**1. Bank Management System**

**🏦 Secure Command-Line Bank Management System**

A Python CLI application simulating core banking operations with secure user authentication, account management, financial transactions (deposits, withdrawals, transfers), transaction history, and an admin panel for oversight.

**✨ Features**

* User registration, login, and password hashing.
* Create and manage multiple savings/checking accounts.
* Perform deposits, withdrawals, and inter-account transfers.
* View detailed transaction history.
* Admin panel: manage users (activate/deactivate), view all accounts and transactions.

**💻 Technologies**

* **Python 3.x**
* **SQLAlchemy** (ORM)
* **SQLite** (Database)
* **bcrypt** (Password Hashing)

**🚀 Getting Started**

1. **Clone the repository.**
2. **Create and activate a virtual environment:**
3. python -m venv .venv
4. # Windows: .\.venv\Scripts\activate
5. # macOS/Linux: source ./.venv/bin/activate
6. **Install dependencies:**
7. pip install sqlalchemy bcrypt
8. **Run the application (all code in main.py):**
9. python main.py

**2. E-commerce Backend Engine**

**🛒 RESTful E-commerce Backend API**

A conceptual RESTful API backend built with Python (Flask/Django) for an e-commerce platform. It provides essential services for user management, product catalog, shopping carts, and order processing, designed for frontend integration.

**✨ Features**

* User authentication (registration, login, RBAC for customers/admins).
* Product management (CRUD operations).
* Shopping cart functionality (add/remove items, update quantity).
* Order processing and history.
* Stock management and validation.

**💻 Technologies**

* **Python 3.x**
* **Flask / Django** (Web Framework)
* **PostgreSQL / SQLite** (Database)
* **SQLAlchemy / Django ORM** (ORM)
* **bcrypt / Django's hashing** (Authentication)

**🚀 Getting Started**

1. **Clone the repository.**
2. **Create and activate a virtual environment** (as above).
3. **Install dependencies** (choose based on Flask or Django):
4. # For Flask example:
5. pip install Flask SQLAlchemy Flask-Migrate bcrypt
6. # For Django example:
7. pip install Django djangorestframework
8. **Configure database and run migrations** (specifics vary by framework).
9. **Run the server:**
10. # Flask: flask run
11. # Django: python manage.py runserver

**3. Multi-User Chat Application**

**💬 Real-time Multi-User Chat (Socket & Threading)**

A foundational real-time chat application demonstrating direct network communication using Python's socket module and concurrent client handling via threading. It features a server that broadcasts messages, supports custom nicknames, and includes basic moderation commands.

**✨ Features**

* Instant message exchange between multiple clients.
* Server handles concurrent connections.
* Clients use unique nicknames.
* Messages broadcast to all participants.
* Basic admin commands (e.g., /kick).

**💻 Technologies**

* **Python 3.x**
* **socket module** (Network communication)
* **threading module** (Concurrency)

**🚀 Getting Started**

1. **Clone the repository (ensure server.py and client.py are present).**
2. **Create and activate a virtual environment** (as above).
3. **No external dependencies** required.
4. **Open two (or more) separate terminal windows.**
5. **In the first terminal, run the server:**
6. python server.py
7. **In the second (and subsequent) terminals, run the client:**
8. python client.py
   * Enter a nickname when prompted.

**4. Custom File Encryption/Decryption Tool**

**🔒 Secure File Encryption/Decryption Tool (AES-256)**

A command-line utility in Python for advanced file encryption and decryption. It employs AES-256 symmetric encryption with robust PBKDF2 key derivation from a user-defined passphrase, ensuring strong data confidentiality.

**✨ Features**

* Encrypts files using AES-256 in CBC mode.
* Securely derives keys from passwords using PBKDF2 with SHA256.
* Generates unique cryptographic salts and Initialization Vectors (IVs) per encryption.
* Supports large files by processing data in chunks.
* Includes comprehensive error handling for file operations and password validation.

**💻 Technologies**

* **Python 3.x**
* **cryptography library** (AES, PBKDF2, padding primitives)
* **secrets module** (Cryptographically strong randomness)

**🚀 Getting Started**

1. **Clone the repository (ensure encrypt\_tool.py is present).**
2. **Create and activate a virtual environment** (as above).
3. **Install dependencies:**
4. pip install cryptography
5. **Run the tool:**
6. python encrypt\_tool.py
   * Follow the interactive prompts to encrypt or decrypt files.