Nonprofit Fund Accounting System - Administrator's Guide (v8.8)

# Nonprofit Fund Accounting System - Administrator's Guide (v8.8)

## 1. Introduction and Scope

Welcome to the Administrator's Guide for the Nonprofit Fund Accounting System. This document provides comprehensive instructions for system administrators responsible for deploying, managing, and maintaining the application.

This guide is intended for technical users with command-line access to the server environment. It covers installation, configuration, security, maintenance, and troubleshooting for both traditional Linux and Docker-based deployments. For end-user functionality, please refer to the `nonprofit_accounting_user_guide.html`.

---

## 2. System Requirements

### 2.1. Traditional Linux Deployment

- **Operating System**: Ubuntu Server 22.04 LTS (Recommended) or other modern Debian/RHEL-based distro.
- **Software**:
  - Node.js v18.x or later
  - PostgreSQL v16 or later
  - Nginx (or another web server as a reverse proxy)
  - Git
- **Hardware**: Minimum 1 vCPU, 1 GB RAM, 25 GB Storage.

### 2.2. Docker Deployment (Windows or Linux)

- **Docker Desktop for Windows** with WSL 2 backend enabled, or **Docker Engine** on Linux.
- **Docker Compose**
- **Git**
- **Hardware**: Minimum 2 CPU cores, 4 GB RAM, 50 GB Storage.

---

# 3. Installation Procedures

## 3.1. Traditional Linux Server

This is a summary of the full deployment process.

1. **Connect & Update**: `ssh user@your-server-ip` then `sudo apt update && sudo apt upgrade -y`.
2. **Create App User**: `sudo adduser npfa` and `sudo usermod -aG sudo npfa`. Switch to user with `sudo su - npfa`.
3. **Install Dependencies**:
   - **Node.js**: Use `nodesource` to install v18.
   - **PostgreSQL**: `sudo apt install postgresql postgresql-contrib`.
   - **Nginx**: `sudo apt install nginx`.
4. **Clone Repository**: `git clone -b v8.8 https://github.com/tpfbill/nonprofit-fund-accounting.git` into `/home/npfa/`.
5. **Install App Dependencies**: `cd nonprofit-fund-accounting && npm install`.
6. **Configure Environment**: Create a `.env` file with your database credentials and `NODE_ENV=production`.
7. **Set up Database**: Create the user and database in PostgreSQL.
8. **Initialize Database**: Run the comprehensive database initialization script:

   ```
   psql -h localhost -U postgres -d fund_accounting_db -f db-init.sql
   node add-test-data.js
   ```

9. **Create Systemd Service**: Create `/etc/systemd/system/npfa.service` to run the app as a service.
10. **Configure Nginx**: Set up a reverse proxy in `/etc/nginx/sites-available/` to forward requests to the Node.js app on port 3000.
11. **Enable Firewall & SSL**: Use `ufw` and `certbot` to secure the server.

## 3.2. Docker Deployment (Windows/Linux)

For detailed instructions, refer to **`DOCKER_SETUP_WINDOWS.md`**. The process is:

1. **Clone Repository**: `git clone -b v8.8 https://github.com/tpfbill/nonprofit-fund-accounting.git`.
2. **Configure Environment**: Rename `.env.docker` to `.env`.
3. **Build & Run**: `docker-compose up -d --build`.
4. **Initialize Database**:

```
docker-compose exec app sh
# Inside container:
apk add --no-cache postgresql-client
psql -h db -U postgres -d fund_accounting_db -f db-init.sql
node add-test-data.js
exit
```

5. **Access**: Open `http://localhost:3000` in your browser.

---

# 4. Initial System Configuration

After installation, perform these initial setup steps via the application's UI:

1. **Log In**: Use the default administrator credentials (if any) or create the first user directly in the database.
2. **Organization Settings**: Navigate to `Settings > Organization` and configure your organization's name, address, and financial settings.
3. **Fiscal Years**: Navigate to `Settings > Fiscal Years` to define your fiscal periods.
4. **Chart of Accounts**: Navigate to `Chart of Accounts` and either import your CoA from AccuFund or configure it manually.
5. **Funds**: Navigate to `Funds` and set up your fund structure.

