Project Flaws Report

This document outlines identified flaws in the project, categorized by type.

1. Security Vulnerabilities (CRITICAL)

SQL Injection Risks

- **Direct SQL Injection in `/vente` POST route (app.js):** The `productIds.join(',')` used in `checkQuery` and `updateQuery` is directly vulnerable to SQL injection. Malicious input in `req.body.stock` can lead to arbitrary SQL execution.
- **SQL Injection in `/enregistrer-pharmacie` and `/dernier-enregistrement` routes (app.js):** These routes directly use a `connection` object (likely global), bypassing `req.getConnection`, and lack authentication, exposing them to unauthorized access and potential SQL injection.
- **Potential SQL Injection via `LIKE %...%` (app.js):** Routes like `/admin`, `/vente` (GET), `/verification`, and `/entree` (GET) construct dynamic queries with `LIKE ?` and wildcards (`%`). If user-provided wildcards are not properly escaped, this could lead to injection.

Insecure Password Handling

- **Plain-text password comparison in `/login` POST route (app.js):** Passwords are compared directly (`motDePasse !== utilisateur.mot_de_passe`), indicating storage or transmission in plain text. The `bcrypt` library is imported but unused. This is a critical vulnerability.

Cross-Site Scripting (XSS)

- **CRITICAL XSS in `views/entree.ejs`:** The `remplirFormulaire` JavaScript function directly embeds dynamic EJS variables into `onclick` attributes without proper JSON encoding. This allows an attacker to inject arbitrary JavaScript by manipulating input data (e.g., product names).
- **CRITICAL XSS in `views/pers.ejs`:** Similar to `entree.ejs`, the `remplirFormulaire` and `supprimerEmploye` JavaScript functions directly embed dynamic EJS variables into `onclick` attributes without proper JSON encoding, creating critical XSS vulnerabilities.

Session Security

- **Insecure `cookie` configuration (app.js):** `cookie: { secure: false }` is set in `app.js`. If the application is not consistently run over HTTPS, session cookies are vulnerable to Man-in-the-Middle attacks.
- **Weak `session` secret (app.js):** `secret: 'votre_secret_key'` is a placeholder. A strong, random, and securely stored secret (e.g., via environment variables) is required to prevent session hijacking.

Other Security Concerns

- **Lack of CSRF Protection:** No apparent Cross-Site Request Forgery (CSRF) protection implemented, making the application vulnerable to unintended actions by authenticated users.
- **Sensitive Data Logging (app.js):** `console.log("Corps de la requête reçu :", req.body);` in the `/login` route logs sensitive user input (including passwords) to the console or

server logs, posing a security risk.
- **Lack of Server-Side Validation:** Critical business logic, particularly for sales transactions (e.g., quantity checks in `/vente` POST route), relies heavily on client-side validation. This can be easily bypassed by malicious users, leading to data inconsistencies or fraud.

2. Code Quality & Best Practices

Database Interaction Inconsistencies

- **Mixed Database Drivers (app.js, package.json):** The project mixes `express-myconnection` (using the older `mysql` package) with `mysql2/promise` within `app.js`. This creates confusion, potential resource leaks, and inconsistent API usage. The `/vente` POST route specifically exemplifies this problematic mix.
- **Conflicting Database Configurations (app.js, configs/db.js):** `optionBd` is hardcoded in `app.js` with insecure defaults (e.g., empty password, `root` user, `localhost`), contradicting `configs/db.js` which correctly uses environment variables. The hardcoded configuration is used by `express-myconnection`.
- **Lack of Connection Error Handling for Pool (configs/db.js):** `configs/db.js` does not include explicit error handling for initial database connection failures or for errors that might occur during connection acquisition from the `mysql2/promise` pool.

