req.query.limit, 10) || 50;
    const skip = (page -1) * limit;
    const filters = {};
    if (req.query.type) filters.type = req.query.type;
    if (req.query.search) {
      const q = req.query.search.trim();
      filters.$or = [{ title: { $regex: q, $options: "i"} }, { message: { $regex: q,
$options: "i" } }];
    }

    const [total, data] = await Promise.all([
      Notification.countDocuments(filters),
      Notification.find(filters).sort({ createdAt: -1 }).skip(skip).limit(limit)
    ]);
    res.json({ meta: { total, page, limit, pages: Math.ceil(total/limit)||1 }, data });
  } catch (err) {
    console.error("public notifications error:", err);
    res.status(500).json({ msg: "Server error" });
  }
});

module.exports = router;
```

```
// server/routes/settingsRoutes.js
const express = require("express");
const router = express.Router();
const authMiddleware = require("../middleware/authMiddleware");
const { updateProfile } = require("../controllers/settingsController");

// ✅ Update user settings (profile + password + security)
router.put("/update-settings", authMiddleware, updateProfile);

module.exports = router;
```

```
// server/routes/user.js
const express = require("express");
const router = express.Router();
const auth = require("../middleware/authMiddleware");
const User = require("../models/users"); // your user model

// GET /api/user/me  -> protected, returns current user's data
router.get("/me", auth, async (req, res) => {
  try {
    // req.user.userId comes from authMiddleware
```

```
    const user = await User.findById(req.user.userId).select("-password -securityAnswer");
    if (!user) return res.status(404).json({ msg: "User not found" });
    res.json(user);
  } catch (err) {
    console.error("GET /api/user/me error:", err);
    res.status(500).json({ msg: "Server error" });
  }
});

module.exports = router;
```

// index.js

```
require("dotenv").config();
const express = require("express");
const cors = require("cors");
const connectDB = require("./config/db");
const attendanceRoutes = require("./routes/attendanceRoutes");
const leaveRoutes = require("./routes/leaveRoutes");
const settingsRoutes = require("./routes/settingsRoutes");

const app = express();
connectDB();

app.use(cors({ origin: process.env.CLIENT_URL || "http://localhost:5173" }));
app.use(express.json({ limit: "10mb" }));
app.use('/uploads', express.static('uploads'));

// routes
app.use("/api", require("./routes/auth"));

app.use("/api/attendance", attendanceRoutes);
app.use("/api/leaves", leaveRoutes);
app.use("/api/settings", settingsRoutes);



app.use("/api/manager/tasks", require("./routes/manager/taskRoutes"));
const managerNotifications = require("./routes/manager/notificationRoutes");
const publicNotifications = require("./routes/notificationPublicRoutes");

app.use("/api/manager/notifications", managerNotifications);
app.use("/api/notifications", publicNotifications); // authenticated public view for
employees
// notification uploads
const path = require("path");
// serve notification uploads
app.use("/notification_uploads", express.static(path.join(__dirname,
"notification_uploads")));


const managerUserRoutes = require("./routes/manager/userRoutes");


// for user management for manager
app.use("/api/manager", managerUserRoutes);

// server/app.js (or server.js)
const managerLeaveRoutes = require("./routes/manager/leaveRoutes");
app.use("/api/manager/", managerLeaveRoutes);
app.use("/api/manager/leaves", managerLeaveRoutes);

// Protected user routes


app.use("/api/user", require("./routes/user"));
```

```
const authMiddleware = require("./middleware/authMiddleware");
app.get("/api/me", authMiddleware, async (req, res) => {
  res.json({ userId: req.user.userId, role: req.user.role });
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
```

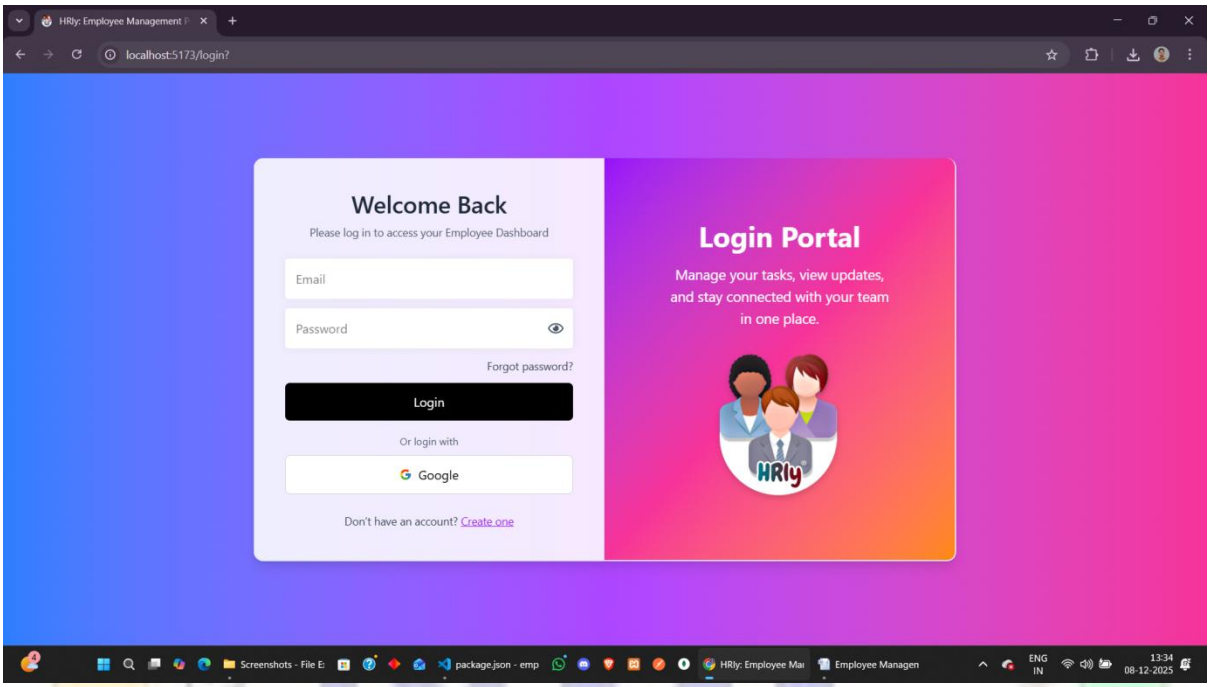
// Package.json

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "keywords": [],
  "author": "Somenath Paul",
  "license": "ISC",
  "type": "commonjs",
  "dependencies": {
    "bcryptjs": "^3.0.2",
    "cors": "^2.8.5",
    "dotenv": "^17.2.3",
    "express": "^5.1.0",
    "jsonwebtoken": "^9.0.2",
    "moment": "^2.30.1",
    "mongoose": "^8.19.2",
    "multer": "^2.0.2"
  },
  "devDependencies": {
    "nodemon": "^3.1.10"
  }
}
```

SNAPSHOTS

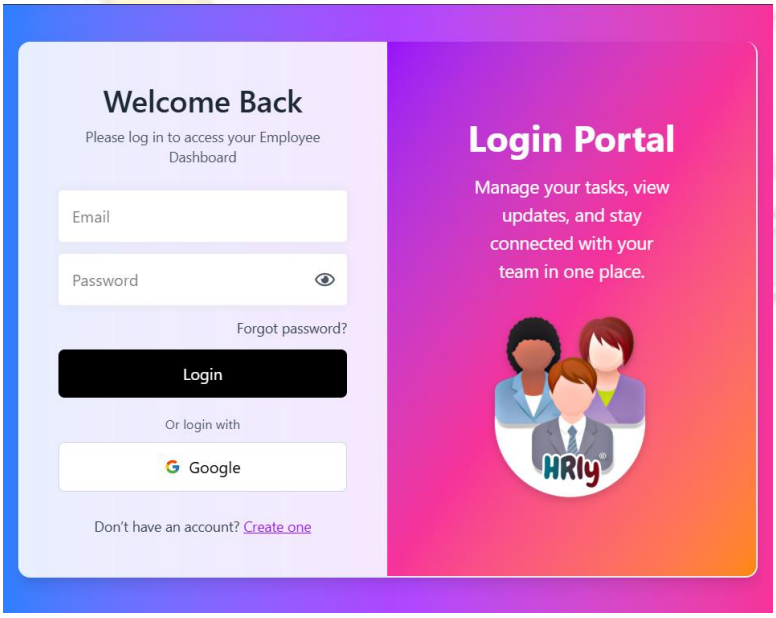
Login page

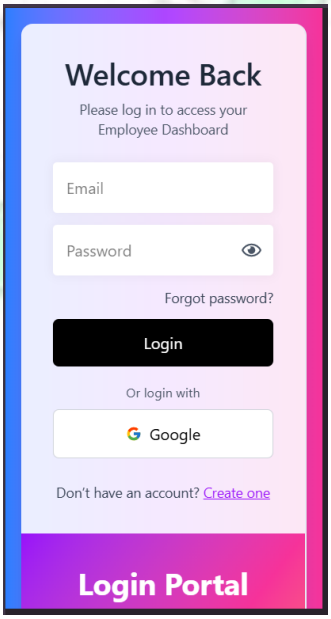
Purpose: Allows users to securely access the system.
Features: Email/password authentication with role-based login.