---

# 5. Database Administration

- **Read-Only User for BI Tools**: It is highly recommended to create a separate, read-only user for connecting third-party BI tools like Metabase or Tableau.

```
-- Connect to psql as the postgres user
CREATE ROLE readonly_user LOGIN PASSWORD 'your_secure_password';
GRANT CONNECT ON DATABASE fund_accounting_db TO readonly_user;
GRANT USAGE ON SCHEMA public TO readonly_user;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly_user;
```

```
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES
        TO readonly_user;
```

- **Comprehensive Database Initialization**: As of v8.8, a comprehensive database initialization script (`db-init.sql`) is provided that creates all necessary tables with proper relationships. This script includes:

  - Creation of the pgcrypto extension for UUID generation
  - Tables for users, entities, funds, accounts, journal entries, and bank accounts
  - Appropriate indexes and constraints
  - Default permissions

  To use this script:

  ```
  psql -h localhost -U postgres -d fund_accounting_db -f db-init.sql
  ```

- **Manual Migrations**: If a future update requires a database schema change, a migration script (`.sql` file) will be provided. Run it using `psql`. Always back up the database first.

---

# 6. User Management

User management is handled within the application UI: * **Location**: `Settings > Users` * **Actions**: * **Add User**: Click "Add User" and fill in the details. * **Edit User**: Click the "Edit" button next to a user to change their name, email, or role. * **Deactivate User**: Edit the user and change their status to "Inactive".

---

# 7. Entity and Fund Hierarchy Administration

The organizational structure is critical for proper reporting. * **Reference**: `docs/HIERARCHY_GUIDE.md` provides a deep dive into the architecture. * **Management**: Handled in `Settings > Entities`. * **Key Principles**: * There should be **one** top-level entity with `is_consolidated = true`. * Child entities should correctly point to their parent. * Funds are always associated with a single entity. * Use the "All Entities" filter on the Funds and Journal Entries pages to manage items before deleting an entity.

## 7.4 Inter-Entity Transfer Configuration

The **Inter-Entity Transfer** feature allows you to move cash between sibling entities while automatically creating a balanced pair of journal entries and maintaining accurate **Due To / Due From** balances.

### 1. Purpose

- Eliminate manual dual-entry work when one legal entity advances or reimburses funds to another.
- Preserve an auditable trail by linking the two journal entries with a shared `matching_transaction_id`.

### 2. Required Account Types

Each entity that will participate in transfers must have:

| Account Type | Typical Code Range | Example Name |
|---|---|---|
| **Asset** | 19-xxxx | "Due From <Other Entity>" |
| **Liability** | 29-xxxx | "Due To <Other Entity>" |

These accounts should be **Active** and mapped to the correct entity.

### 3. Setup Checklist

1. Navigate to **Chart of Accounts → Add Account**.
2. Select the **Entity** field so the account belongs to the right legal entity.
3. Choose **Type = Asset** (for "Due From") or **Liability** (for "Due To").
4. Use consistent codes (e.g., `1901`, `2901`) so the dropdown filters in the wizard recognise them.
5. Repeat for every pair of entities that may transact.

### 4. Best Practices

- Create one dedicated Due To / Due From account **per counter-party** instead of a single pooled account.
- Agree on a naming convention (e.g., "Due From – TPF" / "Due To – TPF-ES") to make reconciliation easier.
- Post transfers **as-of the actual cash movement date** to keep books aligned with bank activity.
- Review the **Inter-Entity Transfers** list (Reports → Inter-Entity) monthly to ensure all pairs are present and balanced.

### 5. Troubleshooting

| Issue | Likely Cause | Resolution |
|---|---|---|
| "No Due From accounts | Asset account not matching filter (name lacks "Due From" or code | Edit or create the correct account and retry. |

| Issue | Likely Cause | Resolution |
|---|---|---|
| found" in wizard | not 19…) | |
| Transfer posts but entity balances don't match | One of the journal entries was edited or deleted | Use the **matching_transaction_id** to locate both entries, correct or repost. |
| Wizard hangs on submission | Database constraint failure (missing fund/account ID) | Check server logs; ensure all selected IDs exist and are active. |

# 8. Bank Account Management System

The Bank Account Management System (added in v8.7) provides comprehensive tools for managing bank accounts, tracking balances, and monitoring connections.

