ChatGPT

Rencana Refactoring dan Integrasi Sistem Informasi Laporan Harian Karyawan Utility

Pendahuluan

Project *Sistem Informasi Reporting Harian Karyawan Utility* saat ini menggunakan **React + Vite + Tailwind CSS** untuk *frontend* dan **Express.js** untuk *backend*, dengan rencana migrasi database dari **MySQL (phpMyAdmin)** ke **PostgreSQL**. Namun, integrasi antar komponen masih bermasalah – misalnya operasi CRUD tidak tersimpan di database – dan kode front-end/back-end belum terhubung dengan baik. Kami akan melakukan refactoring menyeluruh: memperbaiki bug integrasi, menyusun ulang arsitektur proyek, menyelesaikan fitur-fitur (login, dashboard, laporan, manajemen karyawan, checklist harian/mingguan/tahunan dengan export PDF), serta melakukan migrasi database ke PostgreSQL. Langkah-langkah perbaikan dijelaskan secara terperinci di bawah ini.

Identifikasi Masalah Integrasi

Beberapa permasalahan yang teridentifikasi dalam proyek saat ini:
- **Operasi CRUD gagal tersimpan**: Request dari front-end mungkin tidak mencapai back-end atau gagal diproses di server, sehingga perubahan (Create/Update/Delete) tidak masuk ke database. Penyebabnya bisa karena endpoint API tidak sesuai, middleware Express (seperti body parser) belum dikonfigurasi, atau kesalahan query database.
- **CORS/Proxy Issues**: Jika front-end (Vite dev server) berjalan di origin berbeda (misal port 5173) dari back-end (misal port 3000), maka permintaan AJAX bisa diblok oleh kebijakan *browser* (CORS). Ini perlu diatasi dengan mengizinkan CORS di server atau mengkonfigurasi proxy di Vite [1] [2].
- **Kode tidak terstruktur**: Front-end dan back-end mungkin dikembangkan terpisah tanpa kesepakatan format data/endpoint. Akibatnya integrasi sulit. Contohnya, front-end mungkin mengirim request ke URL atau metode yang tidak ada di back-end, atau format datanya berbeda (misal nama field JSON berbeda).
- **Migrasi DB belum tuntas**: Awalnya menggunakan MySQL, migrasi ke PostgreSQL mungkin belum selesai. Perbedaan sintaks (contoh: *auto-increment* di MySQL versus *serial* di PostgreSQL [3]) bisa menyebabkan query gagal. Koneksi database juga harus disesuaikan (ganti driver MySQL ke PostgreSQL).
- **Kurangnya error handling**: Bug bisa juga muncul karena error tidak ditangani. Misal, jika query DB error namun tidak ada log/response yang jelas, front-end tidak tahu apa yang salah. Middleware order di Express penting – pastikan *error-handling middleware* diletakkan di akhir [4] agar error tidak terlewat.

Arsitektur Teknologi dan Setup Dasar

Untuk memastikan integrasi yang mulus, kita akan menyusun ulang proyek dengan pembagian jelas antara *frontend* dan *backend*, serta memastikan keduanya berkomunikasi lewat API REST yang konsisten. Berikut rencana setup dasar:

Frontend: React + Vite + Tailwind CSS

- **Inisialisasi Proyek React**: Buat ulang proyek React menggunakan Vite (misal `npm create vite@latest my-app -- --template react`). Vite memberikan *development server* cepat dengan *hot module replacement*. Struktur folder standar akan digunakan (`src/` untuk komponen, dsb.). Pastikan Tailwind CSS terkonfigurasi (file `tailwind.config.js`, import CSS di `index.css`).
- **Konfigurasi Proxy (Opsional)**: Selama development, untuk kemudahan, kita bisa mengaktifkan proxy di Vite agar request API diarahkan ke server Express. Ini mencegah isu CORS tanpa harus mengubah konfigurasi browser [2]. Contoh di file `vite.config.js` untuk meneruskan semua request berawalan `/api` ke `localhost:3000` (port Express):

```
// vite.config.js
export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:3000',
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, '')
      }
    }
  }
})
```

