Database Options for FIA Cybersecurity App

Overview

As part of Sprint 2, we evaluated three database solutions that integrate easily with Visual Studio Code and can support future deployment of the FIA Cybersecurity App for Women. Our goal was to move beyond our initial XML files toward a more scalable, relational store for users, roles, and permissions. The options considered were:

1. **PostgreSQL** – used either in a self-hosted (containerized) form or via a managed service.
2. **Supabase Postgres** – a hosted service built on Postgres, adding built-in authentication, REST and GraphQL APIs, and row-level security.
3. **SQLite locally with a migration path to Azure SQL** – begin with a lightweight local database then migrate to a managed Azure SQL database when we deploy.

This document summarizes the key features of each candidate – reliability, JSON support, migration tooling, hosting and cost, and access control mechanisms – and provides a high-level sample schema.

Evaluation Criteria and Findings

Feature	PostgreSQL (Managed or Containerized)	Supabase Postgres	SQLite → Azure SQL
ACID Reliability	Postgres has been ACID-compliant since 2001, offering reliable transaction support for complex workloads[1].	Supabase uses a full Postgres database, so it shares the same ACID properties[2].	SQLite transactions are ACID-compliant[3]; Azure SQL is fully ACID and durable.
JSON Support	Supports both `JSON` and `JSONB`; the latter stores JSON as a binary type for efficient search and indexing[4].	Because Supabase is built on Postgres, it inherits `JSONB` and full JSON functions[5].	SQLite stores JSON as text – there is no native JSON type[6]. Azure SQL now offers a native `json` data type that stores JSON in binary format for efficient reads and writes[7] and is generally available in Azure SQL Database and Managed Instance[8].
Migration Tooling	Works seamlessly with Entity	Supabase provides automatic migration	SQLite supports EF Core and Flyway for migrations;

Feature	PostgreSQL (Managed or Containerized)	Supabase Postgres	SQLite → Azure SQL
	Framework (EF) Core migrations; also supported by Prisma and Flyway.	helpers and can be used with EF Core or Prisma, though not all features are supported (e.g., triggers).	migrating to Azure SQL later would require a data transfer script but benefits from EF Core/Prisma tooling.
Hosting & Cost	Self-host in Docker (free beyond hosting resources) or use managed services such as Supabase, Heroku, or AWS RDS; costs vary based on size and throughput.	Supabase includes a managed Postgres database with free and paid tiers, plus built-in auth and API layers[2].	SQLite is free for local development. Azure SQL has multiple pricing tiers with free development options, scaling to production as needed. Hosted SQLite is not practical for production, so migration is required.
Role-Based Access Control	Postgres supports robust role and privilege management. Row-level security (RLS) can enforce per-record access rules[9].	Supabase exposes row-level security policies and uses Postgres roles/GRANTs to secure the auto-generated REST API[10].	SQLite has no built-in RBAC; enforcement must be done in application code. Azure SQL supports roles and GRANT/DENY syntax for fine-grained permissions[11].

Additional Considerations

- **Managed vs. Hosted:** Running Postgres yourself via Docker gives complete control but also more operational burden. Supabase handles backups, patching, and provides extras like authentication and real-time APIs out of the box. Migrating from SQLite to Azure SQL allows simple local development and a path to a managed service when needed.
- **Tooling Support:** All three candidates work with modern ORMs like Entity Framework Core. Prisma (JavaScript/TypeScript) also supports Postgres and SQLite; Flyway can manage schema versions across all options.
- **Development Experience:** Postgres and Supabase provide strong introspection and management tools, plus row-level security. SQLite offers simplicity and zero configuration but lacks advanced features until migrated.

Sample Schema for Users and Roles

All three databases can represent a simple role-based access model with a junction table for many-to-many relationships:

```
-- Users table
CREATE TABLE users (
  id UUID PRIMARY KEY,
  username VARCHAR(50) NOT NULL UNIQUE,
  email VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  is_deleted BOOLEAN NOT NULL DEFAULT FALSE
);

-- Roles table
CREATE TABLE roles (
  id UUID PRIMARY KEY,
  name VARCHAR(50) NOT NULL UNIQUE,
  description TEXT,
  is_deleted BOOLEAN NOT NULL DEFAULT FALSE
);

-- Junction table linking users to roles
CREATE TABLE user_roles (
  user_id UUID REFERENCES users(id),
  role_id UUID REFERENCES roles(id),
  assigned_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (user_id, role_id)
);

-- Example: soft delete flag (is_deleted) allows records to be marked
inactive without removal.
```

Summary

PostgreSQL and Supabase are well-suited for the FIA Cybersecurity App. Both provide full ACID compliance, native JSON/JSONB support, powerful role-based access control with row-level security, and rich migration tooling[1][4][9]. Supabase adds authentication and auto-generated REST APIs out of the box[10], which reduces implementation work. SQLite is attractive for initial prototyping due to its simplicity and zero-config setup, but its JSON support is limited[6] and it lacks native RBAC. Migrating to Azure SQL provides a managed environment with a native JSON data type[7][8] and robust role management[11]. Given these considerations, a managed Postgres solution (either self-hosted or via Supabase) is likely the best fit for our next prototype, with the SQLite → Azure SQL path serving as an optional lightweight entry point.

---

[1] [4] [9] PostgreSQL: About

https://www.postgresql.org/about/

[2] Supabase | The Postgres Development Platform.

https://supabase.com/

[3] Features Of SQLite

https://sqlite.org/features.html

[5] [10] REST API | Supabase Docs

https://supabase.com/docs/guides/api

[6] JSON Functions And Operators

https://sqlite.org/json1.html

[7] [8] JSON Data Type - SQL Server | Microsoft Learn

https://learn.microsoft.com/bg-bg/sql/t-sql/data-types/json-data-type

[11] Database-level roles - SQL Server | Microsoft Learn

https://learn.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles