FAMILY ASSET REGISTRY - TECHNICAL SPEC (JavaScript Version)

0. Tech Stack
- Next.js 14 (App Router)
- MongoDB with Mongoose
- JavaScript only (no TypeScript)
- Authentication using JWT stored in HTTP-only cookie
- Passwords hashed with bcrypt
- Server Components by default, Client Components only where necessary (forms, interactive pages)

1. Purpose of the App
This is a private internal dashboard for one family.

We want to track every asset:
- Land / plots
- Houses / apartments
- Vehicles
- Business shares / other valuables

For each asset we store:
- What it is (title, type, description)
- Where it is (location)
- Who owns how much (owners, with %)
- Legal status (clean or in dispute)
- How and when it was acquired
- Attached supporting documents (scan URLs)
- History timeline of actions (tax paid, mutation done, etc.)

We also track all "people" in the family and link assets to them.

There are only two user roles:
- admin: can add/edit/delete
- viewer: read-only

2. Pages / Routes

/login
- Public
- Login form with email and password
- Calls POST /api/auth/login
- On success -> redirect to /dashboard

/dashboard
- Protected (auth required)
- Shows summary cards:
  - Total assets
  - Assets in dispute
  - Total people
- Shows "recent assets" list (last 5 added)

/assets
- Protected
- Table of all assets:
  - Title
  - Type
  - City / Area
  - Status
  - Owners (names + %)
- If current user is admin, show "Add Asset" button (form)

/assets/[id]
- Protected
- Full detail of one asset:
  - Basic info
  - Location
  - Ownership breakdown
  - Acquisition info
  - Dispute info
  - History timeline
  - Attached documents
- If role is admin, show "Edit" button placeholder

/people
- Protected
- Table of all people:
  - Full name
  - Relation to family
  - CNIC (optional)
  - Status (alive / deceased)
- If role is admin, show "Add Person" form button

3. Auth / Security

Roles
- admin
- viewer

Session
- On successful login:
  - Create a JWT with {_id, role, email}
  - Sign it with process.env.JWT_SECRET
  - Store it in an HTTP-only cookie, name: family_assets_session
  - Cookie: sameSite=strict, secure in production, maxAge 7 days

Current User
- getCurrentUser() helper:
  - Reads and verifies the cookie
  - Looks up user in DB
  - Returns {_id, fullName, email, role} or null
  - Used by server components and API routes

Route Protection
- middleware.js:
  - Allow public: /login, /api/auth/login, and Next.js static assets
  - Block everything else if no valid session cookie
  - If blocked, redirect to /login
- Server components and API routes must still call getCurrentUser() to validate and enforce role.

4. Environment Variables (.env.local)

MONGODB_URI="mongodb+srv://<username>:<password>@<cluster>.mongodb.net/family_assets_db"
JWT_SECRET="<long random string at least 64 chars>"
NODE_ENV="development"

- MONGODB_URI: Atlas or local
- JWT_SECRET: long random string
- NODE_ENV: standard

5. Directory / File Structure

```
project-root/
  .env.local
  next.config.js
  package.json
  middleware.js

  lib/
    db.js
    auth.js

  models/
    User.js
    Person.js
    Asset.js

  app/
    globals.css
    layout.js

    login/
      page.js
      LoginForm.js

    dashboard/
      page.js

    assets/
      page.js
      AddAssetForm.js
      [id]/
```

```
    page.js

  people/
    page.js
    AddPersonForm.js

  api/
    auth/
      login/route.js
    assets/
      route.js
    assets/[id]/
      route.js
    people/
      route.js
```

6. MongoDB Connection Helper (lib/db.js)

- Use mongoose
- Use a cached global connection to avoid multiple connections in dev
- Export connectDB()
- Throw helpful error if MONGODB_URI is missing

Example pseudocode:

```
import mongoose from "mongoose";

let cached = global._mongooseConn;
if (!cached) {
  cached = global._mongooseConn = { conn: null, promise: null };
}

export async function connectDB() {
  if (cached.conn) return cached.conn;
  if (!cached.promise) {
    if (!process.env.MONGODB_URI) {
      throw new Error("MONGODB_URI not set in .env.local");
    }
    cached.promise = mongoose
      .connect(process.env.MONGODB_URI, { dbName: "family_assets_db" })
      .then((m) => m);
  }
  cached.conn = await cached.promise;
  return cached.conn;
}
```

7. Mongoose Models

7.1 models/User.js
Users who can log in.