  Dengan konfigurasi di atas, pemanggilan `fetch('/api/...')` di front-end akan otomatis diteruskan ke `localhost:3000` [5]. Ini memudahkan integrasi dan *relative URL* bisa dipakai.
- **State Management & Routing**: Gunakan React Router untuk mengatur navigasi (halaman login, dashboard, dsb.). State user login bisa disimpan via Context API atau state management lain (Redux, dll.) untuk menentukan role (super user atau user biasa) dan mengendalikan akses UI.
- **Komunikasi API**: Gunakan `fetch` API atau library seperti Axios untuk CRUD ke endpoint Express. Pastikan endpoint URL dan method sesuai dengan definisi di back-end. Contoh:

```
// Contoh request GET data karyawan
fetch('/api/employees')
  .then(res => res.json())
  .then(data => setEmployees(data))
  .catch(err => console.error('Error fetching data:', err));
```

  (Dengan proxy, `/api/employees` akan sampai ke Express). Semua permintaan harus menyertakan header `Content-Type: application/json` bila mengirim body JSON, dan jika memakai autentikasi JWT, sertakan juga header `Authorization: Bearer <token>`.
- **Tailwind UI**: Pastikan komponen-komponen (form login, tabel laporan, dsb.) menggunakan kelas utility Tailwind untuk tampilan konsisten. Uji antarmuka agar responsif dan mudah digunakan.

Backend: Express.js + PostgreSQL

- **Inisialisasi Server Express**: Buat proyek Node.js baru (`npm init`) dan pasang Express (`npm install express`). Buat file `index.js` (atau `app.js`) sebagai entry point server. Gunakan middleware `express.json()` (atau *body-parser*) untuk parsing JSON request body [6], dan aktifkan `cors()` middleware jika perlu mengizinkan akses dari origin front-end [1] (jika tidak pakai proxy). Contoh:

```
const express = require('express');
const cors = require('cors');
const app = express();
app.use(express.json());
app.use(cors()); // membuka akses CORS untuk semua origin (bisa dibatasi
domain)
```

  Langkah di atas mencegah masalah umum seperti *blocked by CORS policy* dan `req.body` undefined karena lupa parsing JSON.

- **Koneksi ke PostgreSQL**: Gunakan library **node-postgres (`pg`)** untuk terhubung ke database PostgreSQL. Install terlebih dahulu (`npm install pg`). Buat konfigurasi koneksi menggunakan `Pool` dari pg dengan parameter sesuai kredensial PostgreSQL (user, host, database name, password, port). Contoh inisialisasi koneksi:

```
const { Pool } = require('pg');
const pool = new Pool({
  user: 'db_user',
  host: 'localhost',
  database: 'nama_database',
  password: 'password_db',
  port: 5432
});
```

  Inisialisasi di atas membuat pool koneksi yang siap digunakan untuk query SQL [7]. Selanjutnya, setiap operasi CRUD akan menggunakan pool ini, misal:

```
// Contoh query: mengambil data users
pool.query('SELECT * FROM users', (err, result) => { ... });
```

  atau dengan *parameterized query* untuk keamanan:

```
pool.query('SELECT * FROM users WHERE id=$1', [id], ... );
```

  Catatan migrasi MySQL→PostgreSQL: Jika sebelumnya menggunakan MySQL, sesuaikan perintah SQL dengan sintaks PostgreSQL. Contoh, MySQL menggunakan `AUTO_INCREMENT` untuk primary key, sedangkan PostgreSQL memakai tipe `SERIAL` atau `GENERATED AS IDENTITY` untuk auto-increment [3]. Pastikan skema tabel dibuat ulang di PostgreSQL sesuai kebutuhan sistem (tabel pengguna, laporan, karyawan, dll.), dengan penyesuaian tipe data

(contoh: `INT(11)` bisa diganti `INTEGER`, tipe `VARCHAR` serupa, dll.). Gunakan **pgAdmin** atau CLI `psql` untuk membuat database dan tabel awal di PostgreSQL.

