ProU Technology

Fullstack Web Application Assignment

Internship Coding Challenge – Track 3: Fullstack (Frontend + API + Database)

1. Scenario

You've joined a small software team developing an **Employee Task Tracker** - a complete internal tool for managing tasks within a company.
The product team has provided the functional requirements, and you are responsible for **building both the frontend and backend** that work together as one integrated system**.**

The system should allow:

- Viewing all employees and their tasks
- Adding and updating tasks
- Filtering tasks by status or employee
- Viewing a simple dashboard summary (e.g., total tasks, completion rate)

You can use **any modern tech stack** you're comfortable with (for example):

- **Frontend:** React / Vue / Angular
- **Backend:** Node.js + Express / Python + FastAPI / Java + Spring Boot
- **Database:** SQLite / MySQL / PostgreSQL / MongoDB

2. Expected Outcome

A **fully working web app that**:

- Displays and manages employees and tasks
- Persists data using a real database
- Communicates between frontend and backend via REST API
- Has a clean, responsive user interface

3. Deliverables

Please submit the following via GitHub (single repository or monorepo structure):

- **Frontend Source Code**
- **Backend Source Code**
- **Database Schema / Migration Script**
- **Sample Data** (optional)
- **README.md** that includes:
    - Tech stack and architecture overview
    - Setup & run instructions (for both frontend & backend)
    - API endpoint documentation
    - Screenshots or screen recording (if possible)
    - Any assumptions or limitations

4. Instructions

- The frontend must communicate with the backend API (no mock data).
- Store all data in a proper database.
- Use modern best practices:
    - **Frontend**: Component structure, state management, responsive design
    - **Backend:** RESTful design, validation, error handling
    - **Database:** Relationships and foreign keys (Employee ↔ Task)
- Use environment variables for API URLs and config
- Keep the UI simple but functional

5. Focus Areas (Evaluation Criteria)

Area	What we look for
Architecture	Clear separation between frontend, backend, and DB layers
API Integration	Smooth communication between UI and API (no hardcoding
Code Quality	Modular, readable, and consistent naming conventions
UI/UX	Clean, intuitive design and responsiveness
Data Persistence	Proper CRUD operations with a database
Documentation	Setup clarity, endpoint list, and how to test the app

6. Bonus Challenge (Optional)

Add **user authentication and role-based access**:

- Admin can add/update/delete tasks and employees
- Regular users can view their assigned tasks only

This demonstrates your understanding of authentication, authorization, and real-world app flow.

7. Estimated Time

8–10 hours (spread over 2–3 sittings).

8. Example Architecture

```
frontend/
  ├── src/
  │    ├── components/
  │    ├── pages/
  │    └── services/ (API calls)
  └── package.json

backend/
  ├── app/
  │    ├── routes/
  │    ├── models/
  │    ├── controllers/
  └── requirements.txt or package.json

database/
  ├── schema.sql or migrations/
```

9. Example API Flow

Frontend Actions → Backend Endpoints → Database

Action	API Endpoint	Database Table
View all employees	GET /employees	Employees
View all tasks	GET /tasks	Tasks
Add new task	POST /tasks	Tasks
Update task status	PUT /tasks/:id	Tasks
View summary	GET /dashboard	Calculated from both tables