Login page in small screens

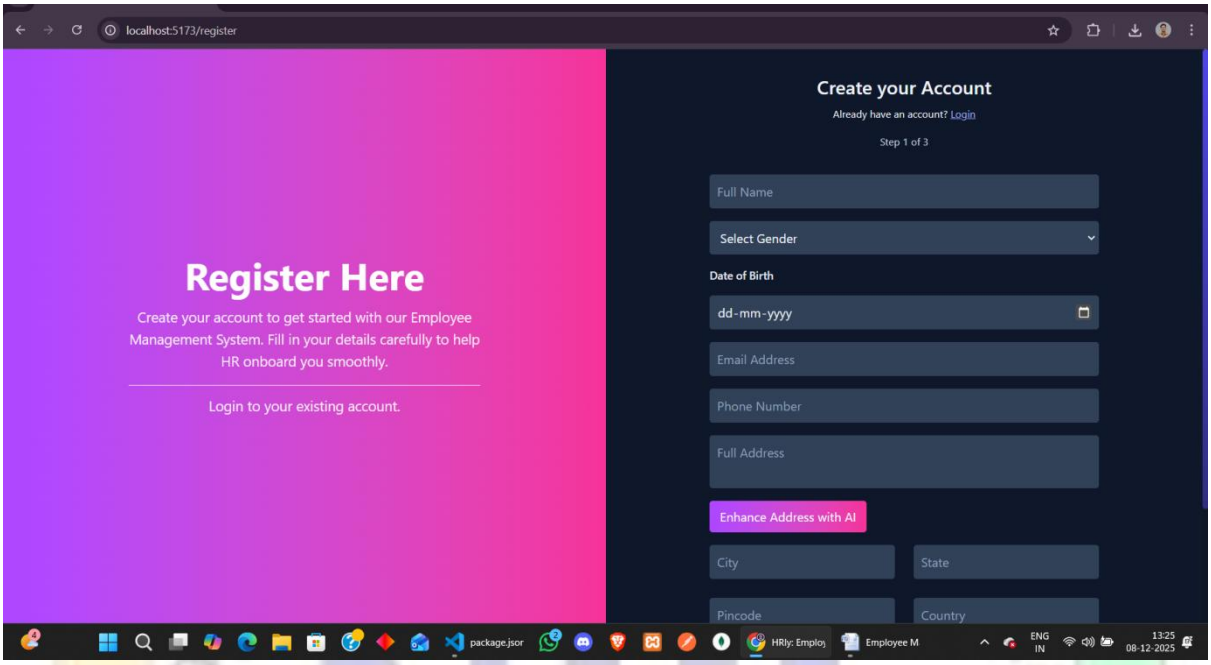
Purpose: Ensures smooth login experience on mobile/tablets.
Features: Fully responsive UI with optimized input layout.





Register/Signup page

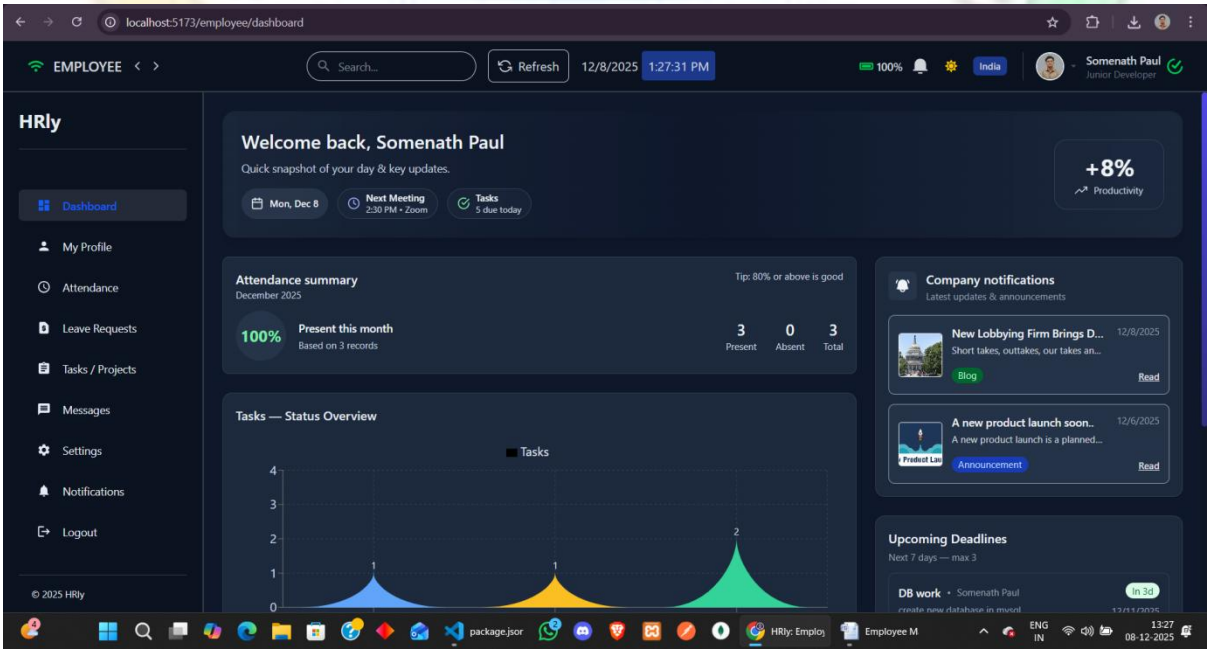
Purpose: Enables new users to create an account.
Features: Form validation, role selection, and secure data submission.



Employee pages

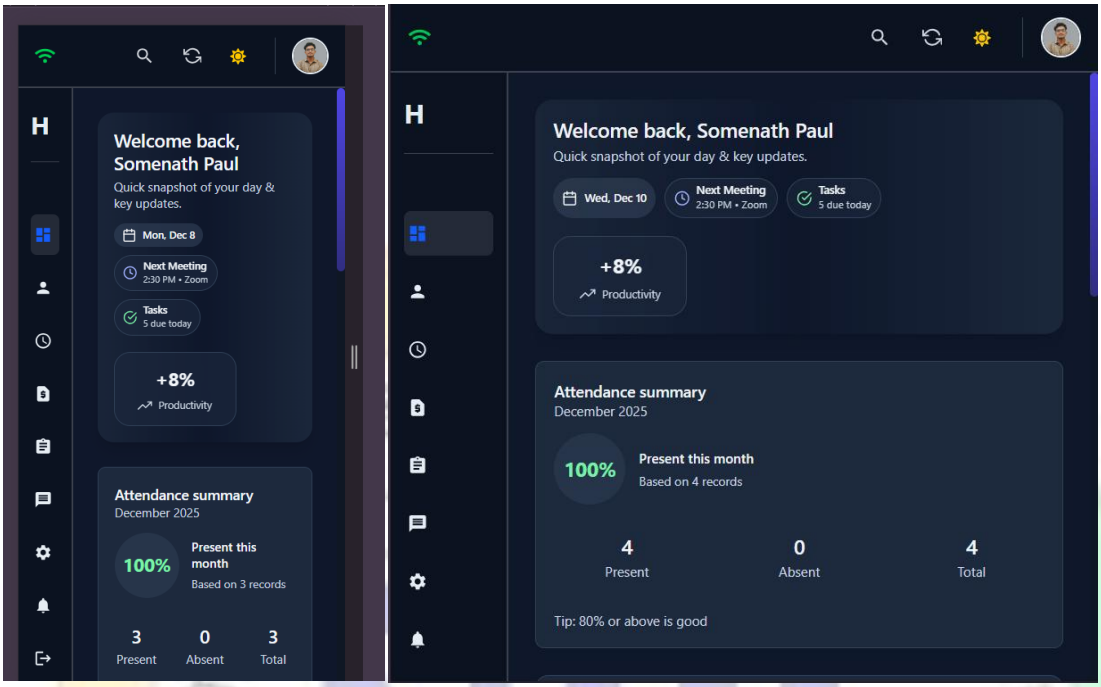
Employee Dashboard

Purpose: Provides employees a quick overview of daily activities.
Features: Task summary, attendance stats, notifications, and shortcuts.



