# Project Documentation: Express Cron SQL

## 1. Introduction

This document provides detailed setup, configuration, and implementation details for the Express Cron SQL project, a backend application demonstrating a RESTful API, scheduled data collection, and SQL data management. The project includes three main components:

- **Backend API**: A Node.js/Express server with in-memory storage for managing product data.
- **Scheduled Data Collection**: Cron jobs for automated data collection and file cleanup.
- **SQL Data Processing**: SQL scripts for managing employee data in a PostgreSQL database.
- **Repository**: https://github.com/ranandasatria/express-cron-sql
- **References**: For quick setup instructions, see `README.md`. For backend system architecture, see `./docs/system_design_backend.pdf`.

## 2. Backend API

### 2.1 Overview

The backend is a Node.js/Express server that provides RESTful API endpoints to manage product data, secured with JWT authentication. Data is stored in an in-memory array, suitable for lightweight applications.

### 2.2 Implementation Details

- **System Design**: Refer to `./docs/system_design_backend.pdf` for detailed architecture, including routes, controllers, services, and in-memory storage.
- **Authentication**:
  - Uses JSON Web Tokens (JWT) for securing endpoints.
  - Users register with email/password, stored in-memory with hashed passwords (bcrypt).
  - Login returns a JWT token, used in `Authorization: Bearer <token>` header for protected routes.
- **Endpoints**:

| Method | Endpoint | Description | Auth Required | Request Body Example |
|---|---|---|---|---|
| POST | /auth/register | Register a new user | No | `{"email": "user@example.com", "password": "123"}` |
| POST | /auth/login | Login and receive JWT token | No | `{"email": "user@example.com", "password": "123"}` |
| GET | /products | List all products | No | `{"id": "d5b4958e-3e73-4e5c-9c3d-fa97b49addd4", "name": "Laptop", "description": "High performance laptop","price": 1500, "stock": 10}` |
| GET | /products/:id | Get product details by ID | No | - |
| POST | /products | Create a new product | Yes | `{"name": "Laptop", "description": "High perf", "price": 1500, "stock": 10}` |
| PATCH | /products/:id | Update product details | Yes | `{"price": 1600}` |
| DELETE | /products/:id | Delete a product | Yes | - |

- **Product Attributes**:
  - `id`: UUID string
  - `name`: String
  - `description`: String
  - `price`: Number
  - `stock`: Number
- **API Documentation**: Available at `http://localhost:8080/api-docs` (Swagger UI).
- **Storage**: In-memory array (`products[]`) in `src/services/product.service.js`.

### 2.3 Setup and Configuration

- **Prerequisites**: Node.js (v18+), npm.
- **Steps**:
  1. Clone repository: `git clone https://github.com/ranandasatria/express-cron-sql.git`
  2. Install dependencies: `cd express-cron-sql && npm install`
  3. Create `.env` file based on `.env_example`:

```
PORT=8080
APP_SECRET=your_secret_key
```

4. Run server: `node index.js`
5. Access server at `http://localhost:8080` (or configured PORT).

- **Health Check**: `GET /health` returns `{ "status": "ok" }`.
- **Troubleshooting**:
  - If port 8080 is in use, change `PORT` in `.env` (e.g., `PORT=8081`).
  - If `npm install` fails, delete `node_modules` and `package-lock.json`, then retry.
  - Ensure `APP_SECRET` is a strong, unique string.

# 3. Scheduled Data Collection

## 3.1 Overview

This component includes cron jobs to:

1. Collect weather data three times daily (08:00, 12:00, 15:00 WIB) from an external API and save to CSV files.
2. Clean up CSV files older than 1 month, running daily at 00:00 WIB.

## 3.2 Implementation Details

- **Data Collection** (`automation/cron_collect.js`):
  - Fetches hourly temperature data from: `https://api.open-meteo.com/v1/forecast?latitude=-6.1818&longitude=106.8223&hourly=temperature_2m&timezone=Asia%2FBangkok`.
  - Saves to `./home/cron/cron_DDMMYYYY_HH.MM.csv` (e.g., `cron_23082025_08.00.csv`).
  - CSV format: Columns `Time`, `Temperature_2m`.
  - Schedule: `0 8,12,15 * * *` (runs at 08:00, 12:00, 15:00 WIB).
- **Data Cleansing** (`automation/cron_cleanup.js`):
  - Scans `./home/cron` for files matching `cron_*.csv`.
  - Deletes files with modification time (mtime) older than 1 month.
  - Schedule: `0 0 * * *` (runs at 00:00 WIB).
- **Path**: Uses relative path `./home/cron` for cross-platform compatibility (Windows/Linux/Mac).

