Backend for admin

1. User Management Backend

- **Database Models**:
  - `User`: Includes fields for user details (name, email, role, status, registration date, activity logs, etc.).
  - `Role`: Define roles (Admin, Moderator, User) with associated permissions.
  - `ActivityLogs`: Store user activity data, such as login history, postings, transactions, etc.
- **APIs**:
  - `GET /users`: Fetch users with filters (name, email, role, etc.).
  - `POST /users`: Create new users.
  - `PUT /users/{id}`: Update user roles, status, or personal information.
  - `DELETE /users/{id}`: Deactivate or delete user accounts.
- **Features**:
  - Bulk actions (e.g., batch update roles, bulk bans).
  - Logging and auditing of user-related changes.

---

2. Product Management Backend

- **Database Models**:
  - `Product`: Includes fields for product details (title, description, price, category, status, images, owner, etc.).
  - `Category`: Hierarchical categories for products.
  - `FlaggedContent`: Stores details of flagged products.
- **APIs**:
  - `GET /products`: Fetch product listings with filters (status, category, owner, etc.).
  - `POST /products`: Add new product listings.
  - `PUT /products/{id}`: Update product details (images, descriptions, status).
  - `DELETE /products/{id}`: Delete products.
- **Features**:
  - Approve/reject product workflows.
  - Flagged product moderation with reasons.

---

3. Transaction Management Backend

- **Database Models**:
  - `Transaction`: Stores barter, bid, buy, or resell transactions, including user IDs, product IDs, status, and timestamps.
  - `Dispute`: Tracks transaction disputes and resolution details.
- **APIs**:
  - `GET /transactions`: Fetch transactions with filters (type, user, status).
  - `PUT /transactions/{id}`: Update transaction status (e.g., approve or deny).
  - `POST /transactions/{id}/dispute`: Add a dispute for a transaction.
- **Features**:
  - Automatic notifications for completed transactions.
  - Tools for admins to approve, deny, or resolve disputes.

---

4. Communication & Notifications Backend

- **Database Models**:
  - `Notifications`: Stores system-generated notifications for users.
  - `Messages`: Stores internal communication between users and admins.
- **APIs**:
  - `POST /notifications`: Send notifications to users.
  - `GET /notifications`: Fetch notification history for a user.
  - `POST /messages`: Send messages between users and admins.
- **Features**:
  - Real-time notifications using WebSocket or push notifications.
  - Broadcast functionality for admin announcements.

---

5. Analytics & Reporting Backend

- **Database Models**:
  - `Reports`: Stores aggregated data for analytics (e.g., user activity, transaction stats, earnings).
- **APIs**:
  - `GET /analytics/user-activity`: Fetch user activity reports.
  - `GET /analytics/revenue`: Fetch revenue reports.
  - `GET /analytics/flagged-content`: Fetch trends in flagged content.
- **Features**:
  - Generate and export reports in CSV or PDF.
  - Data visualization tools for real-time dashboards (e.g., charts, graphs).

---

6. Settings & Customization Backend

- **Database Models**:
  - `Settings`: Store platform-wide configurations (e.g., transaction fees, payment settings, themes).
- **APIs**:
  - `GET /settings`: Fetch platform settings.
  - `PUT /settings`: Update platform settings.
- **Features**:
  - Update content moderation rules dynamically.
  - Theme customization for frontend layout changes.

---

7. System Security Backend

- **Authentication**:
  - Use OAuth 2.0 or JWT for secure authentication.
  - Implement multi-factor authentication (MFA) for admin accounts.
- **Encryption**:
  - Encrypt sensitive data (e.g., passwords, transaction details) using industry standards like AES-256.
- **Access Control**:
  - Role-based access control (RBAC) to ensure admins have only necessary permissions.

- **Audit Logs**:
  - Log all admin actions in a secure, immutable database table for accountability.

---

8. Scalability & Performance

- **Caching**:
  - Use a caching layer (e.g., Redis) for frequently accessed data like user profiles or product categories.
- **Database Optimization**:
  - Implement indexing, partitioning, and query optimization for handling large datasets.
- **Load Balancing**:
  - Use load balancers to distribute API traffic across multiple servers.
- **Asynchronous Processing**:
  - Use message queues (e.g., RabbitMQ, Kafka) for long-running tasks like report generation or bulk actions.

---

9. Flagging & Moderation Backend

- **Database Models**:
  - `FlagReports`: Stores information about flagged products, users, or comments.
- **APIs**:
  - `GET /flagged-content`: Fetch flagged content with details (reason, reporter, date).
  - `PUT /flagged-content/{id}`: Update the status of flagged content (approved, rejected, escalated).
- **Features**:
  - Integrate automated moderation using machine learning for detecting inappropriate content.
  - Provide tools for admins to add notes or actions taken on flagged content.