Imports

from flask_sqlalchemy import SQLAlchemy

from werkzeug.security import generate_password_hash, check_password_hash

from datetime import datetime

1. flask_sqlalchemy.SQLAlchemy – A Flask extension that provides ORM (Object Relational Mapping) for database operations.
2. werkzeug.security – Provides functions to **hash passwords** securely (generate_password_hash) and **verify** them (check_password_hash).
3. datetime – Used to store timestamps for chat history.

Database Initialization

```
db = SQLAlchemy()
```

- Creates a SQLAlchemy instance db that will be linked to the Flask app later.

```
def init_db(app=None):
    if app:
        app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///agri_chatbot.db"
        app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
        db.init_app(app)
        with app.app_context():
            db.create_all()
```

- init_db(app=None) – Function to initialize the database with the Flask app.
- SQLALCHEMY_DATABASE_URI – Sets the database type and location (SQLite file here).
- SQLALCHEMY_TRACK_MODIFICATIONS – Disables extra tracking to save memory.
- db.init_app(app) – Links SQLAlchemy to the Flask app.
- with app.app_context(): db.create_all() – Creates all tables defined in the models if they don't exist.

User Model

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password_hash = db.Column(db.String(200), nullable=False)
```

```
chats = db.relationship("ChatHistory", backref="user", lazy=True)
```

- User represents registered users.
- id – Unique primary key for each user.
- username – Must be unique and non-empty.
- password_hash – Stores hashed password (never plain text).
- chats – Relationship with ChatHistory, allows user.chats to fetch all messages from this user.

```
def set_password(self, password):
    self.password_hash = generate_password_hash(password)
```

- Hashes and stores the user's password securely.

```
def check_password(self, password):
    return check_password_hash(self.password_hash, password)
```

- Verifies if a provided password matches the stored hash.

```
@staticmethod
def create(username, password):
    u = User(username=username)
    u.set_password(password)
    db.session.add(u)
    db.session.commit()
    return u
```

- Helper method to **create a new user**, hash the password, and save to the database.

```
@staticmethod
def get_by_username(username):
    return User.query.filter_by(username=username).first()
```

- Fetches a user by their username. Returns None if not found.

ChatHistory Model

```
class ChatHistory(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"), nullable=False)
    message = db.Column(db.Text, nullable=False)
    response = db.Column(db.Text, nullable=False)
```

timestamp = db.Column(db.DateTime, default=datetime.utcnow)

- ChatHistory stores individual chat messages.
- id – Primary key for each message.
- user_id – Foreign key linking to a user.
- message – User's message text.
- response – Bot's response text.
- timestamp – Automatically stores the time the message was created.

@staticmethod

def create(user_id, msg, resp):

ch = ChatHistory(user_id=user_id, message=msg, response=resp)

db.session.add(ch)

db.session.commit()

return ch

- Helper method to **create and save a chat record** for a specific user.

Summary (Short Version)

- User model: Manages users and password security.
- ChatHistory model: Tracks chat messages and responses with timestamps.
- init_db(app) initializes SQLite database with Flask.
- Helper methods simplify creating and retrieving users or chat history.

Explanation

- **One-to-Many Relationship:**
  - A **single User** can have **many ChatHistory** entries.
  - user.chats allows you to access all chat messages for that user.
- **Foreign Key:**
  - ChatHistory.user_id links to User.id.
- **Back Reference:**
  - Using backref="user", you can access the user from a chat with chat.user.