Music Store System Installation and Configuration Guide, v2 for Linux

This document should provide a comprehensive guide for setting up the Music Store System on a Linux platform, including detailed steps for installation, configuration, and database setup. The guide is designed for IT students withbasic knowledge of Flask, Python, HTML, and databases.

Index:

1. Introduction

The Music Store System is a web application built using Flask, a web framework for Python. This guide will walk you through the steps to install and configure the application on a Linux environment.

2. Prerequisites

Before you continue, please, ensure you have the following installed on your system:

Python 3.8 or newer

PostgreSQL

Git for cloning the repository

3. Installing Python and Flask

Install Python:

Open a terminal.

Update the package list:

sudo apt update

Install Python:

sudo apt install python3 python3-venv python3-pip

Install Flask:

Create a virtual environment:

mkdir myproject

cd myproject

python3 -m venv venv

Activate the virtual environment:

source venv/bin/activate

Install Flask:

pip install Flask

4. Setting Up the Music Store Application

Clone the repository:

If you have Git installed, you can clone the repository:

git clone <repository_url>

cd music_store

Install Dependencies:

Ensure your virtual environment is activated and install the required packages:

pip install -r requirements.txt

5. Configuring PostgreSQL Database

Install PostgreSQL:

Download and install PostgreSQL:

sudo apt install postgresql postgresql-contrib

Create a Database:

Switch to the PostgreSQL user:

sudo -i -u postgres

Open the PostgreSQL prompt:

psql

Create a new database:

CREATE DATABASE MStore_v1;

Create the schema:

CREATE SCHEMA mstore_v1;

6. Creating and Populating the Database

Creating Tables:

Within the mstore_v1 schema, create tables for your application:

```
CREATE TABLE mstore_v1.users (
   id SERIAL PRIMARY KEY,
   username VARCHAR(50) NOT NULL,
   email VARCHAR(100) NOT NULL,
   password VARCHAR(100) NOT NULL
);
```

```
CREATE TABLE mstore_v1.products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    stock INT NOT NULL
);
```

AI-generated code. Review and use carefully. More info on FAQ.

Populating Tables:

Insert initial data into the tables: (This is just an EXAMPLE, not the MStore_v1 db)

```
INSERT INTO mstore_v1.users (username, email, password) VALUES
('john_doe', 'john@example.com', 'securepassword'),
('jane_doe', 'jane@example.com', 'anothersecurepassword');
INSERT INTO mstore_v1.products (name, price, stock) VALUES
('Guitar', 199.99, 10),
('Piano', 499.99, 5);
```

See db-documents in my MusicStore repo.

7. Running the Application

Starting the Flask Application:

Ensure your virtual environment is activated.

Run the application:

```
flask run
```

8. Directory Structure

An overview of the directory structure of the project.

See the GIT MusicStore repo structure.

9. User Interface Overview

Provide an overview of the user interface and its features.

TBD later on...

10. Using .env Files

Benefits of Using .env Files:

Security: .env files help keep sensitive information like database credentials, API keys, and other configuration details out of your source code

Environment-Specific Configurations: They allow you to easily switch between different configurations for development, testing, and production environments without changing your code.

Simplified Configuration Management: By centralizing configuration settings in a single file, .env files make it easier to manage and update configurations.

How to Use .env Files:

Create a .env File: In the root directory of your project, create a file named .env.

Add Environment Variables: Add your environment-specific variables in the format KEY=VALUE.

For example: (look also .env file in repo MusicStore)

FLASK_APP=app.py

FLASK_ENV=development

DATABASE_URL=postgresql://user:password@localhost/MStore_v1

Load Environment Variables: Use a library like python-dotenv to load these variables into your application. Install it using:

pip install python-dotenv

Then, in your Python application, load the variables:

from dotenv import load_dotenv

import os

```
load_dotenv()

database_url = os.getenv('DATABASE_URL')
```