# Task Management Application – Project Documentation

## Project Overview

This project is a **Task Management Application** inspired by tools like Jira/Trello.
It enables users to **register, log in, and manage their personal tasks** in an organized manner.
Every task is **securely linked to its owner (user)**, ensuring accountability and personalization.

The application follows a **microservice-style modular design** with two main services:

- **User Service** → handles user registration, authentication, and session management.
- **Task Service** → manages the full lifecycle of tasks (create, update, delete, list).

Both services are exposed as **REST APIs** and are also integrated into a **frontend** for ease of use.

## Technologies Used

- **Backend:** Spring Boot (Spring MVC, Spring Data JPA, Hibernate, Validation)
- **Frontend:** Thymeleaf templates, HTML
- **Database:** MySQL (depending on setup)
- **Build Tool:** Maven
- **Server:** Embedded Tomcat (default in Spring Boot)
- **Language:** Java 17+

## MVC File Explanation

### Model

- User.java – Entity class for user details (id, username, email, password).
- Task.java – Entity class for task details (id, title, description, dueDate, status, user).

### View

- register.html – Form for new user registration.
- login.html – User login page.
- welcome.html – Main dashboard where tasks are listed and managed.
- Thymeleaf is used for binding model data and displaying validation errors.

### Controller

- UserController.java
    - Handles registration, login, logout.
- TaskController.java
    - Handles task CRUD operations (save, update, delete, list).

### Service Layer

- UserService.java & TaskService.java
  - Business logic, interaction with repositories.

Repository Layer

- UserRepository.java & TaskRepository.java
  - Extends Jpa Repository for database CRUD operations.

## Architecture & Flow

### Components

- User Service → Registers users, manages login, handles session and validation.
- Task Service → Provides CRUD operations for tasks, ensuring each task is linked to the correct user.
- Database → Stores users and their tasks with referential integrity.
- Frontend (Thymeleaf UI) → Provides a responsive interface for non-technical users.
- REST API Layer → Enables Postman testing, third-party integrations, and future frontend extensions (React, Angular, etc.).

### Flow of Operations

1. Registration/Login → User creates an account or logs in.
2. Session Management → After successful login, a session is created to identify the user.
3. Task Operations →
   - Create: User adds a new task.
   - Update: User modifies existing task details.
   - Delete: User removes a task.
   - View: User lists all tasks belonging to them.
4. Data Persistence → Tasks and users are stored in the relational database.
5. API & UI Access →
   - UI: Simple responsive Thymeleaf pages.
   - API: Exposed via REST endpoints for testing in Postman and future frontend apps.

## Database Details

### User Table

| Column | Type | Constraints |
|---|---|---|
| Id | Long (PK) | Auto-generated |
| Username | String | Unique, Not Null |
| Email | String | Valid Email |
| Password | String | Not Null |

Task Table :

| Column | Type | Constraints |
|---|---|---|
| Id | Long (PK) | Auto-generated |
| Title | String | Not Blank |
| Description | String | Optional |
| due_date | Date | Not Null |
| Status | Enum | PENDING / COMPLETED |
| user_id | FK | References User(id) |

## Project Workflow

1. User Registration
    - The user navigates to the /register page.
    - They provide **username, email, and password**.
    - Validation ensures:
        - Username is unique.
        - Email is valid and not blank.
        - Password is not blank.
    - On successful submission, the user's details are stored in the **User table** in the database.
    - If validation fails, error messages are shown on the same page.
2. User Login
    - The user navigates to the /login page.
    - They enter their registered **username and password**.
    - If credentials match a user in the database, their userId is stored in the **HTTP session**.
    - If invalid, an error message ("Invalid credentials") is displayed, and the login page reloads.
3. Welcome Page (Dashboard)
    - After successful login, the user is redirected to /welcome.
    - The system retrieves all tasks associated with the logged-in user (userId) and displays them in a table.
    - Features on this page:
        - **Add Task:**
            - User fills out title, description, due date, and status in the form.

        - Validation ensures:
            - Title is not blank.
            - Due date is required and cannot be in the past.
        - If validation fails → error shown under the input field.
        - If valid → task is saved in the DB and linked to the logged-in user.
    - Edit Task:
        - Each task row has an **Edit** button.
        - When clicked, the task details are pre-filled into the form.
        - User can update title, description, due date, or status.
        - On save, changes are updated in the database.

Delete Task:

Each task row has a **Delete** button.

On click, the task ID is sent to the backend.

The system verifies that the task belongs to the logged-in user (by matching userId).

If valid → task is deleted from the DB.

If not → action is ignored (prevents deleting others' tasks).

Access Control

Only tasks linked to the current user are fetched from the database and displayed.

Even if someone tries to tamper with a task ID in the URL, the backend checks that the task's userId matches the logged-in user before allowing update/delete.

Logout

When the user clicks the **Logout** button, the system invalidates the session.

This ensures no user information is kept in memory.

The user is redirected back to /login.

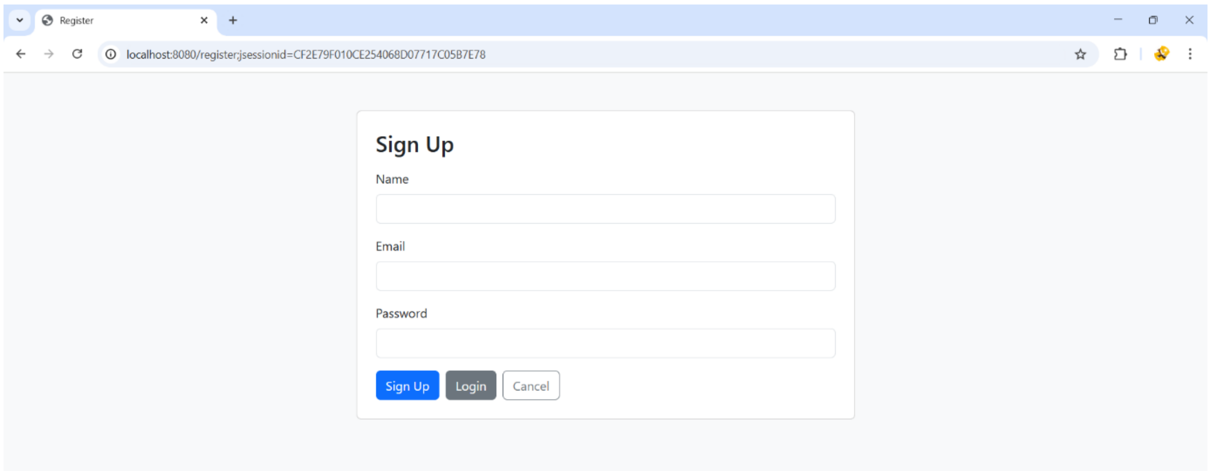
## Features Implemented

- User Management
    - Register new users via **UI form** and **REST API**.
    - Username validation ensures no duplicates.
    - Secure login/logout with session tracking.
    - Validation for empty/invalid fields (UI + API).
- Task Management