## 8.1. Features

- **Account Management**: Create, edit, and delete bank accounts
- **Connection Testing**: Verify connectivity to bank accounts
- **Synchronization**: Update account information and balances
- **Balance Tracking**: Monitor account balances across all connected accounts

## 8.2. Configuration

Bank accounts are managed through the application UI:

1. **Access**: Navigate to `Settings > Bank Accounts`
2. **Add Account**: Click "Add Bank Account" and fill in the required details:
    - Bank Name: Select from predefined options or enter a custom name
    - Account Name: Descriptive name for the account
    - Account Number: The actual account number (masked in the UI for security)
    - Routing Number: Bank routing number
    - Account Type: Checking, Savings, Credit Line, etc.
    - Connection Method: Manual, Online Banking, or API Integration
    - Status: Active, Inactive, or Pending
    - Initial Balance: Starting balance for the account
    - Description: Optional notes about the account
3. **Test Connection**: Use the "Test" button to verify connectivity
4. **Sync All Accounts**: Click "Sync All Accounts" to update all account information

## 8.3. Database Schema

The bank account data is stored in the `bank_accounts` table:

```sql
CREATE TABLE IF NOT EXISTS bank_accounts (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    bank_name VARCHAR(100) NOT NULL,
    account_name VARCHAR(100) NOT NULL,
    account_number VARCHAR(50),
    routing_number VARCHAR(50),
    type VARCHAR(50) NOT NULL,
    status VARCHAR(20) DEFAULT 'Active',
    balance DECIMAL(15,4) DEFAULT 0,
    connection_method VARCHAR(50) DEFAULT 'Manual',
    description TEXT,
    last_sync TIMESTAMPTZ,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

## 8.4. API Endpoints

The following API endpoints are available for bank account management:

- `GET /api/bank-accounts`: List all bank accounts
- `POST /api/bank-accounts`: Create a new bank account
- `PUT /api/bank-accounts/:id`: Update an existing bank account
- `DELETE /api/bank-accounts/:id`: Delete a bank account
- `POST /api/bank-accounts/sync`: Synchronize all bank accounts
- `POST /api/bank-accounts/:id/test-connection`: Test connection to a specific bank account

## 8.5. Troubleshooting

| Issue | Likely Cause | Resolution |
|---|---|---|
| Connection test fails | Network connectivity issues or incorrect credentials | Check network settings, verify account credentials, ensure the bank's API is operational |
| Bank accounts not displaying | Database connectivity issue or permission problem | Verify database connection, check if the bank_accounts table exists, ensure proper permissions |
| Sync operation hangs | Timeout connecting to bank API or rate limit- | Check server logs, increase timeout settings, implement retry logic with exponential |

| Issue | Likely Cause | Resolution |
|---|---|---|
| | ing | backoff |
| Balance discrepancies | Manual edits or failed sync operations | Reconcile with bank statements, check transaction history, perform manual balance adjustment |

### 8.6. Security Considerations

- **Sensitive Data**: Account numbers and routing numbers should be encrypted in the database
- **API Credentials**: Store bank API credentials securely using environment variables
- **Access Control**: Restrict bank account management to administrative users only
- **Audit Trail**: Maintain logs of all changes to bank account information

---

# 9. Backup and Recovery Procedures

## 9.1. Backup

Automated backup scripts are included in the `scripts/` directory.

- **Linux**: Schedule `scripts/update-linux.sh` to run daily via `cron`. It automatically creates backups before updating. For manual backups:

  ```
  pg_dump -U postgres -h localhost fund_accounting_db > backup.sql
  ```

- **Docker**: Schedule `scripts/update-docker-windows.ps1` to run daily. For manual backups:

  ```
  docker-compose exec -T db pg_dump -U postgres -d
          fund_accounting_db > backup.sql
  ```

## 9.2. Recovery

In case of failure, use the backups to restore the system.

- **Linux**:
  1. Stop the application: `sudo systemctl stop npfa`.
  2. Restore the database: `sudo -u postgres psql -d fund_accounting_db -f /path/to/backup.sql`.
  3. Restore application files if needed from the `.tar.gz` backup.
  4. Start the application: `sudo systemctl start npfa`.

- **Docker**:
  1. Stop the containers: `docker-compose down`.
  2. Restore the database: `cat /path/to/backup.sql | docker-compose exec -T db psql -U postgres -d fund_accounting_db`.
  3. Start the containers: `docker-compose up -d`.

---

# 10. System Monitoring and Maintenance

- **Checking Service Status**:
  - **Linux**: `sudo systemctl status npfa.service`
  - **Docker**: `docker-compose ps`
- **Viewing Logs**:
  - **Linux**: `sudo journalctl -u npfa -f` (live logs)
  - **Docker**: `docker-compose logs -f app` (live logs)
- **Log Rotation**: Ensure `logrotate` is configured for Nginx and the application logs to prevent disk space issues.
- **Database Maintenance**: Periodically run `VACUUM` and `ANALYZE` on the PostgreSQL database to maintain performance.

---

# 11. Security Considerations

- **Firewall**: Use `ufw` on Linux to allow only necessary ports (SSH, HTTP, HTTPS).
- **SSL/HTTPS**: Always deploy with an SSL certificate. Use Certbot with Nginx for free, auto-renewing certificates.
- **Database Security**:
  - Use a strong, unique password for the `postgres` user.
  - Configure `pg_hba.conf` to only allow connections from trusted hosts.
  - Use the dedicated read-only user for external BI tools.
- **Application Security**:
  - Run the Node.js process as a non-root user (`npfa`).
  - Keep Node.js and npm packages updated (`npm audit`).
- **SSH Security**: Disable password authentication and use SSH keys for server access.

---

# 12. Performance Tuning

- **Database**: Ensure PostgreSQL has adequate `shared_buffers` and `work_mem` configured in `postgresql.conf` based on server RAM.
- **Nginx**: Enable gzip compression and caching for static assets in your Nginx site

configuration.
- **Node.js**: For high-traffic environments, use a process manager like `pm2` to run the Node.js application in cluster mode, leveraging all CPU cores.

---

# 13. Troubleshooting Common Issues

| Symptom | Likely Cause | Solution |
|---|---|---|
| **502 Bad Gateway** | The Node.js application (`npfa.service` or `app` container) is not running or has crashed. | 1. Check the application logs for errors (`journalctl` or `docker-compose logs`).<br>2. Ensure the database is running.<br>3. Restart the application service. |
| **"DB Offline"** | The Node.js application cannot connect to the PostgreSQL database. | 1. Verify PostgreSQL is running.<br>2. Check the `.env` file for correct database credentials (host, user, password, dbname).<br>3. Ensure network connectivity between the app and DB servers (or containers). |
| **404 Not Found on API routes** | Nginx is not proxying requests correctly, or the Node.js routes are not registered. | 1. Ensure the API routes are defined *before* the `app.use(express.static(...))` line in `server.js`.<br>2. Check the Nginx configuration for a correct `proxy_pass` directive. |
| **Permission Denied errors** | File system permissions are incorrect. | Ensure the `npfa` user owns the application directory: `sudo chown -R npfa:npfa /home/npfa/nonprofit-fund-accounting`. |

---

# 14. Update and Upgrade Procedures

The repository includes automated update scripts in the `scripts/` directory. **It is highly recommended to use these scripts for all updates.**

- **Linux**: `scripts/update-linux.sh`
- **Docker on Windows**: `scripts/update-docker-windows.ps1`

These scripts automatically handle backups, code pulling, dependency updates, service restarts, and verification. Refer to the comments within the scripts for detailed information.

---

# 15. Integration Management

- **Natural Language Query (NLQ)**: The NLQ engine relies on pattern matching defined in `natural-language-queries.html`. To add new query types, new patterns must be added to the `NLQ_PATTERNS` object in that file.
- **Custom Reports Builder**: This feature uses a secure `REPORT_BUILDER_FIELD_MAP` in `server.js` to whitelist accessible tables and fields. To expose new data sources or fields to the report builder, they must be added to this map.
- **BI Tools**: Connect using the dedicated read-only database user. The BI tool will have direct access to all tables and can build reports on the same data as the main application.

---

# 16. Best Practices for Production

1. **Automate Backups**: Ensure the backup script is running daily via `cron` or Windows Task Scheduler.
2. **Separate Environments**: Maintain separate development, staging, and production environments. Test all updates in staging before deploying to production.
3. **Use Version Control**: All changes, including configuration, should be committed to Git. Use feature branches for development and merge to the main production branch (`v8.8` in our case) for deployment.
4. **Monitor Actively**: Use tools like `htop` for real-time monitoring and set up alerts for service failures or high resource usage.
5. **Secure Credentials**: Never commit secrets (like passwords or API keys) directly to Git. Use environment files (`.env`) and add them to `.gitignore`.

This guide provides the foundation for successfully administering the Non-Profit Fund Accounting System. For further technical details, refer to the other documentation files in the repository.