- **Struktur Routes dan Controller**: Buat modul terpisah untuk route dan query agar rapi. Misal, folder `routes` atau `controllers` yang memuat fungsi-fungsi handler. Beberapa endpoint yang diperlukan:

- `POST /api/auth/login` – menangani login user (mengautentikasi username/password).
- `GET /api/employees` – mendapatkan daftar karyawan (hanya untuk super user).
- `POST /api/employees` – menambah karyawan baru (super user).
- `PUT /api/employees/:id` – mengubah data karyawan (super user).
- `DELETE /api/employees/:id` – menghapus karyawan (super user).
- `GET /api/reports?date=...` – mengambil data laporan harian, atau mungkin `GET /api/reports/daily`, `/weekly`, `/annual` tergantung implementasi.
- `POST /api/reports` – (opsional, jika user menginput laporan). Namun dalam skenario ini laporan harian kemungkinan diunggah oleh superuser sebagai checklist, jadi endpoint ini bisa tidak ada untuk user biasa.
- `GET /api/checklists/:period` – superuser dan user akses untuk lihat checklist harian/mingguan/tahunan. `period` bisa berupa `daily`, `weekly`, `annual`.
- `POST /api/checklists/:period` – unggah checklist (file atau data) untuk periode tertentu (super user saja).
- `GET /api/checklists/:period/download` – unduh file checklist (user bisa unduh).
- `GET /api/checklists/:period/pdf` – (opsional) endpoint untuk menghasilkan PDF dari checklist (jika diperlukan server-side).

Masing-masing route sebaiknya dipecah ke handler fungsi. Contoh pola sederhana dengan Express Router:

```
const express = require('express');
const router = express.Router();
const EmployeeController = require('./controllers/employee');
router.get('/employees', EmployeeController.getAll);
router.post('/employees', EmployeeController.create);
// ...dst
app.use('/api', router);
```

Dalam handler `EmployeeController.getAll`, lakukan query ke DB via `pool.query('SELECT * FROM employees', ...)` dan kembalikan hasil sebagai JSON.

- **Autentikasi & Otorisasi**:
- **Login**: Saat user login, validasi kredensial terhadap database. Simpan password terenkripsi (gunakan bcrypt untuk hashing password saat membuat user). Jika valid, terbitkan **JWT (JSON Web Token)** untuk sesi user tersebut. JWT akan dikirim ke client, dan disimpan (misal di `localStorage` atau sebagai cookie HTTP-Only). Untuk setiap request selanjutnya, client kirim token via header `Authorization: Bearer <token>`.
    - Di server, siapkan middleware untuk memverifikasi JWT pada endpoint yang butuh login. Jika token valid, *decode* untuk mendapatkan informasi user (termasuk peran/role).

    - Opsi lain adalah menggunakan **Passport.js** strategi JWT atau lokal. Passport memudahkan integrasi berbagai metode auth [8], tapi untuk konteks ini JWT custom sudah memadai.

- **Role-based Access**: Tandai user dengan role, misal field `role` di tabel users (nilai bisa `admin`/`superuser` vs `user`). Middleware otorisasi bisa dibuat, contoh `isAdmin` yang mengecek `req.user.role` hasil decode JWT. Jika bukan admin, balas 403 Forbidden [9]. Terapkan middleware ini di routes sensitif (seperti CRUD karyawan, upload checklist). Dengan mekanisme ini, superuser memiliki akses penuh (kelola data karyawan, upload laporan), sedangkan user biasa hanya bisa melihat/mengunduh.

- **Upload File (Checklist)**: Fitur checklist harian/mingguan/tahunan kemungkinan melibatkan upload file (misal format PDF atau Excel berisi daftar tugas/ceklist). Untuk menangani *file upload*, gunakan **Multer**, middleware khusus Express untuk `multipart/form-data` [10]. Install Multer (`npm install multer`) dan konfigurasi misal:

```
const multer = require('multer');
const upload = multer({ dest: 'uploads/' }); // simpan file ke folder
uploads/
app.post('/api/checklists/:period', isAuth, isAdmin,
upload.single('file'), (req, res) => {
  // req.file berisi file yang diunggah [11]
  // req.body berisi field text lainnya (jika ada)
  const period = req.params.period; // "daily" / "weekly" / "annual"
  // TODO: simpan informasi file (misal path atau nama) ke database agar
bisa diakses user
  res.status(200).json({ message: 'Checklist uploaded', fileInfo:
req.file });
});
```

Pastikan form upload di front-end menggunakan `enctype="multipart/form-data"` dan field name sesuai (`name="file"` pada input type file). Setelah file di-upload dan misalnya disimpan di folder server atau cloud, simpan referensi (path atau URL) di database *Checklists* dengan kolom `period` dan mungkin `uploaded_at`.