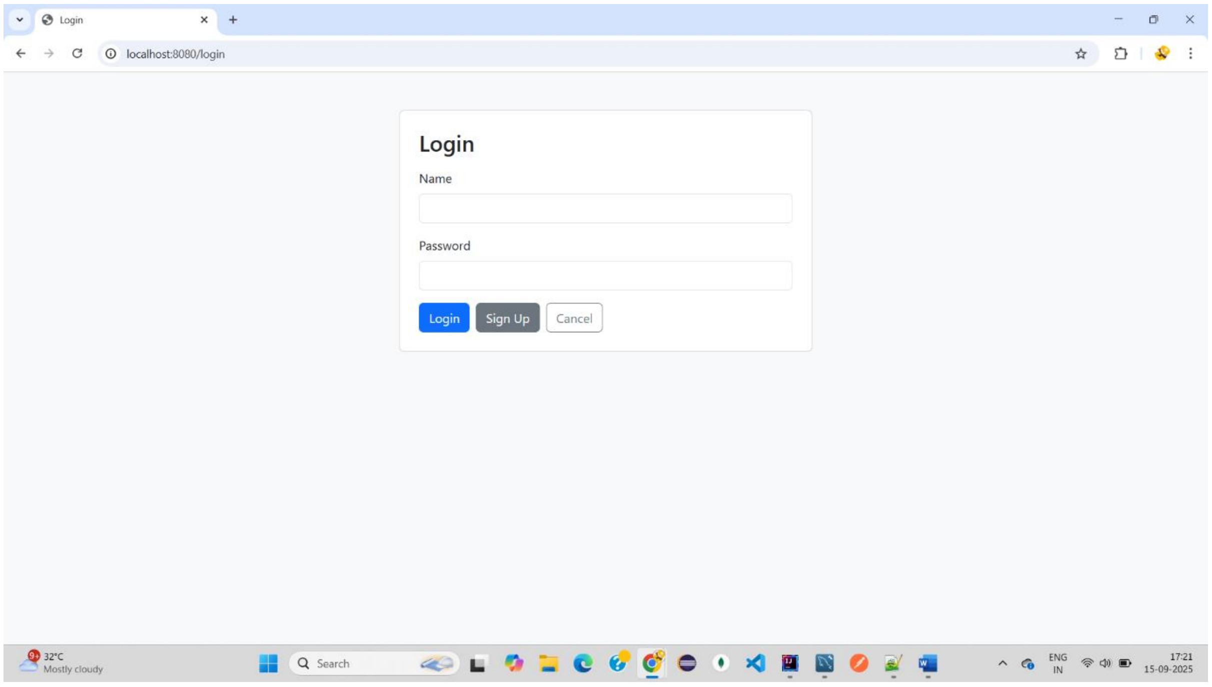
    - **Create Task** → add title, description, due date, and status.
    - **Update Task** → edit task details while preserving ownership.
    - **Delete Task** → remove tasks (soft restriction: only owner can delete).
    - **View Tasks** → list all tasks sorted by due date.
    - **Validation** → ensures all mandatory fields are filled.
- REST API Endpoints
    - User API (/api/users) → create user, fetch users, login via API.
    - Task API (/api/tasks) → CRUD tasks with ownership check.
    - Supports Postman testing for independent verification.
- Frontend (Thymeleaf UI)
    - Simple, responsive web pages for non-technical users.
    - Provides full flow: register → login → create/update/delete/view tasks → logout.

## Task Management App- Test Result

Register page,


Register
localhost:8080/register;jsessionid=CF2E79F010CE254068D07717C05B7E78
Sign Up
Name
Email
Password
Sign Up
Login
Cancel

Login page:

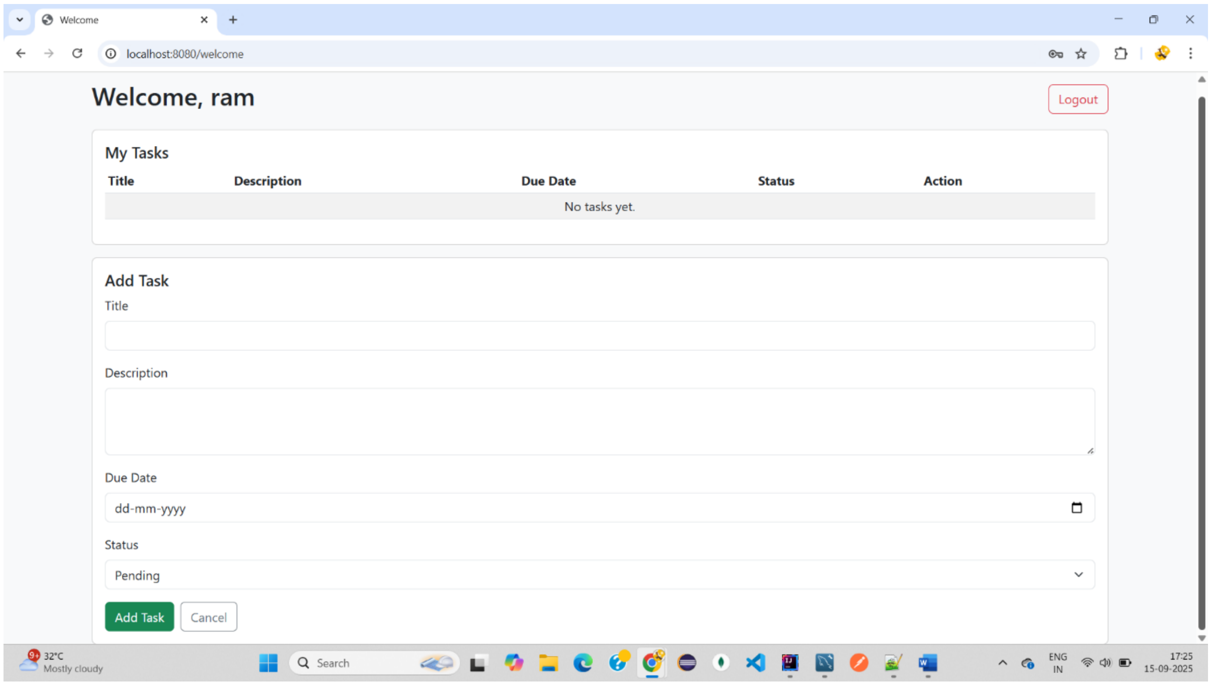

localhost:8080/login

**Login**

Name

Password

After Register and Login-> Welcome task page :


localhost:8080/welcome

**Welcome, ram**

My Tasks

| Title | Description | Due Date | Status | Action |
|---|---|---|---|---|
| No tasks yet. | | | | |

