**Sperm Bank Security & Accountability — Demo App (HTML+ CSS+JS + Firebase + MUI + Next.js)**

Implements email/password auth, optional **passkey (WebAuthn)** MFA, donor records, a sperm retrieval confirmation flow, and an **audit trail**.

**Table of Contents**

* [Features](#features)
* [Tech Stack](#tech-stack)
* [Prerequisites](#prerequisites)
* [Getting Started](#getting-started)
* [Environment Variables](#environment-variables)
* [Firebase Setup](#firebase-setup)
* [Desktop-only Enforcement](#desktop-only-enforcement)
* [Project Structure](#project-structure)
* [Data Model (Firestore)](#data-model-firestore)
* [Available Pages](#available-pages)
* [API Routes](#api-routes)
* [Scripts](#scripts)
* [How It Works](#how-it-works)
* [Testing Checklist](#testing-checklist)
* [Troubleshooting](#troubleshooting)
* [Security Notes](#security-notes)
* [License](#license)

**Features**

* **Desktop-only access**
  + Middleware blocks mobile/tablet user-agents and rewrites to /unsupported.
  + Optional client guard as a second safety net.
* **Authentication**
  + Firebase **Email/Password**.
  + **Passkeys (WebAuthn)**: register, authenticate, stored per user.
  + Settings page: view account, send password reset, manage passkeys.
* **Donor Records**
  + List donors (MUI DataGrid), **Create / Edit / Delete**.
* **Sperm Retrieval Workflow**
  + Minimal confirmation dialog; on confirm, logs an **audit** event.
* **Audit Logging**
  + Client helper posts to /api/audit/log.
  + Logs viewer (MUI DataGrid) with newest first.
  + Log fields: uid, email, action, details, ip, userAgent, env, tsMillis.

**Tech Stack**

* **Frontend**: HTML, CSS, JS, TypeScript, MUI / MUI X DataGrid, Next.js (App Router)
* **Auth & DB**: Firebase Web SDK (Auth, Firestore)
* **Server (API routes)**: Firebase **Admin SDK**
* **WebAuthn**: @simplewebauthn/browser, @simplewebauthn/server

**Prerequisites**

* Node.js 18+ and npm
* Firebase project (Auth + Firestore enabled)
* Service Account (for Admin SDK)

**Getting Started**

# 1) Install dependencies

npm install

# 2) Create .env.local (see below)

# 3) Start dev

npm run dev

# 4) Build & start

npm run build

npm run start

If ESLint blocks your build, you can temporarily use:

// package.json

"build": "next build --no-lint"

**Environment Variables**

Create **.env.local** at the project root:

# Firebase (client)

NEXT\_PUBLIC\_FIREBASE\_API\_KEY=...

NEXT\_PUBLIC\_FIREBASE\_AUTH\_DOMAIN=...

NEXT\_PUBLIC\_FIREBASE\_PROJECT\_ID=...

NEXT\_PUBLIC\_FIREBASE\_STORAGE\_BUCKET=...

NEXT\_PUBLIC\_FIREBASE\_MESSAGING\_SENDER\_ID=...

NEXT\_PUBLIC\_FIREBASE\_APP\_ID=...

# Firebase Admin (server)

FIREBASE\_PROJECT\_ID=...

FIREBASE\_CLIENT\_EMAIL=...@...iam.gserviceaccount.com

# If your key has line breaks, escape them or wrap in quotes

FIREBASE\_PRIVATE\_KEY="-----BEGIN PRIVATE KEY-----\nABC...\n-----END PRIVATE KEY-----\n"

# WebAuthn

NEXT\_PUBLIC\_WEBAUTHN\_ORIGIN=http://localhost:3000

NEXT\_PUBLIC\_WEBAUTHN\_RP\_ID=localhost

**Production tips**

* Use your HTTPS origin for NEXT\_PUBLIC\_WEBAUTHN\_ORIGIN.
* Use your effective domain for NEXT\_PUBLIC\_WEBAUTHN\_RP\_ID (e.g., example.edu).
* Add the domain to Firebase **Authorized Domains**.

**Firebase Setup**

1. Create a Firebase project.
2. **Authentication → Sign-in methods**: enable **Email/Password**.
3. **Firestore**: enable (start in test mode for demo).
4. **Service Account**: Project settings → Service accounts → copy credentials into .env.local.

**Demo Firestore rules** (simple; tighten for production):

rules\_version = '2';

service cloud.firestore {

match /databases/{database}/documents {

// Donors - let authenticated users read/write (demo)

match /donors/{donorId} {

allow read, write: if request.auth != null;

}

// Audit logs - read allowed; writes via Admin SDK (API route only)

match /auditLogs/{logId} {

allow read: if request.auth != null;

allow write: if false;

}

// WebAuthn creds - user can read/delete own credentials

match /webauthnCredentials/{uid}/creds/{credId} {

allow read, delete: if request.auth != null && request.auth.uid == uid;

allow create, update: if false;

}

// WebAuthn challenges - API only

match /webauthnChallenges/{uid} {

allow read, write: if false;

}

}

}

**Desktop-only Enforcement**

**Server-side (src/middleware.ts)**

* Checks User-Agent; mobile/tablet → rewrite to /unsupported.
* Excludes /\_next/\*, /api/\*, etc.

**Client-side (optional)**

* A small DesktopOnly component double-checks UA and shows a message if not desktop.

