Backend Planning: User Authentication & Local Data Storage

1. Architecture Overview

Backend Style:

- Lightweight backend using **Python + Postgre Sql**
- Optional **FastAPI** layer for scalability and async support
- Authentication logic separated from UI (Streamlit)

Why SQLite?

- Local storage
- Zero configuration
- Ideal for capstone and offline academic tools
- Easily upgradeable to PostgreSQL later

2. Technology Stack

Layer	Technology
Backend Logic	Python
API Layer (Optional)	FastAPI
Database	**Postgre Sql**
ORM	SQLAlchemy
Authentication	bcrypt / passlib
Session Handling	Streamlit session state

Security	Password hashing + input validation

3. Database Design (Postgre Sql)

3.1 Users Table

```
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    institution TEXT,
    role TEXT DEFAULT 'user',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3.2 User Sessions (Optional)

```
CREATE TABLE user_sessions (
    session_id TEXT PRIMARY KEY,
    user_id INTEGER,
    login_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY(user_id) REFERENCES users(user_id)
);
```

3.3 User Activity / History (Optional)

```
CREATE TABLE user_activity (
    activity_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    activity_type TEXT,
    activity_data TEXT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY(user_id) REFERENCES users(user_id)
);
```

4. Authentication Workflow

🔑 Registration Flow

1. User submits **name, email, password, institution**
2. Backend validates input
3. Password hashed using **bcrypt**
4. User info stored in SQLite
5. Success message returned

🔓 Login Flow

1. User enters email + password
2. Fetch user record from **Postgre Sql**
3. Verify password hash
4. Store login status in **Streamlit session state**
5. Redirect to dashboard

🔒 Logout Flow

1. Clear session state
2. Optional session deletion from DB

5. Password Security Strategy

Feature	Implementation
Hashing	bcrypt / passlib
Plain Password Storage	❌ Never
Salted Hash	☑ Yes
Brute-force Protection	Login attempt limit (optional)

Example (conceptual):

hash = bcrypt.hashpw(password.encode(), bcrypt.gensalt())

6. Backend Module Structure

```
backend/
│
├── database/
│   ├── db.py              # Postgre Sql connection
│   ├── models.py          # SQLAlchemy models
│
├── auth/
│   ├── register.py        # Registration logic
│   ├── login.py           # Login validation
│   ├── security.py        # Password hashing
│
├── services/
│   ├── user_service.py    # CRUD operations
│
└── utils/
    ├── validators.py      # Email, password validation
```

7. Streamlit Integration

Session Management

```
if "user" not in st.session_state:
    st.session_state.user = None
```

Login State Control

- Show login page if user is None
- Show dashboard if user is authenticated
- Prevent unauthorized access to feature

8. Role-Based Access (Optional)

Role	Permissions
User	Upload, chat, write, cite

Admin	View all users, usage stats
Faculty	Review student work

Stored in `users.role`

9. Data Stored Per User

Data Type	Storage
Profile Info	SQLite
Uploaded Papers	Local filesystem + DB reference
Chat History	SQLite (JSON/Text)
Citations Library	SQLite
Notes & Drafts	SQLite

10. Future Upgrade Path

Feature	Upgrade
Multi-user deployment	PostgreSQL
Token-based auth	JWT
OAuth login	Google / ORCID
Cloud deployment	AWS / Railway

Encryption at rest	SQLCipher