Add Task

Title

Description

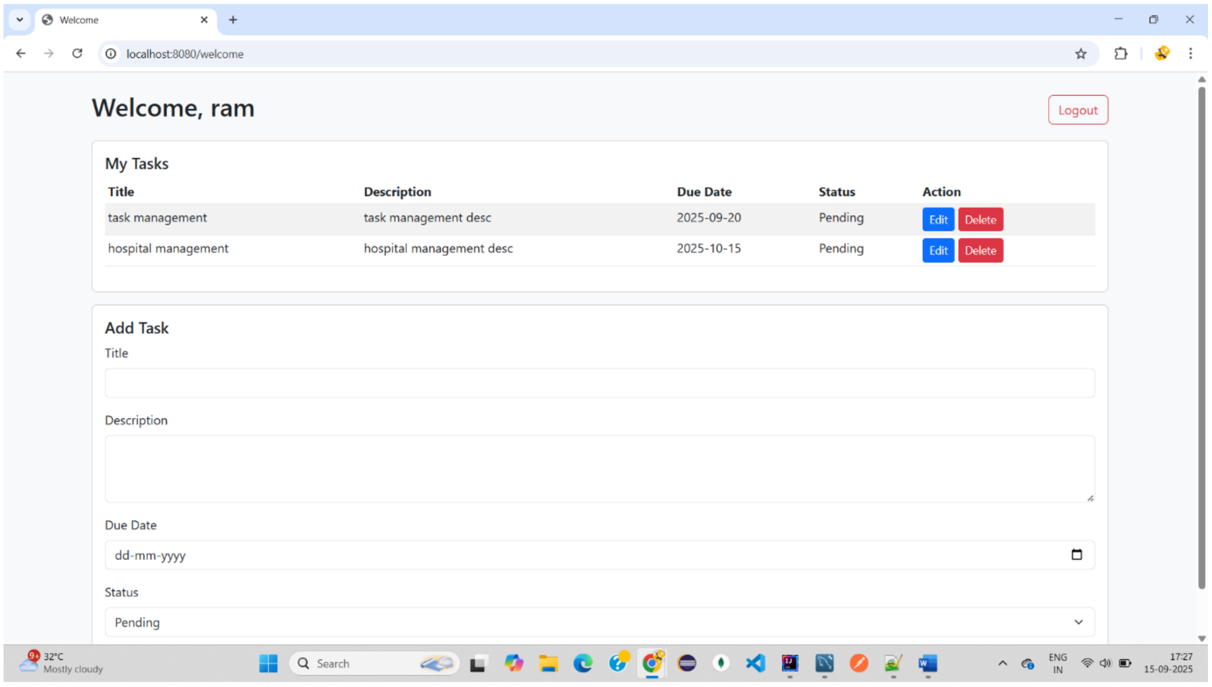
Due Date

dd-mm-yyyy

Status

Pending

Added task for ram user:



localhost:8080/welcome

Welcome, ram

Logout

My Tasks

| Title | Description | Due Date | Status | Action |
|---|---|---|---|---|
| task management | task management desc | 2025-09-20 | Pending | Edit Delete |
| hospital management | hospital management desc | 2025-10-15 | Pending | Edit Delete |

Add Task

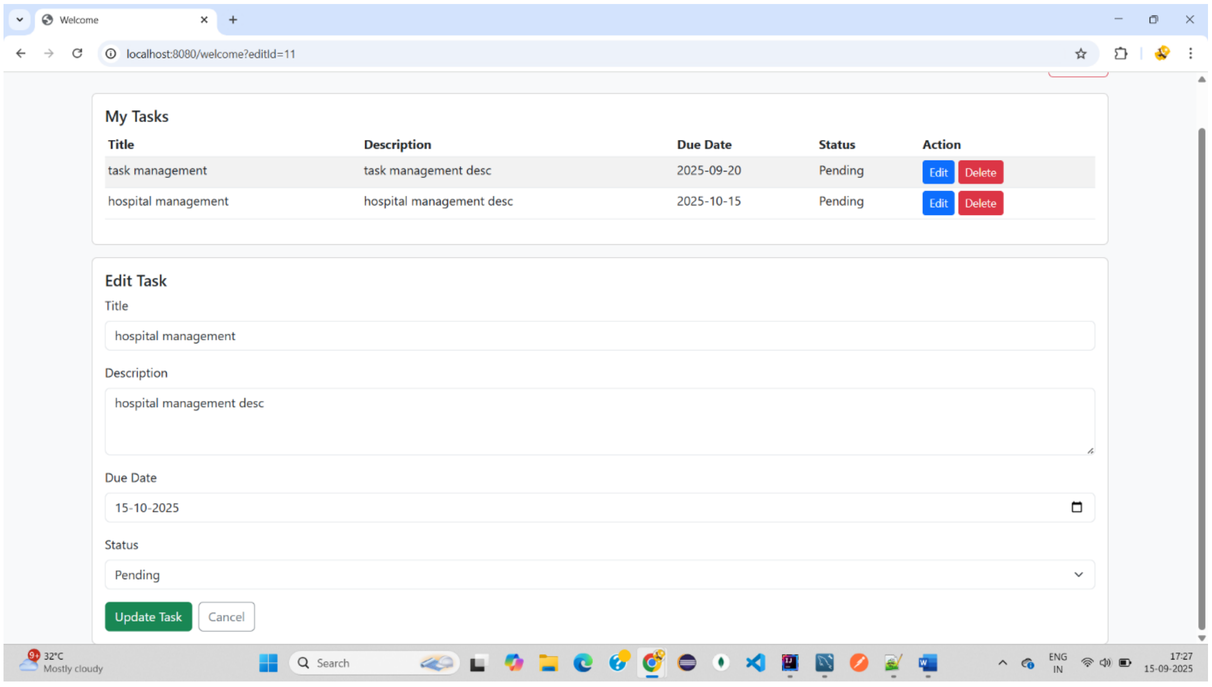
Title:

Description:

Due Date: dd-mm-yyyy

Status: Pending

Update Due date for Hospital management task:



localhost:8080/welcome?editId=11

My Tasks

| Title | Description | Due Date | Status | Action |
|---|---|---|---|---|
| task management | task management desc | 2025-09-20 | Pending | Edit Delete |
| hospital management | hospital management desc | 2025-10-15 | Pending | Edit Delete |

Edit Task

Title: hospital management

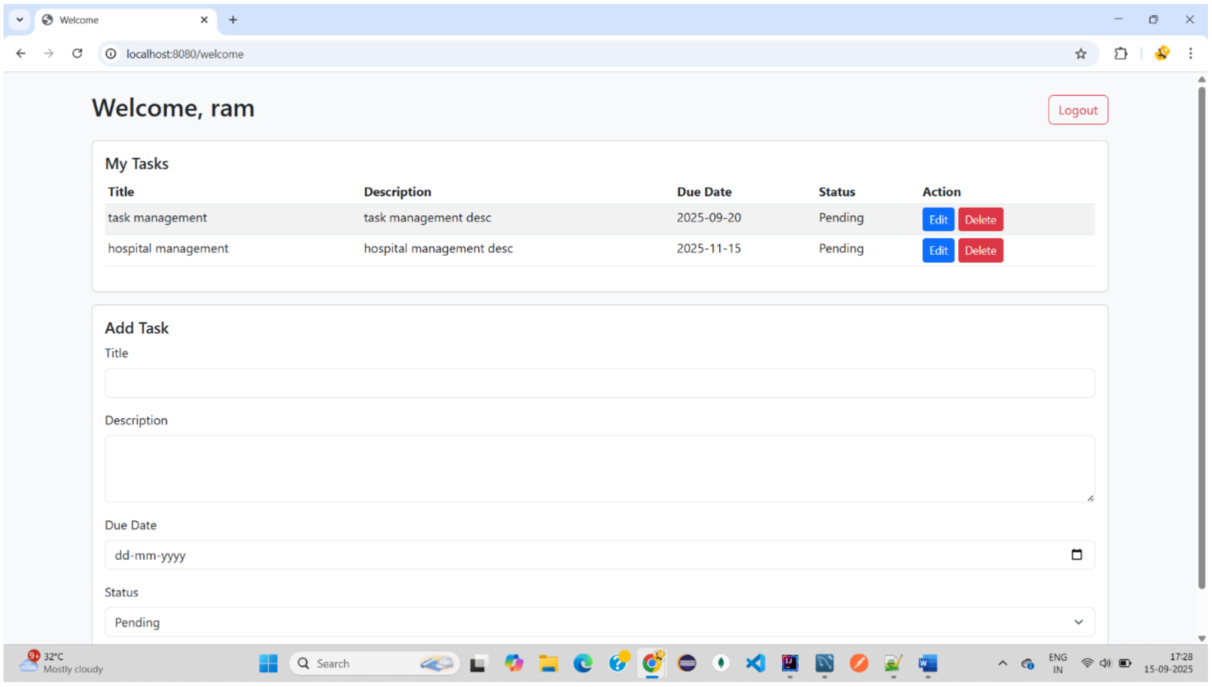
Description: hospital management desc

Due Date: 15-10-2025

Status: Pending

Update Task   Cancel

Updated due date for hospital management:



