Admin Complete Guide

This guide covers all admin features in the getJOBS Freelance App, including Role-Based Access Control (RBAC), user management, content approval, and system monitoring.

Table of Contents

Admin Roles & Permissions

Available Roles

1. Super Admin

- **Description**: Full system access - manages all features and admins
- **Permissions**:
- `manage_admins` - Create, update, and remove admin users
- `manage_roles` - Assign and modify admin roles
- `approve_content` - Approve/reject jobs, companies, gigs
- `manage_finances` - Handle payments and transactions
- `view_analytics` - Access analytics dashboard
- `manage_users` - Suspend, warn, or remove users
- `manage_disputes` - Resolve user disputes
- `system_settings` - Configure platform settings
- **Use Case**: Primary administrator with complete control

2. Moderator

- **Description**: Content moderation - approves jobs, companies, and user content
- **Permissions**:
- `approve_jobs` - Review and approve/reject job postings

- `approve_companies` - Review and approve company profiles
- `approve_gigs` - Review and approve freelance gigs
- `manage_disputes` - Resolve user disputes
- `issue_warnings` - Issue warnings to users for violations
- `suspend_users` - Temporarily suspend user accounts
- **Use Case**: Content quality control and user behavior management

3. Analyst

- **Description**: Analytics and reporting - generates reports and insights
- **Permissions**:
- `view_analytics` - Access analytics dashboard
- `generate_reports` - Create custom reports
- `view_metrics` - View platform metrics
- `export_data` - Export data for analysis
- **Use Case**: Data analysis and business intelligence

4. Financial

- **Description**: Finance management - handles payments and transactions
- **Permissions**:
- `view_transactions` - View all transactions
- `process_refunds` - Process refund requests
- `view_wallet` - View user wallets and balances
- `generate_financial_reports` - Create financial reports
- **Use Case**: Payment processing and financial management

5. Support

- **Description**: Customer support - manages tickets and user inquiries
- **Permissions**:
- `manage_disputes` - Handle user disputes
- `contact_users` - Contact users directly
- `view_tickets` - Access support tickets
- `resolve_issues` - Mark issues as resolved
- **Use Case**: Customer support and user assistance

RBAC Management

Accessing RBAC Settings

1. Open the app as a Super Admin
2. Navigate to **Admin Panel** (usually in admin drawer)
3. Select **Admin Roles & Access Control**

Current Implementation

File: lib/screens/admin/admin_rbac_screen.dart

The RBAC screen has two tabs:

Tab 1: Role Definitions

Shows all available roles with their descriptions and associated permissions.

```
 // Role definitions in admin_rbac_screen.dart
final Map<String, String> _roleDescriptions = {
  'superAdmin': 'Full system access - manages all features and admins',
  'moderator': 'Content moderation - approves jobs, companies, and user content',
  'analyst': 'Analytics and reporting - generates reports and insights',
  'financial': 'Finance management - handles payments and transactions',
  'support': 'Customer support - manages tickets and user inquiries',
};

final Map<String, List<String>> _rolePermissions = {
  'superAdmin': [
    'manage_admins',
    'manage_roles',
    'approve_content',
    'manage_finances',
    'view_analytics',
    'manage_users',
    'manage_disputes',
    'system_settings',
  ],
  // ... other roles
};
```

Tab 2: Admin Users

Shows all current admin users and allows modification of their roles.

Assigning Roles to New Admins

Email-Based User Lookup:

1. Click **"Add Admin"** button
2. Enter the **user's email address**

3. Click **"Find User"** to verify the user exists
4. If found, select a **role** for the user
5. Click **"Add Admin"** to confirm