## 3.3 Setup and Configuration

- **Prerequisites**: Node.js, npm.
- **Steps**:
  1. Navigate to automation folder: `cd automation`
  2. Install dependencies: `npm install`
  3. Run data collection: `node cron_collect.js`
  4. Run data cleansing: `node cron_cleanup.js`
- **Testing Tips**:
  - For quick testing, modify `cron_collect.js` schedule to `* * * * *` (every minute), run, and check for new CSV files in `./home/cron`.
  - For cleansing, change `subtract(1, 'month')` to `subtract(1, 'minute')` temporarily, then revert.
- **Troubleshooting**:
  - Ensure `./home/cron` directory exists; create it if missing (`mkdir home/cron`).
  - Check timezone (Asia/Jakarta for WIB) in cron configuration.
  - If API fails, verify internet connection or API availability.

# 4. Data Processing

## 4.1 Overview

This component includes SQL scripts to manage employee data in a PostgreSQL database, supporting operations like adding, updating, and querying employee records.

## 4.2 Implementation Details

- **Table Structure** (`sql/employee_queries.sql`):

```
CREATE TABLE employees (
    name VARCHAR(100),
    position VARCHAR(100),
    join_date DATE,
    release_date DATE,
    year_of_experience FLOAT,
    salary INTEGER
);
```

- **Sample Queries**:
  1. Add a new employee:

```
INSERT INTO employees (name, position, join_date, year_of_experience, salary)
VALUES ('Albert', 'Engineer', '2024-01-24', 2.5, 50);
```

2. Update salary for Engineers:

```
UPDATE employees SET salary = 85 WHERE position = 'Engineer';
```

3. Calculate total salary for active employees in 2021:

```
SELECT SUM(salary) AS total_salary_2021
FROM employees
WHERE join_date <= '2021-12-31'
AND (release_date IS NULL OR release_date > '2021-01-01');
```

4. List top 3 employees by experience:

```
SELECT name, position, year_of_experience
FROM employees
ORDER BY year_of_experience DESC
LIMIT 3;
```

5. List Engineers with <= 3 years experience:

```
SELECT name, year_of_experience
FROM employees
WHERE position = 'Engineer' AND year_of_experience <= 3;
```

### 4.3 Setup and Configuration

- **Prerequisites**: Docker, PostgreSQL.
- **Steps**:
  1. Install Docker.
  2. Run PostgreSQL container:

```
docker run -d --name postgres_db -e POSTGRES_USER=admin -e POSTGRES_PASSWORD=admin123
-e POSTGRES_DB=employee_db -p 5432:5432 postgres:latest
```

  3. Connect to psql:

```
docker exec -it postgres_db psql -U admin -d employee_db
```

  4. Copy-paste queries from `./sql/employee_queries.sql`.
  5. Alternatively, use VS Code:
     - Install "Database Client JDBC" extension.
     - Connect: host=localhost, user=admin, password=admin123, database=employee_db, port=5432.
     - Open `./sql/employee_queries.sql`, select all, right-click, "Run Query".
- **Troubleshooting**:
  - If port 5432 is in use, use `-p 5433:5432` and update connection settings.
  - Ensure Docker is running and container is up ( `docker ps` ).
  - If queries fail, verify table creation and data insertion order.

## 5. General Setup and Configuration

### 5.1 Project Structure

```
├─ automation
│  ├─ cron_cleanup.js
│  └─ cron_collect.js
├─ docs
│  ├─ system_design_backend.pdf
│  └─ project_documentation.docx
├─ index.js
├─ package-lock.json
├─ package.json
├─ README.md
├─ sql
│  └─ employee_queries.sql
└─ src
   ├─ controllers
   │  ├─ auth.controller.js
   │  └─ product.controller.js
   ├─ middlewares
   │  └─ auth.middleware.js
   ├─ services
   │  ├─ auth.service.js
   │  └─ product.service.js
   ├─ swagger
   │  └─ swagger.js
   └─ validators
      └─ auth.dto.js
```

### 5.2 Additional Notes

- **Cross-Platform**: Relative paths (e.g., `./home/cron`) ensure compatibility across Windows/Linux/Mac.
- **Testing**: All components tested locally with Node.js v22.14.0, Docker 28.1.1, and PostgreSQL 17.5.
- **Assumptions**: The application assumes a frontend sends JSON data to `/products` endpoint; no frontend code is included.
- **Troubleshooting**:
  - Dependency issues: Run `npm install` for consistent installs.
  - Cron failures: Verify API connectivity.
  - SQL errors: Check table schema and data types match query expectations.

## 6. Conclusion

The Express Cron SQL project provides a robust backend solution with a RESTful API, automated data collection, and SQL-based data management.

All components are fully functional, documented, and designed for ease of use. Refer to `README.md` for quick setup and `./docs/system_design_backend.pdf` for backend architecture details.