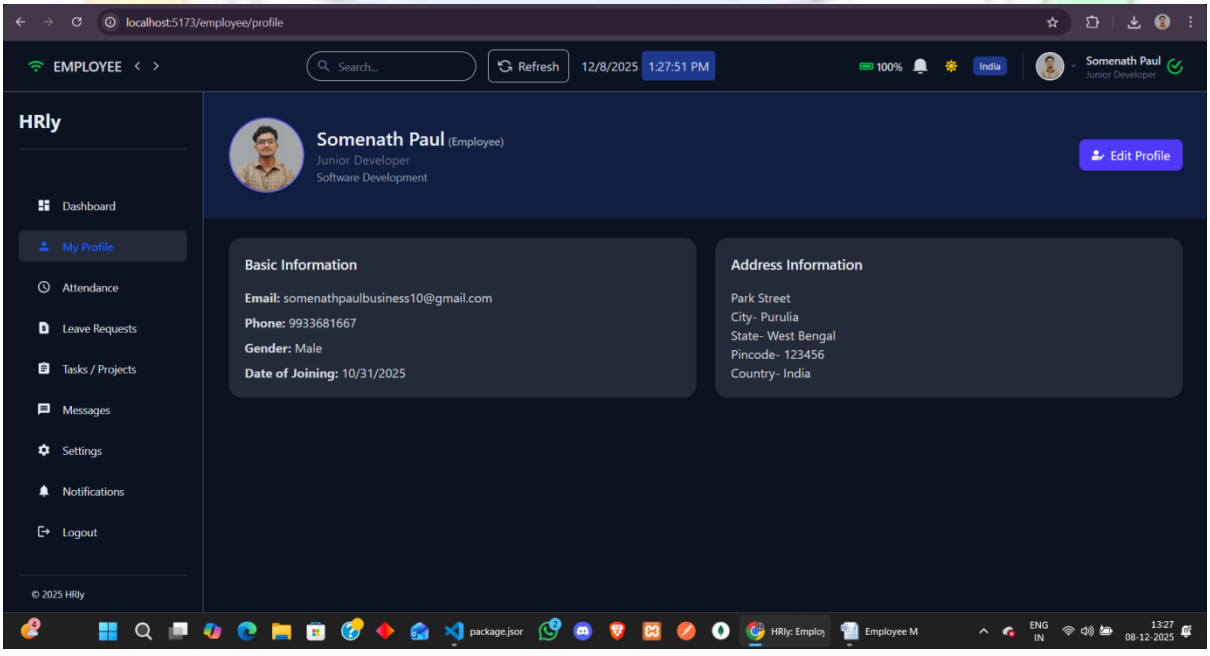
Employee Dashboard: for small screens

Purpose: Makes dashboard fully usable on mobile/tablet devices.
Features: Responsive cards, adaptive layout, and simplified navigation.



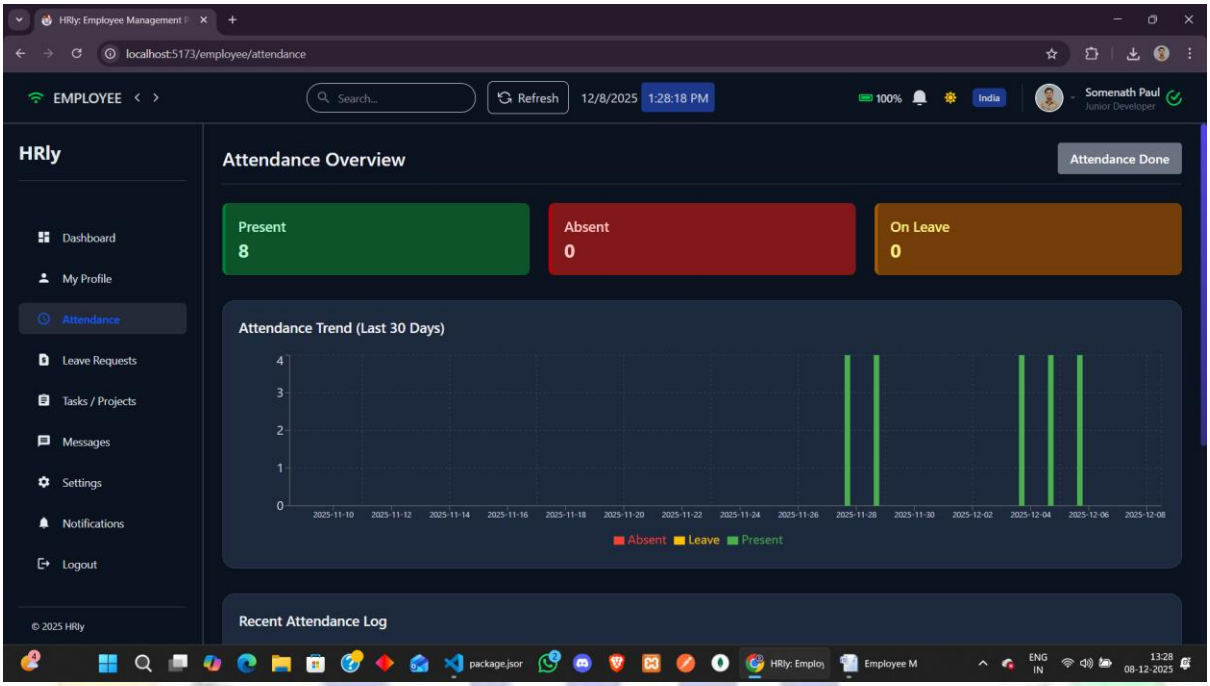
View/edit Employee Profile page

Purpose: Lets employees view and update their personal details.
Features: Editable fields, profile picture display, clean UI.



