EIC Inspection App - Maintenance Guide

Overview

This guide provides comprehensive instructions for maintaining the EIC Inspection App, including routine maintenance tasks, troubleshooting procedures, performance optimization, and system monitoring.

Table of Contents

System Requirements

Development Environment

- **Node.js**: Version 16.x or higher
- **Git**: Version 2.x or higher
- **Modern Browser**: Chrome 90+, Firefox 88+, Safari 14+
- **Code Editor**: VS Code recommended with extensions:
- ES6 String HTML
- Firebase Explorer
- Tailwind CSS IntelliSense
- GitLens

Production Environment

- **Firebase Project**: Active Firebase project with billing enabled
- **Domain**: Custom domain with SSL certificate
- **CDN**: Content delivery network for static assets
- **Monitoring**: Firebase Performance Monitoring enabled

Routine Maintenance

Daily Tasks

1. System Health Check (5 minutes)

```
# Check application status
curl -I https://your-domain.com

# Verify Firebase services
firebase projects:list

# Check recent logs
firebase functions:log --limit 50
```

2. Monitor Key Metrics

- **User Activity**: Active users, login success rate
- **Performance**: Page load times, API response times
- **Errors**: JavaScript errors, Firebase errors
- **Security**: Failed login attempts, permission denials

3. Review Alerts

- Check monitoring dashboard for any alerts
- Review error logs for new issues
- Verify backup completion status
- Monitor resource usage trends

Weekly Tasks

1. Performance Review (30 minutes)

```
// Check user management performance
const userManager = new EnhancedUserManager();
const stats = userManager.getUserStats();
console.log('User Statistics:', stats);

// Review log statistics
const logs = logger.getLocalLogs();
const errorLogs = logs.filter(log => log.level === 'ERROR');
console.log('Error Count:', errorLogs.length);
```

2. Security Audit

- Review user access patterns
- Check for suspicious login attempts
- Verify role assignments are appropriate
- Review Firebase security rules
- Update dependencies with security patches

3. Data Cleanup

```
// Clean up old logs (keep last 30 days)
const thirtyDaysAgo = new Date();
thirtyDaysAgo.setDate(thirtyDaysAgo.getDate() - 30);

// Archive old reports
const oldReports = reports.filter(r => r.createdAt < thirtyDaysAgo);
```

Monthly Tasks

1. Comprehensive System Review (2 hours)

- **Performance Analysis**: Review all performance metrics
- **Security Assessment**: Complete security audit
- **User Feedback Review**: Analyze user feedback and support tickets
- **Capacity Planning**: Review resource usage and plan for growth

2. Documentation Updates

- Update API documentation
- Review and update troubleshooting guides
- Update system architecture diagrams
- Review and update maintenance procedures

3. Backup Verification

- Test backup restoration procedures
- Verify backup integrity
- Update disaster recovery procedures
- Test failover mechanisms

Monitoring & Alerting

Key Performance Indicators (KPIs)

1. Application Performance

- **Page Load Time**: Target < 2 seconds
- **API Response Time**: Target < 100ms
- **Error Rate**: Target < 1%
- **Uptime**: Target > 99.9%

2. User Experience

- **User Satisfaction**: Survey scores > 4.0/5.0
- **Feature Adoption**: Track new feature usage
- **Support Tickets**: Target < 5 per week
- **User Retention**: Monthly active users growth

3. System Health

- **Memory Usage**: Target < 80%
- **CPU Usage**: Target < 70%
- **Database Performance**: Query time < 50ms
- **Storage Usage**: Monitor growth trends

Monitoring Setup

1. Firebase Monitoring

```javascript
// Enable performance monitoring
import { getPerformance } from 'firebase/performance';
const perf = getPerformance();

// Custom metrics
const customMetric = perf.newTrace('user_management_operation');
customMetric.start();
// ... operation code ...
customMetric.stop();
```

2. Custom Logging

```javascript
// Monitor critical operations
await logger.info('System health check completed', {
  timestamp: new Date(),
  metrics: {
    activeUsers: userCount,
    errorRate: errorRate,
    responseTime: avgResponseTime
  }
}, 'SYSTEM_HEALTH');
```

