Task Management System

Application Document

Document Version: 1.0

Date: 23 March 2025

Author: Anshika Gupta

Table of Contents

1. Introduction

This full-stack **Task Management System** is built to simplify how users manage tasks, offering features like task creation, editing, filtering, undo actions, and role-based access. The app ensures business compliance by blocking task creation on public holidays and weekends.

It is designed with scalability, performance, and security in mind, following modern DevOps and engineering practices.

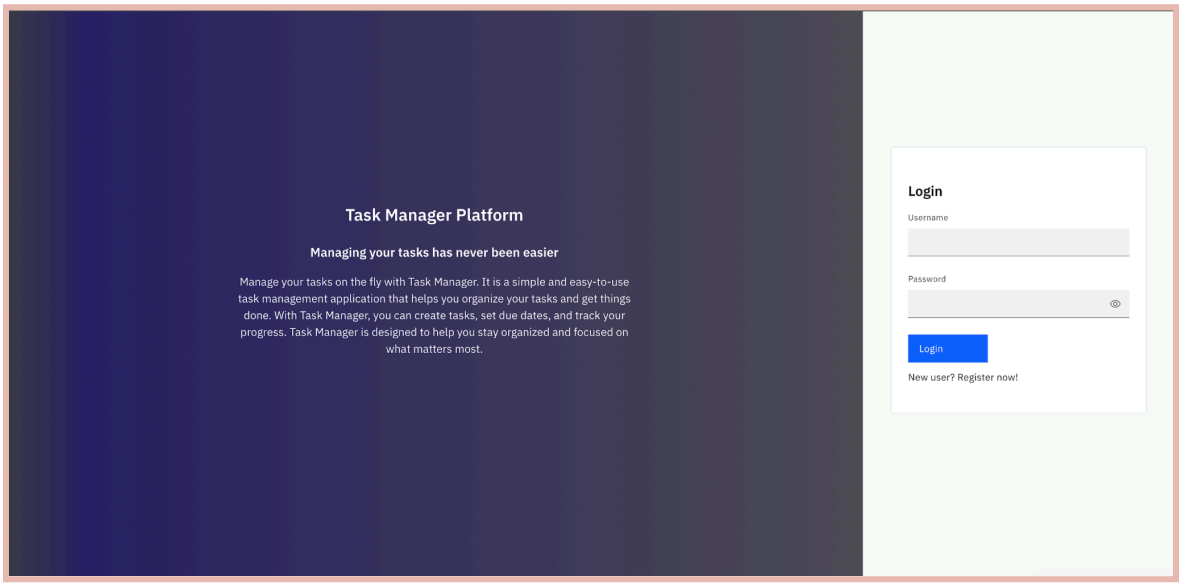
To streamline development and ensure high code quality, the project adopts a **monorepo structure** housing both frontend and backend. **GitHub** is used for version control, with **CodeRabbit AI** (trial mode) integrated to assist in PR reviews. **Pre-commit hooks** ensure code quality checks before submission, enforced by **ESLint** and **Prettier** for formatting and linting. The entire codebase is written in **TypeScript**, promoting strong typing and maintainability throughout the system.

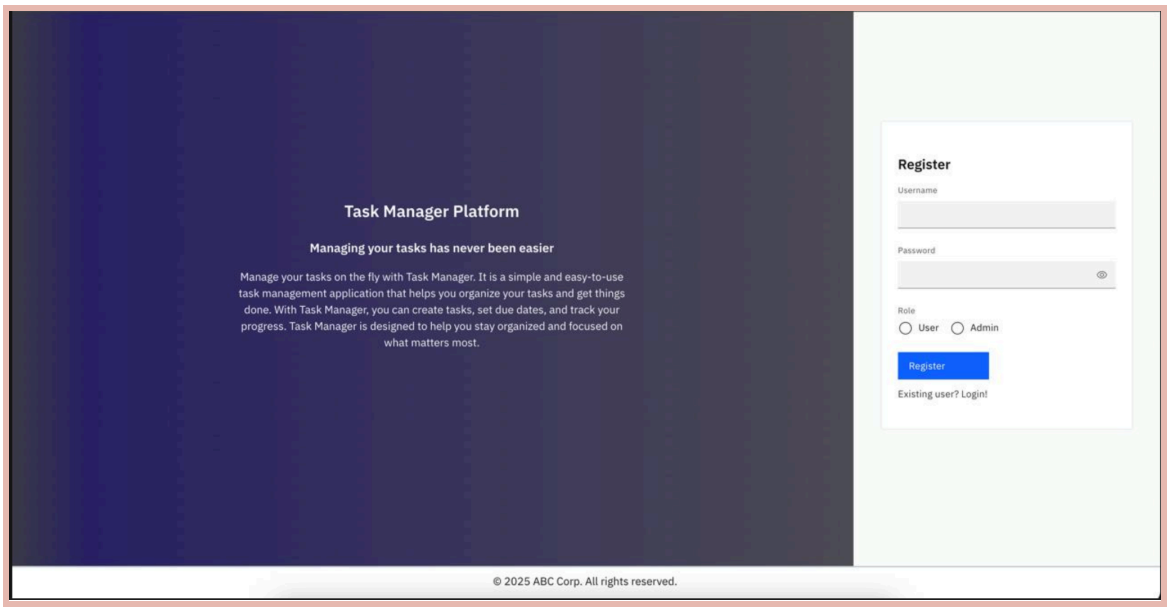
The system includes:

Frontend: Angular 19 using standalone APIs and Carbon Design System.
Backend: Node.js with Express, PostgreSQL, Sequelize ORM.
CI/CD & Deployment: Docker, CircleCI, GitHub Actions.
Authentication: JWT-based login and registration.