Implementation:

```
 // Email lookup implementation
Future<void> _lookupUserByEmail(String email) async {
  // Query users collection by email
  final userQuery = await _firestore
      .collection('users')
      .where('email', isEqualTo: email.trim().toLowerCase())
      .limit(1)
      .get();

  if (userQuery.docs.isEmpty) {
    // User not found
    return;
  }

  // Extract user ID and name
  final userData = userQuery.docs.first;
  final userId = userData.id;
  final userName = userData.data()['name'] ?? 'Unknown';

  setState(() {
    _lookedUpNewAdminId = userId;
    _lookedUpNewAdminName = userName;
  });
}

// Assign role
Future<void> _assignRole(String userId, String newRole) async {
  // Create admin_roles document
  await _firestore.collection('admin_roles').doc(userId).set({
    'role': newRole,
    'permissions': _rolePermissions[newRole] ?? [],
    'assignedAt': FieldValue.serverTimestamp(),
    'updatedAt': FieldValue.serverTimestamp(),
  });

  // Update user document
  await _firestore.collection('users').doc(userId).update({
    'isAdmin': true,
```

```
    'adminRole': newRole,
  });

  // Log audit event
  await _firestore.collection('admin_audit_logs').add({
    'timestamp': FieldValue.serverTimestamp(),
    'action': 'assign_admin_role',
    'targetUserId': userId,
    'adminId': _getCurrentAdminId(),
    'role': newRole,
    'type': 'role_management',
  });
}
```

Database Structure

admin_roles collection:

```
{
  "userId": "user123",
  "role": "moderator",
  "permissions": ["approve_jobs", "approve_companies", ...],
  "assignedAt": "2024-01-15T10:30:00Z",
  "updatedAt": "2024-01-15T10:30:00Z"
}
```

users document updates:

```
{
  "isAdmin": true,
  "adminRole": "moderator"
}
```

Changing Admin Roles

1. In the **Admin Users** tab, click on an admin
2. Select a new role from the available options
3. The role updates immediately in Firestore

Removing Admin Access

1. In the **Admin Users** tab, click on an admin
2. Click **"Remove Admin Access"**
3. The admin role and permissions are revoked

User Suspension

Purpose

Temporarily or permanently suspend users who violate platform policies.

Accessing User Suspension

1. Open Admin Panel
2. Navigate to **User Suspension**
3. Two tabs available:
4. **Active Suspensions**: Current and active suspensions
5. **Suspension History**: All past suspensions

Current Implementation

File: lib/screens/admin/admin_user_suspension_screen.dart

Creating a User Suspension

Step 1: Email-Based User Lookup

```
Future<void> _lookupUserByEmail(String email) async {
  if (email.trim().isEmpty) {
    showSnackBar('Please enter a user email');
    return;
  }

  try {
    setState(() => _isLookingUpUser = true);

    // Query users by email
    final userQuery = await FirebaseFirestore.instance
        .collection('users')
        .where('email', isEqualTo: email.trim().toLowerCase())
        .limit(1)
        .get();

    if (userQuery.docs.isEmpty) {
      setState(() => _isLookingUpUser = false);
      showSnackBar('User with email "$email" not found');
      return;
    }

    // Extract user information
    final userData = userQuery.docs.first;
    final userId = userData.id;
    final userName = userData.data()['name'] ?? 'Unknown';
```

```
    setState(() {
      _lookedUpUserId = userId;
      _lookedUpUserName = userName;
      _isLookingUpUser = false;
    });

    showSnackBar('Found: $userName');
  } catch (e) {
    setState(() => _isLookingUpUser = false);
    showSnackBar('Error looking up user: $e');
  }
}
```

Step 2: Enter Suspension Details

In the suspension dialog: 1. **User Email**: Enter the user's email 2. Click **"Find User"** to verify 3. **Reason**: Enter reason for suspension (e.g., "Posting inappropriate content") 4. **Duration**: Choose: - **Temporary**: Select end date - **Permanent**: Toggle permanent flag 5. Click **"Suspend User"**

