PROJECT DOCUMENTATION

Squad E3.1

Platform Infrastructure — Admin User Management

---

1. Overview

The **Admin User Management Module** is a **critical component** of a super app's platform infrastructure. It plays a central role in ensuring that administrators can effectively monitor, manage, and control user accounts, thereby maintaining the overall **security, accountability, and compliance** of the system.

1.1 A Critical Module in a Super App's Platform Infrastructure

- **Description:**
  In a super app, multiple services (e.g., payments, shopping, transport, social features) are integrated into one ecosystem. The **admin user management module** forms the backbone of governance across all these services.
- **Details:**
  - Provides a unified interface for managing users across services.
  - Ensures consistency in user status (active/suspended) across the entire ecosystem.
  - Serves as a single source of truth for administrators.
- **Purpose:**
  Guarantees that the platform operates securely and efficiently by centralizing control of user accounts.

1.2 Provides Lifecycle Management for User Accounts

- **Description:**
  User accounts follow a **lifecycle** from creation, active usage, suspension (if required), and restoration or deletion. This module handles every stage of that lifecycle.
- **Details:**

    - **Onboarding:** Registers and tracks new users.
    - **Active State:** Allows users to fully access services.
    - **Suspended State:** Restricts access due to violations, risks, or administrative actions.
    - **Restoration:** Re-enables suspended accounts when justified.
- **Purpose:**
  Ensures that users are managed systematically, with clear transitions between account states.

1.3 Ensures System Security, Accountability, and Compliance

- **Description:**
  Protects the platform from misuse, ensures accountability for administrative actions, and supports compliance with regulatory standards.
- **Details:**
    - **Security:** Prevents unauthorized users from accessing services by enabling suspensions.
    - **Accountability:** Tracks every admin action in detailed logs.
    - **Compliance:** Meets requirements of regulations (e.g., GDPR, AML/KYC, audit standards).
- **Purpose:**
  Maintains trust and prevents legal, financial, and reputational risks.

1.4 Gives Administrators Visibility and Control Over Accounts

- **Description:**
  Provides admins with real-time access to all user information and tools to take necessary actions.
- **Details:**
    - Centralized dashboard to view all users.
    - Search and filtering capabilities for quick access.
    - Action tools (suspend/restore) to manage accounts instantly.
- **Purpose:**
  Empowers administrators to respond quickly to fraud, policy violations, and user requests while maintaining full oversight of the system.

2. Purpose

The purpose of the **Admin User Management Module** is to empower administrators with the necessary tools to effectively manage user accounts, ensure compliance with policies, and maintain system integrity. This module ensures that the platform remains secure, trustworthy, and auditable.

---

1. Allow Admins to View All User Profiles

- **Description:**
  Provides administrators with a centralized dashboard to access user information.
- **Details:**
  - Displays key user attributes such as ID, name, email, status, and timestamps.
  - Supports search and filter functions to quickly locate specific users.
  - Ensures that admins have complete visibility into the user base.
- **Purpose:**
  Helps in monitoring user activity, identifying problematic accounts, and improving decision-making.

---

2. Suspend Accounts for Policy Violations, Fraud, or Security Risks

- **Description:**
  Enables admins to take **immediate corrective action** by suspending accounts that pose risks to the platform.
- **Details:**
  - Accounts marked as `suspended` lose access to login and services.
  - Helps protect the platform from malicious activity, fraud, or regulatory non-compliance.
  - Suspension may be temporary or permanent based on the severity of the violation.
- **Purpose:**
  Safeguards the system, prevents damage to other users, and ensures platform integrity.

---

3. Restore User Accounts When Issues Are Resolved

- **Description:**
  Provides flexibility to reinstate users once their issues are cleared.
- **Details:**
  - Restores account status from `suspended` → `active`.
  - Enables users to regain access without losing historical data.
  - Allows admins to correct false suspensions or reverse decisions.
- **Purpose:**
  Maintains fairness by ensuring users are not permanently penalized for temporary issues or mistakes.

---

4. Maintain Detailed Logs of All Admin Actions for Auditing

- **Description:**
  Ensures that every admin action is tracked in a secure, tamper-proof log.
- **Details:**
  - Logs include admin ID, affected user, type of action (suspend/restore), and timestamp.
  - Provides a complete historical record for accountability.
  - Supports internal investigations and compliance checks.
- **Purpose:**
  Strengthens governance, prevents misuse of admin privileges, and enables regulatory reporting.