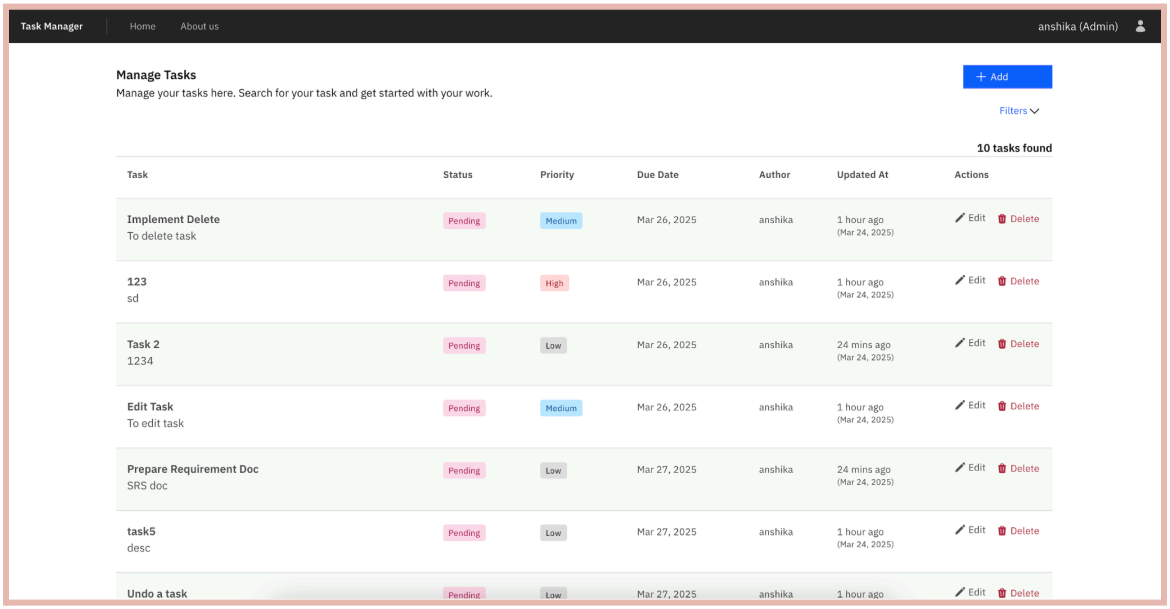
2. Application View

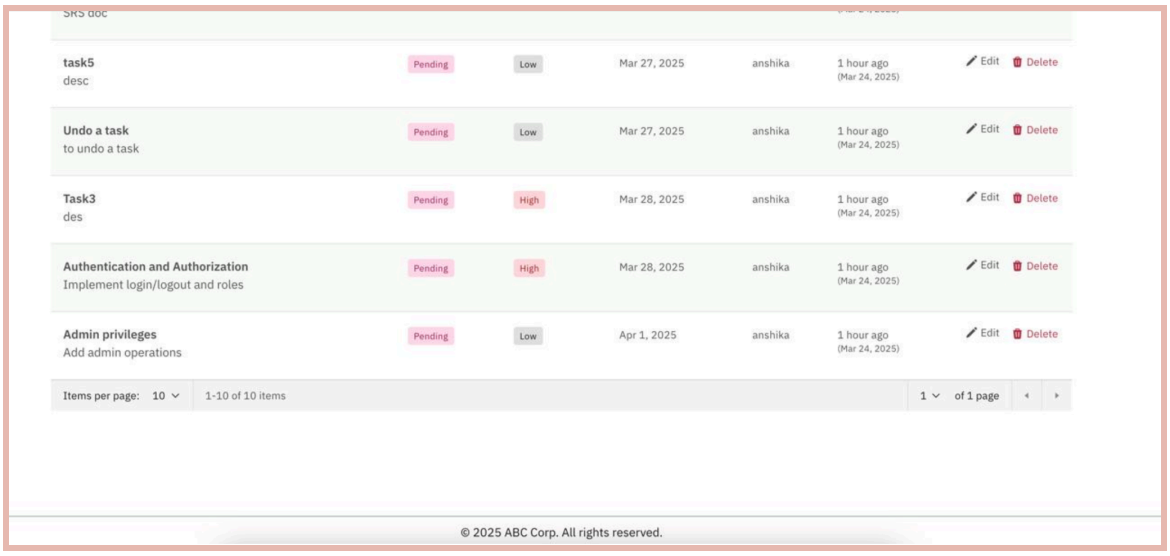
2.1 Login/Register – Secure user authentication using JWT.

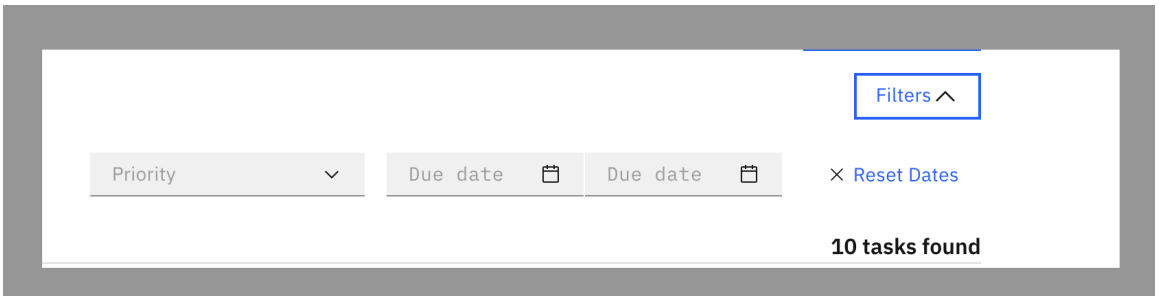




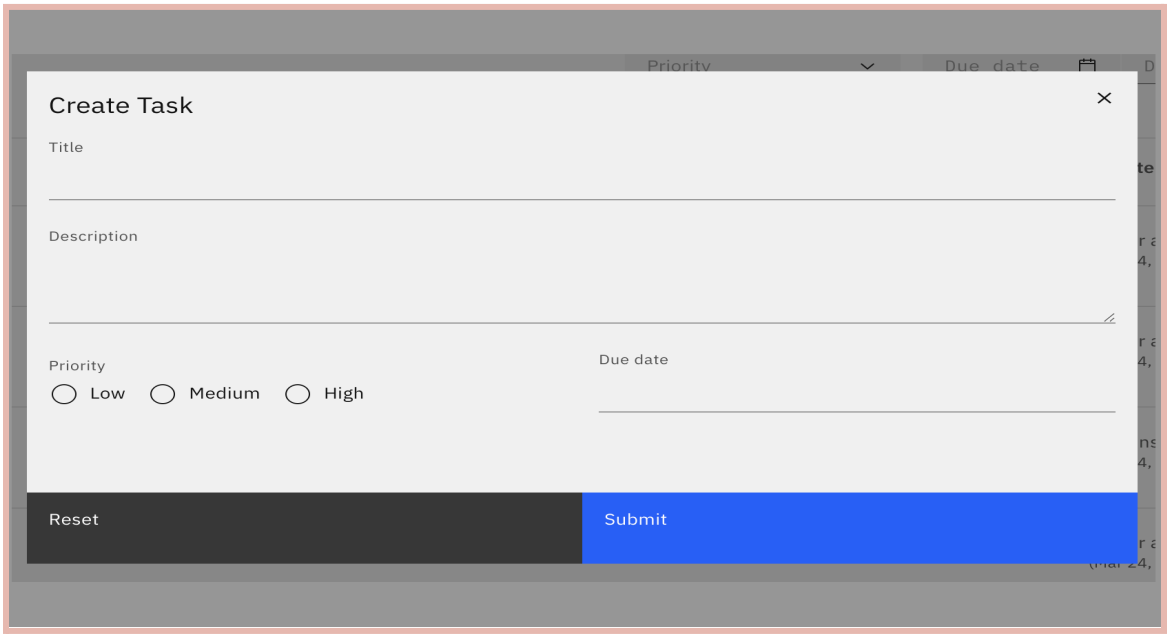
2.3 Task List Page – Table/grid to view all tasks with filters.