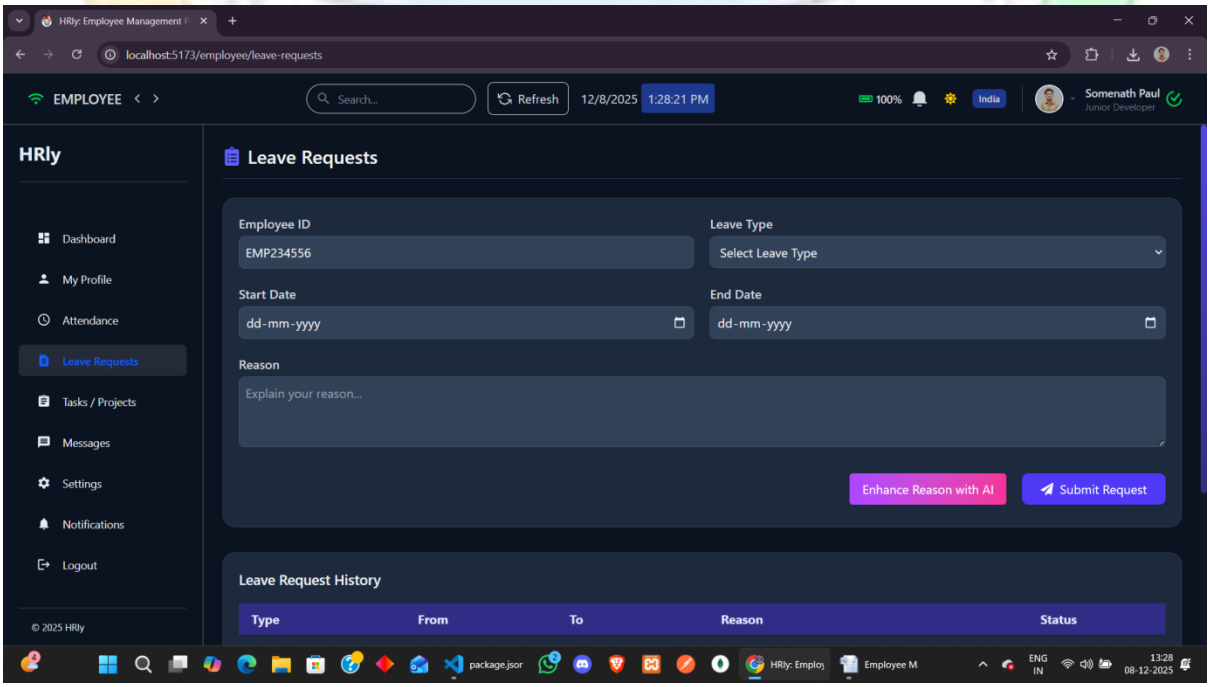
Employee Attendance page

Purpose: Allows employees to give their daily attendance and track monthly attendance.
Features: Daily status list, summary statistics, and date filtering.



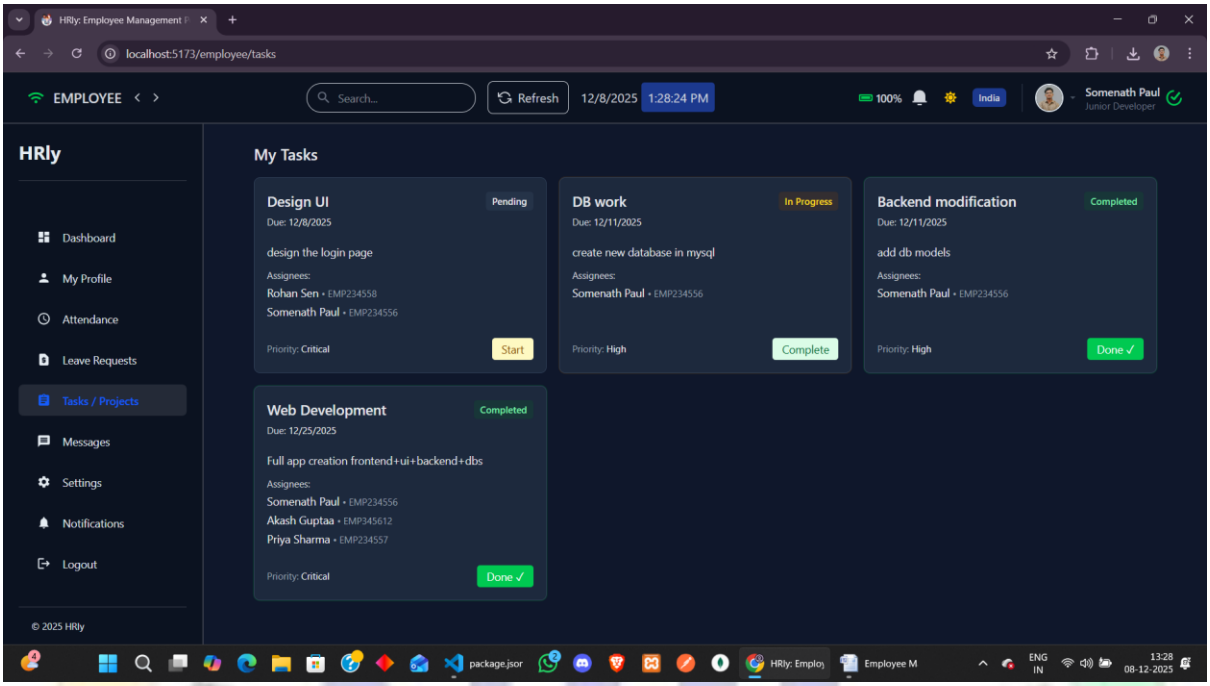
Employee Request Leave page

Purpose: Enables employees to submit leave applications online.
Features: Leave form, date validation, and submission tracking.



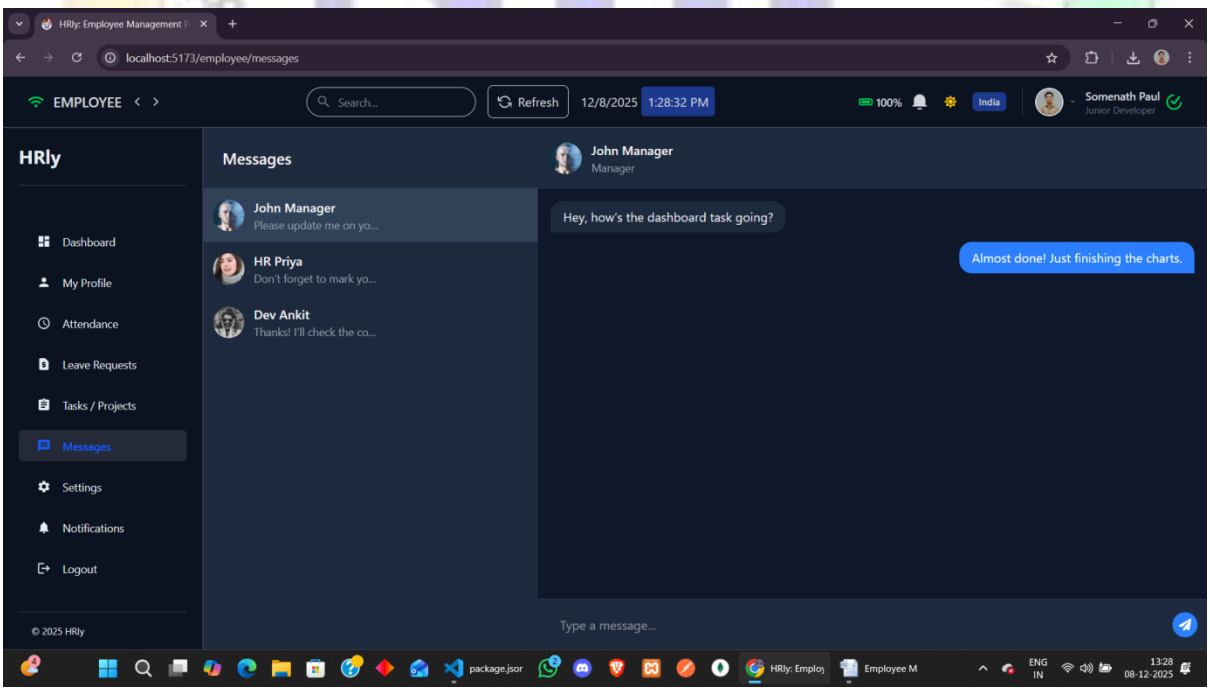
Employee Task page

Purpose: Shows all assigned tasks and their progress.
Features: Task cards, status updates (Start/Complete), responsive layout.