Welcome, ram

Logout

My Tasks

| Title | Description | Due Date | Status | Action |
|---|---|---|---|---|
| task management | task management desc | 2025-09-20 | Pending | Edit Delete |
| hospital management | hospital management desc | 2025-11-15 | Pending | Edit Delete |

Add Task

Title

Description

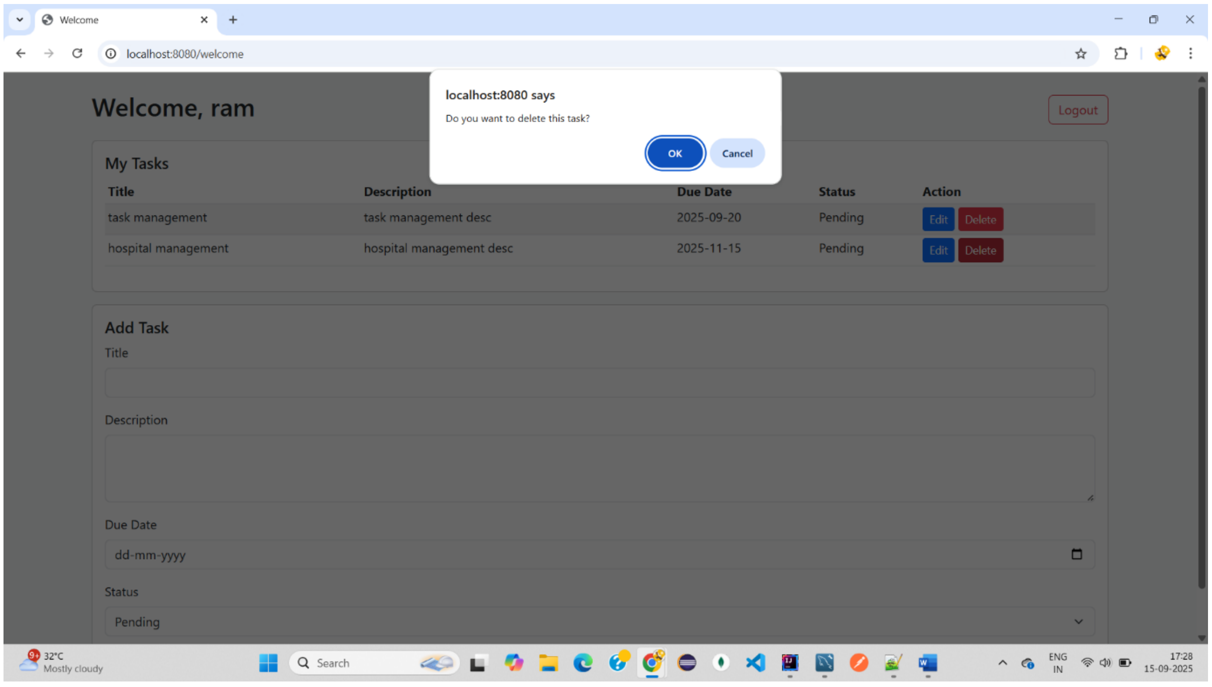
Due Date

dd-mm-yyyy

Status

Pending

Delete task :



localhost:8080 says

Do you want to delete this task?

OK Cancel

Welcome, ram

Logout

My Tasks

| Title | Description | Due Date | Status | Action |
|---|---|---|---|---|
| task management | task management desc | 2025-09-20 | Pending | Edit Delete |
| hospital management | hospital management desc | 2025-11-15 | Pending | Edit Delete |

Add Task

Title

Description

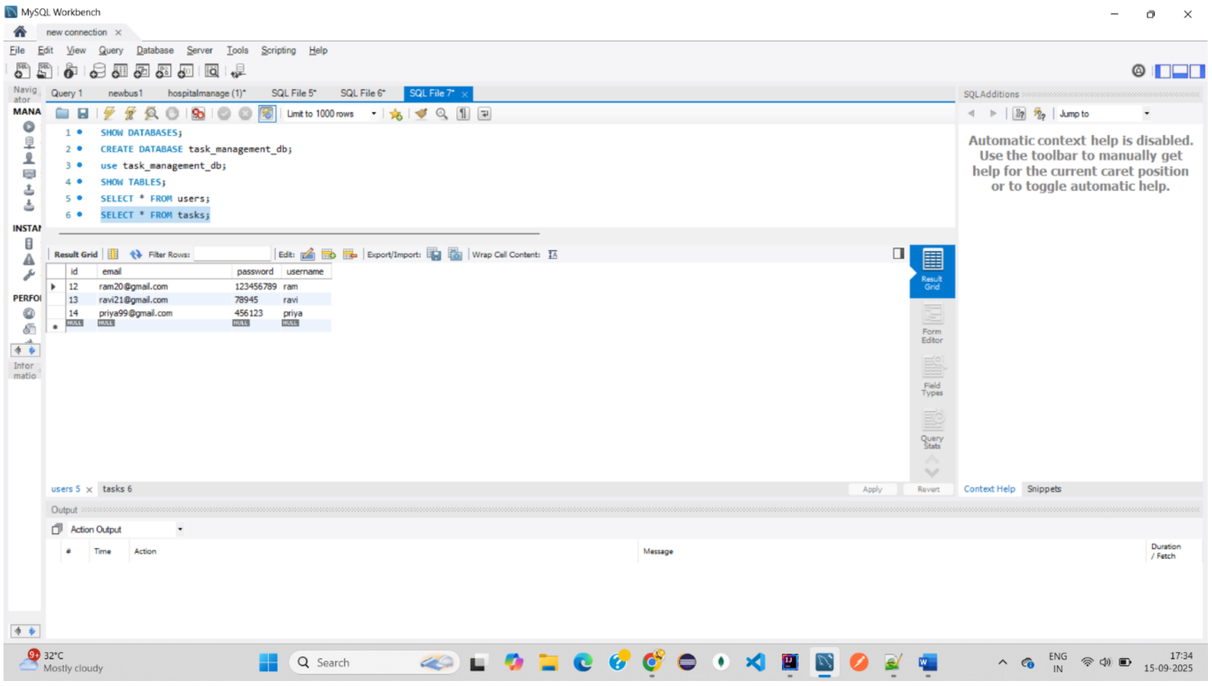
Due Date

dd-mm-yyyy

Status

Pending

DB check: User table Different user :