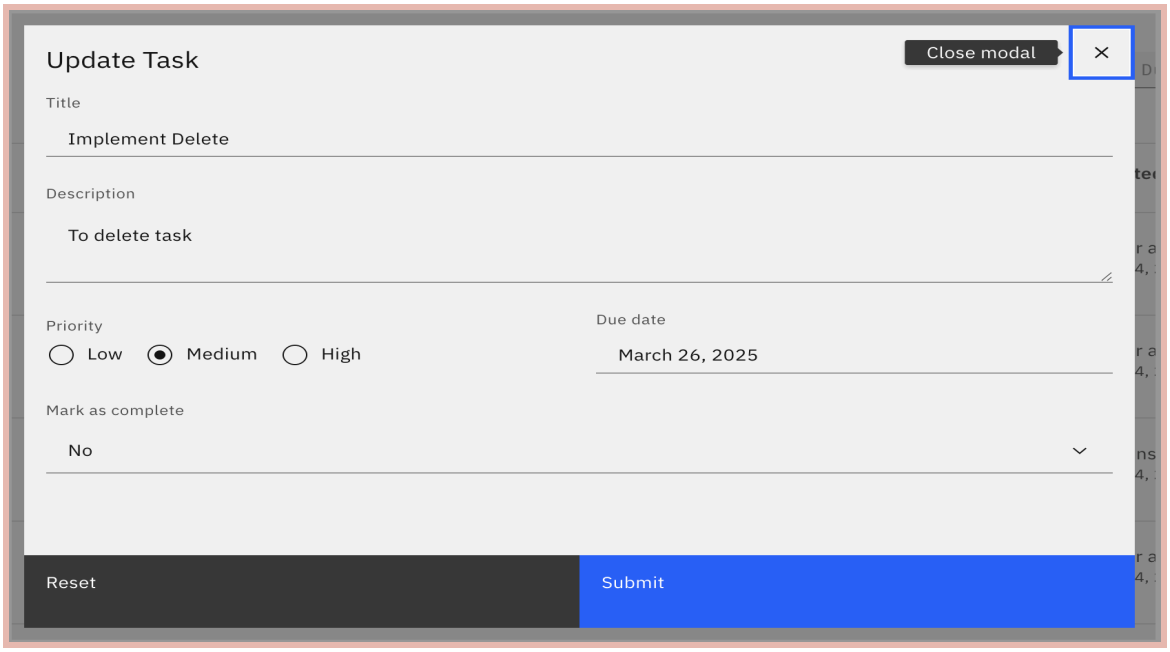




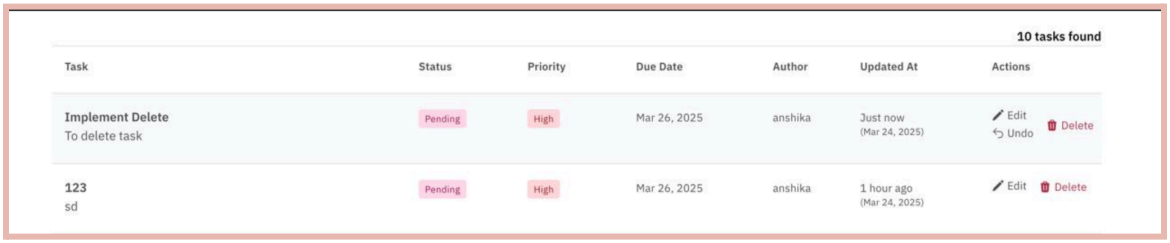


2.4 Task Form Page – Create/Edit tasks with validation.





2.5 Undo Feature – Shows “Undo” button within 5 minutes of the last update.



3. Success User Stories

I	User Story	Benefit
1	As a user, I want to create tasks.	Manage personal workload
2	As a user, I want to filter by priority/due date.	Efficient task search
3	As a user, I want to undo a task update.	Revert accidental changes
4	As an admin, I want to delete tasks.	Manage global task visibility
5	As a system, I want to restrict task creation on holidays/weekends.	Business logic compliance

4. Acceptance Criteria

Feature	Criteria Fulfilled
JWT Authentication	Token-based login/register with secure storage.
Task CRUD	All tasks can be created, viewed, updated, and deleted.
Public Holiday Validation	Validated using metadata API. disabled the dates on the frontend date picker
Undo Functionality	Available within 5 minutes post-update using task-history tal
Admin Role Privileges	Delete access restricted to admin. (soft deletion pending row is getting deleted as of now)

5. Architecture Diagram & Description

The Task Management System follows a modern full-stack architecture designed for scalability, modularity, and performance. The system is divided into three major layers: Frontend, Backend API, and Database, with DevOps support through Docker and CI/CD pipelines using CircleCI.

Components:

1. **Angular 19 Frontend**:
   The user interface is developed using Angular 19's standalone component architecture and MobX for state management. It includes reusable UI components built with Carbon Design System and TailwindCSS for responsive design.

2. **Node.js + Express Backend**:
   The backend API is built using Express.js. It handles business logic, routing, authentication (JWT), RBAC, and validation. API routes are RESTful and documented using Swagger/OpenAPI.

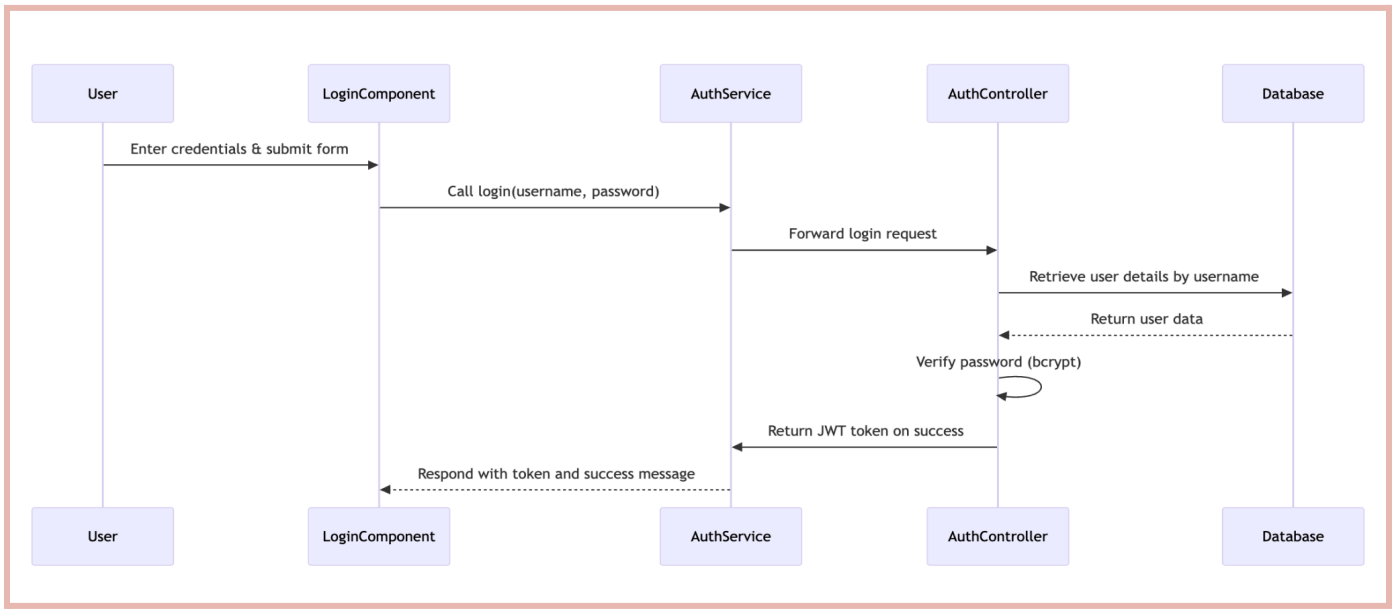
3. **PostgreSQL Database**:
   A relational database stores persistent data for tasks, users, roles, public holidays, and task history (for undo functionality). Task-related queries use indexed fields for performance.

