Government Auction Tender - Project Codebase Overview

1. System Architecture

Type: REST API Backend
Framework: FastAPI (Python)
Database: PostgreSQL (Production/NeonDB) / SQLite (Local connection fallback)
Validation: Pydantic

The application follows a standard modular FastAPI structure, separating concerns into models, schemas, routers, and database configuration.

2. Directory Structure & Module Breakdown

app/main.py:
Entry point. Initializes FastAPI app and handles database table creation via lifespan events.

app/db.py:
Manages database sessions. Configured with fallback: uses NeonDB (PostgreSQL) by default, but falls back to SQLite if the remote DB is unreachable and 'sqlite' URL is set.

app/models.py:
SQLAlchemy ORM models:
- User: Stores credentials and roles (admin/bidder)
- Item: Auction items
- Bid: Bids placed on items

app/schemas.py:
Pydantic models for data validation (UserSignin, Post_new_item, etc.)

app/auth.py:
API Routers for authentication:
- POST /signin: Registers users
- POST /login: Authenticates users

app/settings.py:
Configuration loader. Reads .env.local for secrets.

app/utils.py:
Password hashing utilities using passlib (Argon2).

3. Key Logic Flows

Database Connection:
The settings module loads the DATABASE_URL. db.py creates an engine. main.py creates tables on startup.

Authentication:
1. User provides email/password.
2. schemas. UserSignin validates format.
3. auth.create_user hashes password and saves to DB.
4. auth.login verifies password hash for login.

4. Current Status

State: Functionally correct and bug-free.
Configuration: Currently set to use NeonDB (PostgreSQL) in app/.env.local.
Known Issue: User network blocks NeonDB DNS. Verified workaround: Google DNS (8.8.8.8) or switching to local SQLite in .env.local.