Step 3: Create Suspension Record

```
Future<void> _createSuspension() async {
  // Validate inputs
  if (_userEmailController.text.isEmpty) {
    showSnackBar('Please enter a user email');
    return;
  }

  if (_lookedUpUserId == null) {
    showSnackBar('Please click "Find User" first');
    return;
  }

  if (_reasonController.text.isEmpty) {
    showSnackBar('Please enter a reason');
    return;
  }

  if (!_isPermanent && _suspensionEndDate == null) {
    showSnackBar('Please select an end date or mark as permanent');
    return;
  }

  try {
    final userId = _lookedUpUserId!;
```

```
// Create suspension record
await FirebaseFirestore.instance
    .collection('user_suspensions')
    .add({
      'userId': userId,
      'userEmail': _userEmailController.text.trim().toLowerCase(),
      'userName': _lookedUpUserName,
      'reason': _reasonController.text.trim(),
      'suspendedAt': FieldValue.serverTimestamp(),
      'suspendedUntil': _isPermanent
          ? null
          : Timestamp.fromDate(_suspensionEndDate!),
      'isPermanent': _isPermanent,
      'adminEmail': FirebaseAuth.instance.currentUser?.email,
      'status': 'active',
      'createdAt': FieldValue.serverTimestamp(),
    });

// Update user document
await FirebaseFirestore.instance
    .collection('users')
    .doc(userId)
    .update({
      'isSuspended': true,
      'suspensionReason': _reasonController.text.trim(),
      'suspendedUntil': _isPermanent
          ? null
          : Timestamp.fromDate(_suspensionEndDate!),
    });

// Log audit event
await FirebaseFirestore.instance
    .collection('admin_audit_logs')
    .add({
      'action': 'suspend',
      'adminEmail': FirebaseAuth.instance.currentUser?.email,
      'targetType': 'user',
      'targetId': userId,
      'targetEmail': _userEmailController.text.trim().toLowerCase(),
      'reason': _reasonController.text.trim(),
      'timestamp': FieldValue.serverTimestamp(),
      'changes': {
```

```
            'isSuspended': {'before': false, 'after': true},
            'suspendedUntil': {
              'before': null,
              'after': _isPermanent ? 'permanent' : _suspensionEndDate
            }
          },
        });

    // Show success message
    showSnackBar('User suspended successfully');

    // Reload suspensions list
    _loadSuspensions();
    Navigator.pop(context);
  } catch (e) {
    showSnackBar('Failed to suspend user: $e');
  }
}
```

Database Structure

user_suspensions collection:

```
{
  "userId": "user456",
  "userEmail": "violator@example.com",
  "userName": "John Doe",
  "reason": "Posting inappropriate content",
  "suspendedAt": "2024-01-15T10:30:00Z",
  "suspendedUntil": "2024-02-15T10:30:00Z",
  "isPermanent": false,
  "adminEmail": "admin@example.com",
  "status": "active",
  "createdAt": "2024-01-15T10:30:00Z"
}
```

users document updates:

```
{
  "isSuspended": true,
  "suspensionReason": "Posting inappropriate content",
  "suspendedUntil": "2024-02-15T10:30:00Z"
}
```

Lifting a Suspension

Step 1: Navigate to **Active Suspensions** tab

Step 2: Find the suspension record

Step 3: Click **"Lift Suspension"**

Implementation:

```
Future<void> _liftSuspension(String suspensionId, String userId) async {
  try {
    // Update suspension status to lifted
    await FirebaseFirestore.instance
        .collection('user_suspensions')
        .doc(suspensionId)
        .update({
          'status': 'lifted',
          'liftedAt': FieldValue.serverTimestamp(),
        });

    // Update user document
    await FirebaseFirestore.instance
        .collection('users')
        .doc(userId)
        .update({
          'isSuspended': false,
          'suspensionReason': null,
          'suspendedUntil': null,
        });

    // Log audit event
    await FirebaseFirestore.instance
        .collection('admin_audit_logs')
        .add({
          'action': 'lift_suspension',
          'adminEmail': FirebaseAuth.instance.currentUser?.email,
          'targetType': 'user',
          'targetId': userId,
          'timestamp': FieldValue.serverTimestamp(),
          'changes': {
            'isSuspended': {'before': true, 'after': false},
          },
        });

    showSnackBar('Suspension lifted successfully');
    _loadSuspensions();
  } catch (e) {
    showSnackBar('Failed to lift suspension: $e');
```

```
  }
}
```

Suspension Workflow

Suspended Users: - Cannot log in - Cannot post jobs or gigs - Cannot apply for positions - Cannot access chat - Can view their suspension details in app

Permission Check: Moderators with `suspend_users` permission can create suspensions.

Content Approval

Types of Content Requiring Approval

1. **Job Postings**: New jobs posted by employers
2. **Company Profiles**: New company registrations
3. **Freelancer Gigs**: New gig listings by freelancers
4. **Profile Changes**: Major profile updates (education, experience, etc.)