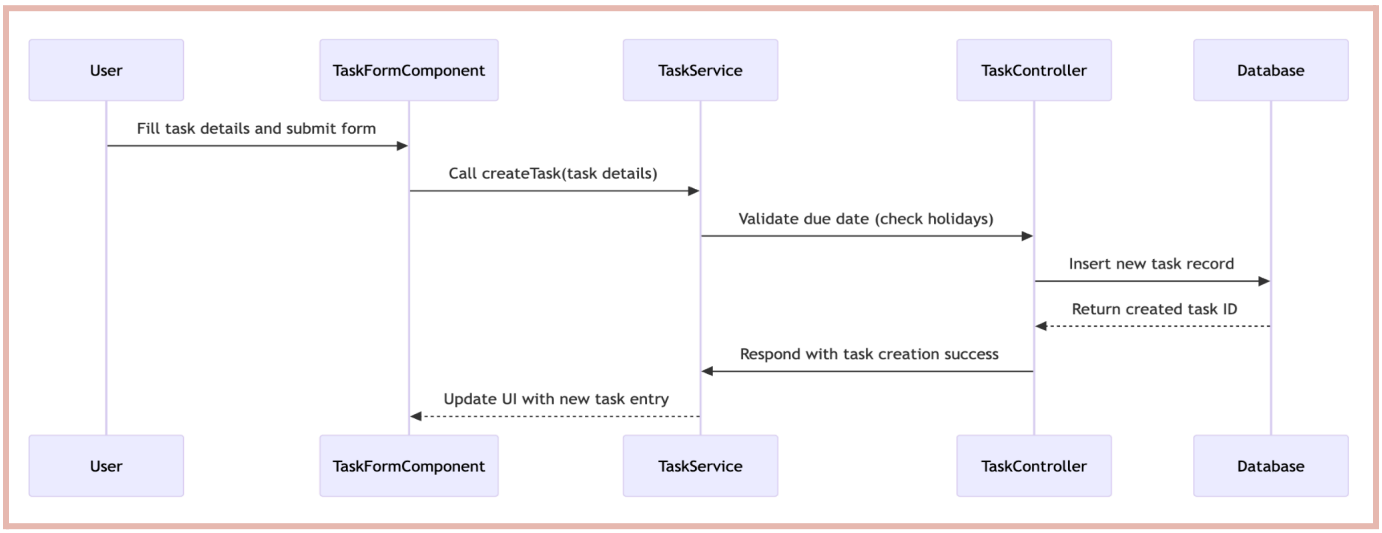
4. **Sequelize ORM**:
   Sequelize provides a model-based abstraction for DB interactions, including associations,

soft deletes, and validations. It helps enforce schema consistency and secure queries.

5. **Docker & CircleCI**:
   Both frontend and backend are containerized using Docker for consistent development and deployment. CircleCI automates testing, building, and deployment pipelines

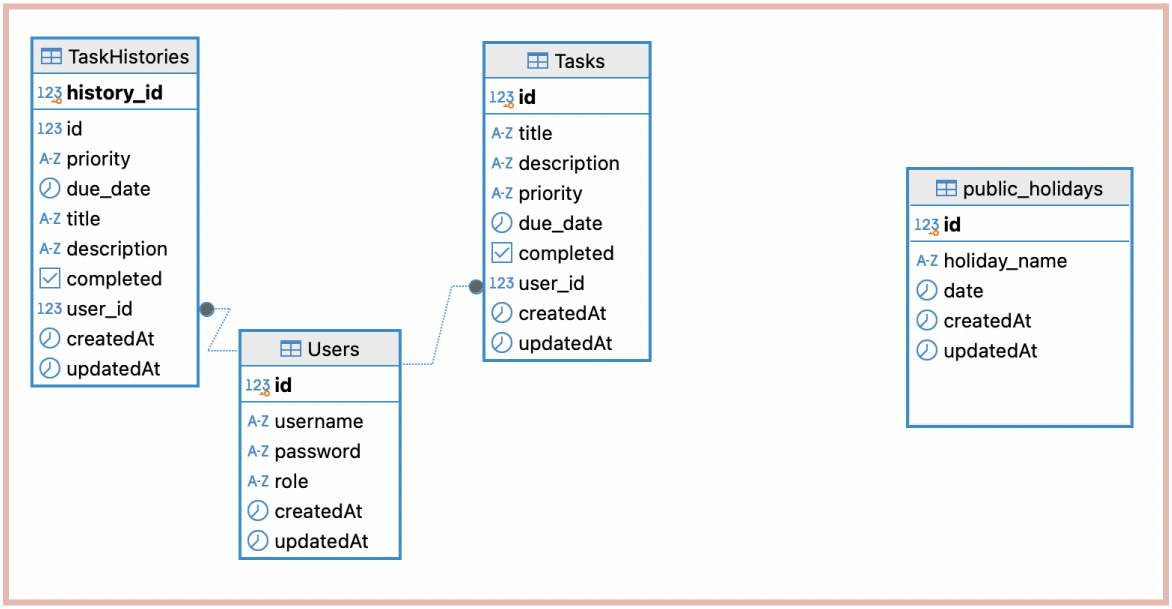
Flow:





6. Database Schemas and Optimization Methods

The above architecture and backend application will create



Tables:

- **Users**: Auth and role info

```
-- public."Users" definition

-- Drop table

-- DROP TABLE public."Users";

CREATE TABLE public."Users" (
    id serial4 NOT NULL,
    username varchar(255) NOT NULL,
    "password" varchar(255) NOT NULL,
    "role" varchar(255) DEFAULT 'user'::character varying NOT NULL,
    "createdAt" timestamptz NOT NULL,
    "updatedAt" timestamptz NOT NULL,
    CONSTRAINT "Users_pkey" PRIMARY KEY (id),
    CONSTRAINT "Users_username_key" UNIQUE (username)
);
```

- **Tasks**: Task info with indexing on priority, due_date

```sql
-- public."Tasks" definition

-- Drop table

-- DROP TABLE public."Tasks";

CREATE TABLE public."Tasks" (
    id serial4 NOT NULL,
    title varchar(255) NOT NULL,
    description text NULL,
    priority public."enum_Tasks_priority" NOT NULL,
    due_date timestamptz NOT NULL,
    completed bool DEFAULT false NULL,
    user_id int4 NOT NULL,
    "createdAt" timestamptz NOT NULL,
    "updatedAt" timestamptz NOT NULL,
    CONSTRAINT "Tasks_pkey" PRIMARY KEY (id)
);
CREATE INDEX idx_task_due_date ON public."Tasks" USING btree (due_date);
CREATE INDEX idx_task_priority ON public."Tasks" USING btree (priority);
CREATE INDEX idx_task_priority_due_date ON public."Tasks" USING btree (priority, due_date);


-- public."Tasks" foreign keys

ALTER TABLE public."Tasks" ADD CONSTRAINT "Tasks_user_id_fkey" FOREIGN KEY (user_id) REFERENCES public."Users"(id) ON UPDATE CASCADE;
```

- **TaskHistory**: Records previous state for undo

```sql
-- public."TaskHistories" definition

-- Drop table

-- DROP TABLE public."TaskHistories";

CREATE TABLE public."TaskHistories" (
    history_id serial4 NOT NULL,
    id int4 NOT NULL,
    priority public."enum_TaskHistories_priority" NOT NULL,
    due_date timestamptz NOT NULL,
    title varchar(255) NOT NULL,
    description text NULL,
    completed bool DEFAULT false NULL,
    user_id int4 NOT NULL,
    "createdAt" timestamptz NOT NULL,
    "updatedAt" timestamptz NOT NULL,
    CONSTRAINT "TaskHistories_pkey" PRIMARY KEY (history_id)
);


-- public."TaskHistories" foreign keys

ALTER TABLE public."TaskHistories" ADD CONSTRAINT "TaskHistories_user_id_fkey" FOREIGN KEY (user_id) REFERENCES public."Users"(id) ON UPDATE CASCA
```

- **PublicHolidays**: Caches upcoming holidays

```sql
-- public.public_holidays definition

-- Drop table

-- DROP TABLE public.public_holidays;

CREATE TABLE public.public_holidays (
    id serial4 NOT NULL,
    holiday_name varchar(255) NOT NULL,
    "date" date NOT NULL,
    "createdAt" timestamptz NOT NULL,
    "updatedAt" timestamptz NOT NULL,
    CONSTRAINT public_holidays_date_key UNIQUE (date),
    CONSTRAINT public_holidays_holiday_name_key UNIQUE (holiday_name),
    CONSTRAINT public_holidays_pkey PRIMARY KEY (id)
);
```

Optimization:

- Composite indexes for frequent query patterns
- Task history cleanup via cron (records older than 20 mins)
- Enum constraints for role/priority
- Normalized schema for scalability

7. Backend API Servers

Stack: Node.js + Express
Features:

- JWT Auth Middleware
- RBAC Middleware
- Swagger setup at /api-docs
- Global error handling with ApiError middleware
- Logging via custom LoggerService
- CRON job to delete the task_history table content for data maintenance purging (not done in progress)

Undo feature explanation:

1. **Task Update Trigger:**
    - When a user updates a task, the previous version of that task is saved in the task_history table before the update is applied.
2. **TaskHistory Table Schema:**
    - Mirrors the structure of the tasks table.
    - Includes createdAt and updatedAt timestamps.
    - Stores the last state of a task for rollback purposes.
3. **Undo Endpoint Logic:**
    - API: PUT /api/tasks/:id/undo
    - Authenticated users can trigger undo only for tasks they own (or admins).
    - On API hit:
        - Fetch the latest record from task_history for the specified task ID and user.
        - Check if the createdAt timestamp is less than 5 minutes old.
        - If valid, restore the task by updating the tasks table with that backup.
        - Delete the used history entry after undo to prevent repeat restores.
4. **Show Undo Button (UI Logic):**
    - When listing tasks, the backend also checks if a corresponding history record exists for each task within the last 5 minutes.
    - A boolean showUndoButton: true/false is returned for each task in the API response
    - The frontend conditionally renders the Undo button using this flag.
5. **Validation Rules**
    - Only one-level undo is allowed.
    - Undo is only available within 5 minutes of the last update.
    - Only the owner or admin can undo a task.
    - If the task has no recent history record, undo is not permitted.

8. Frontend Web API

Stack: Angular 19 (standalone APIs) + Carbon Design + tailwind
State Management: MobX
Highlights:

1. Dynamic rendering based on user role
2. The undo button rendered only if the undo history exists
3. Filtering & pagination are handled via reactive forms
4. Flatpickr is used for date input with validation

9. Security Considerations

Implemented:

- **JWT Authentication** – Users sign in via **JWT tokens** to access APIs.
- **Role-Based Access Control (RBAC)** – Admins/validators can delete, users cannot.
- **Database Indexing** – Speeds up querying and prevents performance bottlenecks.
- **Data Encryption** – Passwords are stored using **bcrypt**.
- **Helmet.js** for setting secure HTTP headers.
- Environment variables are managed via .env for secrets like JWT_SECRET..

Pending:

- **Soft Deletes for Audit** – Deleted tasks remain stored but hidden.
- **Input Validation** – **Zod/Yup validation** to prevent bad inputs.
- **Rate Limiting –** Express Rate Limit to prevent API abuse.

10. Performance Optimizations

Completed:

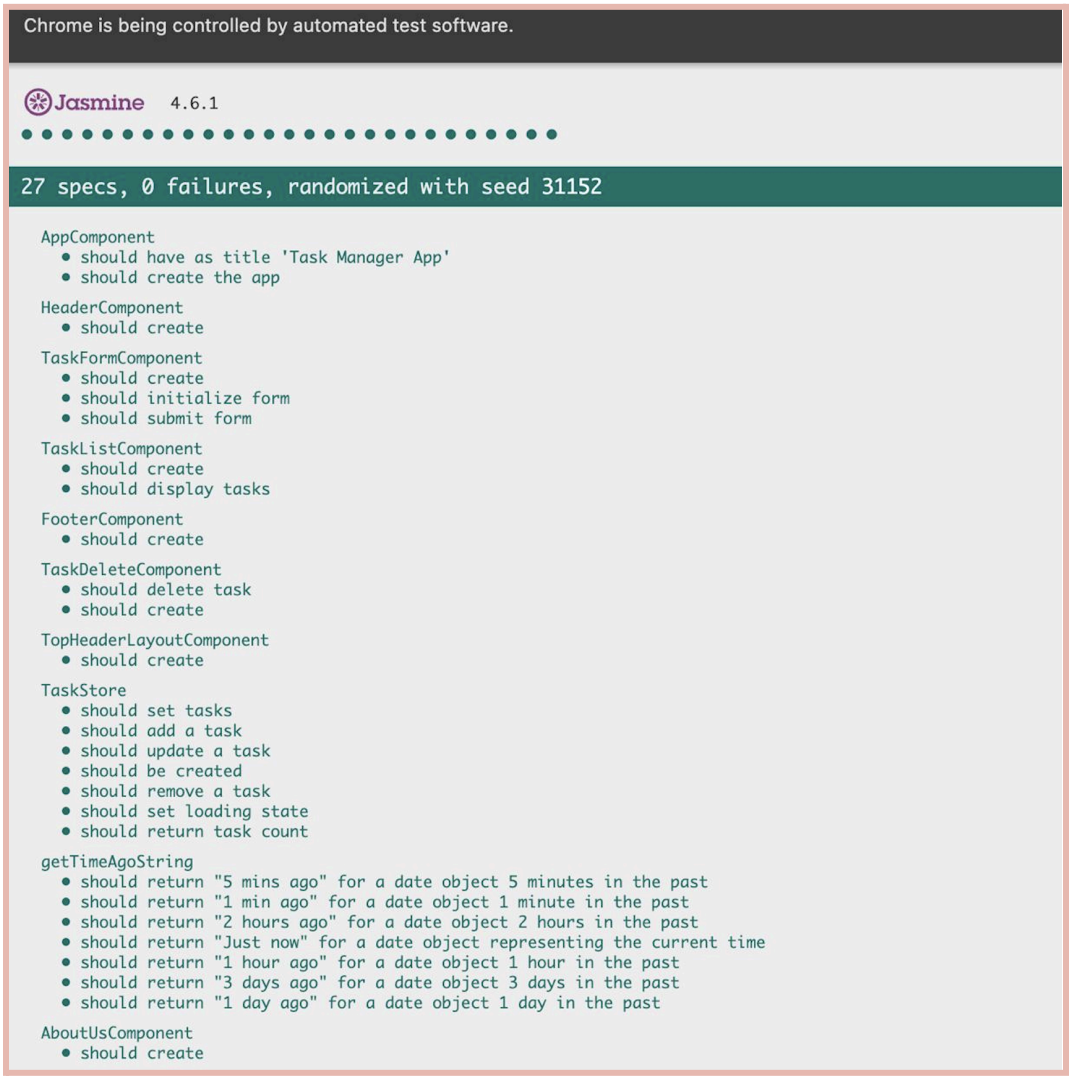
- Indexed DB queries for fast database access.
- Pagination on task APIs to limit large responses.
- **Public holiday data is cached in a DB table** to avoid external API calls.
- Frontend responsiveness: Used CDS and tailwind.
- Angular lazy loading and module chunking to reduce load time.
- Optimized filtering using Sequelize query conditions.

