**📄 Payment Oversight Module – Technical Documentation (Squad E3.4)**

**1. Overview**

The Payment Oversight Module is a critical backend service within the super app’s platform infrastructure. It ensures secure handling of financial transactions, transparent oversight of disputes, and regulatory compliance.

**1.1 A Critical Module in Financial Oversight**

* **Description**:  
  Safeguards the payment ecosystem by identifying risks, resolving disputes, and maintaining immutable audit trails.
* **Details**:
  + Detects and manages flagged (suspicious) transactions.
  + Provides structured workflows for dispute resolution.
  + Ensures transparency via audit logs.
  + Alerts and notifies admins/users of resolution outcomes.
* **Purpose**:  
  Maintains security, compliance, and accountability in financial operations.

**1.2 Lifecycle of Payment Oversight**

* **Description**:  
  Handles flagged transaction detection, dispute management, resolution, and closure.
* **Details**:
  + Transaction flagged → pending review.
  + Dispute created → lifecycle transitions (open → under\_review → resolved/escalated).
  + Final state recorded in immutable audit logs.
* **Purpose**:  
  Guarantees traceability and fairness in financial dispute handling.

**1.3 Ensures System Security, Compliance, and Transparency**

* **Security**: End-to-end encryption (AES-256, TLS 1.3).
* **Compliance**: AML/KYC, GDPR adherence.
* **Auditability**: Immutable logs (append-only).
* **Transparency**: Structured workflows prevent bias and ensure consistency.

**2. Purpose**

The purpose of the Payment Oversight Module is to:

1. Detect suspicious or anomalous transactions.
2. Allow structured dispute resolution with clear statuses.
3. Maintain tamper-proof audit trails for compliance.
4. Provide notifications to all stakeholders during dispute lifecycle.
5. Enable monitoring and analytics for fraud detection and oversight.

**3. Scope (MVP)**

**3.1 MVP Features**

1. **Flagged Transaction Management**
   * Admins can view all flagged transactions.
   * Supports filtering by status (pending, reviewed, cleared).
2. **Dispute Resolution Workflow**
   * Disputes can be created, reviewed, resolved, or escalated.
   * Admins update status (open, under\_review, resolved, escalated).
3. **Immutable Audit Logging**
   * Every admin action logged with admin\_id, target\_id, action\_type, timestamp, and metadata.
   * Stored in append-only DB or ELK stack.
4. **Notification Integration**
   * Alerts sent via email, SMS, or push to notify users/admins of outcomes.

**3.2 Future Enhancements**

* **ML-based Fraud Detection** for auto-flagging anomalies.
* **Chatbot-based Dispute Query System** for faster resolution.
* **Blockchain-based Audit Logs** for immutable compliance trails.
* **Bulk Dispute Handling** for large-scale fraud events.

**4. Functional Requirements**

**4.1 Flagged Transactions**

* View flagged transactions with details (reason, timestamp, status).
* Supports filtering and search (transaction\_id, status).

**4.2 Dispute Lifecycle Management**

* Create new disputes linked to transactions.
* Update status (open → under\_review → resolved).
* Allow escalation for unresolved disputes.

**4.3 Notifications**

* Email/SMS/Push integration for informing stakeholders.
* Retry + backoff mechanism for delivery failures.

**4.4 Audit Logs**

* Record every admin action (view, resolve, escalate, notify).
* Immutable, append-only storage.

