

Admin Console Implementation Guide

Date: October 27, 2025
Status: In Progress

Overview

The Admin Console provides comprehensive system management capabilities for administrators including:

1. User Management (Add, Edit, Delete)
2. Vendor Management
3. Department Code Management
4. Budget Management
5. Requisition Reassignment
6. Reports Generation

Access the Admin Console

URL: <http://localhost:3000/admin.html>

Login Credentials:

- Username: admin
- Password: admin123

Implementation Status

Completed

1. Admin Console UI structure with tabbed interface
2. Authentication check (Admin role required)
3. Dashboard with statistics overview
4. Navigation between different management sections

To Implement

The following sections need full implementation. I'll provide the code structure for each:

1. Backend API Endpoints Needed

Add these endpoints to `backend/server.js`:

```
// =====
// ADMIN ENDPOINTS
// =====

// Get admin statistics
app.get('/api/admin/stats', authenticate, authorize('admin'), (req, res, next) => {
    try {
        db.get(`SELECT COUNT(*) as total FROM users`, (err, userCount) => {
            db.get(`SELECT COUNT(*) as total FROM requisitions`, (err2, reqCount) => {
                db.get(`SELECT COUNT(*) as total FROM vendors`, (err3, vendorCount) => {
                    db.get(`SELECT COUNT(DISTINCT department) as total FROM users WHERE department IS NOT NULL`, (err4, deptCount) => {
                        res.json({
                            totalUsers: userCount?.total || 0,
                            totalRequisitions: reqCount?.total || 0,
                            totalVendors: vendorCount?.total || 0,
                            activeDepartments: deptCount?.total || 0
                        });
                    });
                });
            });
        });
    } catch (error) {
        next(error);
    }
});

// ===== USER MANAGEMENT =====

// Get all users
app.get('/api/admin/users', authenticate, authorize('admin'), (req, res, next) => {
    try {
        db.all(`
            SELECT id, username, full_name, email, role, department, created_at
            FROM users
            ORDER BY created_at DESC
        `, (err, users) => {
            if (err) {
                logError(err, { context: 'get_all_users' });
            }
        });
    }
});
```

```

        return next(new AppError('Database error', 500));
    }
    res.json(users);
  });
} catch (error) {
  next(error);
}
});

// Create new user
app.post('/api/admin/users', authenticate, authorize('admin'), (req, res, next) => {
  try {
    const { username, full_name, email, password, role, department } = req.body;

    // Validation
    if (!username || !full_name || !password || !role) {
      return res.status(400).json({ error: 'Required fields missing' });
    }

    // Check if username exists
    db.get(`SELECT id FROM users WHERE username = ?`, [username], (err, existing) => {
      if (existing) {
        return res.status(400).json({ error: 'Username already exists' });
      }

      // Hash password
      const bcrypt = require('bcrypt');
      const hashedPassword = bcrypt.hashSync(password, 10);

      db.run(
        `INSERT INTO users (username, full_name, email, password, role, department)
        VALUES (?, ?, ?, ?, ?, ?)`
      , [username, full_name, email, hashedPassword, role, department], function(err) {
        if (err) {
          logError(err, { context: 'create_user' });
          return next(new AppError('Failed to create user', 500));
        }

        res.json({
          success: true,
          message: 'User created successfully',
          userId: this.lastID
        });
      });
    } catch (error) {
      next(error);
    }
  });
}

// Update user
app.put('/api/admin/users/:id', authenticate, authorize('admin'), (req, res, next) => {
  try {
    const userId = req.params.id;
    const { full_name, email, role, department, password } = req.body;

    let query =
      `UPDATE users
       SET full_name = ?, email = ?, role = ?, department = ?
      `;
    let params = [full_name, email, role, department];

    // If password is provided, update it
    if (password) {
      const bcrypt = require('bcrypt');
      const hashedPassword = bcrypt.hashSync(password, 10);
      query =
        `UPDATE users
         SET full_name = ?, email = ?, role = ?, department = ?, password = ?
        `;
      params = [full_name, email, role, department, hashedPassword];
    }

    query += ` WHERE id = ?`;
    params.push(userId);

    db.run(query, params, function(err) {
      if (err) {
        logError(err, { context: 'update_user', user_id: userId });
        return next(new AppError('Failed to update user', 500));
      }

      res.json({
        success: true,
        message: 'User updated successfully'
      });
    });
  }
});