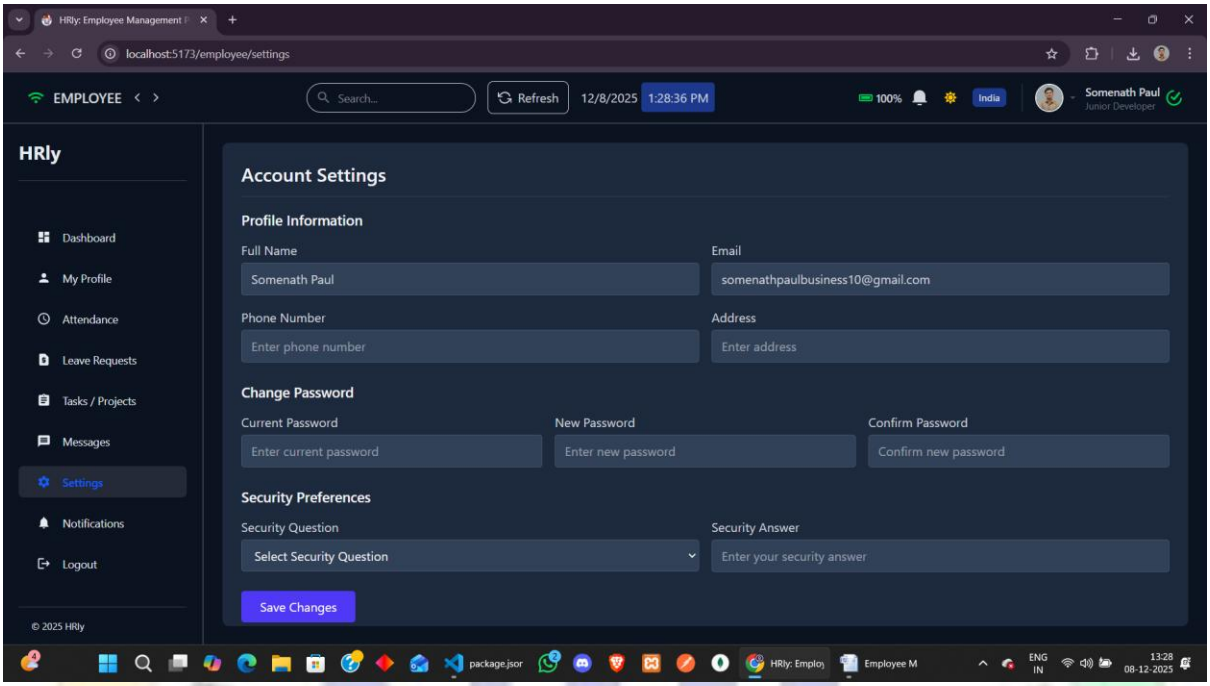
Employee Massage page

Purpose: Provides internal messaging between employees and managers.
Features: Chat UI, real-time message updates, user list.



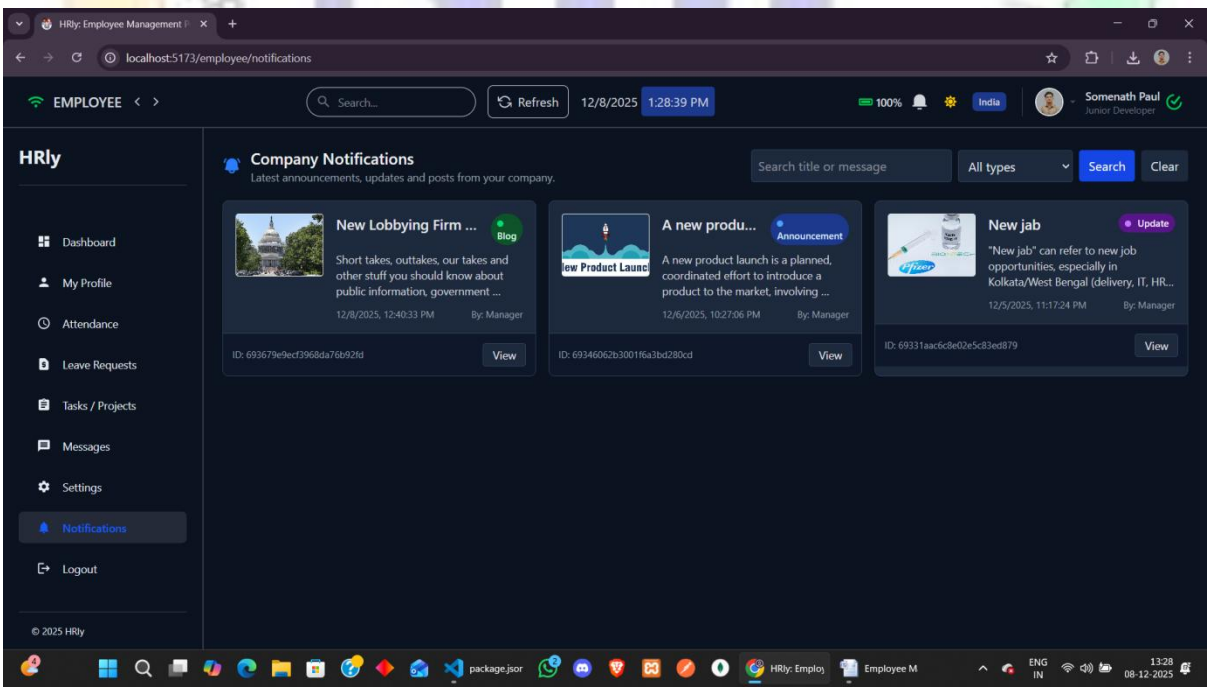
Employee Settings page

Purpose: Allows users to configure their account preferences.
Features: profile settings, update options.



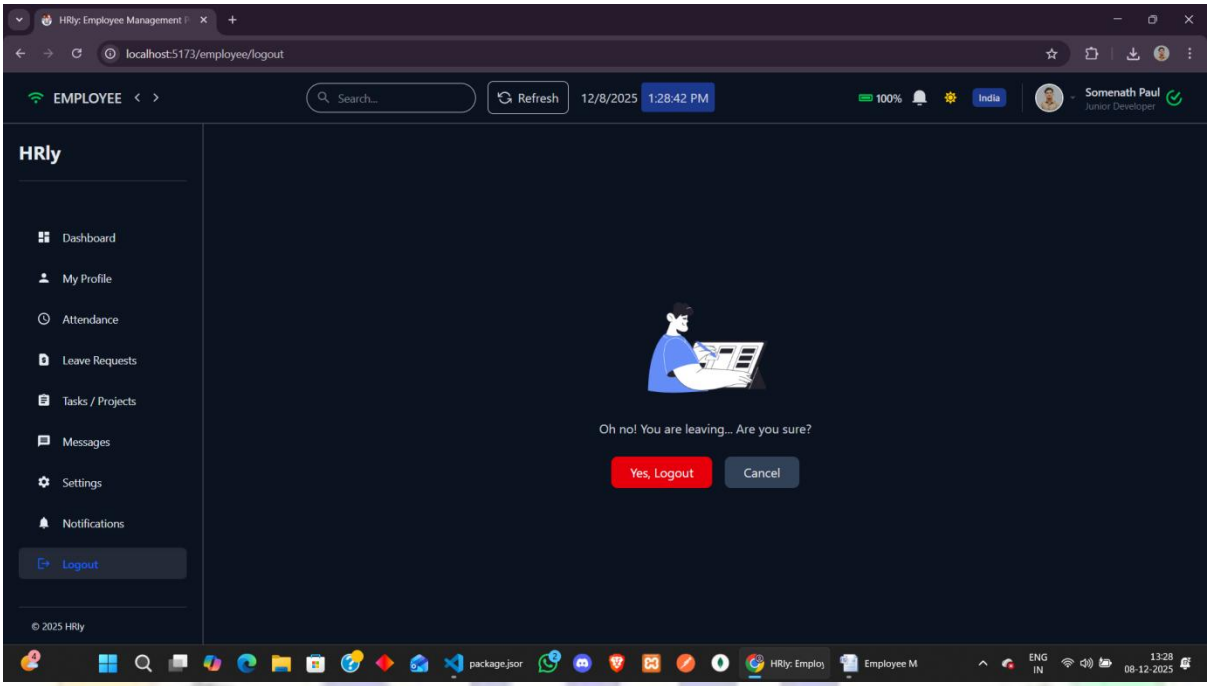
Employee Notifications page

Purpose: Displays company announcements and updates.
Features: Search, filters, categorized notifications.



