Dotix Technologies — Full Stack Developer Skill Test

Scenario-Based Assignment: “Job Scheduler & Automation System”

Role: Full Stack Web Developer (0–1+ Yrs) **Mode:** Remote • Take-Home Test • AI Allowed (with disclosure)

⁉️ Scenario

Dotix is building a mini automation engine used internally to run background tasks such as sending emails, generating reports, syncing data, etc.

You are required to build a simplified version of this system — a **Job Scheduler & Automation Dashboard** — that allows users to create tasks, run them, track their status, and trigger webhooks when tasks complete.

This test evaluates your full-stack thinking: **Frontend** → **Backend** → **Database** → **Integrations** → **Logical Design** → **Documentation** → **GitHub Workflow.**

⁉️ What You Need to Build

Your project must include a **frontend**, **backend**, **database**, and a **webhook integration**.

1️⃣ Create Job (UI + API)

Fields:

1. `taskName` (string)
2. `payload` (JSON textarea)
3. `priority` (Low / Medium / High)

When the user creates a job:

1. Save it to the database with status = `"pending"`.

2️⃣ Job Runner API

Create an endpoint:

POST /run-job/:id

Simulate a background job:

1. Set status → `"running"`
2. Wait 3 seconds (simulate processing)
3. Set status → `"completed"`
4. Trigger an **outbound webhook** (see #4)

3Dashboard to Track Jobs

Create a React/Next.js dashboard with:

1. Table listing jobs
2. Filters: Status (pending/running/completed), Priority
3. “Run Job” button → calls `/run-job/:id`
4. Job detail view (show payload in readable format)

UI quality matters. Use **Tailwind**.

4Webhook Trigger (Outbound) When

job is completed, send POST request to a test URL:

POST https://webhook.site/<your-id>

Payload should include:

1. jobId
2. taskName
3. priority
4. payload
5. completedAt

Log webhook request/response (console or DB — your choice).

5API Summary

You must at least implement:

1. `POST /jobs` (create job)

2. `GET /jobs` (list jobs)
3. `GET /jobs/:id` (job detail)
4. `POST /run-job/:id` (simulate processing)
5. `POST /webhook-test` (optional; local webhook receiver for testing)

6Database

Use **MySQL**.

Suggested table: `jobs`

id (PK)
taskName
payload (JSON)
priority
status
createdAt
updatedAt

⬜⬜ Tech Stack Requirements

Frontend:

1. React or Next.js (Next.js preferred)
2. Tailwind CSS & Library (Shadcn preferred)

Backend:

1. Node.js (Express or Fastify)
2. REST APIs

Database:

1. SQLite or MySQL

Optional:

1. Typescript
2. Prisma / Sequelize
3. Docker (Using these tools gives bonus points but not mandatory.)

⁉ AI Usage Policy (Allowed but must be documented)

You are **free to use any AI tool**, including:

1. ChatGPT
2. GitHub Copilot
3. Gemini
4. Claude
5. Cursor
6. Copilot Workspace
7. ANY LLM or coding assistant

But you **must include**:

README must contain:

1. **AI tools used** (e.g., ChatGPT, Gemini, Copilot, GitHub Models, Llama, etc.) 2. **Model names** (e.g., GPT-4.1, Claude 3.7 Sonnet, Gemini 2.0 Pro) 3. **Exact prompts used** (copy/paste, or attach screenshots / transcript — your choice) 4. **What part of the project AI helped with**
5. UI design?
6. Writing backend logic?
7. Debugging?
8. Documentation?

AI is optional — we evaluate your thinking, not prompt engineering.

⁉ GitHub

Requirements Your project **must be**

public and include:

Repository Structure:

/frontend
/backend
/README.md

README should include:

1. Setup instructions
2. Tech stack
3. ER diagram or schema design
4. Architecture explanation
5. API documentation
6. How webhook works
7. AI usage log (as described above)

Commit messages should be meaningful — not “final”, “fix”, “lol”, etc.

?? How You Will Be

Evaluated Your performance will be judged on:

1. Architecture & Code Structure

1. Clean project structure
2. Modular backend
3. Reusable frontend components
4. Clear naming

2. UI/UX

1. Clean, modern UI
2. Good use of Tailwind
3. Logical user flow
4. Dashboard usability

3. Backend Logic

1. API quality
2. Status transitions
3. Error handling
4. Async job handling
5. Webhook implementation

4. Database Design

1. Proper schema
2. Indexes if needed
3. Clean migrations / SQL file

5. Problem-Solving / Logical Thinking

6. Production Readiness

1. ENV management
2. Config separation
3. Input validation
4. Error logging
5. Secure patterns (even basics)
6. Code readability

7. GitHub Skills

1. Repo organization
2. Clear commits
3. Readable README

8. AI Skills (Optional Bonus)

1. How effectively you used AI
2. Prompts clarity
3. Understanding when to NOT use AI

9. Feature Completeness

1. All required features implemented
2. No broken flows

10. Documentation Quality

1. Architecture explanation
2. Setup instructions
3. API usage
4. Webhook behavior
5. Screenshots (bonus)

⏳ Time

You may take **24–72 hours** based on your availability.

⁉️ Submission

Share **GitHub repo link** and a **live demo URL** if deployed (Vercel/Render/Railway/etc).
Deployment is optional but gives bonus points.