Job Scheduler Application – Project Documentation

The Job Scheduler Application is a full-stack web application that allows users to create jobs, view all jobs, trigger job execution, and track job status (pending, running, completed). The system is designed with a modern frontend, a RESTful backend, and a cloud-hosted relational database.

Tech Stack Used

Frontend

- **Next.js (App Router)** – UI framework
- **React** – Component-based UI development
- **TypeScript** – Type safety and maintainability
- **Tailwind CSS** – Styling and responsive design
- **Vercel** – Frontend deployment platform

Backend

- **Node.js** – Runtime environment
- **Express.js** – REST API framework
- **MySQL2** – Database driver
- **Axios** – HTTP client for webhook triggering
- **CORS** – Cross-origin request handling
- **Render** – Backend deployment platform

Database

- **MySQL** – Relational database
- **Railway** – Cloud-hosted MySQL database service

Application Architecture

Frontend (Next.js – Vercel)

|

| REST API calls

v

Backend (Express – Render)

|

| SQL Queries

v

Database (MySQL – Railway)

Deployment Overview

- **Frontend** deployed on **Vercel**
- **Backend** deployed on **Render**
- **Database** hosted on **Railway**
- Secure communication using **environment variables**

Environment Variables Used

- **Frontend (Vercel)**

```
NEXT_PUBLIC_API_URL = https://<render-backend-url>
```

- **Backend (Render)**
  - MYSQLHOST
  - MYSQLUSER
  - MYSQLPASSWORD
  - MYSQLDATABASE
  - MYSQLPORT
  - PORT

Step-1:

Project Setup

- Created Next.js frontend using App Router
- Designed basic pages: Home, Create Job, View Jobs
- Set up Express backend with initial routes
- Configured MySQL database schema

Step-2:

Backend & Database Integration

- Implemented /jobs POST API to create jobs
- Implemented /jobs GET API to fetch all jobs
- Connected backend to MySQL database
- Added job status management (pending, running, completed)

Step-3:

Job Execution Logic

- Implemented /run-job/:id API
- Added simulated job execution using delay
- Updated job status dynamically in database
- Integrated webhook trigger after job completion

Step-4:

Frontend Integration

- Connected frontend with backend APIs
- Displayed jobs in table format
- Added Run Job button with status handling
- Implemented dynamic UI updates using React hooks

Step-5:

Deployment & Debugging

- Deployed frontend to Vercel
- Deployed backend to Render
- Created MySQL database on Railway
- Fixed CORS, environment variables, and connection issues
- Resolved UI visibility and data rendering issues

Step-6:

Final Testing & Optimization

- Verified full job lifecycle execution
- Tested API endpoints in production
- Ensured real-time job status updates
- Final UI polish and validation

Step-7:

Final Outcome

- Users can create jobs successfully
- Jobs are stored securely in the database
- Job execution updates status correctly
- Fully deployed and accessible application
- End-to-end cloud-based full-stack system

Deployment Steps

Step 1: Prepare the Project Structure

- Split the project into two folders:
    - frontend/ → Next.js application
    - backend/ → Express.js API server
- Ensure both have their own package.json
- Push the complete project to GitHub

Step 2: Deploy the Frontend (Next.js) on Vercel

1. Login to Vercel
2. Click New Project
3. Import the GitHub repository
4. Select frontend as the Root Directory
5. Set framework preset as Next.js
6. Add Environment Variable:

NEXT_PUBLIC_API_URL = https://<render-backend-url>

7. Click Deploy
8. Vercel builds and deploys the frontend successfully

Step 3: Create MySQL Database on Railway

1. Login to Railway
2. Create a New Project
3. Add MySQL service
4. Open the Connect → Public Network tab
5. Copy the following credentials:
    - Host
    - Port
    - Username
    - Password
    - Database name
6. Create required tables (jobs) using the Railway database UI

Step 4: Deploy the Backend (Express) on Render

1. Login to Render
2. Click New → Web Service
3. Connect the GitHub repository
4. Select backend as the Root Directory
5. Set:
    - Build Command: npm install
    - Start Command: node server.js
6. Add Environment Variables:

MYSQLHOST

MYSQLUSER

MYSQLPASSWORD

MYSQLDATABASE

MYSQLPORT

PORT

7. Click Deploy

Step 5: Connect Frontend and Backend

- Update frontend API calls to use:

process.env.NEXT_PUBLIC_API_URL

- Ensure CORS is enabled in backend:

app.use(cors());

- Redeploy frontend if environment variables are updated

Step 6: Production Testing

1. Open the frontend URL (Vercel)
2. Create a new job
3. Verify job is stored in MySQL (Railway)
4. Navigate to View Jobs
5. Click Run Job

Step 7: Final Validation

- Verified API endpoints using browser and logs
- Checked database insertions and updates
- Confirmed UI rendering and button actions
- Ensured secure usage of environment variables