```

```

        });
    } catch (error) {
        next(error);
    }
});

// Delete user
app.delete('/api/admin/users/:id', authenticate, authorize('admin'), (req, res, next) => {
    try {
        const userId = req.params.id;

        // Prevent deleting the admin user
        if (userId === req.user.id) {
            return res.status(400).json({ error: 'Cannot delete your own account' });
        }

        db.run(`DELETE FROM users WHERE id = ?`, [userId], function(err) {
            if (err) {
                logError(err, { context: 'delete_user', user_id: userId });
                return next(new AppError('Failed to delete user', 500));
            }

            res.json({
                success: true,
                message: 'User deleted successfully'
            });
        });
    } catch (error) {
        next(error);
    }
});

// ===== VENDOR MANAGEMENT =====

// Create vendor
app.post('/api/admin/vendors', authenticate, authorize('admin'), (req, res, next) => {
    try {
        const { name, contact_person, email, phone, address } = req.body;

        if (!name) {
            return res.status(400).json({ error: 'Vendor name is required' });
        }

        db.run(`
            INSERT INTO vendors (name, contact_person, email, phone, address)
            VALUES (?, ?, ?, ?, ?)
        `, [name, contact_person, email, phone, address], function(err) {
            if (err) {
                logError(err, { context: 'create_vendor' });
                return next(new AppError('Failed to create vendor', 500));
            }

            res.json({
                success: true,
                message: 'Vendor created successfully',
                vendorId: this.lastID
            });
        });
    } catch (error) {
        next(error);
    }
});

// Update vendor
app.put('/api/admin/vendors/:id', authenticate, authorize('admin'), (req, res, next) => {
    try {
        const vendorId = req.params.id;
        const { name, contact_person, email, phone, address } = req.body;

        db.run(`
            UPDATE vendors
            SET name = ?, contact_person = ?, email = ?, phone = ?, address = ?
            WHERE id = ?
        `, [name, contact_person, email, phone, address, vendorId], function(err) {
            if (err) {
                logError(err, { context: 'update_vendor', vendor_id: vendorId });
                return next(new AppError('Failed to update vendor', 500));
            }

            res.json({
                success: true,
                message: 'Vendor updated successfully'
            });
        });
    } catch (error) {
        next(error);
    }
});

```

```

        }
    });

// Delete vendor
app.delete('/api/admin/vendors/:id', authenticate, authorize('admin'), (req, res, next) => {
    try {
        const vendorId = req.params.id;

        db.run(`DELETE FROM vendors WHERE id = ?`, [vendorId], function(err) {
            if (err) {
                logError(err, { context: 'delete_vendor', vendor_id: vendorId });
                return next(new AppError('Failed to delete vendor', 500));
            }

            res.json({
                success: true,
                message: 'Vendor deleted successfully'
            });
        });
    } catch (error) {
        next(error);
    }
});

// ===== DEPARTMENT & BUDGET MANAGEMENT =====

// Get all departments with budgets
app.get('/api/admin/departments', authenticate, authorize('admin'), (req, res, next) => {
    try {
        db.all(`
            SELECT DISTINCT department,
            (SELECT SUM(total_amount) FROM requisitions WHERE department = users.department AND status = 'completed') as spent
            FROM users
            WHERE department IS NOT NULL
            ORDER BY department
        `, (err, departments) => {
            if (err) {
                logError(err, { context: 'get_departments' });
                return next(new AppError('Database error', 500));
            }
            res.json(departments || []);
        });
    } catch (error) {
        next(error);
    }
});

// ===== REQUISITION REASSIGNMENT =====

// Get requisitions pending approval
app.get('/api/admin/pending-requisitions', authenticate, authorize('admin'), (req, res, next) => {
    try {
        db.all(`
            SELECT r.*, u.full_name as created_by_name, u.department
            FROM requisitions r
            JOIN users u ON r.created_by = u.id
            WHERE r.status IN ('pending_hod', 'pending_md', 'procurement_completed')
            ORDER BY r.created_at DESC
        `, (err, requisitions) => {
            if (err) {
                logError(err, { context: 'get_pending_requisitions' });
                return next(new AppError('Database error', 500));
            }
            res.json(requisitions || []);
        });
    } catch (error) {
        next(error);
    }
});

// Reassign requisition
app.put('/api/admin/reassign/:id', authenticate, authorize('admin'), (req, res, next) => {
    try {
        const reqId = req.params.id;
        const { new_approver_id, comments } = req.body;

        if (!new_approver_id || !comments) {
            return res.status(400).json({ error: 'New approver and comments are required' });
        }

        // Log the reassignment
        db.run(`
            INSERT INTO audit_log (requisition_id, user_id, action, details)
            VALUES (?, ?, 'reassigned', ?)
        `, [reqId, req.user.id, `Reassigned by admin. ${comments}`], function(err) {
            if (err) {