- **Export PDF**: Ada dua skenario export PDF dalam aplikasi: (1) superuser mengunggah file PDF langsung sebagai checklist (kemudian user tinggal download), atau (2) data laporan dibangkitkan dari database dan user menekan tombol "Export PDF" untuk mengunduhnya. Untuk skenario (2), kita perlu menghasilkan PDF dari data secara dinamis. Solusinya:

- **Di front-end**: Gunakan library seperti **jsPDF** atau **html2canvas** + jsPDF. Dengan jsPDF, kita bisa menangkap konten HTML (misal tabel laporan) dan menghasilkan file PDF di browser. Contoh penggunaan: install `jspdf` (`npm install jspdf jspdf-autotable`), lalu di halaman laporan:

```
import jsPDF from 'jspdf';
import 'jspdf-autotable';
// ...
function exportPDF() {
```

```
  const doc = new jsPDF();
  doc.text("Laporan Harian", 10, 10);
  // Misal menambahkan tabel ke PDF menggunakan autotable:
  doc.autoTable({ html: '#id-tabel-laporan' });
  doc.save('laporan-harian.pdf');
}
```

Pendekatan ini *client-side*, cocok untuk data yang sudah ter-render. Memberikan kemampuan konversi data ke dokumen PDF yang rapi dan bisa diunduh user [12].

- **Di back-end**: Alternatifnya, buat endpoint misal `GET /api/reports/:id/pdf` yang ketika diakses akan mengambil data dari DB dan mengembalikan respon PDF (dengan header `application/pdf`). Di Node.js bisa gunakan pustaka seperti **pdfkit** atau **puppeteer** (merender HTML ke PDF). Namun ini lebih kompleks di server. Untuk aplikasi ini, solusi front-end (jsPDF) sudah memadai, kecuali jika PDF harus terarsip di server.

Implementasi Fitur Utama

Berikut penjelasan implementasi masing-masing fitur sesuai kebutuhan sistem:

- **1. Login & Authentication**: Halaman login di React akan meminta username/email dan password. Ketika submit, front-end memanggil endpoint `POST /api/auth/login` dengan kredensial. Back-end memverifikasi: cari user di tabel `users`, cek hash password. Jika cocok, back-end mengirim JWT token (atau session cookie). Front-end kemudian:
- Menyimpan token (misal di `localStorage` atau cookie).
- Mengarahkan user ke halaman dashboard.
- Menggunakan token ini untuk header di request selanjutnya.

- Menyetel state global bahwa user `isLoggedIn` dan apakah `isAdmin` (bisa decode token yang memuat role).

- **2. Dashboard**: Halaman dashboard (setelah login) menampilkan ringkasan informasi. Misalnya: jumlah karyawan, ada laporan terbaru, pengumuman, dsb. Dashboard untuk superuser mungkin berbeda (menampilkan modul admin seperti *Manage Employees*). Secara implementasi, sediakan beberapa endpoint pendukung, contoh:

- `GET /api/stats/summary` – mengembalikan jumlah karyawan, jumlah laporan harian minggu ini, dll., untuk ditampilkan di dashboard.
  Front-end akan fetch data tersebut dan menampilkan dalam kartu-kartu statistik.

- **3. Data Karyawan (Manajemen User)**: Fitur ini hanya untuk **Super User**. Halaman *Data Karyawan* menampilkan tabel daftar karyawan dan form tambah/edit. Implementasi:

- **Back-end**: Endpoint CRUD seperti yang sudah diuraikan (`/api/employees`). Pastikan semua endpoint ini dilindungi `isAdmin` middleware [9]. Contoh, `GET /api/employees` akan menjalankan `SELECT * FROM employees` dan mengirim hasil ke front-end. `POST /api/employees` menerima JSON data karyawan baru, menjalankan `INSERT` ke DB. Gunakan parameter query ($1, $2, ...) untuk mencegah SQL injection.
- **Front-end**: Buat halaman `EmployeesPage` yang fetch data dari `/api/employees` saat load (useEffect). Tampilkan dalam tabel. Sediakan tombol "Add Employee" yang membuka form modal, serta tombol edit/hapus di setiap baris. Saat submit form, panggil API `POST /api/`

`employees` (untuk tambah) atau `PUT /api/employees/:id` (untuk edit), lalu refresh data tabel. Tampilkan notifikasi sukses atau error sesuai respon server.