Approval Workflow

Status Flow:

```
Pending → Approved (Live)
  ↓
Rejected (With Reason)
```

Accessing Content Approval

1. Open Admin Panel
2. Select **Content Approval**
3. Available tabs:
4. **Jobs** - Job postings
5. **Companies** - Company profiles
6. **Gigs** - Freelancer gigs
7. **Profiles** - Profile updates

Approving Content

Step 1: View pending content in respective tab

Step 2: Click on content item

Step 3: Review details (title, description, images, etc.)

Step 4: Choose action: - **Approve**: Content goes live - **Reject**: Must provide rejection reason

Rejection Reasons

Common reasons for rejection:

For Jobs: - Incomplete job description - Inappropriate content - Suspicious requirements - Salary/budget not specified

For Companies: - Unverified company information - Missing business details - Inappropriate company name - Red flags for fraud

For Gigs: - Unclear service description - Pricing issues - Suspicious content - Quality concerns

---

Finance Management

Purpose

Manage platform monetization settings, fees, and financial transactions.

Accessing Finance Settings

1. Open Admin Panel
2. Navigate to **Finance & Monetization**
3. Two tabs available:
4. **Settings** - Fee configuration
5. **Transactions** - Payment history

Monetization Settings

File: lib/screens/admin/admin_finance_screen.dart

Configurable Fees

```
 // Finance settings model
class MonetizationSettingsModel {
  bool isIndividualMonetizationEnabled;      // Allow individuals to post jobs
  bool isCompanyMonetizationEnabled;          // Allow companies to post jobs
  double companyJobPostFee;                   // Fee for company job posting
  double individualJobPostFee;                // Fee for individual job posting
  double applicationFee;                      // Fee per job application
  double bluePageListingFee;                  // Premium listing fee
  double globalDiscountPercentage;            // Platform-wide discount
  String bankDetails;                         // Platform bank account
}
```

Fee Configuration Steps

Step 1: Open Finance & Monetization screen

Step 2: Navigate to Settings tab

Step 3: Update fees: - **Company Job Post Fee**: Cost for companies to post jobs (₱) - **Individual Job Post Fee**: Cost for individuals to post jobs (₱) - **Application Fee**: Cost per application submission (₱) - **Blue Page Listing Fee**: Premium featured listing cost (₱) - **Global Discount %**: Platform-wide discount percentage (0-100) - **Bank Details**: Platform's bank account information

Step 4: Toggle monetization by user type: - Enable/disable individual monetization - Enable/disable company monetization

Step 5: Save settings

Code Example:

```
Future<void> _saveSettings() async {
  final settings = MonetizationSettingsModel(
    isIndividualMonetizationEnabled: _isIndividualMonetizationEnabled,
    isCompanyMonetizationEnabled: _isCompanyMonetizationEnabled,
    companyJobPostFee: double.tryParse(_companyJobCostController.text) ?? 50.0,
    individualJobPostFee: double.tryParse(_individualJobCostController.text) ?? 5.0,
    applicationFee: double.tryParse(_appCostController.text) ?? 1.0,
    bluePageListingFee: double.tryParse(_bluePageCostController.text) ?? 100.0,
    globalDiscountPercentage: double.tryParse(_discountController.text) ?? 0.0,
    bankDetails: _bankController.text,
  );

  await _walletService.updateSettings(settings);
  SnackbarHelper.showSuccess(context, 'Settings Saved!');
}
```

Transaction Management

View Transactions: 1. Open Finance screen 2. Select Transactions tab 3. See all platform transactions: - Date and time - Transaction type - Amount - User/Company - Status

Filter Transactions: - By date range - By type (deposit, withdrawal, fee) - By status (completed, pending, failed) - By user

Financial Reports

Available Reports: - Total revenue by time period - Fee collection by type - Transaction breakdown - Outstanding payments - Refund history

---

Ratings System

Purpose

Manage user ratings and reviews across the platform.

