Technical Report: Secure Auth Platform

Author: Jason Lian **Date**: November 24, 2025 **Repository**: secure-auth-platform

1. Executive Summary

This report details the architecture, design decisions, and implementation of a secure, production-ready authentication platform. The system features a robust **Go backend** implementing RESTful APIs, a responsive **React frontend**, and a **PostgreSQL database**, all containerized for seamless deployment. Key achievements include a comprehensive test suite (Unit, Integration, E2E), adherence to security best practices (Argon2id, CSP, Rate Limiting), and a clean, maintainable codebase following SOLID principles.

2. Architecture & Design

2.1 Backend Architecture (Clean Architecture)

The backend follows **Clean Architecture** principles to ensure separation of concerns and testability:

- **Handler Layer** (`internal/api/handlers`): Manages HTTP requests/responses and input validation.
- **Service Layer** (`internal/service`): Contains business logic (e.g., auth flows, user registration) and is decoupled from the database.
- **Repository Layer** (`internal/repository`): Handles data persistence using **pgx** for high-performance PostgreSQL interaction.
- **Domain Layer** (`internal/models`): Defines core entities and interfaces.

Why this approach? * **Testability**: Dependencies (Repositories, Services) are injected via interfaces, allowing for easy mocking in unit tests. * **Flexibility**: The database implementation can be swapped without affecting business logic.

2.2 Frontend Architecture

The frontend is built with **React** and **TypeScript**, utilizing modern patterns:

- **Context API**: A dedicated `AuthContext` manages global authentication state, ensuring consistent user experience across the app.
- **Protected Routes**: A `ProtectedRoute` component acts as a guard, redirecting unauthenticated users from private pages (`/me`).
- **OpenAPI Client**: An auto-generated API client guarantees type safety and synchronization with the backend contract.

2.3 Database Schema

A normalized PostgreSQL schema ensures data integrity: * **Users Table**: Stores `id` (UUID), `email` (Unique, Indexed), `password_hash`, and timestamps.

* **Security**: Sensitive data is never returned in API responses.

3. Security Implementation

Security was a primary focus throughout development:

1. **Password Storage**: Implemented **Argon2id** hashing, the winner of the Password Hashing Competition, providing superior resistance to GPU/ASIC attacks compared to bcrypt.
2. **Authentication**: Stateless **JWT (JSON Web Tokens)** authentication.
   - Tokens are signed with a secure secret.
   - Frontend stores tokens in `localStorage` (with potential upgrade path to HttpOnly cookies).
3. **Rate Limiting**: Middleware implements a Token Bucket algorithm to limit request rates per IP, mitigating brute-force and DDoS attacks.
4. **Secure Headers**:
   - **HSTS**: Enforces HTTPS connections.
   - **X-Content-Type-Options**: Prevents MIME sniffing.
   - **X-Frame-Options**: Prevents clickjacking.
   - **CSP**: Strict Content Security Policy to mitigate XSS.
5. **Graceful Shutdown**: The server handles `SIGINT/SIGTERM` signals to ensure in-flight requests complete before shutting down.

4. Quality Assurance & Testing

A multi-layered testing strategy ensures reliability:

- **Backend Unit Tests**: Validates individual components (Services, Handlers) using **Testify** and mocks.
- **Database Integration Tests**: Verifies actual database interactions using a test container or isolated DB instance.
- **Frontend Unit Tests**: Tests React components and hooks using **Vitest** and **React Testing Library**.
- **E2E Tests**: **Playwright** tests run in a Dockerized environment, simulating real user flows (Sign Up -> Sign In -> Profile -> Logout) and verifying accessibility (WCAG compliance).

5. Future Improvements

1. **Refresh Tokens**: Implement a refresh token rotation mechanism using HttpOnly cookies to enhance security and user session management.
2. **Distributed Rate Limiting**: Migrate from in-memory rate limiting to **Redis** to support horizontal scaling across multiple backend replicas.
3. **MFA**: Add Multi-Factor Authentication (TOTP) for an additional layer of security.
4. **Email Verification**: Implement email verification flow (send link with token) to verify user identity upon registration.

5. **Monitoring**: Integrate **Prometheus** and **Grafana** for real-time metrics and alerting.

6. Conclusion

The Secure Auth Platform demonstrates a professional, scalable, and secure approach to modern web application development. By combining strict architectural patterns with rigorous testing and security controls, it provides a solid foundation for production deployment.