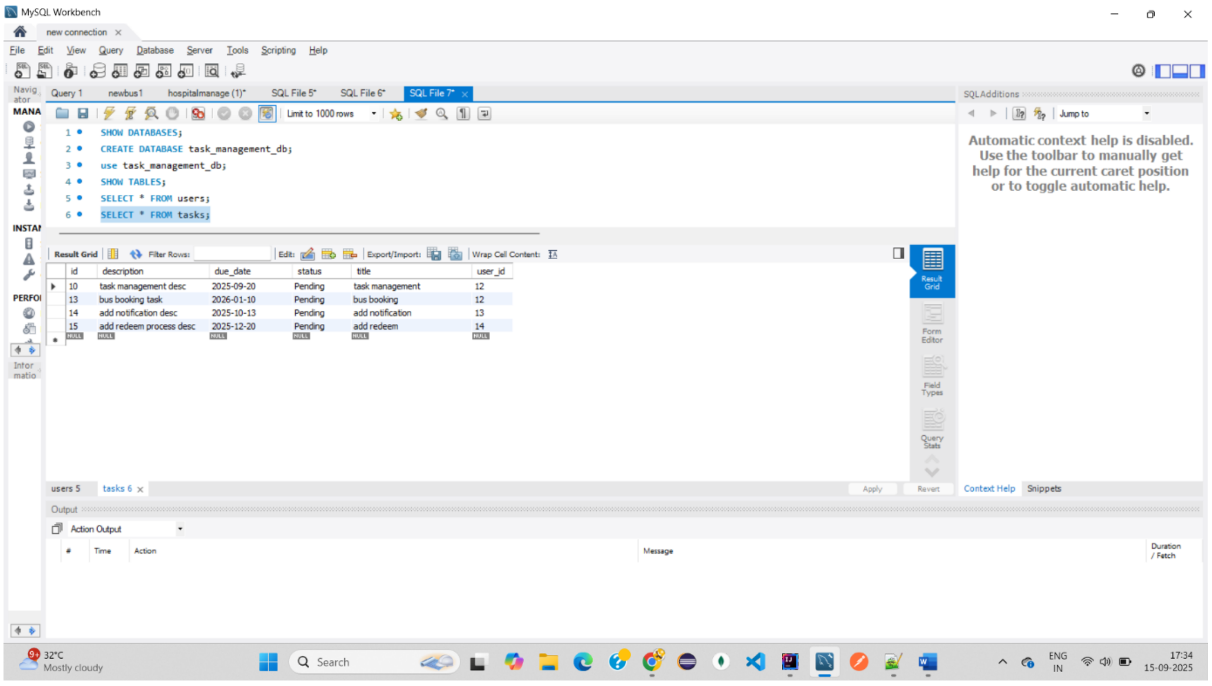


```
SHOW DATABASES;
CREATE DATABASE task_management_db;
use task_management_db;
SHOW TABLES;
SELECT * FROM users;
SELECT * FROM tasks;
```

| id | email | password | username |
|---|---|---|---|
| 12 | ram20@gmail.com | 123456789 | ram |
| 13 | ravi21@gmail.com | 78945 | ravi |
| 14 | priya99@gmail.com | 456123 | priya |
| NULL | NULL | NULL | NULL |

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

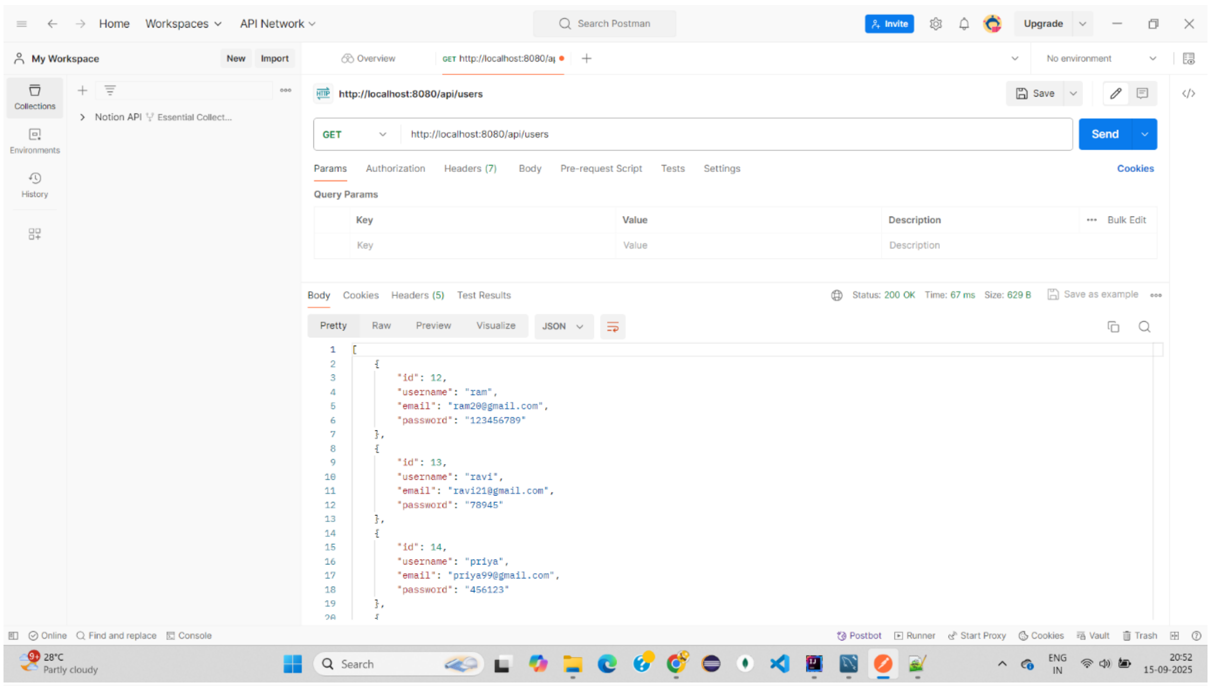
Task Table different task different user :



```
SHOW DATABASES;
CREATE DATABASE task_management_db;
use task_management_db;
SHOW TABLES;
SELECT * FROM users;
SELECT * FROM tasks;
```

| id | description | due_date | status | title | user_id |
|---|---|---|---|---|---|
| 10 | task management desc | 2025-09-20 | Pending | task management | 12 |
| 13 | bus booking task | 2026-01-10 | Pending | bus booking | 12 |
| 14 | add notification desc | 2025-10-13 | Pending | add notification | 13 |
| 15 | add redeem process desc | 2025-12-20 | Pending | add redeem | 14 |
| NULL | NULL | NULL | NULL | NULL | NULL |