Accessing Ratings Moderation

1. Open Admin Panel
2. Navigate to **Rating Moderation**
3. View all ratings with filters:
4. All Ratings
5. Flagged Ratings

Ratings Overview

File: lib/screens/admin/admin_ratings_moderation_screen.dart

Managing Ratings

View Rating Details: 1. Open Ratings Moderation screen 2. See all platform ratings 3. Filter by status (all, flagged) 4. Click on rating to view: - Rater information - Rating score (1-5 stars) - Review text - Recipient (trainee, freelancer, employer) - Date posted - Flag status and reason

Rating Moderation Actions:

Flag Suspicious Rating: 1. Open rating 2. Tap **"Flag Rating"** 3. Select reason: - Inappropriate language - Fake/spam review - Fraudulent claim - Harassment - Irrelevant content 4. Add notes 5. Tap **"Confirm Flag"**

Remove Flagged Rating: 1. Open flagged rating 2. Review evidence 3. If confirmed violation: - Tap **"Remove Rating"** - Provide removal reason - Confirm deletion 4. Rating removed from public view 5. Audit log entry created

Code Structure:

```
 // Rating moderation workflow
class AdminRatingsModerationScreen extends StatefulWidget {
  // Manage ratings with filtering and moderation actions

  void _initializeStream() {
    if (_filterStatus == 'flagged') {
      _ratingsStream = FirebaseFirestore.instance
          .collection('ratings')
          .where('isFlagged', isEqualTo: true)
          .orderBy('flaggedAt', descending: true)
          .snapshots();
    } else {
      _ratingsStream = FirebaseFirestore.instance
          .collection('ratings')
          .orderBy('createdAt', descending: true)
          .snapshots();
    }
  }
}
```

Rating Quality Standards

What to Flag: - ✅ Profanity or insults - ✅ Off-topic reviews - ✅ Spam or ads - ✅ Fake reviews - ✅ Harassment or threats - ✅ Competitors attacking competitors

What to Keep: - ✅ Honest negative reviews - ✅ Constructive criticism - ✅ Legitimate complaints - ✅ Detailed feedback

---

Disputes & Conflicts

Purpose

Resolve disputes between users (job seekers, employers, trainers).

Accessing Disputes

1. Open Admin Panel
2. Navigate to **Disputes**
3. Manage by status:
4. **Open** - New disputes
5. **In Review** - Being handled
6. **Resolved** - Closed

Dispute Management

File: lib/screens/admin/admin_disputes_screen.dart

Dispute Resolution Workflow

Step 1: Review Dispute Details 1. Open Disputes screen 2. Click on dispute 3. Review: - Dispute type - Involved parties - Timeline of events - Messages between parties - Evidence provided

Step 2: Investigate 1. Check both parties' history 2. Review transaction details 3. Check previous disputes 4. Assess credibility

Step 3: Take Action 1. Assess evidence 2. Choose resolution: - **Favor Complainant** - Award to person who filed dispute - **Favor Respondent** - Keep with original party - **Split Decision** - Partial compensation - **Request More Info** - Ask for clarification

Step 4: Communicate 1. Add message to dispute thread 2. Explain resolution 3. Notify both parties 4. Provide next steps

Step 5: Close Dispute 1. After resolution applied 2. Mark status as **Resolved** 3. Dispute history preserved 4. Audit log created

Common Dispute Types

Type	Typical Resolution
Non-payment	Verify transaction, release funds
Incomplete work	Review deliverables, assess quality
Miscommunication	Clarify expectations, refund if needed
Quality issues	Evaluate against requirements
Late delivery	Check deadline, assess impact
Fraud claim	Investigate thoroughly

Dispute Checklist

Before resolving disputes, verify: - ✅ Both parties' communications reviewed - ✅ All evidence examined - ✅ Relevant dates and times verified - ✅ Contract/agreement reviewed - ✅ Party history checked - ✅ Similar cases reviewed - ✅ Decision clearly documented - ✅ Both parties notified

---

Bulk Approval

Purpose

Efficiently approve multiple pending items at once.

Accessing Bulk Approval

1. Open Admin Panel
2. Navigate to **Bulk Approval**
3. Select content type to approve

Bulk Approval Process

File: lib/screens/admin/admin_bulk_approval_screen.dart

Step 1: Select Content Type - Jobs - Companies - Gigs - Profiles

Step 2: Review Items 1. See all pending items of selected type 2. Checkboxes to select multiple items 3. Preview each item before approval 4. Filter by: - Submission date - User/company - Category

Step 3: Batch Operations 1. Select multiple items with checkboxes 2. Choose bulk action: - **Approve All** - Approve selected items - **Reject All** - Reject with reason - **Request Changes** - Send back for revision 3. Add notes (optional)

Step 4: Confirm 1. Review selections 2. Confirm action 3. All items processed simultaneously 4. Notifications sent to users 5. Audit log created

Approval Checklist

Before bulk approval, verify each item: - ✅ Content is complete - ✅ Information is accurate - ✅ No policy violations - ✅ No spam or duplicates - ✅ Appropriate category - ✅ Contact info valid - ✅ Images/media appropriate - ✅ Pricing reasonable

---

Content Moderation

Purpose

Monitor and enforce content policies across the platform.