Fields:
- fullName: String, required
- email: String, required, unique, lowercase
- passwordHash: String, required (bcrypt hash)
- role: "admin" | "viewer" (default "viewer")
- timestamps: true

7.2 models/Person.js
Represents a person that may hold ownership.

Fields:
- fullName: String, required
- fatherName: String (optional)
- cnic: String (optional)
- relationToFamily: String, example "father", "sister", "uncle"
- status: "alive" | "deceased", default "alive"
- notes: String (optional)
- timestamps: true

7.3 models/Asset.js
Main asset model. Includes embedded subdocuments.

Top-level fields:

assetType:
- "land_plot"
- "house"
- "apartment"
- "vehicle"
- "business_share"
- "other"

title: String, required
  Example: "I-8 Plot 432", "White Corolla 2020"

description: String (optional)

location: {
  country: String
  city: String
  areaOrSector: String   // e.g. "I-8/3", "DHA Phase 4", "Village Dhudial"
  addressDetails: String   // plot number, khasra number etc.
  geoCoordinates: {
    lat: Number
    lng: Number
  }
}

currentStatus:

- "clean"
- "in_dispute"
- "under_transfer"
- "sold_but_not_cleared"
- "unknown"
default "clean"

owners: Array of:
{
  personId: ObjectId -> Person (required)
  percentage: Number (e.g. 50, 100) required
  ownershipType: String ("legal owner", "inherited", "benami", etc.)
}

acquisitionInfo: {
  acquiredDate: Date
  acquiredFrom: String
  method:
    "purchased"
    "gifted"
    "inherited"
    "transferred"
    "settlement"
    "other"
  priceOrValueAtAcquisition: Number
  notes: String
}

disputeInfo: {
  isInDispute: Boolean, default false
  type: String // "family dispute", "court case"
  startedDate: Date
  details: String
  lawyerName: String
  caseNumber: String
  nextHearingDate: Date
}

documents: Array of:
{
  label: String, required // "Registry Deed"
  fileUrl: String, required // link to file in private storage
  notes: String
  uploadedAt: Date (default now)
}

history: Array of:
{
  date: Date (default now)
  action: String, required // "Acquired", "Mutation done"

  details: String
}

tags: [String] // e.g. ["ancestral", "rental", "village"]

timestamps: true

8. Auth Helpers (lib/auth.js)

Responsibilities:
- Sign and verify JWT
- Set / clear cookie
- Return current user

Important details:
- Cookie name: family_assets_session
- Cookie httpOnly, sameSite strict, secure in production, maxAge 7 days

Functions:

signJWT(payload)
- uses process.env.JWT_SECRET
- expiresIn "7d"

verifyJWT(token)
- uses process.env.JWT_SECRET

createSession(userId, role, email)
- sign JWT with {_id, role, email}
- set cookie family_assets_session with httpOnly etc.

destroySession()
- clear cookie

getCurrentUser()
- read cookie from next/headers cookies()
- verify JWT
- fetch user from DB
- return {_id, fullName, email, role}
- return null if invalid

9. middleware.js

Behavior:
- Protect everything except:
  - /login
  - /api/auth/login
  - static assets like /_next and favicon
- If user is not logged in (no cookie family_assets_session):
  - redirect to /login

Server components and API routes must still call getCurrentUser() to do final
role checks and to enforce admin-only actions.

10. API Route Handlers (App Router, route.js files)

All handlers live in app/api/.../route.js.
Use Next.js Request and Response.

