Full Stack Web Application for Task Management To-do list

Features-

Minimum Requirements-

- Personal Task List
    - Add/Remove/Update/Delete Tasks
    - Specify Due Date for tasks.
- Implement the backend in one of the desired Tech-Stacks provided below.
- Backend stores all the tasks data received from the Frontend and stores it in the Database.
- Implemented a Database in the Backend which should store all this structured data.
- The data sharing between Frontend and Backend done in JSON format rendered over REST APIs.

Intermediate Requirements-

- Personal Task List
    - Implemented feature to set Labels to Tasks like ‘Personal’, ‘Work’, ‘Shopping’ and ‘Others’.
    - Implemented feature to set the Status of the Tasks like ‘New’, ‘In progress’ and ‘Completed’.
    - Implemented feature to set the Priority of the Tasks like ‘High’, ‘Medium’ and ‘Low’.
- Implemented these features for an End-to-End stack, implementation will go on the Frontend as well as Backend.
- Stored the relevant flags in the database.

Extra Requirements-

- Implemented Signup and Login/Logout functionality. Created user-auth schema in the database.
- Personal Task List
    - Implemented Search feature(search substring in task title).
    - Filtered Tasks based on Priority, Label, Status (with default:all option) and a combination of at least two or more also works.
    - Sort By based on Priority (Highest => high to low, Lowest => low to high) or Date (Earliest => future due date tasks with earliest first, followed by past tasks with earliest first, followed by no due date tasks.)
    - Implemented combination Sort By and Filter By.

Additional Work-

- Validation check to title field while adding tasks.No empty task can be submitted.
- Redirect to login page if the user is not logged in.
- Built redirection page, in case the page does not exist for the logged in user.

- Built Team
    - User can create/be a member of a team with other existing users.
    - Team List shows names of all teams in which the user is a member of, along with moderator name, the team creator.
    - Each Team has a seperate team task list with the same functionalities of personal task list along with assigning a task to a team member.
    - Tasks shared across all team members.
    - Implemented a feature to add comments to tasks in the team.
        - Add, Delete comments on task of a team done by any team member.
        - Comment list shows username of commenter, created at time.
        - Comment list appears in reverse sorted order by date-time.
- Added an Email service.
    - An email is sent when a task is assigned to a person.
    - A reminder email is sent one day prior to the due date of any task(for personal, team as well).
- User can view personal tasks as well as the tasks shared in the team(created by them) on the /me/tasks list.
- Implemented **Cascading** as well *(In case you delete a team, all the tasks related to that team, and all the comments belonging to those tasks get deleted as well)*
- Implemented **JWT AUTHENTICATION** *(used JWT Authentication for our system's authentication.Expiration time of the token is 10 days.)*
- **MIDDLEWARE**
    - Implemented middleware to check if the user is authenticated whenever a user makes a request to the server.
    - Added checks in the backend, making sure that a person can only view/edit tasks created by them, or of a team that they're part of. Other users are denied access.
- **CRON JOBS**
    - Using CRON jobs to send reminder email to the task creators one day prior to the due date. [for these reminder emails, as heroku doesn't let CRON jobs run on their machine, rather it provides its own job scheduler. So we're using the Heroku job scheduler for this script]
- Deployed both the servers , backend(Express) and frontend in React, on Heroku platform.
- Using Mysql as our database, and the database provider for the database is the ClearDB Myqsl add-on provided by heroku.
- **DATABASE SEEDING**
    - Created database seeders for user, labels, statuses, priorities.
    - Need to run this once when you set up your database.

Instructions to run locally

Backend Server
Make sure you have mysql installed in your system. Start the mysql service on your machine. We are using mySQL as our database service, so we need it running on our machine.

Create a database , you may use Mysql client on your machine or install MySQLWorkbench for that.[It's a GUI platform and is handy to use]

Create a .env file in the root folder of the project and fill in the details according to your setup. I'm sharing a sample .env file here.

```
DB_HOST = "localhost"
DB_NAME = ""
DB_USERNAME = ""
DB_PASSWORD = ""

PORT = 8000
```

What do they mean?

- Let the host be localhost as we are running on our local machine.
- Set the DB_NAME as the name of the database that you've created in your MySQL Service.
- Set DB_USERNAME as the user that has permission to access the database. Usually its kept as root but you can keep the user of your choice
- DB_PASSWORD the password for the DB.
- PORT is the port number that our server will be running on. In order to avoid any errors, make sure no other server is running on the PORT that you wite here. We've kept it as 8000 but you can change it as well. But then you need to change it in the frontend server too, because it'll have to make requests to this server.

Commands to run

- npm install
- npx sequelize db:seed:all
- node servers.js

What do they mean?

- npm install
    - to install all the packages, and dependencies for them, that our project requires.
- npx sequelize db:seed:all

    - Runs the database seeders that we’ve created for labels, priorities, statuses, and two demo users.
- node server.js
    - Command to run the server.

Frontend Server

Commands to run

- Npm install
- Npm start

What do they mean ?

- npm install
    - to install all the packages, and dependencies for them, that our project requires.
- npm start
    - runs the react scripts, thereby redirecting to the browser where our actual website can be seen.