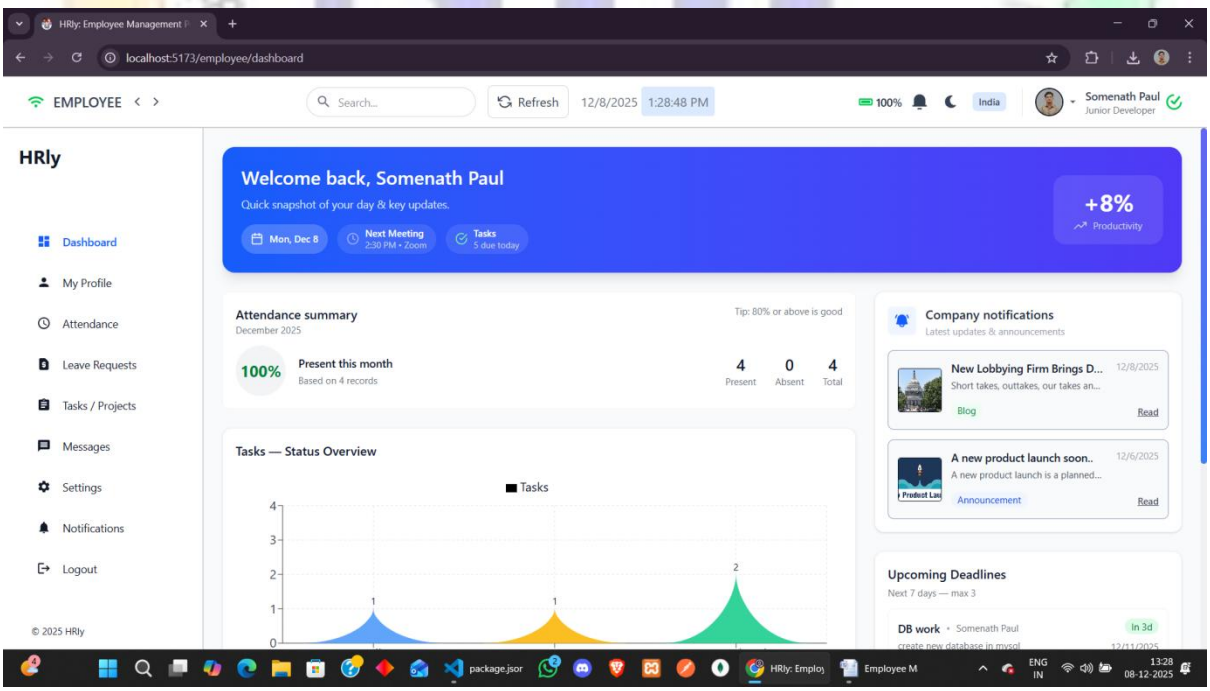
Employee Logout page

Purpose: Securely ends user session.
Features: One-click logout with redirection.



Light mode of the Website

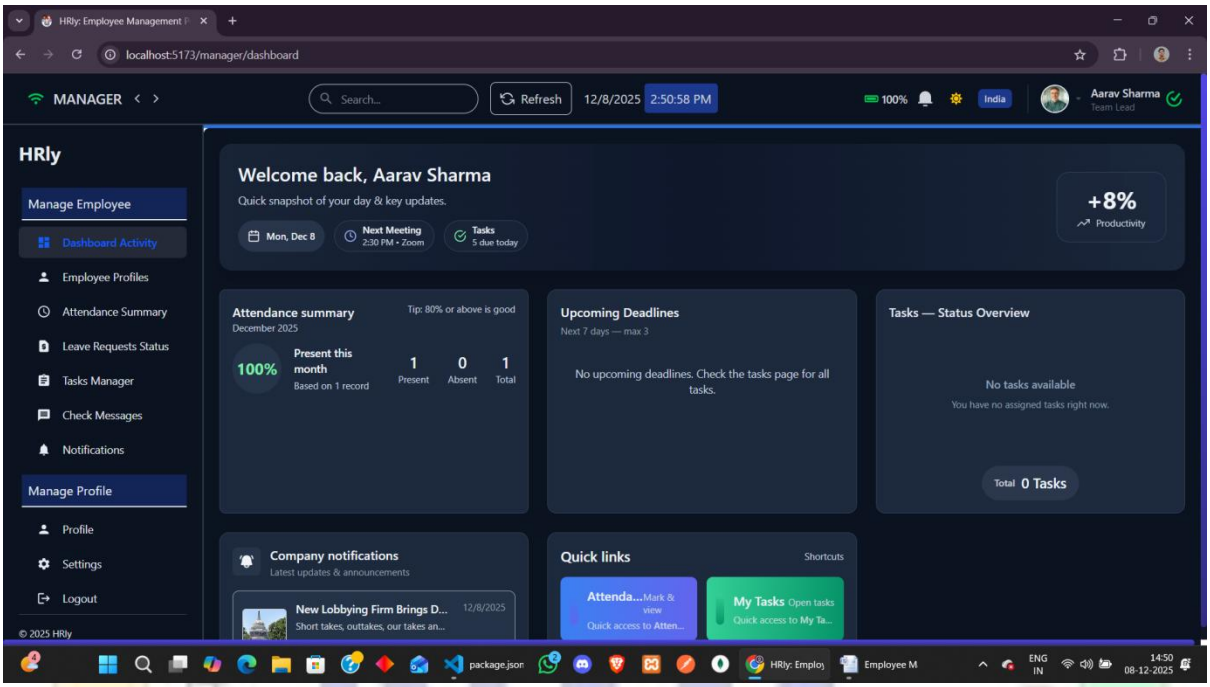
Purpose: Provides an alternate visual theme for better usability.
Features: Smooth theme switching and optimized colors.



Manager pages

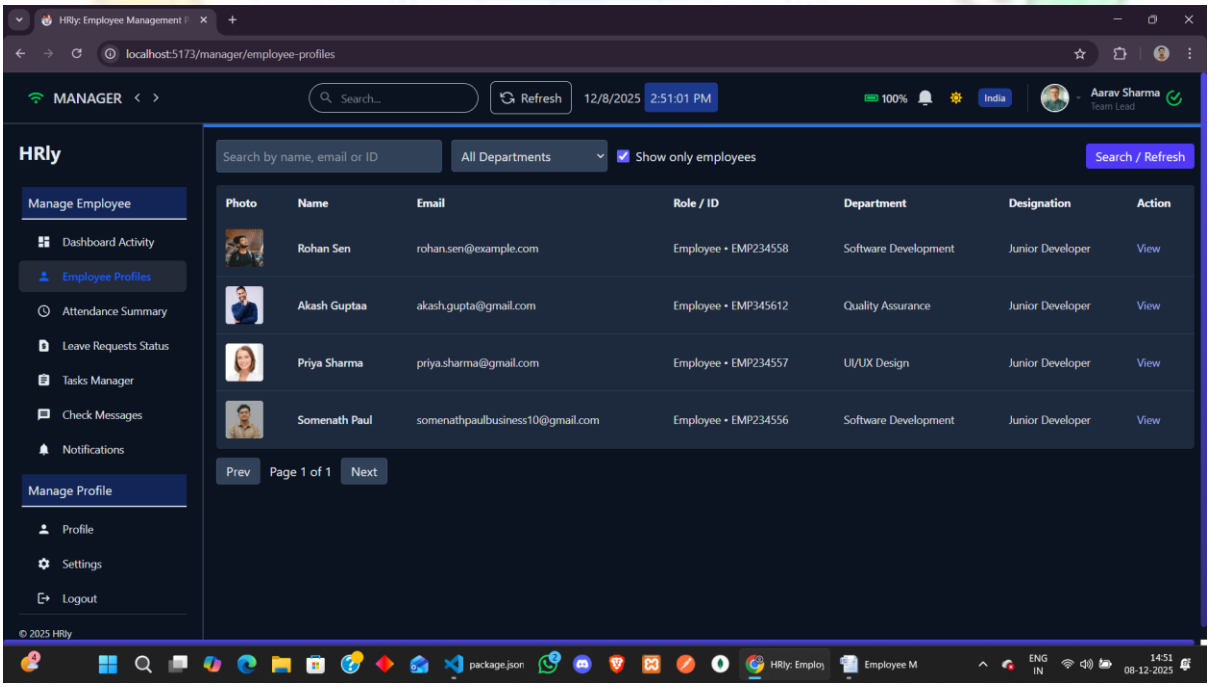
Manager Dashboard

Purpose: Gives managers a quick summary of company activity.
Features: Employee insights, attendance overview, quick actions.