10.1 POST /api/auth/login
- Body: { email, password }
Steps:
1. connectDB()
2. find user by email
3. compare provided password with user.passwordHash using bcrypt.compare
4. if match:
   - createSession(user._id, user.role, user.email)
   - return JSON with _id, fullName, role, email
5. else return 401

10.2 GET /api/people
- Must be authenticated
- Returns list of all people:
  {
    _id,
    fullName,
    relationToFamily,
    status,
    cnic
  }
- Sort by fullName ascending

10.3 POST /api/people
- Must be authenticated
- Must be role admin
- Body:
  {
    fullName (required),
    fatherName?,
    cnic?,
    relationToFamily?,
    status? ("alive" or "deceased"),
    notes?
  }
- Creates a Person document and returns { _id }

10.4 GET /api/assets
- Must be authenticated
- Returns list for table view:
  {

_id,
title,
assetType,
currentStatus,
location: { city, areaOrSector },
owners: [
{ personName, percentage }
]
}
- Should populate owners.personId to get personName

10.5 POST /api/assets
- Must be authenticated
- Must be role admin
- Creates an Asset
- Request body can include:
assetType,
title,
description,
location { country, city, areaOrSector, addressDetails },
currentStatus,
owners [{ personId, percentage, ownershipType }],
acquisitionInfo { acquiredDate, acquiredFrom, method, priceOrValueAtAcquisition, notes },
disputeInfo { isInDispute, type, startedDate, details, lawyerName, caseNumber, nextHearingDate },
tags [...]
- Returns { _id }

10.6 GET /api/assets/[id]
- Must be authenticated
- Returns full expanded asset document:
{
_id,
assetType,
title,
description,
location,
currentStatus,
owners: [
{
personId,
personName,
percentage,
ownershipType
}
],
acquisitionInfo,
disputeInfo,
documents,
history,
tags,

```
    createdAt,
    updatedAt
  }
```
- Populate owners.personId to get personName

11. Frontend Components / Pages

app/layout.js
- Global layout with <html> and <body>
- Can include a top nav or sidebar if user is logged in
- Nav links: Dashboard, Assets, People

app/login/page.js
- Client component
- Renders a login form
- On submit:
  - fetch("/api/auth/login", { method: "POST", body: JSON.stringify({email, password}) })
  - If status 200 -> window.location.href = "/dashboard"
  - Else show error message

app/dashboard/page.js
- Server component
- Calls getCurrentUser()
- Queries DB:
  - total assets count
  - total people count
  - assets in dispute count (disputeInfo.isInDispute === true)
- Renders stat cards
- Shows last ~5 assets (title, status)

app/assets/page.js
- Server component
- Calls getCurrentUser()
- Fetches /api/assets or reads DB directly
- Renders table with columns:
  Title | Type | Location | Status | Owners
- If currentUser.role === "admin":
  - Render <AddAssetForm /> (client component)

AddAssetForm.js
- Client component
- Form fields:
  - title
  - assetType
  - city
  - areaOrSector
  - status
  - owners array (basic for MVP: manual personId + percentage)
- On submit do POST /api/assets
- After success, reload page

app/assets/[id]/page.js
- Server component
- Fetch /api/assets/[id]
- Render sections:
  - Basic info (title, description, type, status)
  - Location block
  - Ownership table (Name | % | ownershipType)
  - Acquisition info
  - Dispute info
  - Documents list (label + fileUrl)
  - History timeline (date, action, details)

app/people/page.js
- Server component
- Fetch /api/people
- Render table:
  Full Name | Relation | CNIC | Status
- If currentUser.role === "admin":
  - Render <AddPersonForm />

AddPersonForm.js
- Client component
- Form fields:
  - fullName
  - fatherName
  - cnic
  - relationToFamily
  - status (alive/deceased)
  - notes
- On submit:
  - POST /api/people
  - Reload page

12. Seeding Initial Admin User (manual script)
We will NOT build signup UI.

Create a one-time Node script (not in production runtime) e.g. scripts/createAdmin.js:
- connect to DB
- hash a password using bcrypt.hash(password, saltRounds)
- insert:

```
{
  fullName: "Your Name",
  email: "your@email.com",
  passwordHash: "<hashed>",
  role: "admin"
}
```

After seeding, delete or keep script private.

13. Future Ideas (Not required MVP)
- File uploads for "documents" using S3 or Google Drive or local private folder
- Reminder system for hearings, tax renewals, token renewals, etc.
- Edit and delete functionality for assets and people
- Audit log of changes (who edited what and when)
- Export to PDF for inheritance/legal discussions

END OF SPEC