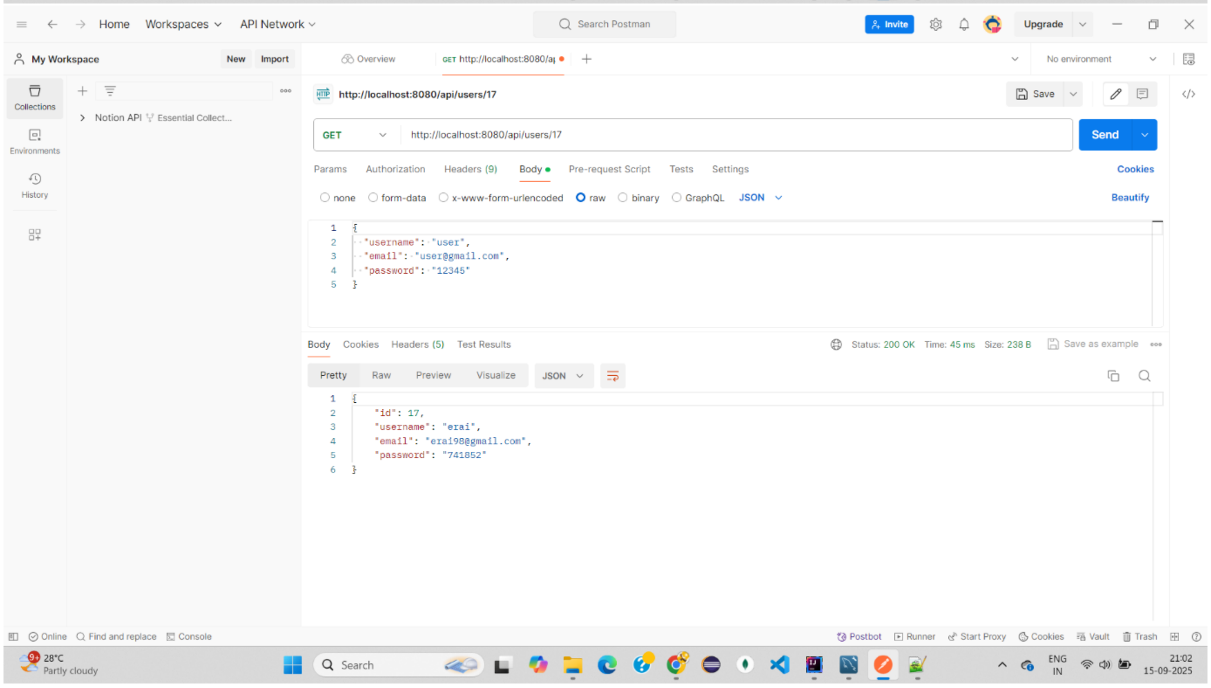
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.



GET http://localhost:8080/api/users

Status: 200 OK Time: 67 ms Size: 629 B

```json
[
    {
        "id": 12,
        "username": "ram",
        "email": "ram20@gmail.com",
        "password": "123456789"
    },
    {
        "id": 13,
        "username": "ravi",
        "email": "ravi21@gmail.com",
        "password": "78945"
    },
    {
        "id": 14,
        "username": "priya",
        "email": "priya99@gmail.com",
        "password": "456123"
    },
    {
```

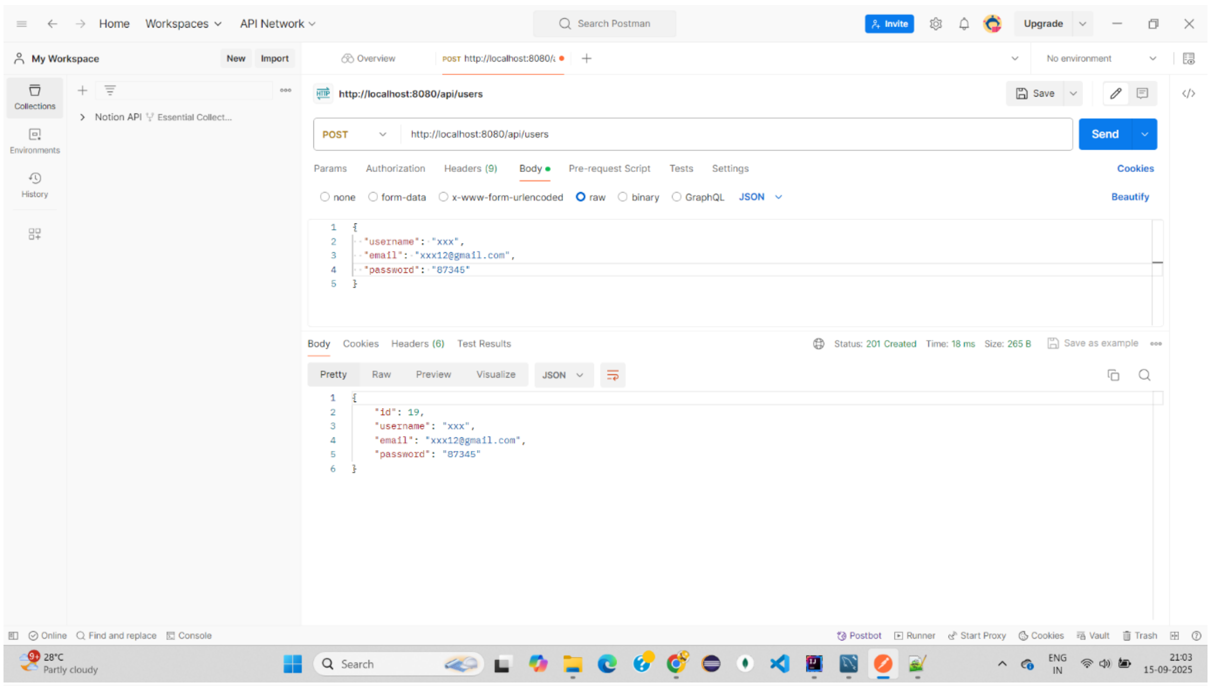


GET http://localhost:8080/api/users/17

```json
{
  "username": "user",
  "email": "user@gmail.com",
  "password": "12345"
}
```

Status: 200 OK Time: 45 ms Size: 238 B

```json
{
    "id": 17,
    "username": "erai",
    "email": "erai98@gmail.com",
    "password": "741852"
}
```