---

5. Build Trust Through Transparency and Compliance

- **Description:**
  Enhances user and stakeholder confidence in the platform by demonstrating fair and transparent account management.
- **Details:**
  - Transparency ensures users know that admin actions are traceable and justified.
  - Compliance with auditing and reporting standards builds credibility with regulators.
  - Prevents legal risks and supports certifications (e.g., GDPR, ISO standards).

- **Purpose:**
  Promotes long-term platform growth by ensuring fairness, trust, and adherence to industry best practices.

3. Scope (MVP)

The **scope** defines what will be delivered in the **Minimum Viable Product (MVP)** and outlines potential **future enhancements** to extend functionality.

3.1 MVP Features

The MVP focuses on providing the **essential admin user management capabilities** required to ensure security, control, and accountability:

1. Display User List with Search and Filter Options

- Admins can view all registered users in the system.
- Supports **search and filtering** based on:
  - **Name** (partial or exact match).
  - **Email** (exact match).
  - **Status** (active / suspended).
- Provides **pagination and sorting** to handle large datasets.
- Enables admins to quickly locate specific users for action.

---

2. Suspend User Accounts (Set Status → Suspended)

- Admins can **disable user accounts** when required.
- On suspension:
  - The user's `status` field is updated to `suspended`.
  - The user loses access to login, transactions, and platform services.
- Provides a mechanism to **temporarily or permanently block users** based on violations, suspicious activity, or admin policy.

---

3. Restore Suspended Accounts (Set Status → Active)

- Allows admins to **reactivate suspended users**.
- On restoration:
  - The user's `status` field is updated to `active`.
  - Access to the platform is reinstated immediately.
- Ensures flexibility in handling cases where:
  - Suspensions were temporary.
  - False positives occurred in fraud detection.
  - Admins decide to reinstate user privileges.

---

4. Log Every Admin Action

- Every action (suspend/restore) performed by an admin is **automatically logged**.
- The system records:
  - **Admin ID** (who performed the action).
  - **Target User ID** (which account was affected).
  - **Action Type** (suspend or restore).
  - **Timestamp** (when the action occurred).
- Provides **auditability, compliance, and accountability**.
- Protects against misuse of admin privileges.

3.2 Future Enhancements

While the MVP delivers core functionality, additional features can significantly improve efficiency, security, and compliance in later iterations:

1. Role & Permission Management

- Define **roles** (e.g., Super Admin, Moderator, Support Agent).
- Configure **permissions** to restrict who can perform sensitive actions (like suspend/restore).
- Implements **Role-Based Access Control (RBAC)** for fine-grained security.

---

2. Bulk Actions (Suspend/Restore Multiple Users)

- Provide the ability to select multiple users and apply actions in one go.
- Reduces repetitive work for admins managing large datasets.
- Useful for handling mass fraud cases or bulk reinstatements.

---

3. Analytics and Reporting

- Generate insights from admin actions and user states.
- Example reports:
  - Number of suspensions per month.
  - Average suspension duration.
  - Admin activity trends.
- Helps in compliance checks, fraud monitoring, and system optimization.

---

4. Third-Party Compliance Integrations

- Integrate with external compliance and auditing tools.
- Examples:
  - **KYC/AML systems** for financial platforms.
  - **Fraud detection services** to automate suspensions.
  - **Audit log exporters** for regulatory bodies.
- Ensures the platform meets **legal and industry standards**.

4. Functional Requirements

The **Admin User Management Module** provides the following essential functions to ensure effective user lifecycle management and compliance:

4.1 User Listing

- **Description:**
  Allows administrators to view a complete list of users within the system.
- **Features:**
  - Retrieve user details such as `id`, `name`, `email`, `status`, and timestamps.
  - Support for **filters**:
    - `name` → Search by partial or full user name.
    - `email` → Search by email address (exact match).
    - `status` → Filter by account state (active / suspended).
  - Pagination and sorting to handle large datasets efficiently.
- **Purpose:**
  - Provides a quick overview of all users.
  - Enables targeted actions (suspend/restore) based on filters.

4.2 Suspend User

- **Description:**
  Allows an admin to disable a user account, preventing further access to the platform.
- **Process:**
  - The system updates the `status` field in the `users` table from `active` → `suspended`.
  - Logs the action into the `admin_actions` table with details of who performed it.
- **Constraints:**
  - Only admins with the required permissions can perform this action.
  - A suspended user cannot log in or perform any transactions.
- **Use Cases:**
  - Violation of terms of service.
  - Suspicious activity or fraudulent behavior.
  - Temporary administrative decision.

4.3 Restore User

- **Description:**
  Allows an admin to reactivate a previously suspended account.

