**Project Documentation — Squad E3.2 Marketplace Oversight**

**1. Purpose**

Marketplaces often face challenges around **fraud prevention, trust, and dispute resolution**. Buyers may report suspicious listings (e.g., fake products, misleading descriptions, offensive material), and disputes may arise when transactions don’t go as expected (e.g., damaged goods, non-delivery, payment issues).

The **Marketplace Oversight module** empowers administrators to:

* **Monitor flagged listings**: Review and take action against items reported by users.
* **Resolve disputes**: Ensure fair treatment of buyers and sellers through timely resolution.
* **Maintain trust**: Demonstrate that the platform actively enforces rules and protects users.

This system establishes **accountability, transparency, and governance** in the marketplace.

**2. Scope (MVP)**

The **Minimum Viable Product (MVP)** focuses on **core oversight functionality** for administrators.

**In-Scope (MVP Features)**

* **Flagged Listings Management**
  + View flagged listings in an admin dashboard.
  + Filter by status (pending, reviewed, resolved).
  + Mark listings as reviewed/resolved with timestamps.
* **Dispute Management**
  + View disputes raised by users.
  + Change dispute status (open → in\_review → resolved).
  + Record resolution timestamps for audit trails.
* **Notifications**
  + Send system notifications (email, in-app) to parties when disputes are resolved.

**Out-of-Scope (Future Enhancements)**

* Automated moderation (AI/ML to auto-detect fraud).
* Multi-level escalation (legal review, compliance teams).
* Dispute analytics dashboards.
* Integration with third-party fraud detection systems.
* User-facing self-resolution workflow (buyer/seller negotiate before admin).

**3. Functional Requirements**

The system must:

1. **Flagged Listings**
   * Retrieve all flagged listings with filters (status/date).
   * Allow admins to mark flagged items as “reviewed” or “resolved”.
   * Record resolution date and admin user ID.
2. **Disputes**
   * Display disputes raised by users (with listing ID, involved users).
   * Allow admins to update dispute status (open, in\_review, resolved).
   * Record resolution details for traceability.
3. **Notifications**
   * Notify buyers/sellers when a dispute is resolved.
   * Support integration with email + push notifications.
4. **Error Handling & Security**
   * Prevent resolving non-existent or already resolved cases.
   * Restrict dispute/flagged listing APIs to admins only.
   * Return meaningful error codes for invalid operations.

**4. Data Models**

A normalized relational database structure will be used.

**Entity Relationship Diagram (ERD)**

(Imagine a diagram showing relationships):

* **Users** (user\_id PK)  
  ↳ has many → **Listings** (listing\_id PK)  
  ↳ can raise many → **Disputes** (dispute\_id PK)
* **Flagged\_Listings** linked to **Listings**
* **Disputes** linked to **Users** and **Listings**

**flagged\_listings Table**

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| id (PK) | INT | Unique record identifier. |
| listing\_id (FK) | INT | Reference to marketplace listing. |
| reason | TEXT | User’s reason for flagging. |
| status | ENUM | pending, reviewed, resolved. |
| created\_at | TIMESTAMP | Time of flag creation. |
| resolved\_at | TIMESTAMP | Time of resolution. |
| resolved\_by (FK) | INT | Admin user who resolved the flag. |

**disputes Table**

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| id (PK) | INT | Unique identifier for the dispute. |
| user\_id (FK) | INT | User who raised the dispute. |
| listing\_id (FK) | INT | Related listing. |
| status | ENUM | open, in\_review, resolved. |
| created\_at | TIMESTAMP | When dispute was created. |
| resolved\_at | TIMESTAMP | When dispute was resolved. |
| resolved\_by (FK) | INT | Admin who resolved dispute. |
| resolution\_notes | TEXT | Notes added by admin. |

**5. API Endpoints**

All endpoints are **admin-only** and secured with authentication & authorization middleware.

**Retrieve Flagged Listings**

* **Method:** GET
* **Endpoint:** /api/v1/admin/marketplace/flagged
* **Params:** status, date\_range (optional).
* **Response:**

[

{

"id": 101,

"listing\_id": 5001,

"reason": "Fake brand item",

"status": "pending",

"created\_at": "2025-09-12T10:15:00Z"

}

]

**Resolve Flagged Listing**

* **Method:** POST
* **Endpoint:** /api/v1/admin/marketplace/flagged/{id}/resolve
* **Payload:**

{

"resolved\_by": 9001,

"notes": "Confirmed fraudulent listing. Removed from marketplace."

}

* **Response:**

{

"id": 101,

"status": "resolved",

"resolved\_at": "2025-09-16T09:00:00Z",

"resolved\_by": 9001

}

**Resolve Dispute**

* **Method:** POST
* **Endpoint:** /api/v1/admin/marketplace/disputes/{id}/resolve
* **Payload:**

{

"resolved\_by": 9001,

"resolution\_notes": "Refund issued to buyer."

}

* **Response:**

{

"id": 202,

"status": "resolved",

"resolved\_at": "2025-09-16T09:30:00Z",

"resolved\_by": 9001

}

**6. Sequence Flow**

**Dispute Resolution Flow**

1. Buyer raises dispute → stored in disputes table.
2. Admin views open disputes in dashboard.
3. Admin investigates and resolves dispute.
4. System updates status, resolved\_at, and logs resolved\_by.
5. Notification service sends messages to buyer/seller.

(Sequence diagram can be provided showing: **Admin → API → DB → Notification Service → Users**).

**7. Testing Plan**

**Unit Tests**

* resolveDispute(id) updates status, resolved\_at.
* resolveFlag(id) prevents multiple resolutions.

**Integration Tests**

* Admin resolves listing → DB reflects status → Notification sent.
* Admin resolves dispute → Buyer & Seller receive notifications.

**Negative Tests**

* Invalid ID returns 404.
* Unauthorized request returns 403.
* Attempting to resolve already resolved item returns 409 Conflict.

**Performance Tests**

* Bulk retrieval of 10k flagged listings under 2 seconds.
* Simulated 500 concurrent dispute resolutions.

**8. Deliverables**

1. **Database Migrations** for flagged\_listings and disputes.
2. **Admin APIs** (GET flagged listings, POST resolve listing, POST resolve dispute).
3. **Notification Integration** (email + in-app).
4. **Test Suite** (unit + integration + load tests).
5. **Documentation**:
   * API reference (Swagger/OpenAPI).
   * Sequence diagrams.
   * User/admin guides.

**9. Project Timeline (8 Weeks)**

| **Week** | **Deliverables** | **Notes** |
| --- | --- | --- |
| 1 | DB schema design & migrations | Review with DBA. |
| 2–3 | Flagged Listings API | GET & resolve endpoints. |
| 4 | Dispute Handling API | Resolve disputes with audit logs. |
| 5 | Notification Service | Email + push notification integration. |
| 6 | Integration Testing | End-to-end flow validation. |
| 7 | Performance Testing | High-volume listings/disputes. |
| 8 | Final Demo + Docs | Stakeholder review + deployment. |

**10. Risks & Mitigation**

* **Risk:** Admins overwhelmed by false reports.
  + **Mitigation:** Add filters, batch resolve, and reporting thresholds (future).
* **Risk:** Notification delivery failure.
  + **Mitigation:** Retry mechanism with fallback to email.
* **Risk:** Security vulnerabilities in admin APIs.
  + **Mitigation:** Role-based access control, audit logging.