11. Scalability

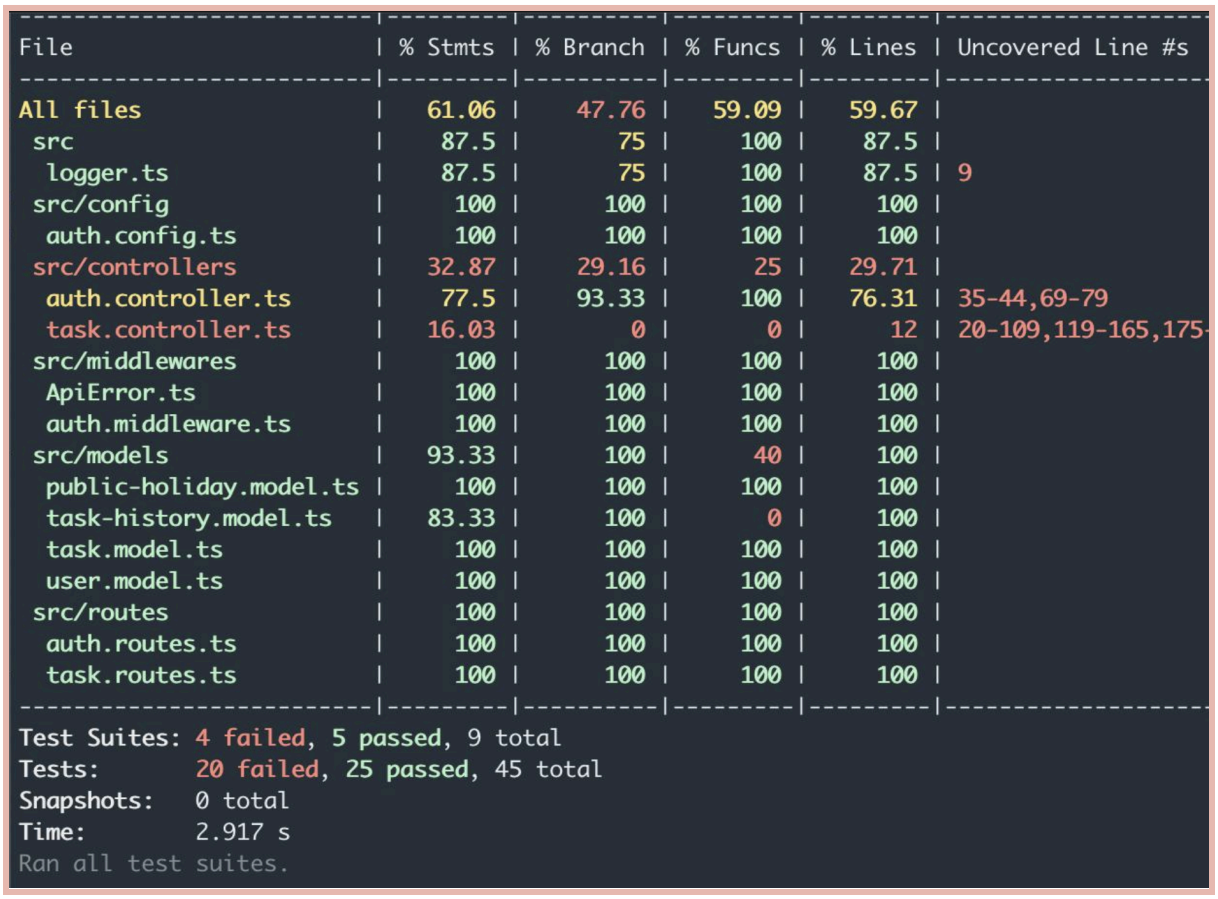
- Modular, component-based Angular 19 frontend.
- API-first design on the backend (Node.js + Express).
- Dockerized app for containerized deployment.
- Supports horizontal scaling via Docker Compose / Kubernetes.
- Redis cache implementation to cache recurring apis
- Grafna for monitoring the app

12. Testing Strategies & Coverage

- **Frontend:** Tool(Jasmine, Karma), Purpose (Component testing)

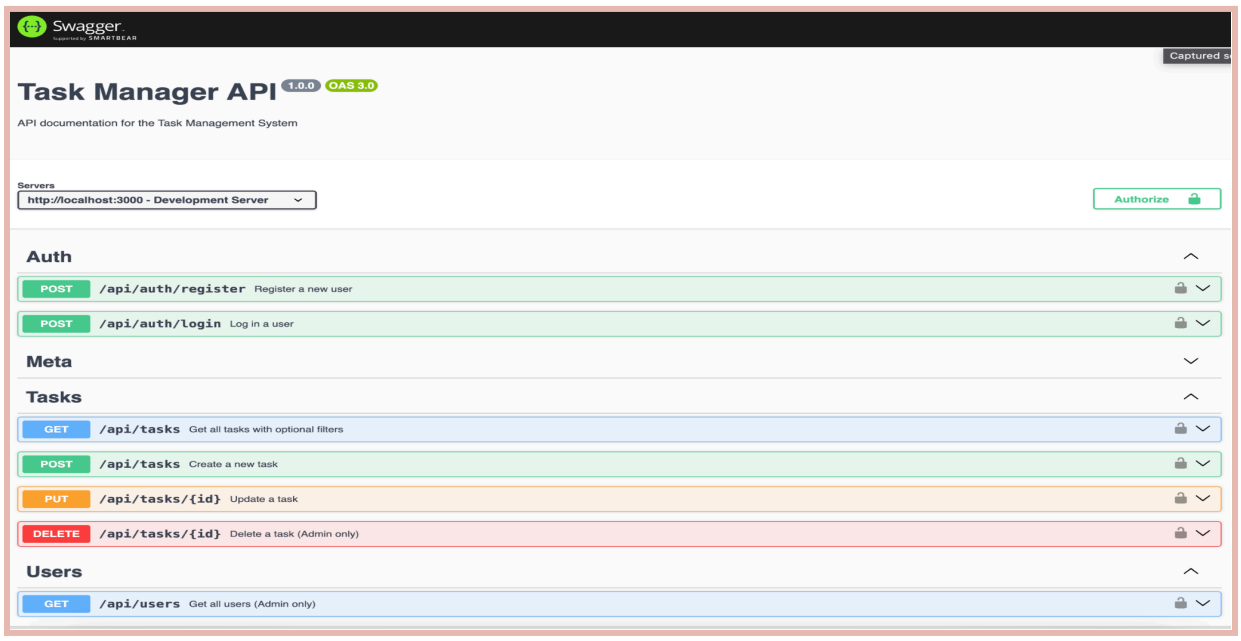


- **Backend:** Tool(Jest) Purpose: (Unit and API tests)