3. Alert Configuration

```javascript
// Configure alert thresholds
const alertThresholds = {
  errorRate: 0.05,        // 5% error rate
  responseTime: 2000,     // 2 seconds
  failedLogins: 10,       // 10 failed attempts
  memoryUsage: 0.8        // 80% memory usage
};
```

Alert Response Procedures

1. Critical Alerts (Response within 15 minutes)

- **System Down**: Application completely unavailable
- **Security Breach**: Unauthorized access detected
- **Data Loss**: Database corruption or data deletion
- **Performance Degradation**: Response time > 5 seconds

2. Warning Alerts (Response within 1 hour)

- **High Error Rate**: Error rate > 5%
- **Slow Performance**: Response time > 2 seconds
- **Resource Usage**: Memory/CPU > 80%
- **Failed Backups**: Backup process failures

3. Informational Alerts (Response within 24 hours)

- **Usage Trends**: Unusual usage patterns
- **Feature Adoption**: Low adoption of new features
- **User Feedback**: Negative feedback trends

- **Maintenance Reminders**: Scheduled maintenance tasks

Troubleshooting

Common Issues

1. Authentication Problems

Symptoms: Users cannot log in, authentication errors
Diagnosis:

```
// Check Firebase Auth status
import { getAuth } from 'firebase/auth';
const auth = getAuth();
console.log('Auth state:', auth.currentUser);

// Check user document
const userDoc = await getDoc(doc(db, 'users', userId));
console.log('User data:', userDoc.data());
```

Solutions:
- Verify Firebase configuration
- Check user permissions in Firestore
- Review Firebase Auth settings
- Clear browser cache and cookies

2. Real-time Updates Not Working

Symptoms: UI not updating when data changes
Diagnosis:

```
// Check listener status
console.log('Active listeners:', userManager.realTimeListeners.length);

// Test manual refresh
await userManager.fetchUsers();
```

Solutions:
- Verify Firebase connection
- Check listener setup
- Review Firestore security rules
- Restart real-time listeners

3. Performance Issues

Symptoms: Slow page loads, unresponsive UI
Diagnosis:

```
// Performance profiling
console.time('user-load');
await userManager.fetchUsers();
console.timeEnd('user-load');

// Memory usage check
console.log('Memory usage:', performance.memory);
```

Solutions:
- Optimize database queries
- Implement pagination
- Add loading states
- Optimize images and assets

4. Permission Errors

Symptoms: Users cannot access features they should have access to
Diagnosis:

```
// Check user role
console.log('User role:', window.eicApp.currentUserRole);

// Verify permissions
const canManage = userManager.canManageUser(currentRole, targetRole);
console.log('Can manage:', canManage);
```

Solutions:
- Verify role assignments
- Check permission hierarchy
- Review Firestore security rules
- Update user roles if necessary

Debugging Tools

1. Browser Developer Tools

```
// Enable debug logging
logger.setLogLevel('DEBUG');
logger.setConsoleLogging(true);

// Monitor network requests
// Use Network tab in DevTools

// Check for JavaScript errors
// Use Console tab in DevTools
```

2. Firebase Debugging

```
# Firebase emulator for local testing
firebase emulators:start

# Firebase logs
firebase functions:log

# Firestore debugging
firebase firestore:indexes
```

3. Performance Profiling

```
// Performance monitoring
const observer = new PerformanceObserver((list) => {
  list.getEntries().forEach((entry) => {
    console.log('Performance entry:', entry);
  });
});
observer.observe({entryTypes: ['measure', 'navigation']});
```

Performance Optimization

Database Optimization

1. Query Optimization

```
// Use indexes for common queries
const usersQuery = query(
  collection(db, 'users'),
  where('role', '==', 'manager'),
  orderBy('createdAt', 'desc'),
  limit(20)
);

// Avoid N+1 queries
const userIds = users.map(u => u.id);
const userDocs = await Promise.all(
  userIds.map(id => getDoc(doc(db, 'users', id)))
);
```

