**🧩 Registration Tab – What You Need to Build**

**1. Frontend (React.js)**

* **Webcam Feed**: Use a package like react-webcam to access the webcam.
* **Name Input**: Simple input field where user enters their name.
* **Capture Button**: Captures the current frame from the webcam.
* **POST Request**: Sends the image + name to your Python backend.

Example UI Components:

* Webcam live view
* Input: Name
* Button: Register Face

**2. Backend (Python API for Face Recognition)**

Use **FastAPI** or **Flask** to:

* Accept the image and name via an API (/register endpoint).
* Detect the face using face\_recognition or dlib.
* Compute **face encoding** (128-d vector).
* Store in a **database**:
  + Name
  + Encoding (as list or blob)
  + Timestamp

Libraries to use:

* face\_recognition
* opencv-python
* numpy
* fastapi or flask
* sqlalchemy or direct sqlite3 / MongoDB

**3. Database**

Store:

* id
* name
* face\_encoding
* registration\_time

You can use:

* **SQLite** (easy for demos)
* **MongoDB**
* **PostgreSQL**

**🔁 End-to-End Flow**

1. User types name, positions face, clicks "Register".
2. React captures an image frame and sends it + name to backend.
3. Python:
   * Detects face.
   * Extracts face encoding.
   * Saves encoding, name, and timestamp.
4. Response: “Face Registered Successfully.”

For this hackathon project, I recommend starting with **SQLite**. Here's a quick comparison to help you decide:

**✅ Why Choose SQLite (Recommended for You)**

| **Feature** | **Reason** |
| --- | --- |
| ✅ Simple setup | No server needed, just a .db file. Perfect for demos or quick work. |
| ✅ Lightweight | Ideal for local development and small apps like this. |
| ✅ Easy integration | Easily works with Python (sqlite3, SQLAlchemy). |
| ✅ No dependencies | No need to install or run a DB server. |

**🤔 When to Use Others**

* **MongoDB**:
  + Good if you're familiar with NoSQL.
  + Easier to store face encodings as arrays.
  + Needs a running MongoDB server (local or Atlas cloud).
* **PostgreSQL**:
  + Great for production-grade apps.
  + Robust relational features and JSONB support.
  + More setup effort than SQLite.

"**Allow multiple unique face registrations**" means your system should:

**✅ Let users register:**

* The **same name** with **different faces** (if needed)
* **Different names** with **different faces**
* But **not** register the **same face twice** under the same name

**🔍 More Precisely:**

1. **Allow**:
   * "Alice" with face A
   * "Alice" again with **another face** (face B)
   * "Bob" with face C
2. **Prevent/Duplicate Check**:
   * If someone tries to register the **same face** again under the **same name**, you should detect that it's already in the DB and **reject or warn**.

**💡 How to Implement This:**

Before saving a new registration:

* Encode the face
* Load existing encodings from DB
* Use face\_recognition.compare\_faces() with a similarity threshold
* If **any match exists** for the **same name**, reject it