```

```

        logError(err, { context: 'reassign_log', requisition_id: reqId });
    });

    res.json({
        success: true,
        message: 'Requisition reassigned successfully'
    });
} catch (error) {
    next(error);
}
});

// ===== REPORTS =====

// Generate summary report
app.get('/api/admin/reports/summary', authenticate, authorize('admin'), (req, res, next) => {
    try {
        const { start_date, end_date, department } = req.query;

        let query = `
            SELECT
                COUNT(*) as total_requisitions,
                SUM(CASE WHEN status = 'completed' THEN 1 ELSE 0 END) as completed,
                SUM(CASE WHEN status = 'rejected' THEN 1 ELSE 0 END) as rejected,
                SUM(CASE WHEN status LIKE '%pending%' THEN 1 ELSE 0 END) as pending,
                SUM(CASE WHEN status = 'completed' THEN total_amount ELSE 0 END) as total_spent
            FROM requisitions r
            JOIN users u ON r.created_by = u.id
            WHERE 1=1
        `;

        const params = [];

        if (start_date) {
            query += ` AND r.created_at >= ?`;
            params.push(start_date);
        }
        if (end_date) {
            query += ` AND r.created_at <= ?`;
            params.push(end_date);
        }
        if (department) {
            query += ` AND u.department = ?`;
            params.push(department);
        }

        db.get(query, params, (err, report) => {
            if (err) {
                logError(err, { context: 'generate_report' });
                return next(new AppError('Failed to generate report', 500));
            }

            res.json(report || {});
        });
    } catch (error) {
        next(error);
    }
});
});

```

2. Frontend Implementation

The admin console UI is already created with tabs. Each tab needs its full implementation.

I've created the basic structure. The file is located at:
frontend/admin.html

3. Access Instructions

1. Start the backend server:

```
cd backend
npm start
```

2. Start the frontend server:

```
cd frontend
python -m http.server 3000
```

3. Access Admin Console:

- Open browser: <http://localhost:3000/admin.html>
- Login as admin: admin / admin123

4. Add link in main dashboard:

In index.html, add admin link for admin users

4. Next Steps

1. Add the backend endpoints to server.js
 2. Implement each tab's UI with forms and tables
 3. Connect frontend to backend APIs
 4. Add validation and error handling
 5. Test all CRUD operations
-

Features Summary

User Management

- View all users in table
- Add new user with role assignment
- Edit user details
- Delete users (except self)
- Password reset capability

Vendor Management

- View all vendors
- Add new vendor
- Edit vendor details
- Delete vendor

Department Management

- View departments with spending
- Monitor budget utilization

Requisition Reassignment

- View pending requisitions
- Reassign to different approver
- Add comments for reassignment

Reports

- Generate summary reports
 - Filter by date range
 - Filter by department
 - Export capability
-

Status: Foundation created. Backend endpoints provided above. Full UI implementation in progress.