2. Caching Strategy

```
// Implement local caching
class CacheManager {
  constructor() {
    this.cache = new Map();
    this.ttl = 5 * 60 * 1000; // 5 minutes
  }

  set(key, value) {
    this.cache.set(key, {
      value,
      timestamp: Date.now()
    });
  }

  get(key) {
    const item = this.cache.get(key);
    if (!item) return null;

    if (Date.now() - item.timestamp > this.ttl) {
      this.cache.delete(key);
      return null;
    }

    return item.value;
  }
}
```

Frontend Optimization

1. Code Splitting

```
// Lazy load modules
const loadUserManagement = async () => {
  const { EnhancedUserManager } = await import('./user-management-enhanced.js');
  return new EnhancedUserManager();
};
```

2. Image Optimization

```
// Optimize images
const optimizeImage = (file, maxWidth = 800, quality = 0.8) => {
  return new Promise((resolve) => {
    const canvas = document.createElement('canvas');
    const ctx = canvas.getContext('2d');
    const img = new Image();

    img.onload = () => {
      const ratio = Math.min(maxWidth / img.width, maxWidth / img.height);
      canvas.width = img.width * ratio;
      canvas.height = img.height * ratio;

      ctx.drawImage(img, 0, 0, canvas.width, canvas.height);
      canvas.toBlob(resolve, 'image/jpeg', quality);
    };

    img.src = URL.createObjectURL(file);
  });
};
```

Memory Management

1. Cleanup Procedures

```
// Cleanup event listeners
class ComponentManager {
  constructor() {
    this.listeners = [];
  }

  addListener(element, event, handler) {
    element.addEventListener(event, handler);
    this.listeners.push({ element, event, handler });
  }

  cleanup() {
    this.listeners.forEach(({ element, event, handler }) => {
      element.removeEventListener(event, handler);
    });
    this.listeners = [];
  }
}
```

2. Memory Leak Detection

```
// Monitor memory usage
const monitorMemory = () => {
  if (performance.memory) {
    const memory = performance.memory;
    logger.info('Memory usage', {
      used: memory.usedJSHeapSize,
      total: memory.totalJSHeapSize,
      limit: memory.jsHeapSizeLimit
    }, 'PERFORMANCE');
  }
};

setInterval(monitorMemory, 60000); // Check every minute
```

Security Maintenance

Regular Security Tasks

1. Access Review (Monthly)

```
// Review user access patterns
const reviewUserAccess = async () => {
  const users = await userManager.fetchUsers();
  const suspiciousUsers = users.filter(user => {
    const lastLogin = new Date(user.lastLogin);
    const ninetyDaysAgo = new Date();
    ninetyDaysAgo.setDate(ninetyDaysAgo.getDate() - 90);

    return lastLogin < ninetyDaysAgo && user.isActive;
  });

  logger.warn('Inactive users found', { count: suspiciousUsers.length }, 'SECURITY');
};
```

2. Permission Audit

```
// Audit role assignments
const auditPermissions = async () => {
  const users = await userManager.fetchUsers();
  const roleDistribution = {};

  users.forEach(user => {
    roleDistribution[user.role] = (roleDistribution[user.role] || 0) + 1;
  });

  logger.info('Role distribution', roleDistribution, 'SECURITY');
};
```

3. Security Log Review

```
// Review security events
const reviewSecurityLogs = async () => {
  const securityLogs = logger.getLocalLogs('ERROR', 'SECURITY');
  const recentLogs = securityLogs.filter(log => {
    const logDate = new Date(log.timestamp);
    const oneDayAgo = new Date();
    oneDayAgo.setDate(oneDayAgo.getDate() - 1);
    return logDate > oneDayAgo;
  });

  if (recentLogs.length > 10) {
    logger.error('High security event count', { count: recentLogs.length }, 'SECURITY')
;
  }
};
```

Security Updates

1. Dependency Updates

```
# Check for security vulnerabilities
npm audit

# Update dependencies
npm update

# Fix vulnerabilities
npm audit fix
```

2. Firebase Security Rules Review

```
// Review and update Firestore rules
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Users collection
    match /users/{userId} {
      allow read, write: if request.auth != null &&
        (request.auth.uid == userId ||
         get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == '
superadmin');
    }

    // Logs collection - admin only
    match /logs/{logId} {
      allow read, write: if request.auth != null &&
        get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role in ['
admin', 'superadmin'];
    }
  }
}
```