- **Process:**
    - The system updates the `status` field in the `users` table from `suspended` → `active`.
    - Logs the action in the `admin_actions` table with timestamp and responsible admin.
- **Constraints:**
    - Can only be performed if the user is currently suspended.
    - Restoration must not bypass compliance or security checks (optional workflow).
- **Use Cases:**
    - Reinstating users after temporary suspension.
    - False positives in fraud detection.
    - Admin decision reversal.

4.4 Admin Action Logging

- **Description:**
  Ensures that all admin activities are tracked for **accountability, transparency, and audit purposes**.
- **Process:**
    - Each action (suspend/restore) automatically creates a record in the `admin_actions` table.
    - Captured fields:
        - `admin_id` → ID of the admin performing the action.
        - `target_user_id` → ID of the affected user.
        - `action` → Type of action (`suspend` or `restore`).
        - `created_at` → Timestamp of action.
- **Purpose:**
    - Provides a reliable audit trail.
    - Helps in internal investigations and compliance reporting.
    - Prevents misuse of admin powers by ensuring every action is recorded.

---

5. Data Models

The Admin User Management module requires **two main database tables**: `users` and `admin_actions`. These ensure both **account lifecycle tracking** and **auditing of admin activities**.

5.1 Users Table

Stores all user account information and current account state.

Schema Fields:

- **id (Primary Key)** → Unique identifier for each user.
- **name** → Full name of the user. Used for identification in admin dashboard and search filters.
- **email (unique)** → Unique login credential and primary identifier. Used for authentication and user communication.
- **status** → Indicates current state of the user account:
  - `active` → User has full access to the platform.
  - `suspended` → User account is disabled (cannot log in, transact, or access services).
- **created_at** → Timestamp when the account was first created. Helps track user onboarding.
- **updated_at** → Timestamp when account details (like status) were last modified. Useful for audit and debugging.

Usage:

- Provides data for user listing in the admin dashboard.
- `status` field is updated during suspend/restore actions.
- `email` and `name` are searchable/filterable fields in APIs.

5.2 Admin_Actions Table

Captures all admin-initiated actions to ensure **transparency, accountability, and compliance**.

Schema Fields:

- **id (Primary Key)** → Unique identifier for each admin action log entry.

- **admin_id** → The ID of the admin who performed the action. Links back to the `admins` or `users` table (depending on system design).
- **action** → The type of action performed:
  - `suspend` → Admin disabled a user account.
  - `restore` → Admin reactivated a suspended account.
- **target_user_id** → The ID of the user account affected by the action.
- **created_at** → Timestamp of when the action was executed. Used for historical records and audits.

Usage:

- Ensures a permanent log of all admin decisions.
- Can be queried to generate reports (e.g., how many suspensions per month).
- Used in **compliance checks** to prove accountability.

5.3 Relationships

- **One-to-Many Relationship** between `admin_id` and actions:
  - One admin can perform many actions.
- **One-to-Many Relationship** between `target_user_id` and actions:
  - One user may have multiple suspend/restore logs over time.

5.4 Example Record

Users Table Example:

id	name	email	status	created_at	updated_at
1	John Doe	john@example.com	active	2025-05-01 10:00:00	2025-09-15 14:30:00

Admin_Actions Table Example:

id	admin_id	action	target_user_id	created_at
101	10	suspend	1	2025-09-16 09:45:00
102	10	restore	1	2025-09-16 12:00:00

---

6. API List

6.1 GET `/api/v1/admin/users` → List users with filters

- **Purpose:**
  Allows administrators to retrieve a list of all users in the system. This is the central view where admins can search and filter users for management.
- **Request Parameters (Query):**
  - `status` → filter by user status (*active* / *suspended*).
  - `email` → search by exact or partial email.
  - `name` → search by user's full name or partial string.
  - `page` → pagination parameter for large datasets.
  - `limit` → number of records per page.
- **Example Request:**
- `GET /api/v1/admin/users?status=active&name=John&page=1&limit=20`
- **Response:**
  - `200 OK` → Returns JSON list of users with details.
  - Each record includes: `id`, `name`, `email`, `status`, `created_at`, `last_login`.
  - `meta` block for pagination (total pages, current page, total records).
- **Usage:**
  Used by the **Admin Dashboard** to display the user list table with search and filter options.