Error Handling & Modularity

- **Scattered Error Handling (app.js):** Error handling logic is inconsistent and repetitively implemented across many routes (`if (err) { ... }`). A centralized error handling middleware would improve maintainability and consistency.
- **Monolithic `app.js`:** The `app.js` file contains a broad range of concerns (routing, authentication, database logic, business logic). This hinders modularity, readability, maintainability, and scalability. Routes and related logic should be modularized.
- **Callback Hell / Nested Callbacks (app.js):** Many routes in `app.js` use deeply nested callbacks for asynchronous database operations (e.g., in the `/` route), making the code harder to read, understand, and debug.

Frontend (EJS Templates)

- **Inline Styles (All EJS files):** Large blocks of `<style>` tags are directly embedded in all EJS templates. This is a maintainability and performance anti-pattern; CSS should be externalized into separate `.css` files.
- **Repetitive Table Structures (accueil.ejs, entree.ejs, pers.ejs, verification.ejs):** Similar table layouts are repeated across multiple templates. These could be abstracted into reusable EJS partials to reduce code duplication.
- **Form Logic in View (entree.ejs, pers.ejs):** JavaScript functions directly embedded in the templates mix presentation with behavior, making the code harder to manage.

Unused/Redundant Code

- `bcrypt` is imported in `app.js` but explicitly not used for password hashing, despite being the correct tool for the job.

- The `mysql2/promise pool` from `configs/db.js` is imported but largely unused in `app.js` for routes that utilize `express-myconnection`.
- Redundant database drivers (`mysql` and `mysql2`) are listed in `package.json`.
- `moment` and `ml-regression` are present in `package.json` but their usage is not evident in the provided codebase snippets.
- `views/facture.ejs` is an incomplete placeholder template, rendering static content and missing intended dynamic data, making it non-functional as an invoice.

Input Validation & Data Handling

- **Incomplete/Missing Input Validation:** Beyond simple presence checks, there is a general lack of comprehensive input validation (e.g., data types, formats, lengths, sanitization) for user inputs across the application.
- **`DB_PORT` Handling (configs/db.js):** `DB_PORT` is converted to a `Number` without validation that it is a valid number, potentially leading to `NaN` issues if the environment variable is missing or malformed.
- **Client-Side Date Formatting (accueil.ejs, admin.ejs, pers.ejs):** Dates are formatted client-side using JavaScript. While functional, server-side pre-formatting could ensure consistency and simplify client logic.

Usability & User Experience

- **Filter Form Persistence (admin.ejs, pers.ejs, vente.ejs, verification.ejs):** Filter forms do not retain their submitted values after page reload or submission, forcing users to re-enter search criteria and negatively impacting usability.
- **Confusing Form Input Fields (entree.ejs):** The interaction between "Choisir un produit existant" (select) and "Ou entrer un nouveau produit" (text input) is unclear, potentially leading to user confusion or incorrect data entry.
- **`select` with single `option` (vente.ejs):** The product selection dropdowns in `vente.ejs` contain only one static `<option>`, making them functionally useless for selection.

Inconsistent Naming & Hardcoding

- **Inconsistent Template Titles (entree.ejs, pers.ejs, verification.ejs):** Titles like `SIH Femmes Enceintes` appear in templates, which seems like a copy-paste error given the project's name is "SIGPP".
- **Inconsistent Variable Naming (admin.ejs):** Using `dateNaissance` to format `log.date_connection`.
- **Hardcoded Select Options (pers.ejs):** Options for `poste` and `contrat` are hardcoded in the EJS template; these should ideally be dynamically managed.
- **Misnamed Filter Field (verification.ejs):** The filter label "Voies" expects input for `voies` but the server-side code in `app.js` expects `voie`, causing the filter to not work as intended.
- **Mismatch in Invoice Display (app.js, vente.ejs):** Client-side JavaScript in `vente.ejs` attempts to redirect to `/facture/${data.idVente}`, expecting an `idVente` parameter, but `app.js` has a static `/facture` route and a commented-out dynamic one.