http://localhost:8080/api/users

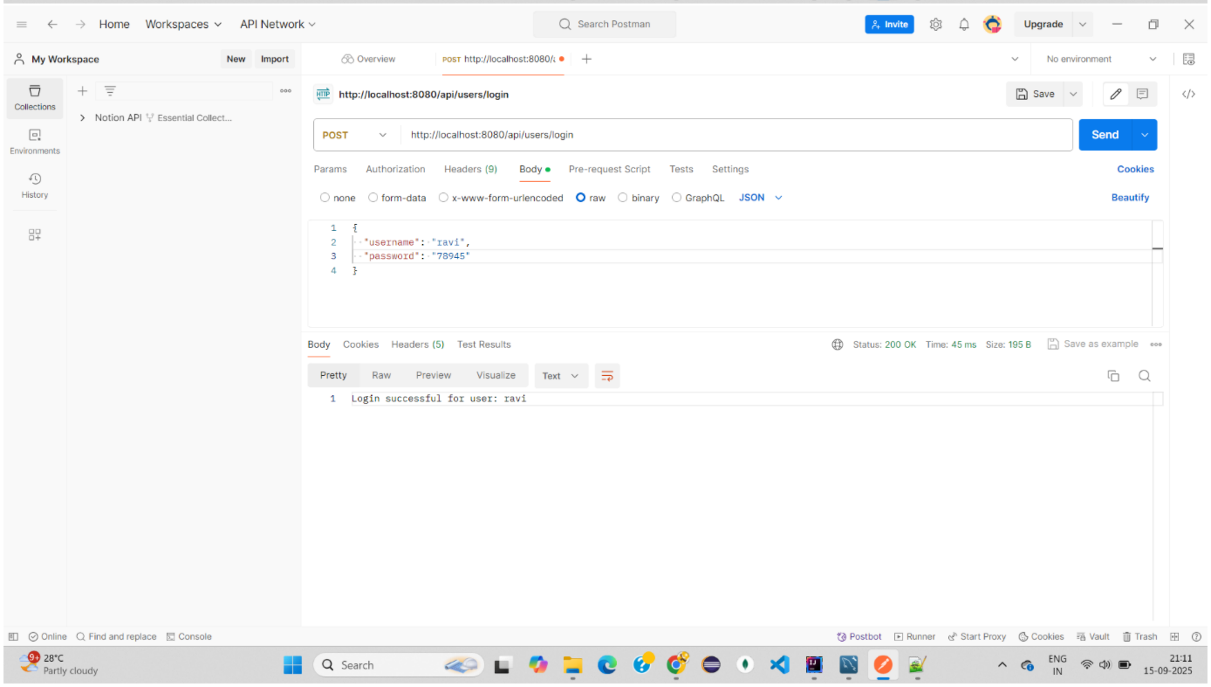
POST http://localhost:8080/api/users

Body: raw JSON

```
{
  "username": "xxx",
  "email": "xxx12@gmail.com",
  "password": "87345"
}
```

Status: 201 Created Time: 18 ms Size: 265 B

```
{
    "id": 19,
    "username": "xxx",
    "email": "xxx12@gmail.com",
    "password": "87345"
}
```


http://localhost:8080/api/users/login

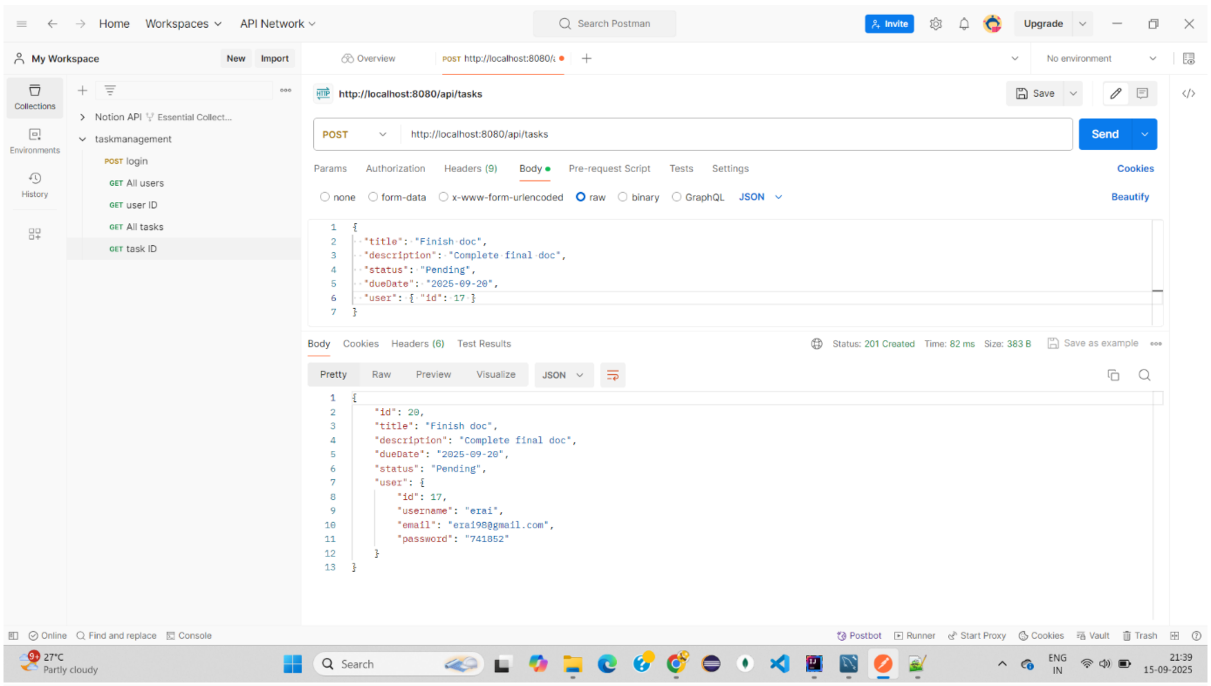
POST http://localhost:8080/api/users/login

Body: raw JSON

```
{
  "username": "ravi",
  "password": "78945"
}
```

Status: 200 OK Time: 45 ms Size: 195 B

```
Login successful for user: ravi
```



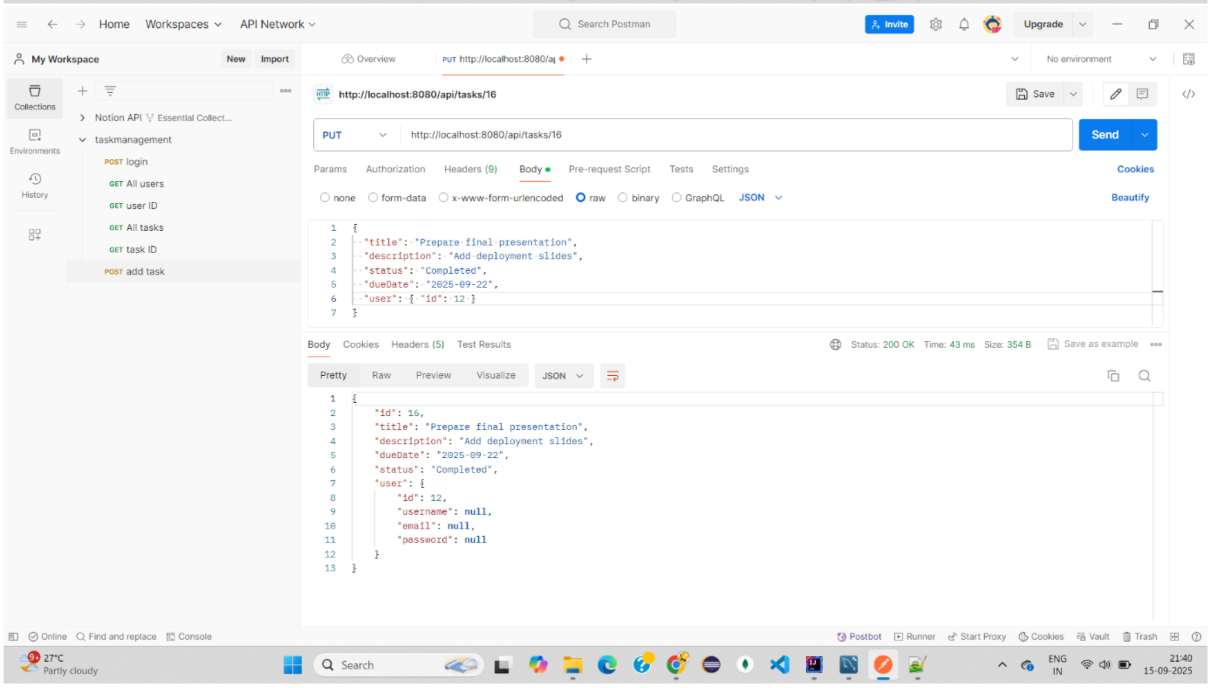
POST http://localhost:8080/api/tasks

Body: raw JSON

```
{
  "title": "Finish doc",
  "description": "Complete final doc",
  "status": "Pending",
  "dueDate": "2025-09-20",
  "user": { "id": 17 }
}
```