- Validasi: Berikan validasi input (nama tidak kosong, email format benar, dsb.) di front-end *dan* back-end (back-end bisa gunakan library seperti *express-validator* untuk cek input [13]).

- **4. Laporan Harian**: *Use case* bagian ini perlu diperjelas. Bisa ditafsirkan sebagai catatan harian setiap karyawan (diisi oleh user biasa), atau sebuah laporan umum yang diunggah admin tiap hari. Mengingat ada fitur checklist harian oleh superuser, kemungkinan laporan harian di sini adalah data yang diinput user.

- Jika **user mengisi laporan harian**: Sediakan form bagi user untuk submit laporan (misal isi kegiatan harian). Endpoint `POST /api/reports` menyimpan ke DB (tabel `reports` dengan kolom user_id, tanggal, isi laporan). User juga bisa melihat laporan-laporan miliknya (filter by user). Admin bisa melihat semua laporan. Front-end: halaman *Laporan* dengan filter tanggal dan tombol tambah laporan baru (jika belum ada untuk hari itu). PDF export bisa memungkinkan user mengunduh laporan tertentu.
- Jika **laporan diunggah admin**: Bisa jadi admin membuat semacam ringkasan harian dan mengupload (mirip checklist). Namun karena sudah ada modul checklist, skenario pertama lebih masuk akal.

- **Keamanan**: Pastikan user biasa hanya bisa CRUD laporan milik sendiri. Admin dapat mengakses semua (bisa gunakan middleware peran lagi).

- **5. Checklist Harian, Mingguan, Tahunan**: Ini adalah fitur yang melibatkan **superuser mengupload checklist**, dan **user biasa hanya dapat melihat atau mengunduh**. Implementasi detail:

- Buat tabel `checklists` di PostgreSQL dengan kolom: `id`, `period` (enum atau text berisi 'daily/weekly/annual'), `file_path` (path file di server atau URL), `uploaded_at`, `uploaded_by`.
- Saat superuser upload file melalui form di UI (pilih file dan period), panggil `POST /api/checklists/{period}` dengan form data. Server menerima file via Multer [11], simpan di folder (misal `uploads/daily/<filename>`). Simpan record di DB. Kembalikan response sukses.
- Front-end superuser: di halaman Checklist, ada form unggah (dropdown pilih periode + pilih file). Juga tampilkan daftar checklist yang sudah ada per periode (misal latest file and date). Setelah upload berhasil, update daftar.
- Front-end user biasa: halaman Checklist yang menampilkan list file checklist yang tersedia per kategori (harian, mingguan, tahunan) dengan tombol *View*/*Download*.
    - Tombol *View* bisa membuka file di tab baru (jika PDF) atau modal preview.
    - Tombol *Download* mengakses endpoint `GET /api/checklists/:period/download` yang mengirim file (set header `Content-Disposition: attachment`).
    - Tombol *Export PDF* – jika file aslinya bukan PDF (misal admin upload format lain atau mungkin admin mengisi form checklist di aplikasi), maka user bisa generate PDF. Namun jika admin sudah upload PDF, user cukup download saja.
- **Catatan**: Bila checklist sebetulnya berupa data isian (bukan file), alternatifnya admin mengisi formulir checklist di aplikasi (misal daftar tugas harian apa saja), disimpan ke DB, dan user bisa melihat (render HTML) lalu export PDF via front-end. Ini opsi lain, tapi yang disiratkan soal "mengupload" cenderung file.

- **Keamanan**: Pastikan hanya admin bisa upload (endpoint sudah dilindungi role). File di server sebaiknya diberi nama unik (misal pakai timestamp) untuk menghindari tertimpa. Juga mungkin perlu *cleanup* file lama jika perlu.

- **6. Export PDF**: Seperti dibahas di atas, pengguna biasa perlu fitur export PDF terutama untuk menyimpan/print laporan atau checklist. Dengan integrasi jsPDF di front-end, implementasinya:

- Tambahkan tombol "Export PDF" di halaman yang relevan (misal di halaman Laporan Harian setelah data ditampilkan, atau di halaman Checklist untuk menyimpan tampilan checklist).
- Event onClick tombol memanggil fungsi yang menggunakan jsPDF untuk mengkonversi elemen HTML menjadi PDF. Bisa menggunakan *html2canvas* untuk screenshot elemen lalu masukkan ke PDF, atau menggunakan *jspdf-autotable* jika datanya tabel.
- Contoh sederhana sudah disampaikan di atas: membuat dokumen PDF, menulis teks atau tabel, lalu `doc.save("nama.pdf")` untuk memicu download [14].
- Uji bahwa hasil PDF rapi (mungkin perlu styling khusus).