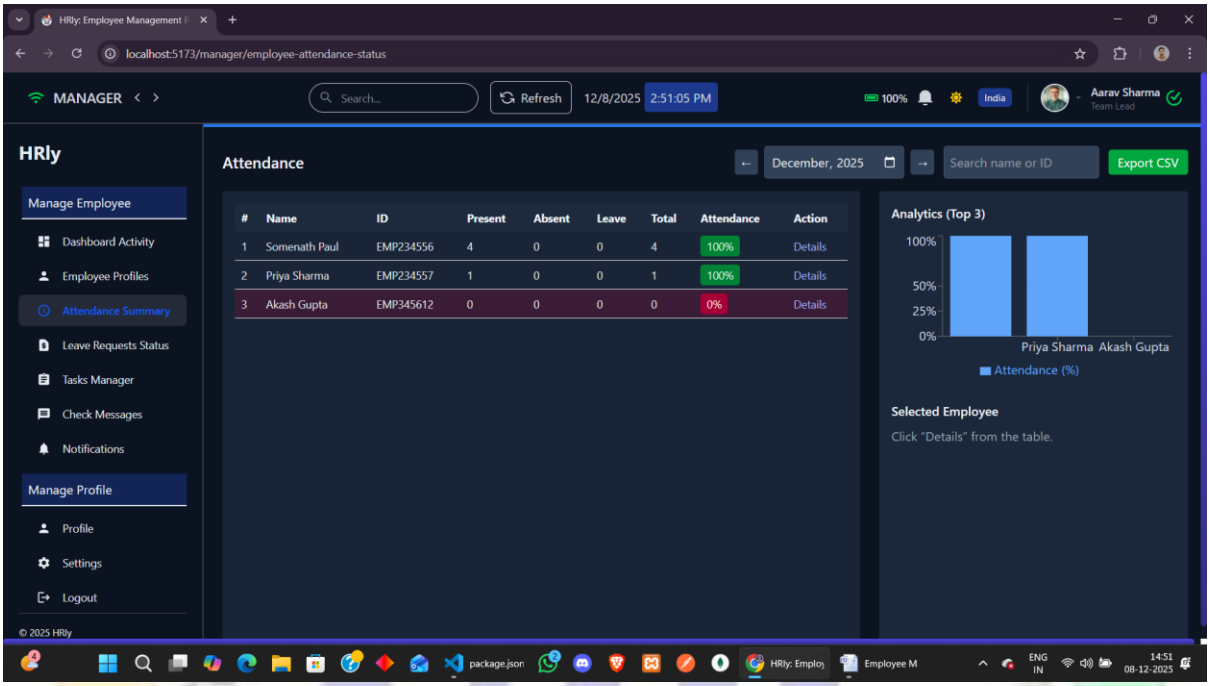
Employee Profiles (Employee data view/edit)

Purpose: Allows managers to manage employee records.
Features: Full profile view, editing, and filtering tools.



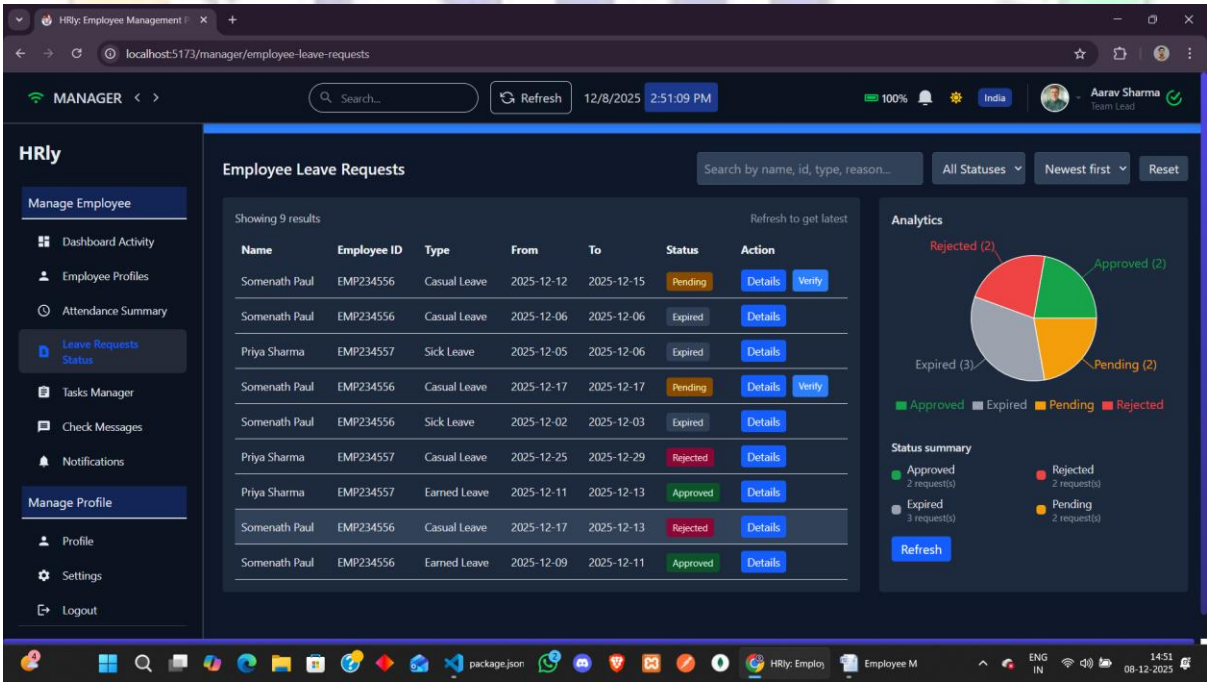
Manager Attendance monitoring page

Purpose: Lets managers monitor employee attendance trends.
Features: Status breakdown, charts, monthly reports.



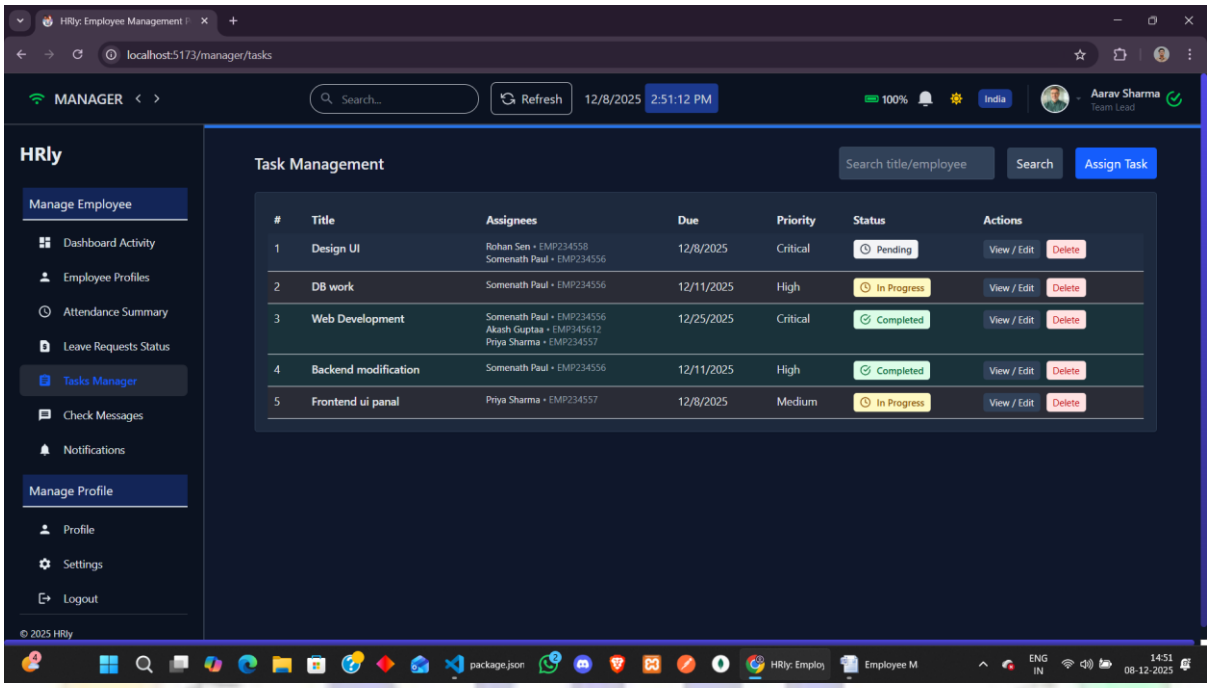
Edit Employee Leave Request page

Purpose: Enables managers to review and approve/reject leaves.
Features: Status controls, leave details, verification steps.



