**Project Documentation — Squad E3.3 Services Oversight**

**1. Purpose**

In service-based marketplaces (e.g., ride-hailing, delivery, home services), **trust and safety** are crucial for both customers and providers. Bookings can sometimes be flagged for suspicious or abusive behavior, and service providers (drivers, vendors, contractors) may violate platform policies.

The **Services Oversight module** gives administrators the tools to:

* **Review flagged bookings** that may involve fraud, misconduct, or service quality issues.
* **Suspend or restore providers** when investigations determine a violation (or clear a provider of accusations).
* **Maintain a secure and auditable environment** by logging every administrative action.

This oversight process ensures fair enforcement, transparent accountability, and ultimately preserves marketplace trust.

**2. Scope (MVP)**

The MVP focuses on **core oversight features** for administrators.

**In-Scope (MVP Features)**

* **Flagged Bookings**
  + Display flagged bookings with reasons.
  + Allow filtering by status (pending, reviewed, resolved).
* **Provider Management**
  + Suspend providers (set status = suspended).
  + Restore providers (set status = active).
* **Action Logging**
  + Record all admin actions, including actor, timestamp, and reason.
  + Ensure logs cannot be deleted/modified (tamper-proof).

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

* Automated fraud detection and AI-based risk scoring.
* Tiered enforcement (warnings, temporary suspensions, permanent bans).
* End-user appeal system for providers.
* Dashboard analytics (e.g., number of suspensions by category).
* Integration with external background check or compliance services.

**3. Functional Requirements**

The system must support the following capabilities:

**Flagged Bookings Management**

* Retrieve flagged bookings with filter options.
* Display details: booking ID, provider ID, reason, status, created date.
* Update status once reviewed or resolved.

**Provider Management**

* Suspend provider accounts:
  + Prevent suspended providers from accepting/initiating bookings.
  + Update provider status to suspended.
* Restore provider accounts:
  + Change status back to active.
  + Allow provider to resume service.

**Audit Logging**

* Record every admin action in admin\_actions.
* Capture details: admin ID, provider ID, action (suspend/restore), reason, timestamp.
* Ensure logs are **immutable** for accountability.

**Security & Validation**

* Restrict endpoints to **admin users** only (role-based access control).
* Validate state transitions:
  + Cannot suspend non-existent provider.
  + Suspending an already suspended provider should not throw error (idempotent).
  + Restoring an already active provider should be safe.

**4. Data Models**

A relational schema will be used to ensure consistency.

**Entity Relationship Overview**

* **Providers** → can be suspended/restored by Admin.
* **Bookings** → may be flagged → stored in flagged\_bookings.
* **Admin\_Actions** → logs suspension/restoration decisions.

**flagged\_bookings Table**

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| id (PK) | INT | Unique identifier. |
| booking\_id (FK) | INT | Reference to the booking. |
| reason | TEXT | Reason provided for flag. |
| status | ENUM | pending, reviewed, resolved. |
| created\_at | TIMESTAMP | When flag was created. |
| resolved\_at | TIMESTAMP | When resolved (nullable). |

**providers Table**

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| id (PK) | INT | Provider ID. |
| name | VARCHAR | Provider’s name. |
| status | ENUM | active, suspended. |
| created\_at | TIMESTAMP | Account creation date. |
| updated\_at | TIMESTAMP | Last modification. |

**admin\_actions Table**

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| id (PK) | INT | Unique log ID. |
| admin\_id (FK) | INT | Admin who performed the action. |
| provider\_id (FK) | INT | Target provider. |
| action | ENUM | suspend, restore. |
| reason | TEXT | Reason provided by admin. |
| created\_at | TIMESTAMP | Action timestamp. |

**5. API Endpoints**

All endpoints must be **authenticated and restricted to admins**.

**Retrieve Flagged Bookings**

* **Method:** GET
* **Endpoint:** /api/v1/admin/services/flagged
* **Query Params:** status, date\_range.
* **Response Example:**

[

{

"id": 301,

"booking\_id": 9001,

"reason": "Customer reported harassment",

"status": "pending",

"created\_at": "2025-09-14T08:00:00Z"

}

]

**Suspend Provider**

* **Method:** POST
* **Endpoint:** /api/v1/admin/providers/{id}/suspend
* **Payload Example:**

{

"admin\_id": 101,

"reason": "Verified multiple user complaints"

}

* **Response Example:**

{

"provider\_id": 501,

"status": "suspended",

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

}

**Restore Provider**

* **Method:** POST
* **Endpoint:** /api/v1/admin/providers/{id}/restore
* **Payload Example:**

{

"admin\_id": 101,

"reason": "Investigation cleared provider"

}

* **Response Example:**

{

"provider\_id": 501,

"status": "active",

"updated\_at": "2025-09-16T11:00:00Z"

}

**6. Sequence Flow**

**Provider Suspension Flow**

1. Admin logs into system → views flagged bookings.
2. Admin reviews case details.
3. If provider fault → admin calls **Suspend API**.
4. System updates provider status → logs action.
5. Provider is blocked from receiving new bookings.
6. Notification service informs provider of suspension.

**Provider Restoration Flow**

1. Admin investigates flagged booking.
2. Admin decides provider is clear of violation.
3. Admin calls **Restore API**.
4. System updates status back to active.
5. Action logged in admin\_actions.
6. Provider notified of restoration.

**7. Testing Plan**

**Unit Tests**

* suspendProvider() sets provider status = suspended.
* restoreProvider() sets provider status = active.
* logAdminAction() records admin action.

**Integration Tests**

* Suspend provider → booking attempts blocked.
* Restore provider → bookings allowed again.
* Admin action correctly logged with timestamp.

**Edge Cases**

* Suspending already suspended provider → idempotent success.
* Restoring already active provider → safe no-op.
* Invalid provider ID → returns 404.

**Security Tests**

* Non-admin request to suspend → 403 Forbidden.
* Tampering with logs → system rejects updates.

**8. Deliverables**

1. **Database Schema & Migrations** for flagged bookings, providers, and admin actions.
2. **Admin APIs** (GET flagged bookings, POST suspend, POST restore).
3. **Logging Mechanism** for immutable admin actions.
4. **Test Suite** (unit, integration, edge, security).
5. **Documentation** (ERD, sequence diagrams, API reference).
6. **Demo Environment** for admin workflows.

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

| **Week** | **Deliverables** | **Notes** |
| --- | --- | --- |
| 1 | Finalize schema & ERD | DBA + dev review |
| 2–3 | Flagged bookings API | Includes GET & filter |
| 4 | Provider suspend/restore endpoints | Role-based access control |
| 5 | Logging implementation | Admin audit trail |
| 6 | Integration testing | Booking flows + provider states |
| 7 | Security hardening | RBAC + immutability |
| 8 | Final demo + docs handoff | Stakeholder sign-off |

**10. Risks & Mitigation**

* **Risk:** False-positive provider suspensions.
  + **Mitigation:** Require reason + allow restoration with audit record.
* **Risk:** Admin misuse of power.
  + **Mitigation:** Immutable admin\_actions table with monitoring.
* **Risk:** Provider dissatisfaction from delays in restoration.
  + **Mitigation:** SLA for admin resolution times.
* **Risk:** System failure during suspension → provider still active.
  + **Mitigation:** Transactional DB updates with rollback.