**Resource Management Application Functional Specification**

**Overview**

A web-based application for managing project resource allocation lifecycle based on SoWs (Statements of Work). The system supports full visibility across resource planning, allocation, project tracking, and fulfillment, enabling proactive bench management and governance.

**User Roles and Access (RBAC)**

* **Super Admin**: Full access; manage users and master data
* **RMT (Resource Management Team)**: Create/update SoW, projects, allocate/release resources
* **PM/Delivery Managers**: Request additions/extensions/releases; update status and lessons learned
* **Finance Controllers**: View financial metrics
* **Practice Heads/IT Solutions Leaders**: Approvals and oversight

**Master Pages (One-time Setup)**

**1. Location Master**

* Add/Edit/Delete locations

**2. Title Master**

* Define resource titles/roles

**3. Skillset Master**

* List skills mapped to resources

**4. Position Type Master**

* Define position types (dropdown values)

**5. Shift Timings Master**

* Configure shift names and times

**6. Allocation Percentage Master**

* Define allocation types (25%, 50%, 100%)

**7. Interview Panel Master**

* Add L1, L2, L3 panel members

**8. IT Verticals/Practice Master**

* Application Development, Data & Analytics, Infrastructure, Platforms

**9. Project Status Master**

* Define statuses (Proposed, In-Flight, Completed, On-Hold)

**10. Resource Master**

* Upload all existing resources and their skills (bulk and individual add/edit)

Each master page includes:

* Landing grid/table
* Add/Edit/Delete/View options
* Audit fields: Created by/date, Last updated by/date

**Transaction Pages**

**1. SoW Upload & Project Creation Page**

* Upload or enter SoW details:
  + Priority, Client Name, Project Name
  + Positions required: title, experience, skills, location, shift
* Store SoW document
* Create project linked to SoW
* Define required roles per project
* Audit capture

**2. Resource Allocation Page**

* View available bench resources by skill/experience/title
* Allocate resource to project-role
* Capture allocation % and timeline
* If no match found:
  + Mark as "To be fulfilled"
  + Raise external fulfillment requirement
* Audit capture

**3. Fulfillment Tracking Page**

* View open positions for fulfillment
* Track request status (open, in progress, fulfilled)
* Add notes and expected closure timeline

**4. Resource Release Page**

* Delivery Managers/PMs submit release request
* Includes reason, replacement (if any), effective date
* Goes to Practice Head/Leader for approval

**5. Resource Additional Request Page**

* Request additional resources mid-project
* Include required title, skill, duration, justification
* Goes to Leaders for approval

**6. Project Extension Request Page**

* Request for extending project and allocations
* Modify timelines and attach updated SoW (if needed)
* Trigger approval flow

**7. Project Status Update Page**

* PMs update status at regular intervals
  + Milestones, Deliverables, Progress %
  + Risks & Issues (log, track, mitigate)
* View-only access to all roles
* Editable by PM/Delivery Manager

**8. Lessons Learned Page**

* Post-project learnings
* Capture success/failure insights
* Searchable by keyword/project

**Reporting Pages**

**1. In-Flight Projects Report**

* Current active projects with summary status

**2. Proposed Projects Report**

* Projects in planning/SoW stage

**3. Spend Tracking Report**

* Planned vs. Actual by project/client/practice

**4. Project Status Report**

* Milestone completion, deliverables progress

**5. Risks & Issues Report**

* Open risks/issues with mitigation plans

**6. Resource Allocation Report**

* Resource utilization across projects
* Filter by skill, title, project, practice

**7. Bench Tracking Report**

* All unallocated or underutilized resources

**8. Forecasting Report**

* Future demand vs. availability by role/skill

**9. Financial Metrics Report**

* Budget burn, cost variance, revenue recognition

**10. Governance Report**

* SoWs, approvals, audit trails

**11. Portfolio Dashboard**

* Health view by client, region, practice

**12. Notifications & Alerts**

* Overdue milestones, tasks, budget alerts

**13. Change Requests Report**

* Scope change status, approval status, impact log

**14. Lessons Learned Repository**

* Keyword-searchable insights

**Integration Readiness**

* Placeholder connectors to integrate with:
  + ERP (SAP, Oracle)
  + CRM (Salesforce, Dynamics)
  + Time Tracking (Harvest, JIRA)

**Audit Logging (Applicable to All Pages)**

* Created By
* Created Date
* Last Updated By
* Last Updated Date

**Key Application Goals**

* Improve visibility and governance
* Enable decision-making on allocation and financials
* Proactively manage bench and reduce revenue leakage
* Provide real-time dashboards and reports
* Ensure compliance with SoWs and approvals

Please help me convert the above mentioned requirement into a full-stack project with the following stack:

Frontend: React (with modern hooks and component structure)

Backend: Node.js with Express.js

Database: PostgreSQL

Project Guidelines:

Frontend (React)

Each "screen" from the document should be a separate route/component.

Use functional components and React Router for page navigation.

Use TailwindCSS or Bootstrap for quick UI styling (optional).

Forms should match the UI controls described in the document (e.g. entity name fields, recurrence date pickers).

Add placeholder pages/screens where screenshot placeholders were mentioned.

Backend (Node.js + Express)

Set up RESTful APIs for each of the key functional components.

Use validation for required fields (e.g., entity name, dates).

Implement endpoints like: for example

POST /entities, GET /entities, PUT /entities/:id

POST /vendors, GET /vendors

POST /jobs, GET /jobs, etc.

Database (PostgreSQL)

Create seperate table for possible requirements with reference between tables.

Use Sequelize or Prisma ORM for easy model and migration management (or raw SQL queries if preferred).

Integration

Connect React frontend to the Node.js backend via Axios.

Backend should connect to the PostgreSQL DB using appropriate drivers (pg/Sequelize).

Extra Notes

Add a .env config for database connection and port management.

Keep folder structure clean (/client, /server, /db, etc.).

Seed some mock data for testing.