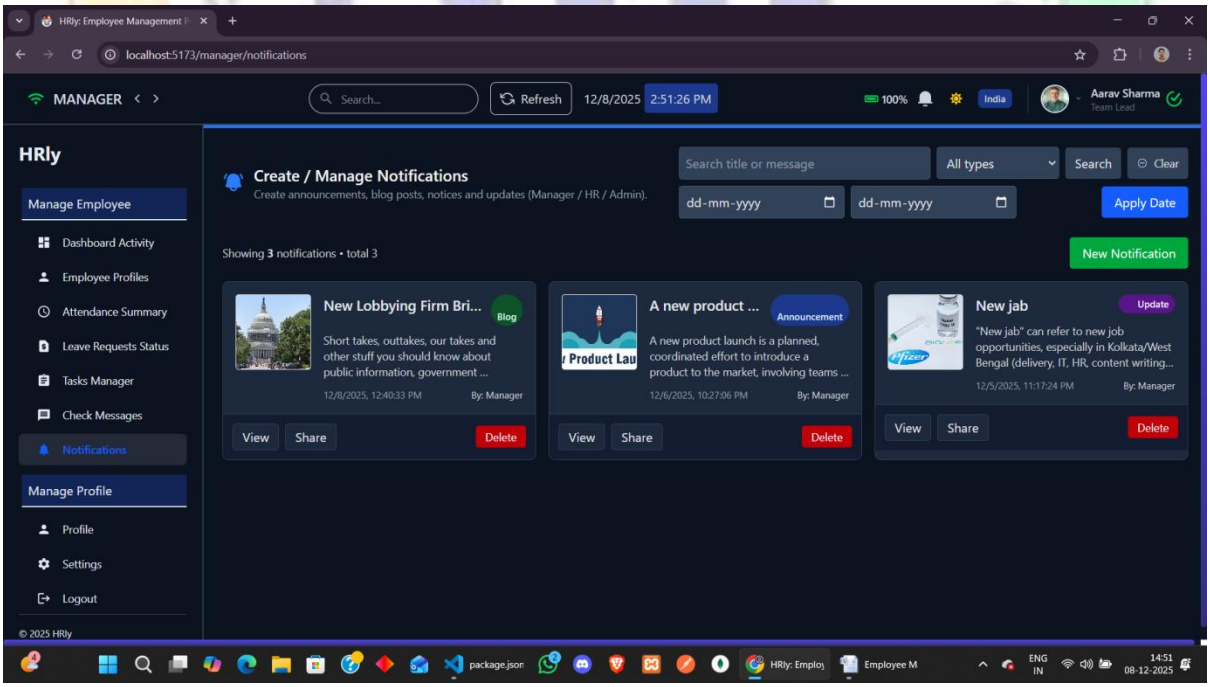
Manager Task manager

Purpose: Allows managers to create and track employee tasks.
Features: Task assignment, editing, deletion, and progress overview.



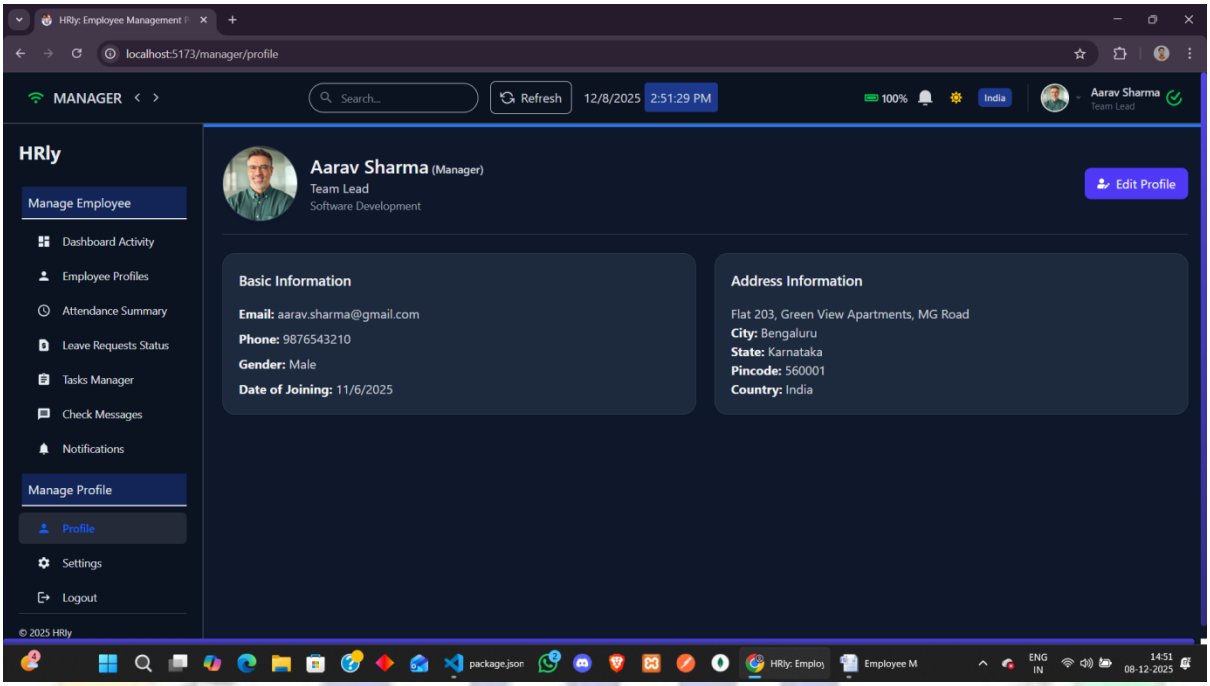
Create notification page

Purpose: Lets managers publish announcements or updates.
Features: Add title/message, upload image, choose notification type.



Manager profile page

Purpose: Displays manager's personal and professional details.
Features: Photo, department info, editable profile fields.



Conclusion

The Employee Management System successfully demonstrates how modern technologies can streamline organizational operations and improve workforce management. Through the development and deployment of this MERN-based system, several key objectives have been achieved:

1. Streamlined Operations:

The system automates essential HR activities such as employee registration, task assignment, attendance tracking, and departmental management. This reduces manual effort, minimizes administrative errors, and ensures efficient workflow across the organization.

2. Enhanced User Experience:

With a clean and intuitive interface built using React and TailwindCSS, employees and administrators can easily navigate the platform, manage tasks, view attendance, and update profiles. The responsive design ensures accessibility across devices and user types.

3. Comprehensive Data Management:

MongoDB and Mongoose provide robust storage for employee records, attendance logs, task histories, and departmental data. Secure and structured data handling supports accurate reporting and helps administrators make informed decisions.

4. Improved Productivity and Collaboration:

By offering real-time updates, clear task assignments, and performance visibility, the system boosts accountability and enhances coordination between employees and management. Automated reminders and progress tracking further improve task completion efficiency.

5. Scalability and Flexibility:

The EMS architecture supports future expansion, including integration of payroll modules, biometric attendance, performance evaluation tools, and third-party APIs. This scalability ensures the system remains adaptable to evolving organizational needs.

Overall Impact:

The Employee Management System has proven to be a valuable tool for modernizing workforce operations. By combining efficient backend logic, dynamic frontend interfaces, and secure data handling, the platform significantly enhances internal management processes. Moving forward, continuous upgrades and feature enhancements will ensure the system evolves with organizational demands, providing a reliable and impactful solution for employee administration.

BIBLIOGRAPHY

Websites and Online Resources

- React Documentation:
https://react.dev/

- Node.js & Express Documentation:
https://nodejs.org/
https://expressjs.com/

- MongoDB & Mongoose Documentation:
https://www.mongodb.com/docs/
https://mongoosejs.com/

- TailwindCSS Documentation:
https://tailwindcss.com/docs

- Vite Documentation:
https://vitejs.dev/

- JWT Authentication Guide:
https://jwt.io/

- Axios Documentation:
https://axios-http.com/

- SweetAlert2 Documentation:
https://sweetalert2.github.io/

- Other Employee Management Systems for understanding workflows and UI/UX patterns.