Accessing Content Moderation

1. Open Admin Panel
2. Navigate to **Content Moderation**
3. Review reported and suspicious content

Moderation Workflow

File: lib/screens/admin/admin_content_moderation_screen.dart

Types of Content Monitored: - Job descriptions - Company profiles - User profiles - Comments and reviews - Messages - Images and media

Reviewing Reported Content

Step 1: Open Content Moderation

Step 2: Filter by: - Content type - Status (reported, reviewed, approved) - Report reason - Date range

Step 3: Review Content 1. Click on reported item 2. View: - Original content - Report reason - Reporter details - Content creator details 3. Assess violation

Step 4: Make Decision 1. **Approve** - Content is acceptable 2. **Remove** - Content violates policy 3. **Warn Creator** - Issue warning without removing 4. **Suspend Creator** - Account action needed

Step 5: Communicate 1. Add decision note 2. Notify reporter 3. Notify content creator 4. Log action

Policy Violations

Common Violations: - Profanity or harassment - Discriminatory language - Sexual content - Spam or advertising - Misleading information - Copyright infringement - Doxxing or privacy violation - Threat or violence

Moderation Actions

Action	Impact	Notes
Approve	None	Content stays live

Action	Impact	Notes
Remove	Content deleted	User notified
Warn	None	Warning logged
Suspend	Account disabled	Temporary/permanent
Ban	Permanent removal	Account deleted

System Monitoring & Health

Purpose

Monitor platform health, performance, and system metrics.

Accessing Monitoring Dashboard

1. Open Admin Panel
2. Navigate to **Monitoring Dashboard**
3. Real-time system status

Dashboard Metrics

File: lib/screens/admin/admin_monitoring_dashboard_screen.dart

Key Metrics to Monitor

User Metrics: - Active users (daily/monthly) - New registrations - User growth rate - Engagement rate - Retention rate - Churn rate

Transaction Metrics: - Total transactions - Transaction volume - Failed transactions - Average transaction value - Payment success rate

Content Metrics: - Jobs posted - Jobs approved - Companies registered - Approval rate - Pending items count - Approval time average

Performance Metrics: - API response time - Error rates - System uptime - Database size - Storage usage - Backup status

Alerts & Thresholds

Set up alerts for: - Error rate exceeds 2% - Response time > 3 seconds - Failed payment rate > 5% - System downtime - Unusual traffic spikes - Database issues

Health Checks

Daily Checks: - Review error logs - Check pending approvals - Verify backups completed - Monitor fraud reports - Check suspension queue

Weekly Checks: - Review analytics - Audit permission logs - Check system performance - Verify security - Review user reports

Monthly Checks: - Generate compliance reports - Audit all admin actions - Review platform growth - Analyze user feedback - Plan improvements

System Maintenance

Scheduled Maintenance: 1. Announce downtime 48 hours prior 2. Backup database 3. Update system 4. Run tests 5. Restore service 6. Verify functionality

Emergency Procedures: - Immediate rollback capability - Incident communication plan - Data recovery procedures - Support escalation - Post-incident review

---

Analytics & Reporting

Dashboard Metrics

Key Metrics: - Total registered users (by type) - Active users (daily/monthly) - Active jobs - Completed contracts - Total transactions - Platform revenue - User growth trends

Available Reports

1. **User Report**: Active, suspended, deleted users
2. **Job Report**: Posted, approved, completed jobs
3. **Financial Report**: Transactions, refunds, revenue
4. **Dispute Report**: Open, resolved disputes
5. **Activity Report**: User activities and login patterns

Generating Custom Reports

1. Navigate to **Analytics & Reporting**
2. Select report type
3. Choose date range
4. Select filters (user type, status, etc.)
5. Click **"Generate Report"**
6. Option to export as CSV/PDF

