A screenshot of a computer

AI-generated content may be incorrect.

**✅ Core Components**

1. **API Gateway**
   * Routes requests to microservices.
   * Handles load balancing, authentication (via JWT), and rate limiting.
   * **Tech:** Spring Cloud Gateway or Kong.
2. **Service Discovery**
   * Allows dynamic registration of microservices.
   * **Tech:** Netflix Eureka or Consul.
3. **Config Server**
   * Centralized configuration for all services.
   * **Tech:** Spring Cloud Config.
4. **Message Broker**
   * For async communication between services (events like “User Created”).
   * **Tech:** Apache Kafka or RabbitMQ.

**✅ User Service Responsibilities**

* **Authentication & Authorization**
  + Login → Generate JWT
  + Logout → Invalidate token (optional if using stateless JWT)
* **User Management**
  + Register new users
  + Update profile
* **Password Management**
  + Secure storage (BCrypt)
  + Reset password
* **Token Refresh**
  + Refresh tokens for long-lived sessions
* **Expose APIs**
  + /auth/login
  + /auth/logout
  + /auth/refresh
  + /users (CRUD for user profile)

**✅ Database Design (User Service)**

**Table: users**

| **Column** | **Type** |
| --- | --- |
| id | UUID |
| username | VARCHAR |
| email | VARCHAR |
| password | VARCHAR |
| roles | VARCHAR |
| created\_at | TIMESTAMP |

**Table: refresh\_tokens** (if using refresh tokens)

| **Column** | **Type** |
| --- | --- |
| id | UUID |
| user\_id | UUID |
| token | VARCHAR |
| expiry\_date | DATE |

**✅ Authentication Flow**

1. **Login**
   * User sends username & password → /auth/login
   * Validate credentials
   * Generate JWT with roles & expiration
   * Return JWT to client
2. **Accessing Other Services**
   * Client sends JWT in Authorization: Bearer <token>
   * API Gateway verifies JWT before forwarding to microservice
3. **Logout**
   * If using JWT only: client deletes token
   * If using refresh tokens: invalidate token in DB
4. **Token Refresh**
   * Client sends refresh token → new JWT issued

**✅ Security**

* Use **Spring Security** for authentication/authorization.
* Hash passwords with **BCrypt**.
* Use **JWT** for stateless sessions.
* Implement role-based access (ADMIN, USER).

**✅ Tech Stack**

* **Backend:** Java 17+, Spring Boot
* **API Gateway:** Spring Cloud Gateway
* **Service Discovery:** Eureka
* **Message Queue:** Kafka
* **Database:** PostgreSQL
* **Security:** Spring Security + JWT
* **Config Management:** Spring Cloud Config
* **Docker & Kubernetes:** For deployment

**✅ Implementation Steps**

1. **Implement User Service**
   1. REST APIs for login, logout, register
   2. Integrate Spring Security + JWT
2. **Set up API Gateway**
3. **Containerize services (Docker)**
4. **Deploy using Kubernetes**
5. **Create Config Server & Eureka Server**
6. **Implement Other Microservices** (Order, Inventory)
   1. Use JWT for securing APIs
7. **Add Kafka for event-driven communication**

**✅Chat GPT Generated**

1. **Create Config Server & Eureka Server**
2. **Set up API Gateway**
3. **Implement User Service**
   * REST APIs for login, logout, register
   * Integrate Spring Security + JWT
4. **Implement Other Microservices** (Order, Inventory)
   * Use JWT for securing APIs
5. **Add Kafka for event-driven communication**
6. **Containerize services (Docker)**
7. **Deploy using Kubernetes**