6.2 POST `/api/v1/admin/users/{id}/suspend` → Suspend user

- **Purpose:**
  Temporarily disables a user account due to policy violations, fraud detection, or manual intervention by admins.
- **Request:**
  - **Path Parameter:** `id` (the unique user ID).
  - **Body:** May include optional metadata such as reason for suspension.
  - `{`
  - `  "reason": "Suspicious login activity detected"`
  - `}`
- **Response:**

    - `200 OK` → Confirmation message with updated user status = `suspended`.
    - Example:
    - `{`
    - `  "message": "User suspended successfully",`
    - `  "user": {`
    - `    "id": 123,`
    - `    "status": "suspended",`
    - `    "suspended_at": "2025-09-16T10:00:00Z"`
    - `  }`
    - `}`
    - `404 Not Found` → If user ID does not exist.
    - `409 Conflict` → If user is already suspended.
- **System Actions:**
    - Updates the `users` table status field.
    - Inserts a record into `admin_actions` with action = `suspend`.
    - Invalidates user sessions/tokens so they can't log in until restored.

6.3 POST `/api/v1/admin/users/{id}/restore` → Restore user

- **Purpose:**
  Reactivates a previously suspended account once the violation is cleared or after manual review.
- **Request:**
    - **Path Parameter:** `id` (the unique user ID).
    - **Body:** Optional reason for restoration.
    - `{`
    - `  "reason": "Account verified after manual review"`
    - `}`
- **Response:**
    - `200 OK` → Confirmation message with updated user status = `active`.
    - Example:
    - `{`
    - `  "message": "User restored successfully",`
    - `  "user": {`
    - `    "id": 123,`
    - `    "status": "active",`
    - `    "restored_at": "2025-09-16T12:30:00Z"`
    - `  }`
    - `}`
    - `404 Not Found` → If user ID does not exist.
    - `409 Conflict` → If user is already active.

- **System Actions:**
  - Updates `users` table status from `suspended` → `active`.
  - Logs the action into `admin_actions` with action = `restore`.
  - Sends notification (email/app push) to user that account is active again.

6.4 Security & Access Control

- All APIs are **protected**:
  - Require authentication via secure tokens (JWT, OAuth, or session-based).
  - Enforced via HTTPS (TLS/SSL).
- **Role-Based Access Control (RBAC):**
  - Only users with `admin` role can call these endpoints.
  - Regular users cannot access these APIs.
- **Audit Logging:**
  - Every call (suspend/restore) is logged with admin ID, user ID, action, timestamp.
  - Ensures accountability and compliance (important for audits).

---

7. Sequence Flow

1. **Admin Authentication**
   - Admin logs into the system through the secure dashboard.
   - Credentials are verified via authentication service (e.g., OAuth/JWT).
   - Role-based access control (RBAC) ensures only authorized admins can proceed.
2. **Access User List Module**
   - Admin navigates to the **User Management** section of the dashboard.
   - A request is sent to the `GET /api/v1/admin/users` API.
   - The API fetches user data from the database with filters (active/suspended).
   - The dashboard displays the user list in a searchable and filterable table.
3. **Action Selection (Suspend/Restore)**
   - Admin chooses an action (suspend or restore) from the list view.

    - The dashboard calls the appropriate API:
        - `POST /api/v1/admin/users/{id}/suspend` OR
        - `POST /api/v1/admin/users/{id}/restore`.
    - Request includes target user ID and optional reason for the action.
4. **System Updates User Status**
    - The backend verifies the request:
        - Confirms admin permissions.
        - Checks current status of the user (to prevent conflicts).
    - Updates the `status` field in the `users` table:
        - `active → suspended` OR
        - `suspended → active`.
5. **Action Logging**
    - A new entry is created in the `admin_actions` table with:
        - `admin_id` (who performed the action)
        - `target_user_id` (affected account)
        - `action` (suspend/restore)
        - `reason` (if provided)
        - `timestamp`
    - This log provides an immutable audit trail for compliance.
6. **Response Returned & Dashboard Update**
    - Backend returns a **200 OK** response with updated user details.
    - Dashboard UI refreshes the user list to reflect the new status.
    - Optional notifications may be sent to the user (e.g., email, push notification).

---

8. Testing Plan

8.1 Unit Tests

- Validate individual methods and functions used in APIs.
- Examples:
    - Confirm that calling `suspendUser(id)` changes the status to `suspended`.
    - Confirm that calling `restoreUser(id)` changes the status back to `active`.

    - Validate error handling (e.g., user not found, invalid ID).

8.2 Integration Tests

- Ensure the **end-to-end workflow** behaves correctly.
- Examples:
    - Admin suspends a user → API updates DB → log entry created.
    - Restoring a user → account active again → notification sent.
- Covers multiple services (API, DB, logging, notifications).

8.3 Edge Case Testing

- Suspending an already suspended user should return a **409 Conflict** error.
- Restoring an already active user should return a **409 Conflict** error.
- If a user ID does not exist, system should return **404 Not Found**.
- Ensure every action (success or failure) is consistently logged in `admin_actions`.