System Settings

Available Settings

Platform Settings: - Platform name and description - Support email and phone - Commission rates - Payment methods

Notification Settings: - Email notifications enabled - Notification templates - Scheduling rules

Security Settings: - Password requirements - Login attempt limits - 2FA requirements - Session timeout

Feature Toggles: - Enable/disable user registration - Enable/disable job posting - Enable/disable gig creation - Maintenance mode

---

Audit Logging

Audit Log Collection

All admin actions are automatically logged in Firestore:

Collection: `admin_audit_logs`

Data Recorded:

```
{
  "timestamp": "2024-01-15T10:30:00Z",
  "action": "approve_job",
  "adminId": "admin123",
  "adminEmail": "admin@example.com",
  "targetType": "job",
  "targetId": "job456",
  "targetEmail": "employer@example.com",
  "reason": "Approved",
  "type": "content_approval",
  "changes": {
    "status": {"before": "pending", "after": "approved"}
  }
}
```

Accessing Audit Logs

1. Go to Admin Panel
2. Select **Audit Logs**
3. Filter by:
4. Date range
5. Admin user
6. Action type
7. Target type

Common Audit Actions

Action	Type	Description
`assign_admin_role`	role_management	New admin role assigned

Action	Type	Description
`remove_admin_role`	role_management	Admin role removed
`approve_job`	content_approval	Job approved
`reject_job`	content_approval	Job rejected
`approve_company`	content_approval	Company approved
`reject_company`	content_approval	Company rejected
`suspend`	user_management	User suspended
`lift_suspension`	user_management	Suspension lifted
`warn_user`	user_management	Warning issued
`change_setting`	system	System setting changed

Security Best Practices

Admin Account Security

1. **Use Strong Passwords**: Minimum 12 characters with special characters
2. **Enable 2FA**: Use authenticator apps (not SMS when possible)
3. **Unique Emails**: Don't share admin email addresses
4. **Regular Updates**: Change passwords quarterly
5. **Monitor Activity**: Check audit logs regularly

Permission Management

1. **Principle of Least Privilege**: Assign minimum necessary permissions
2. **Role Specificity**: Use specific roles, avoid super admin unless needed
3. **Regular Review**: Audit admin permissions monthly
4. **Offboarding**: Remove admin access immediately upon departure

Audit Trail Monitoring

1. **Weekly Reviews**: Check audit logs for unusual activity
2. **Alert Setup**: Monitor for mass suspensions or deletions
3. **Documentation**: Keep records of major admin actions
4. **Escalation**: Report suspicious activity to management

Common Security Patterns

Before Approving Any Content: - ✅ Verify user/company identity - ✅ Check for duplicate/spam accounts - ✅ Review content quality - ✅ Check for policy violations

Before Suspending A User: - ✅ Verify the violation - ✅ Check user history - ✅ Consider first warning - ✅ Document reason clearly

Before Adding New Admin: - ✅ Verify person's identity - ✅ Confirm employment/authorization - ✅ Start with limited permissions - ✅ Require strong password

---

Troubleshooting

Email Lookup Not Finding Users

Problem: "User with email not found" message

Solutions: 1. Verify email spelling (case-insensitive) 2. Check user email in Firestore directly 3. Ensure user account is created 4. Check if email was updated in profile

Suspension Not Taking Effect

Problem: Suspended user can still log in

Solutions: 1. Verify `isSuspended` field is `true` in user document 2. Check `suspendedUntil` date is in future 3. Clear app cache and re-login 4. Verify Firestore rules allow update

Permission Errors

Problem: Can't access certain admin features

Solutions: 1. Verify admin role has required permission 2. Check admin_roles document exists 3. Verify `isAdmin` is `true` in user document 4. Clear cache and re-login as admin

Audit Log Not Recording

Problem: Admin actions not appearing in audit logs

Solutions: 1. Verify Firestore write permissions 2. Check admin_audit_logs collection exists 3. Verify user has email set correctly 4. Check Firestore rules aren't blocking writes

---

Conclusion

This guide covers all major admin features in the getJOBS Freelance App. For additional support:

1. Check Firestore rules and structure
2. Review audit logs for action history
3. Contact technical support with specific issues
4. Refer to code documentation for implementation details

Happy administering! 🎉

Last Updated: 2024 Platform: getJOBS Freelance App