Laitusneo Track - Technical Documentation

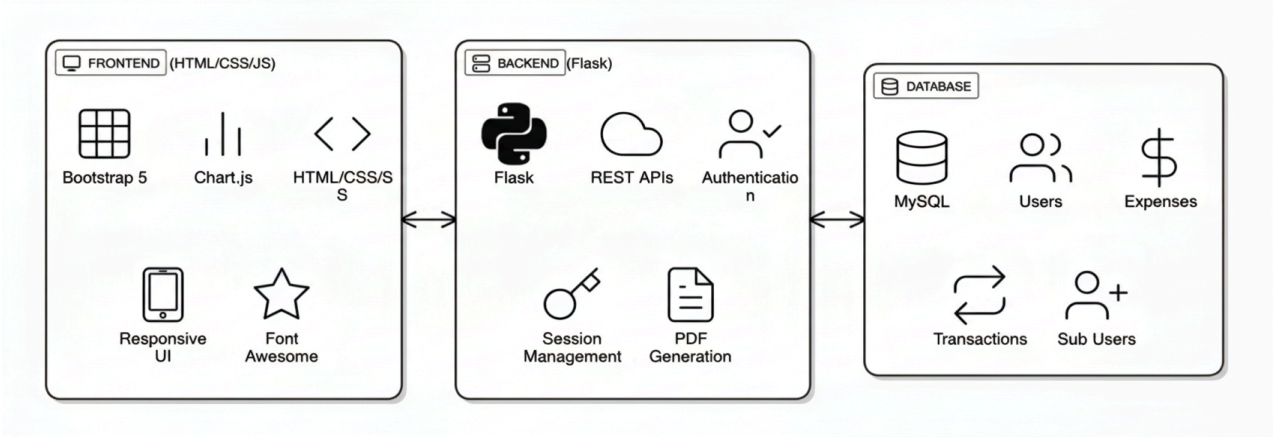
This technical documentation provides a comprehensive overview of the Laitusneo Track

🏗️ System Architecture

Overview

Laitusneo Track is a Flask-based web application with a MySQL database backend, implementing a multi-tier architecture with role-based access control.

Architecture Diagram



Database Design

Entity Relationship Diagram

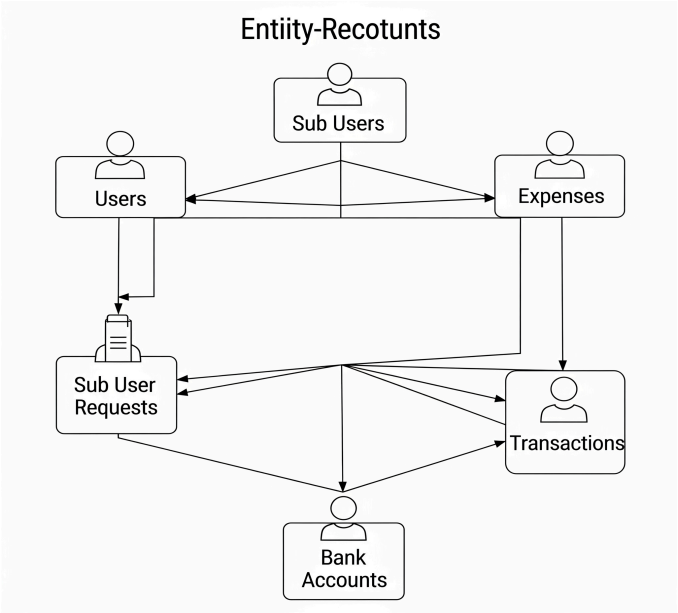


Table Relationships

Primary Tables

1. **users** - Main user accounts
2. **sub_users** - Sub-user accounts (linked to main users)
3. **expenses** - Expense records
4. **transactions** - Transaction records
5. **sub_user_requests** - Sub-user approval requests
6. **bank_accounts** - Bank account information

Key Relationships

- `users.id` → `sub_users.created_by` (One-to-Many)
- `users.id` → `expenses.user_id` (One-to-Many)
- `users.id` → `transactions.user_id` (One-to-Many)
- `sub_users.id` → `sub_user_requests.sub_user_id` (One-to-Many)

🔧 Core Components

1. Authentication System

Password Security (Password hashing using Werkzeug)

```
from werkzeug.security import generate_password_hash, check_password_hash

def hash_password(password):
    return generate_password_hash(password)
def verify_password(password_hash, password):
    return check_password_hash(password_hash, password)
```

Session Management (Flask session configuration)

```
app.secret_key = 'your-secret-key-here'

session['user_id'] = user_id

session['user_type'] = 'main_user'  # or 'sub_user', 'admin'
```

Role-Based Access Control

```
from functools import wraps
from flask import redirect, url_for, session
def require_auth(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_id' not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function
```

2. Unique ID Generation

Algorithm

```
from datetime import datetime
import random
def generate_unique_id(prefix):
    """Generate unique ID with prefix and timestamp"""
    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")
    random_suffix = random.randint(1000, 9999)
    return f"{prefix}{timestamp}{random_suffix}"
```

ID Types

- **EXP**: Expense records
- **TXN**: Transaction records
- **INV**: Invoice records
- **SUB**: Sub-user records

3. Request Approval Workflow

Sub-User Request Flow

Sub-User Creates Request → Pending Status → Main User Reviews → Approved/Rejected → Creates Main User Record → Updates Balance

Implementation

```
def approve_sub_user_expense_request(expense_id):
    # 1. Get request data
    # 2. Create expense record for main user
    # 3. Create transaction record
    # 4. Update account balance
    # 5. Mark request as approved
    pass # Placeholder for implementation
```

4. PDF Generation System

ReportLab Integration

```
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle
from reportlab.lib import colors # Added for colors
def generate_pdf_report(data, filename):
    doc = SimpleDocTemplate(filename, pagesize=letter)
    elements = []
    # Create table with data
    table = Table(data)
    table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 14),
        ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
        ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
        ('GRID', (0, 0), (-1, -1), 1, colors.black)

    ]))
    elements.append(table)
    doc.build(elements)
```

🔌 API Architecture

RESTful Endpoints

Authentication Endpoints

- `POST /login` - User authentication
- `POST /register` - User registration
- `POST /sub-user-login` - Sub-user authentication
- `POST /admin-login` - Admin authentication
- `GET /logout` - Session termination

Data Management Endpoints

- `GET /api/expenses` - List expenses
- `POST /api/expenses` - Create expense
- `DELETE /api/expenses/<id>` - Delete expense
- `GET /api/transactions` - List transactions
- `POST /api/transactions` - Create transaction
- `DELETE /api/transactions/<id>` - Delete transaction

Sub-User Management Endpoints

- `GET /api/sub-user/expense-requests` - List expense requests
- `POST /api/sub-user/expense-requests` - Create expense request
- `POST /api/sub-user/expense-requests/<id>/approve` - Approve request
- `GET /api/sub-user/approved-expenses` - List approved expenses

Response Format

```
{
  "success": true,
  "message": "Operation completed successfully",
  "data": {
    // Response data
  }
}
```

Error Handling

```
{
  "success": false,
  "message": "Error description",
  "error_code": "ERROR_CODE"
}
```

🎨 Frontend Architecture

Component Structure

Base Templates

- `base.html` - Main user base template
- `sub_user_base.html` - Sub-user base template

Page Templates

- `dashboard.html` - Main dashboard
- `expenses.html` - Expense management
- `transactions.html` - Transaction management
- `invoices.html` - Invoice management
- `sub_user_*.html` - Sub-user specific pages

JavaScript Architecture

Global Functions

```
// Notification system
function showNotification(message, type, duration) { /* Implementation */ }

// Data loading
function loadExpenses() { /* Implementation */ }
function loadTransactions() { /* Implementation */ }

// Form handling
function handleExpenseSubmit(event) { /* Implementation */ }
function handleTransactionSubmit(event) { /* Implementation */ }
```

AJAX Implementation

```
// Fetch data from API
async function fetchData(url) {
    try {
        const response = await fetch(url);
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error:', error);
        showNotification('Error loading data', 'error');
    }
}
```

CSS Architecture

Design System

```
/* Color Palette */
:root {
    --primary-color: #667eea;
    --secondary-color: #764ba2;
    --success-color: #48bb78;
    --warning-color: #ed8936;
    --danger-color: #dc3545;
    --info-color: #4299e1;
}
/* Typography */
.font-inter { font-family: 'Inter', sans-serif; }
.font-weight-300 { font-weight: 300; }
.font-weight-400 { font-weight: 400; }
.font-weight-500 { font-weight: 500; }
.font-weight-600 { font-weight: 600; }
.font-weight-700 { font-weight: 700; }
```

Component Classes

```
/* Cards */

.professional-card { /* Card styling */ }

.metric-card { /* Dashboard metric cards */ }

/* Forms */

.form-control { /* Input styling */ }

.btn-auth { /* Authentication buttons */ }

/* Tables */

.detail-table { /* Data table styling */ }
```

architecture, implementation details, and operational procedures.

🔒 Security Implementation

Input Validation

Server-Side Validation

```
def validate_expense_data(data):
    required_fields = ['title', 'amount', 'category', 'date']
    for field in required_fields:
        if not data.get(field):
            raise ValueError(f"{field} is required")
                # Validate amount
    try:
        amount = float(data['amount'])
        if amount <= 0:
            raise ValueError("Amount must be positive")
    except ValueError:
        raise ValueError("Invalid amount format")
```

Client-Side Validation

```
function validateForm(formData) {
   const errors = [];
   if (!formData.title.trim()) {
      errors.push('Title is required');
            }
    if (!formData.amount || formData.amount <= 0) {
      errors.push('Amount must be positive');
            }
   return errors;
}
```

SQL Injection Prevention

Using parameterized queries

```
cursor.execute(
   "SELECT * FROM expenses WHERE user_id = %s AND date >= %s",
   (user_id, start_date)
)
```

File Upload Security

```
def allowed_file(filename):
   return '.' in filename and \
        filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
def secure_filename(filename):
   return werkzeug.utils.secure_filename(filename)
```

📊 Performance Optimization

Database Optimization

Indexing Strategy

```
-- Primary indexes
CREATE INDEX idx_user_id ON expenses(user_id);
CREATE INDEX idx_date ON expenses(date);
CREATE INDEX idx_sub_user_id ON sub_user_requests(sub_user_id);
CREATE INDEX idx_status ON sub_user_requests(status);
-- Composite indexes
CREATE INDEX idx_user_date ON expenses(user_id, date);
CREATE INDEX idx_sub_user_type ON sub_user_requests(sub_user_id, request_type);
```

Query Optimization

```
def get_expenses_paginated(user_id, page=1, per_page=20):
    offset = (page - 1) * per_page
    query = """
        SELECT * FROM expenses
        WHERE user_id = %s
        ORDER BY date DESC
        LIMIT %s OFFSET %s
    """
    return cursor.execute(query, (user_id, per_page, offset))
```

Frontend Optimization

Lazy Loading

```
// Load data on demand
function loadMoreData(page) {
    fetch(`/api/expenses?page=${page}`)
        .then(response => response.json())
        .then(data => appendToTable(data));
}
```

Caching Strategy

```
// Cache frequently accessed data
const cache = new Map();
function getCachedData(key) {
    if (cache.has(key)) {
        return cache.get(key);
            }
    return null;
}
```

🧪 Testing Strategy

Unit Testing

```
import unittest
from app import app, get_db_connection
class TestExpenseManagement(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True
    def test_create_expense(self):
        response = self.app.post('/api/expenses', json={
            'title': 'Test Expense',
            'amount': 100.00,
            'category': 'Office Supplies'
        })
        self.assertEqual(response.status_code, 200)
```

Integration Testing

```
def test_expense_approval_workflow():
        # 1. Create sub-user request
        # 2. Approve request
        # 3. Verify main user record created
        # 4. Verify balance updated
```

📈 Monitoring & Logging

Application Logging

```
import logging
```

Configure logging

```
logging.basicConfig(
  level=logging.INFO,
  format='%(asctime)s %(levelname)s %(message)s',
  handlers=[
    logging.FileHandler('app.log'),
    logging.StreamHandler()
        ]
)
```

Usage

```
logging.info(f"User {user_id} created expense {expense_id}")
logging.error(f"Database connection failed: {error}")
```

Performance Monitoring

```
import time
from functools import wraps
def monitor_performance(f):
  @wraps(f)
  def decorated_function(*args, **kwargs):
    start_time = time.time()
```

```
        result = f(*args, **kwargs)
        end_time = time.time()
        logging.info(f"{f.__name__} executed in {end_time - start_time:.2f}s")
        return result
    return decorated_function
```

🚀 Deployment Architecture

Docker Configuration

```
FROM python:3.9-slim

WORKDIR /app
```

Install system dependencies

```
RUN apt-get update && apt-get install -y \
    gcc \
    default-libmysqlclient-dev \
    pkg-config \
    && rm -rf /var/lib/apt/lists/*
```

Install Python dependencies

```
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
Copy application
COPY . .
```

Create directories

```
RUN mkdir -p uploads exports uploads/templates
```

Expose port

```
EXPOSE 5000
```

Run application

```
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
```

Environment Configuration

docker-compose.yml

```
version: '3.8'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=production
      - DB_HOST=db
    depends_on:
      - db
  db:
    image: mysql:8.0
    environment:
      - MYSQL_ROOT_PASSWORD=rootpassword
      - MYSQL_DATABASE=expense_tracker
    volumes:
      - mysql_data:/var/lib/mysql
volumes:
  mysql_data:
```

🔧 Configuration Management

Environment Variables

```
import os

from dotenv import load_dotenv

load_dotenv()

class Config:

    SECRET_KEY = os.environ.get('SECRET_KEY') or 'dev-secret-key'

    DB_HOST = os.environ.get('DB_HOST') or 'localhost'

    DB_USER = os.environ.get('DB_USER') or 'root'

    DB_PASSWORD = os.environ.get('DB_PASSWORD') or ''

    DB_NAME = os.environ.get('DB_NAME') or 'expense_tracker'

            # File upload settings

    MAX_CONTENT_LENGTH = 16 * 1024 * 1024  # 16MB

    UPLOAD_FOLDER = os.environ.get('UPLOAD_FOLDER') or 'uploads'
```

Feature Flags

```
class FeatureFlags:

    ENABLE_PDF_GENERATION = True

    ENABLE_EMAIL_NOTIFICATIONS = False

    ENABLE_ADVANCED_ANALYTICS = True

    ENABLE_API_RATE_LIMITING = True
```

📚 API Documentation

Authentication

All API endpoints require authentication via session cookies.

Rate Limiting

from flask_limiter import Limiter

from flask_limiter.util import get_remote_address

limiter = Limiter(

app,

key_func=get_remote_address,

default_limits=["200 per day", "50 per hour"]

)

@app.route('/api/expenses')

@limiter.limit("10 per minute")

def get_expenses():

 # Endpoint implementation

Error Codes

- `400` - Bad Request

- `401` - Unauthorized

- `403` - Forbidden

- `404` - Not Found

- `500` - Internal Server Error

🔄 Backup & Recovery

Database Backup

Daily backup script

mysqldump -u root -p expense_tracker > backup_$(date +%Y%m%d).sql

File Backup

Backup uploads and exports

tar -czf files_backup_$(date +%Y%m%d).tar.gz uploads/ exports/

Recovery Procedures

1. Restore database from backup

2. Restore file uploads

3. Verify application functionality

4. Update DNS if necessary

This technical documentation provides a comprehensive overview of the Laitusneo Track system architecture, implementation details, and operational procedures.