**5. Data Models**

**5.1 flagged\_transactions**

| **Field** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| id | UUID | PK | Unique identifier |
| transaction\_id | UUID | FK → transactions | Associated transaction |
| flagged\_reason | TEXT | NOT NULL | Reason flagged (AML, fraud, anomaly) |
| flagged\_at | TIMESTAMP | DEFAULT now() | Time flagged |
| status | ENUM | {pending, reviewed, cleared} | Flag status |

**5.2 payment\_disputes**

| **Field** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| id | UUID | PK | Unique identifier |
| transaction\_id | UUID | FK → transactions | Disputed transaction |
| user\_id | UUID | FK → users | Disputing user |
| dispute\_reason | TEXT | NOT NULL | Reason for dispute |
| status | ENUM | {open, under\_review, resolved, escalated} | Lifecycle state |
| created\_at | TIMESTAMP | DEFAULT now() | Created timestamp |
| resolved\_at | TIMESTAMP | NULLABLE | Resolution timestamp |

**5.3 audit\_logs**

| **Field** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| id | UUID | PK | Unique identifier |
| admin\_id | UUID | FK → admins | Acting admin |
| action\_type | ENUM | {flagged\_view, dispute\_resolved, escalation, notification\_sent} | Admin action |
| target\_id | UUID | FK → {flagged\_transactions, payment\_disputes} | Affected entity |
| timestamp | TIMESTAMP | DEFAULT now() | Action timestamp |
| metadata | JSONB |  | Extra details |

**6. API Endpoints**

**6.1 GET /api/v1/admin/payments/flagged**

* **Description**: Retrieve all flagged transactions pending admin review.
* **Response Example**:

[

{

"id": "uuid-123",

"transaction\_id": "txn-789",

"flagged\_reason": "Unusual amount",

"status": "pending",

"flagged\_at": "2025-09-10T14:22:11Z"

}

]

* **Error Codes**:
  + 401 Unauthorized
  + 404 Not Found
  + 500 Internal Server Error

**6.2 POST /api/v1/admin/payments/disputes/{id}/resolve**

* **Description**: Resolve a payment dispute.
* **Request Body**:

{

"resolution": "Transaction refunded",

"resolved\_by": "admin-456"

}

* **Response Example**:

{

"id": "dispute-123",

"status": "resolved",

"resolved\_at": "2025-09-16T10:00:00Z"

}

* **Error Codes**:
  + 400 Bad Request
  + 404 Not Found
  + 409 Conflict
  + 500 Internal Server Error

**7. Sequence Flow (Dispute Resolution)**

sequenceDiagram

participant Admin

participant API

participant DB

participant User

participant AuditLog

Admin->>API: View flagged transactions

API->>DB: Fetch flagged records

API-->>Admin: Return flagged list

Admin->>API: Resolve dispute (POST)

API->>DB: Update dispute status

API->>User: Notify resolution outcome

API->>AuditLog: Record admin action

**8. Testing Plan**

**8.1 Unit Tests**

* Validate dispute resolution logic.
* Ensure flagged transaction validation works.

**8.2 Integration Tests**

* End-to-end lifecycle of flagged → dispute → resolution.
* Escalation workflow with notifications.

**8.3 Negative Tests**

* Invalid IDs, unauthorized access, duplicate resolution attempts.

**8.4 Performance Tests**

* Stress tests with 10k flagged transactions.
* Load tests for concurrent dispute resolution.

**9. Deliverables**

1. Functional FastAPI endpoints (flagged, disputes, audit logs).
2. PostgreSQL + Redis schema & migrations.
3. Immutable logging service integrated with ELK.
4. CI/CD (GitHub Actions + Helm).
5. Prometheus + Grafana dashboards for monitoring.

**10. Security & Compliance**

* **RBAC**: JWT-based admin authentication.
* **Encryption**: AES-256 (at rest), TLS 1.3 (in transit).
* **Immutable Logs**: Append-only compliance logs.
* **PII Masking**: Hide sensitive data in responses.

**11. Appendices**

**Example SQL Schema**

* flagged\_transactions
* payment\_disputes
* audit\_logs

**Example OpenAPI Spec**

* Endpoints: flagged list, dispute resolution.

✅ This mirrors the **style, structure, and tone** of the Squad E3.1 documentation, but tailored for **Payment Oversight (Squad E3.4)**.

Would you like me to **export this as a PDF (like Squad E3.1)** so it’s presentation-ready, or keep it in editable doc format (Markdown/Word)?