**Project Structure**

src/

app/

page.tsx # Login

mfa/page.tsx # Passkey MFA (optional)

unsupported/page.tsx # For mobile/tablet users

dashboard/

layout.tsx # App shell (MUI AppBar/Drawer)

page.tsx # Overview cards

donors/page.tsx # Donor CRUD (DataGrid + dialog)

retrievals/page.tsx # Sperm retrieval confirm + audit

logs/page.tsx # Audit logs viewer (DataGrid)

settings/page.tsx # Account + passkeys

api/

audit/log/route.ts # POST audit events

webauthn/

register-passkey/

options/route.ts # Attestation options

route.ts # Verify attestation + save credential

authenticate-passkey/

options/route.ts # Assertion options

verify-authentication/

route.ts # Verify assertion

components/

DesktopOnly.tsx

DashboardShell.tsx

context/

AuthContext.tsx # Firebase auth state provider

lib/

firebase.ts # Firebase client init

firebaseAdmin.ts # Admin SDK init

audit.ts # Client audit helper

webAuthCodec.ts # base64url <-> Buffer helpers

middleware.ts # Desktop-only check

**Data Model (Firestore)**

* **donors/{donorId}**  
  name, bloodType, status, createdAt, etc. (demo fields)
* **auditLogs/{autoId}**  
  uid, email, action, details, ip, userAgent, env, tsMillis
* **webauthnChallenges/{uid}**  
  registrationChallenge **or** authenticationChallenge, createdAt
* **webauthnCredentials/{uid}/creds/{credentialID}**  
  credentialID (base64url), credentialPublicKey (base64url), counter, fmt, createdAt

**Available Pages**

* / — Login (Email/Password; optional passkey flows)
* /dashboard — Overview
* /dashboard/donors — Donors CRUD
* /dashboard/retrievals — Sperm retrieval confirm (with audit logging)
* /dashboard/logs — Audit logs viewer
* /dashboard/settings — Account + passkeys management
* /unsupported — Shown to non-desktop devices

**API Routes**

* POST /api/audit/log — write an audit entry (verifies Firebase ID token)
* POST /api/webauthn/register-passkey/options — get registration options
* POST /api/webauthn/register-passkey — verify attestation & store credential
* POST /api/webauthn/authenticate-passkey/options — get authentication options
* POST /api/webauthn/verify-authentication — verify assertion & update counter

**Scripts**

{

"scripts": {

"dev": "next dev",

"lint": "next lint",

"build": "next build", // optionally: "next build --no-lint"

"start": "next start"

}

}

**How It Works**

**Authentication & MFA**

* **Email/Password** via Firebase Auth.
* **Passkeys (WebAuthn)**:
  + **Registration**: client gets options → startRegistration → server verifies and stores the credential’s public key + counter.
  + **Authentication**: client gets options (server builds allowCredentials) → startAuthentication → server verifies signature and updates counter.

**Audit Logging**

* Client calls logAudit(action, details?) with current ID token.
* Server (/api/audit/log) verifies token, captures ip, userAgent, and writes a record to auditLogs.

**Donor CRUD**

* Basic create/edit/delete UI using MUI DataGrid and dialogs.
* On changes, donor\_update audit events are recorded.

**Sperm Retrieval Flow**

* Operator chooses donor → confirm dialog → record sperm\_retrieval\_confirmed with timestamp and donor reference.

**Testing Checklist**

1. **Desktop-only**
   * Open from phone → redirected to /unsupported.
   * Desktop works normally.
2. **Auth**
   * Create account (email/password), sign in/out.
   * Settings → **Add Passkey** (register).
   * (If wired) sign in with passkey or use step-up verification.
3. **Donors**
   * Add donor → appears in list.
   * Edit donor → persists.
   * Delete donor → removed.
4. **Sperm Retrieval**
   * Confirm retrieval → success toast and audit entry.
5. **Audit Logs**
   * /dashboard/logs shows entries with newest first.
6. **Settings**
   * View account info, send password reset.
   * List existing passkeys; add/delete one.

**Troubleshooting**

* **Build fails due to ESLint**
  + Temporarily use next build --no-lint, or relax rules in .eslintrc.json:
  + {
  + "extends": ["next/core-web-vitals", "eslint:recommended", "plugin:@typescript-eslint/recommended"],
  + "rules": {
  + "@typescript-eslint/no-explicit-any": "off",
  + "@typescript-eslint/no-empty-object-type": "off",
  + "@typescript-eslint/ban-ts-comment": "off",
  + "react/display-name": "off",
  + "@typescript-eslint/no-unused-vars": ["warn", { "argsIgnorePattern": "^\_" }]
  + }
  + }
* **WebAuthn “wrong RP ID/origin”**
  + NEXT\_PUBLIC\_WEBAUTHN\_ORIGIN must exactly match your site origin (e.g., http://localhost:3000).
  + NEXT\_PUBLIC\_WEBAUTHN\_RP\_ID must be the domain (e.g., localhost in dev).
* **“challenge undefined”**
  + Always call the corresponding /options endpoint first and pass its response **unchanged** into startRegistration/startAuthentication.
* **“Credential not found” during auth**
  + Ensure registration stored a doc at webauthnCredentials/{uid}/creds/{credentialID} (ID equals base64url credentialID).

**Security Notes**

This demo illustrates MFA and auditability patterns relevant to HIPAA-style environments but is **not** production-ready. For real deployments consider:

* Role-based access control & custom claims
* Tamper-evident, append-only audit logging
* Field-level encryption for PII
* Retention policies & automated cleanup (Cloud Functions)
* Monitoring/alerting on suspicious activity
* Comprehensive WebAuthn UX fallbacks and recovery