Lifeline Accounting System Software Documentation

(Vue.js and Django-Based with WSGI backend and MSSQL Multi-Company Support and Additional Features)

Table of Contents

---

1. Introduction

The Lifeline Accounting System is a robust, web-based accounting software engineered to streamline financial management for businesses, with comprehensive support for multi-company operations. This system leverages Vue.js for a dynamic, responsive front-end and Django for a secure, scalable back-end, replacing the original HTML/CSS/JavaScript and custom Python/WSGI architecture. It ensures strict data isolation per company, implements role-based access control (RBAC), and adheres to FedRAMP Moderate Compliance standards.

This updated version integrates advanced features such as subscription management, document management, automatic bank feeds, advanced inventory tracking, comprehensive payroll, customization options, real-time collaboration, AI-powered insights, and expanded third-party integrations. Hosted internally at Lifeline Data Centers, the system provides a modern, user-friendly solution accessible via web browsers and mobile devices.

Objectives

- Deliver an intuitive, scalable accounting platform with multi-company functionality.
- Ensure data isolation and RBAC tailored to each company.
- Maintain FedRAMP Moderate compliance through secure coding and auditability.
- Support scalability, security, and compliance with financial standards (e.g., GAAP, SOX).
- Facilitate seamless integration with external systems while preserving company boundaries.
- Provide advanced features to meet diverse business needs.

---

2. Requirements

2.1 Functional Requirements

The system must support the following features, all enhanced with multi-company isolation and RBAC:

Core Features

- General Ledger: Double-entry bookkeeping with company-specific accounts.
- Accounts Receivable (AR): Manage customer invoices and payments per company.
- Accounts Payable (AP): Process vendor bills, payments, and credits, isolated by company.
- Bank Reconciliation: Match bank transactions with ledger entries per company.
- Financial Reporting: Generate balance sheets, income statements, and cash flow reports per company.
- Global Admins: Allow users with system-wide access for management.
- Real-Time Dashboards: Provide company-specific financial analytics.
- Workflow Ease: Simplify tasks like invoicing, bill payments, and check printing.

Additional Features

- Invoicing and Billing: Create customizable invoices with company branding.
- Recurring Transactions: Automate invoices and bills with customizable intervals.
- Bill Payment: Select bills, apply credits, and print checks using laser printers.
- Payroll Management: Compute wages, taxes, and deductions per company.
- Tax Management: Calculate taxes and generate compliance reports per tax regime.
- Inventory Management: Track inventory levels and costs per company.
- Budgeting and Forecasting: Create company-specific budgets and forecasts.
- Multi-Currency Support: Handle transactions with company-specific exchange rates.
- Expense Tracking: Categorize expenses per company.
- Project Management: Monitor project budgets and costs per company.
- Time Tracking: Log billable hours for clients within each company.
- Fixed Asset Management: Track assets and depreciation per company.
- Integration: Connect to external systems with company-specific configurations.
- Security and Compliance: Enforce RBAC and maintain audit logs per company.

- Cloud-Based Access: Provide remote access with company-specific views.
- Mobile Support: Ensure mobile-friendly interfaces for key tasks.
- Automation: Automate reminders and AR aging reports.

Proposed Additional Features

- Subscription Management: Manage recurring revenue with automated billing and renewals.
- Advanced Tax Compliance: Support VAT, GST, and other regimes with compliance reports.
- Document Management: Attach and search documents linked to transactions or entities.
- Automatic Bank Feeds: Import and match bank transactions automatically.
- Advanced Inventory Features: Include multi-location tracking and serial number management.
- Comprehensive Payroll Features: Handle benefits, deductions, and compliance.
- Customization Options: Add custom fields, reports, and workflows.
- Robust Import/Export: Support CSV and Excel for data migration.
- Real-Time Collaboration: Enable simultaneous multi-user access with live updates.
- Enhanced Mobile Support: Achieve full feature parity on mobile devices.
- AI-Powered Insights: Offer suggestions, anomaly detection, and predictive analytics.
- Expanded Third-Party Integrations: Integrate with PayPal, Stripe, Shopify, etc.

Multi-Company and RBAC Requirements

- Create New Company: Global admins can initialize companies with a standard Chart of Accounts.
- Data Segregation: Strictly isolate data by CompanyID.
- RBAC: Define roles (e.g., Company Admin, Accountant, Viewer) per company.
- Company Admin: Require at least one admin per company for user and data management.
- User Access Management: Allow admins to assign roles within their company.

- Multi-Company Users: Support distinct roles across multiple companies for a single user.

2.2 Non-Functional Requirements

- Performance: Process 10,000 transactions/month/company with <2-second response times.
- Scalability: Support hundreds of companies and thousands of users.
- Security: Implement HTTPS, encrypt sensitive data, and enforce RBAC.
- Auditability: Use database triggers to log all actions in the AuditLog table.
- Usability: Provide an intuitive UI with company selection and role-based visibility.
- Reliability: Achieve 99.9% uptime with automated backups.
- Compatibility: Support modern browsers and MSSQL 2019+.
- Maintainability: Use modular, well-documented code.
- User Manual: Include a detailed manual via a "Help" link.

---

3. System Architecture

3.1 Overview

The system adopts a three-tier architecture:

- Presentation Tier: Vue.js single-page application (SPA) for a responsive, interactive UI.
- Application Tier: Django back-end managing business logic, API requests, and RBAC.
- Data Tier: MSSQL database with data isolation via CompanyID.

3.2 Components

- Front-end (Vue.js):
  - Vue components for login, company selection, dashboards, etc.
  - Vue Router for navigation.
  - Vuex for state management (e.g., user authentication, selected company).
  - Axios for API communication with the Django back-end.
- Back-end (Django):

    - Django models mapped to database tables.
    - Django REST Framework (DRF) for RESTful API endpoints.
    - Custom permissions for RBAC enforcement.
    - Middleware for authentication and company context.
- Database (MSSQL):
    - Tables with CompanyID for data segregation.
    - Audit triggers for compliance logging.
- External Integrations:
    - RESTful APIs for bank feeds, payment gateways, and third-party apps.

3.3 Data Flow

- User logs in via the Vue front-end, receiving a JWT token.
- User selects a company from their associated companies.
- Vue sends API requests to Django, including the selected CompanyID in headers.
- Django authenticates the request, verifies the user's role, and filters data by CompanyID.
- MSSQL returns company-specific data, which Django serializes and sends to Vue.

---

4. Database Design

4.1 Schema Overview

The database (LifelineAccounting) includes tables for core accounting functions, multi-company support, and proposed features. Each table relevant to company data includes a CompanyID field for isolation, and audit triggers populate the AuditLog table.

4.2 Key Tables and Relationships

Table	**Purpose**	**Key Fields**	**Relationships**
Companies	Stores company info	CompanyID , CompanyName	Links to Users via

		, AdminUserID	AdminUserID
Users	Manages user accounts	UserID , Username , PasswordHash	
UserCompanyRole	Defines RBAC per company	UserCompanyRoleID , UserID , CompanyID , Role	Links to Users and Companies
ChartOfAccounts	Company-specific accounts	AccountID , CompanyID , AccountCode	Links to Companies
GeneralLedger	Financial transactions	TransactionID , CompanyID , AccountID	Links to ChartOfAccounts , Users
Subscriptions	Manages subscription plans	SubscriptionID	Links to

		, CompanyID , CustomerID	Invoices , Customers
Documents	Stores attached files	DocumentID , CompanyID , FilePath	Links to Invoices , Bills , etc.
InventoryLocations	Multi-location inventory tracking	LocationID , CompanyID	Links to Inventory
PayrollDeductions	Payroll benefits/deductions	DeductionID , PayrollID , EmployeeID	Links to Payroll , Employees
CustomFields	Defines custom fields	CustomFieldID , CompanyID , TableName	Links to CustomFieldValues
Integrations	Third-party integration settings	IntegrationID , CompanyID	Company-specific configurations

		, APIKey	

- Data Isolation: Enforced via CompanyID in all relevant tables.
- Audit Logging: Triggers (e.g., TR_Subscriptions_Audit) log all actions.

---

5. Application Design

5.1 Front-end Design (Vue.js)

The front-end is a Vue.js SPA with a modular structure:

- Project Structure:
    - src/components/: Reusable components (e.g., Login.vue, CompanySelector.vue).
    - src/views/: Pages (e.g., Dashboard.vue, Reports.vue).
    - src/store/: Vuex store for state management.
    - src/router/: Vue Router configuration.
- Key Components:
    - Login.vue: Handles user authentication.
    - CompanySelector.vue: Dropdown for company selection.
    - Dashboard.vue: Displays company-specific analytics.
- Routing: Vue Router manages navigation with guards for authentication.
- State Management: Vuex stores user data and selected company.
- API Integration: Axios sends requests with X-Company-ID headers.

Sample Code:

vue

```
<!-- CompanySelector.vue -->

<template>

  <select v-model="selectedCompany" @change="updateCompany">

    <option v-for="company in companies" :value="company.id" :key="company.id">
```

```
      {{ company.name }}
    </option>
  </select>
</template>

<script>
import axios from 'axios';

export default {
  computed: {
    companies() {
      return this.$store.state.user.companies;
    },
    selectedCompany: {
      get() {
        return this.$store.state.selectedCompany.id;
      },
      set(value) {
        this.$store.commit('setSelectedCompany', value);
      },
    },
  },
  methods: {
    updateCompany() {
      axios.defaults.headers.common['X-Company-ID'] = this.selectedCompany;
      this.$store.dispatch('loadCompanyData');
    },
  },
};
</script>
```

5.2 Back-end Design (Django)

The back-end uses Django with a modular app structure:

- Project Structure:
    - apps/accounts/: Models and views for accounts, ledger, etc.
    - apps/subscriptions/: Subscription management logic.
    - apps/documents/: Document handling.
- Models: Map to database tables (e.g., Company, Subscription).
- Serializers: Convert models to JSON for API responses.
- Views: DRF viewsets or API views for CRUD operations.
- Permissions: Custom classes enforce RBAC per company.
- Middleware: Handles JWT authentication and company context.

Sample Code:

python

```
# apps/subscriptions/models.py
from django.db import models


class Subscription(models.Model):
    subscription_id = models.AutoField(primary_key=True)
    company = models.ForeignKey('accounts.Company', on_delete=models.CASCADE)
    customer = models.ForeignKey('accounts.Customer', on_delete=models.CASCADE)
    plan_name = models.CharField(max_length=100)
    billing_cycle = models.CharField(max_length=50)
    renewal_date = models.DateTimeField()
    status = models.CharField(max_length=20, default='Active')
    created_date = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'Subscriptions'

# apps/core/permissions.py
from rest_framework.permissions import BasePermission

class HasCompanyRole(BasePermission):
    def has_permission(self, request, view):
        company_id = request.headers.get('X-Company-ID')
        user = request.user
        role = UserCompanyRole.objects.filter(user=user, company_id=company_id).first()
        return role and role.role in getattr(view, 'allowed_roles', [])

# apps/subscriptions/views.py
from rest_framework import viewsets
from .models import Subscription
from .serializers import SubscriptionSerializer
from apps.core.permissions import HasCompanyRole

class SubscriptionViewSet(viewsets.ModelViewSet):
    serializer_class = SubscriptionSerializer
    permission_classes = [HasCompanyRole]
    allowed_roles = ['Admin', 'Accountant']


    def get_queryset(self):
        company_id = self.request.headers.get('X-Company-ID')
        return Subscription.objects.filter(company_id=company_id)
```

5.3 Integration with MSSQL

- Database Backend: Use django-mssql-backend for connectivity.
- Configuration:

```
python
# settings.py
DATABASES = {
    'default': {
        'ENGINE': 'mssql',
        'NAME': 'LifelineAccounting',
        'USER': 'username',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '1433',
        'OPTIONS': {'driver': 'ODBC Driver 17 for SQL Server'},
    }
}
```

- Audit Triggers: Pre-existing triggers (e.g., TR_Subscriptions_Audit) ensure compliance.

---

6. Technical Details

6.1 Technology Stack

Component	**Technology**
Front-end	Vue.js 3, Vue Router, Vuex, Axios, Bootstrap
Back-end	Django 4, Django REST Framework, djangorestframework-simplejwt
Database	MSSQL 2019+
Authentication	JWT
Security	HTTPS, RBAC, encryption
Integrations	RESTful APIs

6.2 Development Environment Setup

- Front-end:
    - Install Node.js and Vue CLI: npm install -g @vue/cli.
    - Create project: vue create lifeline-frontend.
    - Install dependencies: npm install axios vuex vue-router bootstrap.
- Back-end:
    - Create virtual environment: python -m venv env.
    - Activate and install: pip install django djangorestframework djangorestframework-simplejwt django-mssql-backend.
    - Start project: django-admin startproject lifeline_backend.
    - Configure apps and database in settings.py.

6.3 Security Implementation

- Authentication: JWT via djangorestframework-simplejwt.
- RBAC: Custom permissions check roles per company.
- Encryption: Use Django's encrypt for sensitive fields.
- Audit Logging: Database triggers log all actions.
- Protections: Django's built-in CSRF, XSS, and SQL injection defenses.

6.4 API Endpoints

Endpoint	Method	Description	Headers
/api/auth/login/	POST	Obtain JWT token	
/api/companies/	GET	List user's companies	Authorization
/api/subscriptions/	POST	Create subscription	Authorization, X-Company-ID
/api/documents/	POST	Upload document	Authorization, X-Company-ID
/api/bank/feeds/	GET	Import bank transactions	Authorization, X-Company-ID
/api/insights/	GET	Retrieve AI insights	Authorization, X-Company-ID

---

7. Deployment

- Front-end: Build with npm run build, serve via Nginx.
- Back-end: Deploy with Gunicorn behind Nginx reverse proxy.
- Database: Use a managed MSSQL instance internally at Lifeline Data Centers.
- Security: Enforce HTTPS and FedRAMP Moderate compliance.

---

8. Development Guidelines

8.1 Coding Standards

- Python: Adhere to PEP 8.
- Vue.js: Follow Vue Style Guide.
- Comments: Include docstrings and inline explanations.

8.2 Testing Strategy

- Unit Tests: Cover Django models/views and Vue components.
- Integration Tests: Verify end-to-end workflows.
- Security Tests: Ensure no cross-company data leaks.

8.3 Version Control

- Use Git with feature branches (e.g., feature/subscription-management).

- Require code reviews before merging.

---

9. Future Enhancements

- Real-Time Collaboration: Implement WebSockets with Django Channels.
- AI Insights: Integrate machine learning models (e.g., scikit-learn).
- Native Mobile App: Develop with offline support.

---

10. References

- Vue.js Documentation
- Django Documentation
- Django REST Framework
- MSSQL Documentation
- FedRAMP Moderate Guidelines

This documentation provides a detailed blueprint for developing the Lifeline Accounting System with Vue.js and Django, ensuring all features, security requirements, and development tasks are comprehensively addressed.