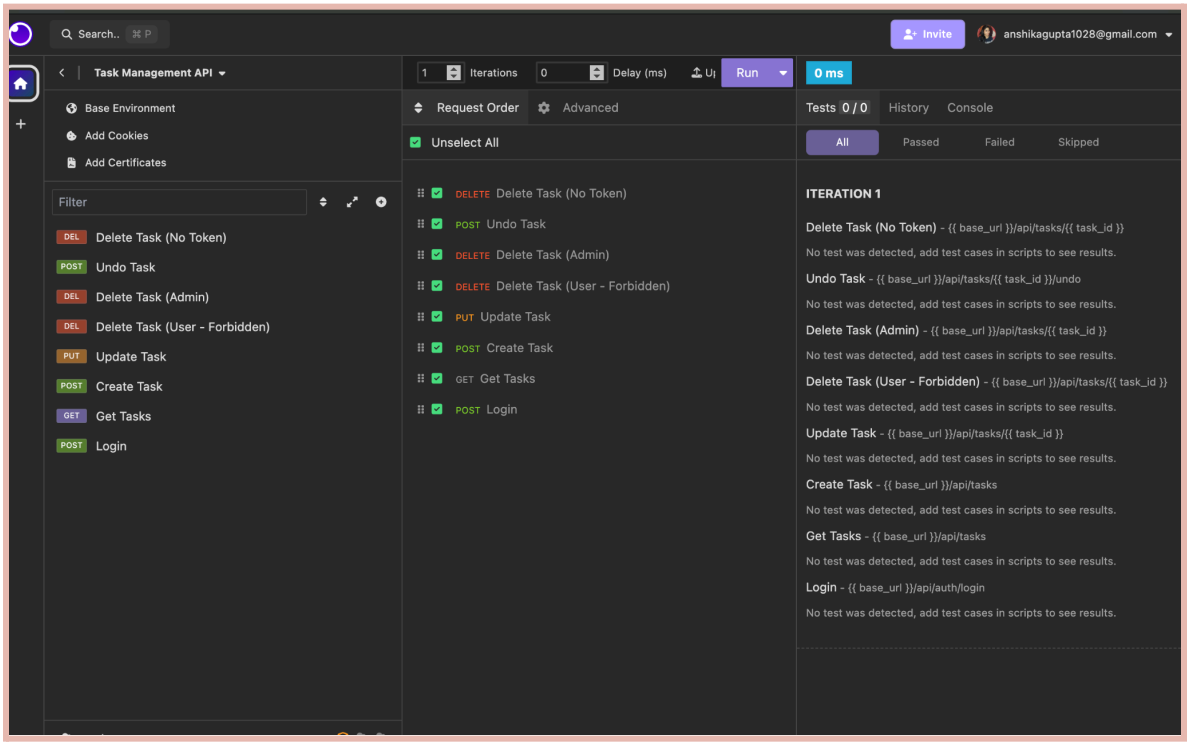


```
---------------------------|---------|----------|---------|---------|-------------------
File                       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------------|---------|----------|---------|---------|-------------------
All files                  |   61.06 |    47.76 |   59.09 |   59.67 |
 src                       |    87.5 |       75 |     100 |    87.5 |
  logger.ts                |    87.5 |       75 |     100 |    87.5 | 9
 src/config                |     100 |      100 |     100 |     100 |
  auth.config.ts           |     100 |      100 |     100 |     100 |
 src/controllers           |   32.87 |    29.16 |      25 |   29.71 |
  auth.controller.ts       |    77.5 |    93.33 |     100 |   76.31 | 35-44,69-79
  task.controller.ts       |   16.03 |        0 |       0 |      12 | 20-109,119-165,175-
 src/middlewares           |     100 |      100 |     100 |     100 |
  ApiError.ts              |     100 |      100 |     100 |     100 |
  auth.middleware.ts       |     100 |      100 |     100 |     100 |
 src/models                |   93.33 |      100 |      40 |     100 |
  public-holiday.model.ts  |     100 |      100 |     100 |     100 |
  task-history.model.ts    |   83.33 |      100 |       0 |     100 |
  task.model.ts            |     100 |      100 |     100 |     100 |
  user.model.ts            |     100 |      100 |     100 |     100 |
 src/routes                |     100 |      100 |     100 |     100 |
  auth.routes.ts           |     100 |      100 |     100 |     100 |
  task.routes.ts           |     100 |      100 |     100 |     100 |
---------------------------|---------|----------|---------|---------|-------------------
Test Suites: 4 failed, 5 passed, 9 total
Tests:       20 failed, 25 passed, 45 total
Snapshots:   0 total
Time:        2.917 s
Ran all test suites.
```

- **Swagger: Tool(**OpenAPI Validator), purpose(Contract compliance validation)



- **E2E: Tool(**insomnia), purpose(Scenario & workflow testing) (not done just created the config this task is in progress need to write more test cases)



13. Deployment Guide

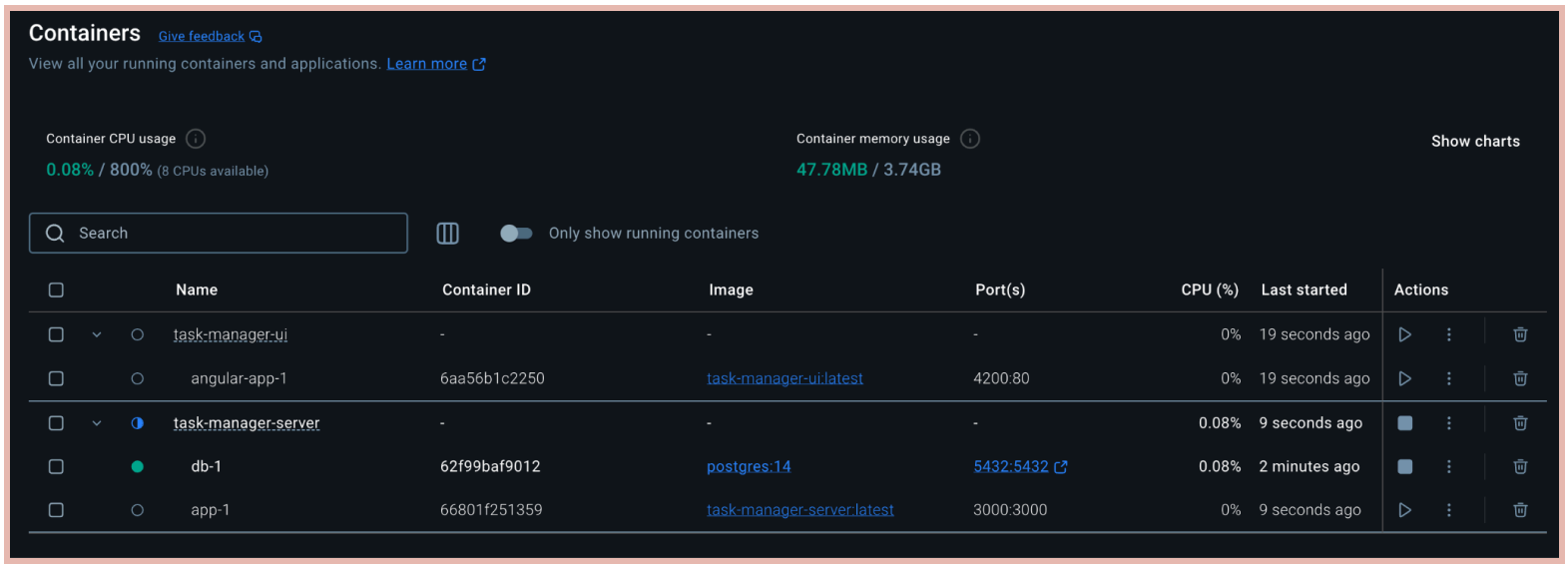
Steps:

1. Clone repo and install dependencies

2. Create .env with secrets (DB, JWT, etc.)

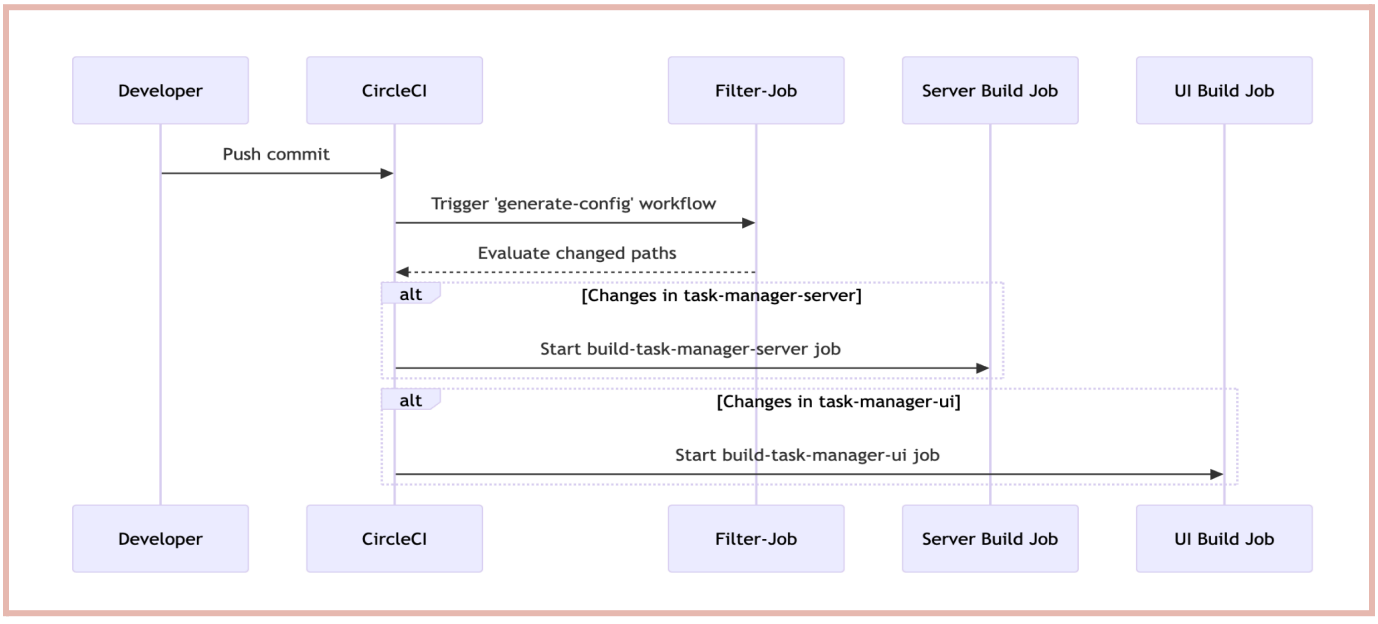
   server repo instructions are here on this link : Server README.md
   UI repo instructions are here on this link : UI README.md

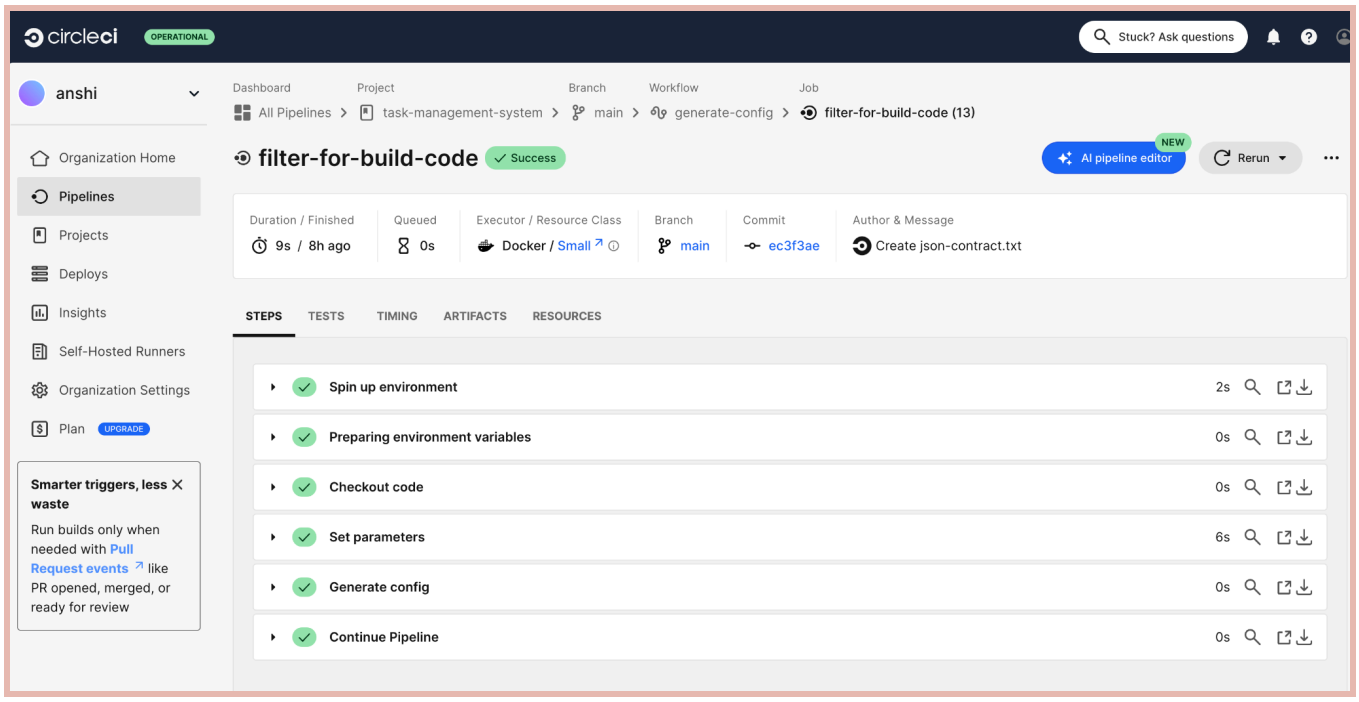
3. Build Docker images go to root directories of server and ui folder and run (
   step 1: docker build -t task-manager-server .
   step 2: docker-compose up
)
4. Access API at localhost:3000, UI at localhost:4200



CI/CD:

- CircleCI for automated tests & Docker builds
- GitHub Actions for alternative workflows
- Deployment ready for Vercel or AWS EC2





14. Known issues

1. Improve validation of task model via Zod.
2. Soft delete not done: table model change (Squelize model paranoid:true) yet
3. code cleanup (partially pending)
4. Improve Swagger API documentation in backend application (docs comments for routes need to modified)
5. CI/CD: Improvements for continuous integration and deployment
6. Add a feedback/confirmation dialog for users before performing any sensitive(write/update) operations
7. Localization and centralization of all the text messages (upload all messages and labels in the const label file in FE , BE to send message codes)

15. Future Enhancements

1. Calendar Sync (Google/Outlook):

Google Calendar Sync (Planned Feature)

To enhance task visibility and time management, a **Google Calendar Sync** feature will be integrated in upcoming releases. This functionality will allow users to automatically sync their tasks with their personal Google Calendars.

- OAuth2 authentication will be used to securely connect user Google accounts.
- Upon task creation or update, events will be automatically inserted or modified in the user's primary calendar.
- Tasks will be synced with details such as title, description, and due date as start/end time.
- A “Connect Google Calendar” button will be available in the UI for users to authorize sync.
- Users can optionally disable or revoke calendar access at any time.
- Events will include links to view the task directly within the application.
- Access tokens will be securely stored and encrypted, with refresh tokens handled for long-lived sessions.

Calendar Import (Planned Feature):
https://www.npmjs.com/package/ical-generator

To enhance productivity and task visibility, the system will support calendar synchronization using the ical-generator package. Each user will be able to generate a secure .ics feed of their tasks, which can be subscribed to in Google Calendar, Outlook, or Apple Calendar. The system will expose a secured API endpoint to serve calendar events, enabling real-time sync of task due dates with external calendar tools.

2. Allow users to share tasks with other people.

The Task Sharing feature enables users to share their tasks with other

registered users, allowing them read-only access. This promotes collaboration while ensuring data integrity and ownership.

📌 Key Highlights:
New Table: task_shared_users created to store task-user sharing relationships.

Permissions:

Task Owner: Full access (edit/delete).

Shared Users: Read-only access.

Backend Changes:

Updated GET /api/tasks to include tasks shared with the logged-in user.

Only owners can update/delete tasks.

Frontend Changes:

Multi-select dropdown in Task Form to choose users to share with.

Tasks shared with the user are tagged accordingly and are read-only.

Security:

Role checks in backend ensure only the owner can modify tasks.

Optional Notification:

Future enhancement to notify shared users via email or toast.

3. Audit Logs for task history
4. Internalization/Localisation of all text label
5. Auth: User HTTPS cookie only token persistence; Implement refresh token policy