Migrasi Database ke PostgreSQL

Transisi dari MySQL ke PostgreSQL membutuhkan langkah-langkah berikut:

- **Pembuatan Skema di PostgreSQL**: Buat database baru (misal named `utility_reports`) dan user di PostgreSQL. Bisa menggunakan CLI:

```
CREATE DATABASE utility_reports;
CREATE USER appuser WITH ENCRYPTED PASSWORD 'appPass';
GRANT ALL PRIVILEGES ON DATABASE utility_reports TO appuser;
```

Lalu konek ke DB tersebut untuk buat tabel. Konversi tabel dari MySQL ke PostgreSQL:
- Ganti tipe auto-increment MySQL dengan `SERIAL` di PostgreSQL untuk kolom ID [3].
- Periksa tipe data lain, misal `VARCHAR(255)` tetap bisa dipakai sama. `DATETIME` MySQL diganti `TIMESTAMP` atau `TIMESTAMPTZ` di PG.
- Jika MySQL pakai engine InnoDB dengan *foreign key*, definisi FK di PostgreSQL sedikit beda (tapi konsep sama).
- Contoh tabel users di MySQL:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100) UNIQUE,
  password VARCHAR(255),
  role ENUM('admin','user') DEFAULT 'user'
);
```

Di PostgreSQL bisa menjadi:

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100) UNIQUE,
```

```
  password VARCHAR(255),
  role VARCHAR(10) DEFAULT 'user'
);
```

(PostgreSQL tidak memiliki ENUM sederhana seperti MySQL, bisa gunakan CHECK constraint atau just use text) [15].

- Migrasikan data jika perlu: ekspor data dari MySQL (CSV atau SQL dump) lalu impor ke PostgreSQL dengan penyesuaian. Jika data tidak banyak, bisa input ulang manual melalui aplikasi.

- **Konfigurasi Koneksi Aplikasi**: Ubah konfigurasi koneksi di *backend* Express dari MySQL ke PostgreSQL:

- Jika sebelumnya pakai library *mysql* atau *sequelize (dialect mysql)*, sekarang gunakan library **pg** atau ORM dengan dialect pg.
- Update *connection string* / credential di environment (misal .env file) untuk PostgreSQL.
- Periksa query di kode backend:
  - Placeholder query MySQL (`?`) harus diubah ke format $1, $2,... untuk pg ketika menggunakan `pg` library. Contoh: MySQL query `WHERE id = ?` dengan values `[id]` menjadi `WHERE id = $1` [16].
  - Fungsi khusus SQL yang beda (jika ada) sesuaikan, misal `NOW()` tetap ada di PG, tapi `DATE_FORMAT` MySQL harus diganti dengan fungsi PG (`TO_CHAR`), dll.
- Uji koneksi: Pastikan server dapat terhubung ke PG (misal log "Connected to Postgres"). Tangani error koneksi (misal wrong credentials) agar mudah debug.

Pengujian dan Penyelesaian Bug

Setelah implementasi, lakukan pengujian menyeluruh:

- **Test CRUD via Postman/Thunder Client**: Sebelum menghubungkan front-end, uji setiap endpoint secara mandiri dengan tool seperti Postman. Kirim request GET/POST/PUT/DELETE ke endpoint Express dan lihat apakah database terupdate sesuai. Ini memastikan alur back-end + database sudah benar [17]. Perbaiki jika ada respon error atau data tidak masuk DB.
- **Cek Integrasi Front-end & Back-end**: Jalankan front-end (npm run dev) dan back-end (npm start) bersama. Coba aksi di UI: login dengan user admin, tambah karyawan, lihat database apakah bertambah. Jika request gagal, lihat konsol browser dan log server:
- Error **Network/CORS**: jika terlihat pesan CORS di konsol, berarti perlu cek middleware cors di Express atau konfigurasi proxy. Pastikan `app.use(cors())` dipanggil *sebelum* route didefinisikan [1], dan kalau pakai proxy, URL fetch harus relative (tanpa domain).
- Error **404/405**: berarti endpoint atau method tidak cocok. Pastikan path dan HTTP method di front-end sesuai dengan yang di back-end.
- Error **500**: lihat error log di terminal server. Tambahkan *error handler* middleware di Express untuk menangkap error dan print `err.message` [18]. Seringkali kesalahan query (syntax SQL salah) bisa dilihat dari pesan error PG di console.
- **UI/UX**: Pastikan navigasi login->dashboard->fitur lain berfungsi. Coba login sebagai user biasa (non-admin) dan pastikan menu admin (karyawan, upload checklist) tidak muncul atau terproteksi. Coba akses langsung URL admin as user, server harus tolak (dapat 403 Forbidden dari middleware isAdmin).