Backup & Recovery

Backup Procedures

1. Automated Backups

```
# Firebase backup script
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/backups/firebase_$DATE"

# Export Firestore data
gcloud firestore export gs://your-backup-bucket/firestore_$DATE

# Export Authentication users
firebase auth:export users_$DATE.json

# Backup configuration files
cp -r src/ $BACKUP_DIR/src/
cp -r docs/ $BACKUP_DIR/docs/
```

2. Manual Backup Verification

```
// Verify backup integrity
const verifyBackup = async () => {
  try {
    // Test data restoration
    const testCollection = collection(db, 'backup_test');
    const testDoc = await addDoc(testCollection, { test: true });
    await deleteDoc(testDoc);

    logger.info('Backup verification successful', null, 'BACKUP');
  } catch (error) {
    logger.error('Backup verification failed', error, 'BACKUP');
  }
};
```

Recovery Procedures

1. Data Recovery

```
# Restore from backup
gcloud firestore import gs://your-backup-bucket/firestore_YYYYMMDD_HHMMSS

# Restore user authentication
firebase auth:import users_YYYYMMDD_HHMMSS.json
```

2. Application Recovery

```
# Rollback to previous version
git checkout v1.0.0-stable

# Redeploy application
firebase deploy

# Verify functionality
curl -I https://your-domain.com
```

Updates & Upgrades

Update Procedures

1. Dependency Updates

```
# Check outdated packages
npm outdated

# Update packages
npm update

# Test after updates
npm test
npm run lint
```

2. Firebase SDK Updates

```
// Update Firebase imports
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

// Test Firebase functionality
const testFirebaseConnection = async () => {
  try {
    const testDoc = doc(db, 'test', 'connection');
    await setDoc(testDoc, { timestamp: new Date() });
    await deleteDoc(testDoc);
    logger.info('Firebase connection test successful', null, 'SYSTEM');
  } catch (error) {
    logger.error('Firebase connection test failed', error, 'SYSTEM');
  }
};
```

Upgrade Planning

1. Version Planning

- **Patch Updates**: Bug fixes, security patches (weekly)
- **Minor Updates**: New features, improvements (monthly)
- **Major Updates**: Breaking changes, architecture updates (quarterly)

2. Testing Strategy

- **Development Testing**: Local testing with emulators
- **Staging Testing**: Full integration testing
- **Production Testing**: Gradual rollout with monitoring
- **Rollback Plan**: Quick rollback procedures

Documentation Maintenance

Documentation Updates

1. Code Documentation

```
/**
 * Enhanced User Manager for comprehensive user management
 * @class EnhancedUserManager
 * @description Provides CRUD operations, real-time updates, and validation for user
management
 * @version 1.0.0
 * @author Development Team
 * @since 2025-06-30
 */
class EnhancedUserManager {
  /**
   * Create a new user with validation
   * @param {Object} userData - User data object
   * @param {string} userData.email - User email address
   * @param {string} userData.password - User password
   * @param {string} userData.displayName - User display name
   * @param {string} userData.role - User role
   * @returns {Promise<Object>} Created user object
   * @throws {Error} Validation or permission errors
   */
  async createUser(userData) {
    // Implementation
  }
}
```

2. API Documentation

```
## User Management API

### Create User
**Endpoint**: `userManager.createUser(userData)`
**Method**: Async function
**Parameters**:
- `userData` (Object): User information
  - `email` (string): Valid email address
  - `password` (string): Minimum 6 characters
  - `displayName` (string): 2-50 characters
  - `role` (string): Valid role ID

**Returns**: Promise<User>
**Throws**: ValidationError, PermissionError
```

Documentation Review Process

1. Monthly Review

- Review all documentation for accuracy
- Update screenshots and examples
- Check for broken links
- Update version numbers

2. Quarterly Audit

- Complete documentation audit
- User feedback integration
- Architecture diagram updates
- Process improvement documentation

Document Version: 1.0.0
Last Updated: June 30, 2025
Next Review: July 30, 2025
Maintained By: Development Team
Emergency Contact: admin@eic.com