Status: 201 Created Time: 82 ms Size: 383 B

```
{
    "id": 20,
    "title": "Finish doc",
    "description": "Complete final doc",
    "dueDate": "2025-09-20",
    "status": "Pending",
    "user": {
        "id": 17,
        "username": "erai",
        "email": "erai98@gmail.com",
        "password": "741852"
    }
}
```



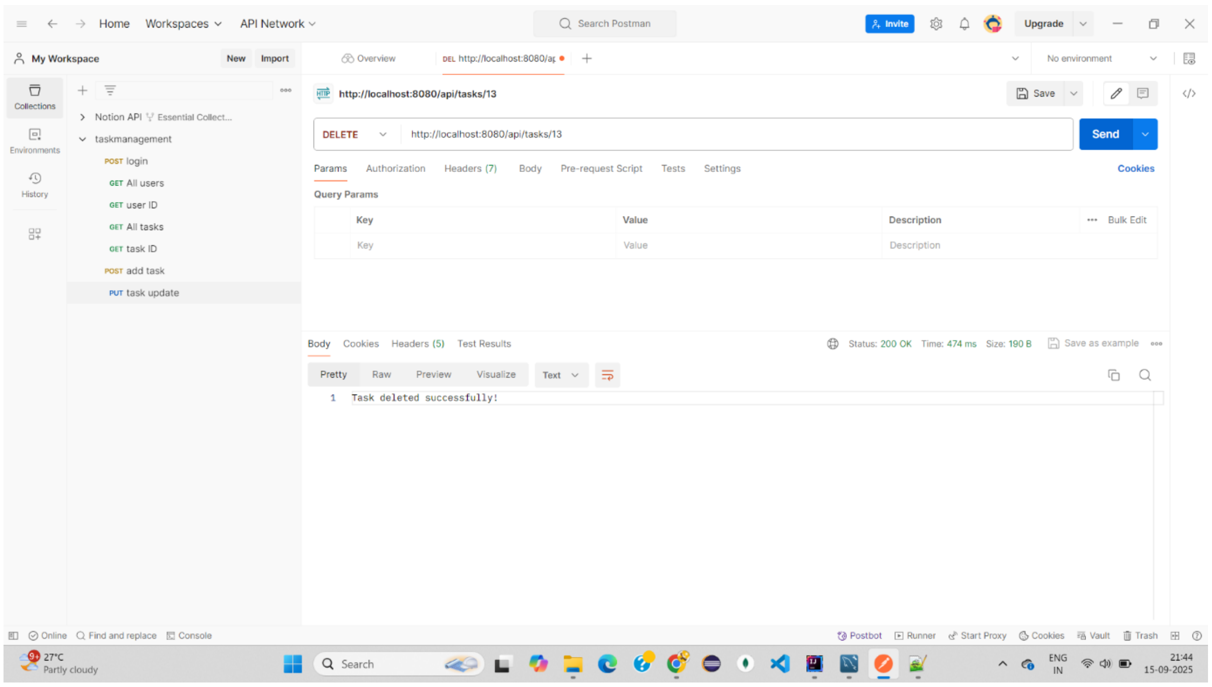
PUT http://localhost:8080/api/tasks/16

Body: raw JSON

```
{
  "title": "Prepare final presentation",
  "description": "Add deployment slides",
  "status": "Completed",
  "dueDate": "2025-09-22",
  "user": { "id": 12 }
}
```

Status: 200 OK Time: 43 ms Size: 354 B

```
{
    "id": 16,
    "title": "Prepare final presentation",
    "description": "Add deployment slides",
    "dueDate": "2025-09-22",
    "status": "Completed",
    "user": {
        "id": 12,
        "username": null,
        "email": null,
        "password": null
    }
}
```



http://localhost:8080/api/tasks/13

DELETE http://localhost:8080/api/tasks/13

Status: 200 OK Time: 474 ms Size: 190 B

```
Task deleted successfully!
```

# Task Management Application- Test Cases

## User Management

| Test Case | Steps | Expected Result |
|---|---|---|
| Register new user (valid data) | Open /register → enter username, email (abc@gmail.com), password → Submit | User saved in DB, redirected to login page with success message |
| Register with missing email | Open /register → leave email empty → Submit | Validation error: 'Email is required' |
| Register with invalid email | Enter abc@gmail without .com → Submit | Validation error: 'Email must be valid' |
| Register duplicate username | Register same username twice | Error: 'Username already exists' |
| Login with correct credentials | Open /login → enter valid username/password → Submit | Session created, redirected to /welcome |
| Login with wrong credentials | Enter wrong username/password → Submit | Error: 'Invalid credentials' |
| Logout | Click logout link | Session cleared, redirected to /login |

## Task Management (UI Workflow)

| Test Case | Steps | Expected Result |
|---|---|---|

| | | |
|---|---|---|
| Add task (all fields valid) | On /welcome → fill title, description, status, due date (today/future) → Save | Task saved and displayed in task list |
| Add task (missing title) | Leave title blank → Save | Validation error: 'Title is required' |
| Add task (past due date) | Select yesterday's date → Save | Validation error: 'Due date must be today or future' |
| Edit task | Click Edit → form prefilled → change description → Save | Updated details saved and shown |
| Delete task | Click Delete on a task → Confirm | Task removed from DB, no longer visible |
| View tasks | Log in as a user with tasks | Only that user's tasks displayed (no other users' tasks) |

### REST API – User Endpoints

| Test Case | Steps | Expected Result |
|---|---|---|
| Get all users | GET /api/users | Returns JSON array of all users |
| Get user by ID (valid) | GET /api/users/1 | Returns JSON of user with ID = 1 |
| Get user by ID (invalid) | GET /api/users/999 | Returns 404 or empty JSON |
| Register via API | POST /api/users with body {username, email, password} | Returns created user JSON |
| Login via API | POST /login with form data | Returns redirect to /welcome (UI) |

### REST API – Task Endpoints

| Test Case | Steps | Expected Result |
|---|---|---|
| Get all tasks | GET /api/tasks | Returns JSON array of all tasks |
| Get task by ID | GET /api/tasks/1 | Returns task JSON with ID = 1 |
| Create task via API | POST /api/tasks with JSON body {title, description, status, dueDate, userId} | Returns saved task JSON |
| Update task via API | PUT /api/tasks/1 with JSON body | Returns updated task JSON |
| Delete task via API | DELETE /api/tasks/1 | Returns 200 OK and task removed |

### Session & Security

| Test Case | Steps | Expected Result |
|---|---|---|
| Access /welcome without login | Enter URL directly | Redirects to /login |
| Access /tasks/delete/{id} of another user | Log in as User A → try deleting User B's task | Not allowed (task not deleted) |

## Conclusion & Future Enhancements

The Task Management Application successfully demonstrates user registration, login, and task management features using Spring Boot MVC. Validation and access control ensure that only authorized users can manage their own tasks.

Future enhancements may include:
- Adding task priority levels (High, Medium, Low).
- Email notifications for upcoming due dates.
- Deployment on cloud platforms for public access.
- REST API support for integration with frontend frameworks like React or Angular.