8.4 Security Tests

- Verify that only authenticated admins can access APIs.
- Attempt API access with:
    - No token → must return **401 Unauthorized**.
    - Invalid/expired token → must return **401 Unauthorized**.
    - Valid token but non-admin role → must return **403 Forbidden**.
- Ensure HTTPS is enforced to prevent data leaks.

---

9. Deliverables

9.1 Functional API Endpoints

- Fully implemented and tested REST APIs for:
    - Listing users
    - Suspending users
    - Restoring users

9.2 Database Schema & Migration Scripts

- SQL scripts to create `users` and `admin_actions` tables.
- Migration files for evolving schema without data loss.
- Indexing for performance (e.g., on `email`, `status`).

9.3 Swagger (OpenAPI) Documentation

- Complete API specification in Swagger/OpenAPI format.
- Includes request/response examples, query params, error codes.
- Interactive API explorer for QA and developers.

9.4 Automated Tests

- **Unit tests** for core functions.
- **Integration tests** covering full flows.
- CI/CD pipeline integration to run tests automatically.

9.5 Logging & Monitoring System Integration

- Admin actions logged in both DB and centralized logging system (e.g., ELK, Datadog).
- Alerts for unusual patterns (e.g., mass suspensions by one admin).
- Dashboards for monitoring API health, error rates, and admin actions.

---

10. 8-Week Timeline

- **Week 1** → Schema design.
- **Weeks 2–3** → User list & search API.
- **Week 4** → Suspend/restore API.
- **Week 5** → Logging integration.
- **Week 6** → Security review.
- **Week 7** → Testing (unit + integration).
- **Week 8** → Documentation & demo.

---

11. Security & Compliance

The **Admin User Management Module** is designed with **security and compliance** at its core. Since administrators have elevated privileges, strict measures must be in place to prevent misuse, protect user data, and meet regulatory obligations.

11.1 Role-Based Access Control (RBAC)

- **Description:**
  Access to the admin APIs is strictly limited to authorized administrators through **Role-Based Access Control (RBAC)**.
- **Details:**
  - Only users with admin roles can call sensitive APIs (e.g., suspend, restore).
  - Roles can be further subdivided:
    - **Super Admin:** Full access (can suspend/restore users, manage other admins).
    - **Support Admin:** Limited access (can only view users, not suspend/restore).
    - **Auditor:** Read-only access to logs for compliance purposes.
  - Prevents unauthorized staff from executing high-risk actions.
- **Purpose:**
  Ensures **least privilege principle**, reducing risks of misuse or accidental errors.

11.2 HTTPS for Encrypted Communication

- **Description:**
  All communication between clients, dashboards, and APIs must be secured using **HTTPS (TLS/SSL encryption)**.
- **Details:**
  - Protects sensitive data (user information, admin credentials, API keys) from interception.
  - Prevents **man-in-the-middle (MITM) attacks**.
  - Ensures integrity of data transmitted between admin dashboards and backend APIs.

- **Purpose:**
  Provides **confidentiality and integrity** of data in transit, a baseline requirement for all secure systems.

11.3 Immutable Audit Logs for Compliance

- **Description:**
  Every admin action is recorded in **audit logs** that cannot be altered or deleted.
- **Details:**
  - Stores action details (admin ID, target user ID, action type, timestamp).
  - Logs are **append-only**, ensuring no tampering.
  - Can be integrated with external logging systems (e.g., Splunk, ELK Stack, AWS CloudTrail).
  - Meets requirements for **regulatory audits** and industry certifications.
- **Purpose:**
  Ensures **accountability and transparency**, supporting investigations and compliance checks (e.g., GDPR, SOC 2, ISO 27001).

11.4 Privacy & Data Protection

- **Description:**
  Sensitive user data is **minimized and protected** to comply with privacy laws.
- **Details:**
  - Admin dashboards display only necessary fields (e.g., name, email, status).
  - Sensitive fields like passwords, payment details, or authentication tokens are never exposed.
  - Follows data protection best practices (masking, encryption at rest, secure access policies).
  - Aligns with privacy regulations such as **GDPR, CCPA, and local data protection laws**.
- **Purpose:**
  Protects user privacy, reduces liability, and ensures trustworthiness of the platform.

12. Appendices

Example SQL Schema:

- `users (id, name, email, status, created_at, updated_at)`
- `admin_actions (id, admin_id, action, target_user_id, created_at)`

Example OpenAPI Spec:

- Endpoints for user listing, suspension, and restoration.