- **Export PDF**: Uji tombol export PDF, apakah file terdownload dan isinya sesuai tampilan. Jika ada issue (misal layout berantakan), atur styling cetak (jsPDF autotable style, dll.).
- **Performa & Keamanan**: Untuk data sensitif (password), pastikan hash berjalan. JWT harus cukup aman (gunakan secret yang kuat, mungkin atur expire token). Gunakan Helmet middleware untuk menambah header keamanan [19]. Juga validasi input mencegah SQL injection (meski pakai parameterized query sudah aman, tetap baik sanitasi).

Kesimpulan

Dengan melakukan perombakan struktur dan kode seperti di atas, sistem informasi reporting harian karyawan Utility akan lebih terintegrasi dan stabil. Memisahkan peran front-end dan back-end dengan API REST yang jelas, serta migrasi ke PostgreSQL yang disesuaikan skemanya, akan menyelesaikan masalah CRUD yang sebelumnya gagal tersimpan. Setiap fitur inti (login, dashboard, laporan, data karyawan, checklist, export PDF) telah diimplementasikan dengan pertimbangan akses berdasarkan peran (superuser vs user biasa) dan kemudahan penggunaan. Hasil akhirnya, aplikasi diharapkan memungkinkan superuser mengelola data dan upload laporan rutin, sementara karyawan biasa dapat melihat laporan dan mengunduhnya (termasuk dalam format PDF) dengan lancar. Semua solusi di atas telah mengikuti praktik standar pengembangan web modern dan merujuk pada sumber terpercaya untuk memastikan kualitasnya. Semoga dengan ini proyek Anda dapat terselesaikan dengan baik.

Referensi:

- Tania Rascia, "CRUD REST API with Node.js, Express, and PostgreSQL", *LogRocket Blog* [7] [1] – (Contoh koneksi PostgreSQL dan solusi CORS di Express).
- GeeksforGeeks, "How to Configure Proxy in Vite?" [2] [5] – (Konfigurasi proxy Vite untuk redirect API di pengembangan, menghindari masalah CORS).
- Sospeter Mong'are, "Migrating from MySQL to PostgreSQL – Key Differences", *DEV Community* [3] – (Perbedaan sintaks MySQL vs PostgreSQL, mis. AUTO_INCREMENT vs SERIAL).
- *Express.js Documentation* – Multer Middleware [11] – (Contoh penggunaan Multer untuk file upload di Express).
- GeeksforGeeks, "Generate PDF File Using jsPDF" [12] – (Penggunaan jsPDF untuk membuat file PDF secara dinamis di aplikasi web).
- LogRocket Blog [8] [9] – (Tips implementasi autentikasi JWT dan middleware otorisasi berbasis role di Node/Express).

---

[1] [4] [6] [7] [8] [9] [13] [16] [17] [18] [19] CRUD REST API with Node.js, Express, and PostgreSQL - LogRocket Blog
https://blog.logrocket.com/crud-rest-api-node-js-express-postgresql/

[2] [5] How to Configure Proxy in Vite? - GeeksforGeeks
https://www.geeksforgeeks.org/reactjs/how-to-configure-proxy-in-vite/

[3] [15] Migrating from MySQL to PostgreSQL Key Query Differences and Considerations - DEV Community
https://dev.to/msnmongare/migrating-from-mysql-to-postgresql-key-query-differences-and-considerations-3831

[10] [11] Express multer middleware
https://expressjs.com/en/resources/middleware/multer.html

[12] [14] Generate PDF File Using jsPDF Library - GeeksforGeeks
https://www.geeksforgeeks.org/html/how-to-generate-pdf-file-using-jspdf-library/