Developer Documentation: Offline Inventory Tracker

Overview

The **Offline Inventory Tracker** is a lightweight, standalone tool for managing inventory in offline environments. Built with Python, it features a graphical interface (Tkinter) and uses SQLite for local data storage.

Project Structure

```
bash
CopyEdit
OfflineInventoryTracker/
├── main.py          # Main application logic
├── inventory.db     # SQLite database (auto-generated)
├── README.md        # Documentation
├── requirements.txt# Dependencies (if needed)
└── dist/            # Generated executables (via PyInstaller)
```

Tech Stack

- **Programming Language**: Python
- **GUI Framework**: Tkinter
- **Database**: SQLite
- **Packaging Tool**: PyInstaller

Key Features

1. **Inventory Management**:
    - Add items with name, quantity, price, and unit of measure.
    - Update existing items.
    - Delete items.
2. **Data Export**:
    - Export inventory data to a CSV file for external use.
3. **Offline Compatibility**:
    - All data is stored locally in an SQLite database.

Setup

Dependencies

If using the source code, ensure the following:

- Python 3.8 or higher
- Required libraries (Tkinter and SQLite are bundled with Python):

  ```
  bash
  CopyEdit
  pip install tkinter sqlite3
  ```

Running Locally

- Run the script:

  ```
  bash
  CopyEdit
  python main.py
  ```

Creating an Executable

To create a standalone executable:

1. Install PyInstaller:

   ```
   bash
   CopyEdit
   pip install pyinstaller
   ```

2. Generate the executable:

   ```
   bash
   CopyEdit
   pyinstaller --onefile --windowed --name InventoryTracker main.py
   ```

3. The executable will appear in the `dist/` folder.

Database Schema

The database file (`inventory.db`) includes one table:

```
sql
CopyEdit
CREATE TABLE IF NOT EXISTS inventory (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    item_name TEXT NOT NULL UNIQUE,
    quantity INTEGER NOT NULL,
    threshold INTEGER NOT NULL,
    price REAL NOT NULL,
    unit TEXT NOT NULL
```

```
);
```

- **id**: Unique identifier for each item.
- **item_name**: Name of the inventory item.
- **quantity**: Current stock level.
- **threshold**: Minimum stock level before a restock is needed.
- **price**: Price per unit.
- **unit**: Unit of measurement (e.g., kg, pcs).

Core Functions

1. **Database Initialization**:
   - `initialize_db`: Creates the database and inventory table if they don't exist.

```
python
CopyEdit
def initialize_db():
    connection = sqlite3.connect("inventory.db")
    cursor = connection.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS inventory (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            item_name TEXT NOT NULL UNIQUE,
            quantity INTEGER NOT NULL,
            threshold INTEGER NOT NULL,
            price REAL NOT NULL,
            unit TEXT NOT NULL
        )
    """)
    connection.commit()
    connection.close()
```

2. **Add Item**:
   - Adds a new inventory item to the database.

```
python
CopyEdit
def add_item(name, quantity, threshold, price, unit):
    connection = sqlite3.connect("inventory.db")
    cursor = connection.cursor()
    cursor.execute("""
        INSERT INTO inventory (item_name, quantity, threshold, price, 
unit)
        VALUES (?, ?, ?, ?, ?)
    """, (name, quantity, threshold, price, unit))
    connection.commit()
    connection.close()
```

3. **Update Item**:
   - Updates existing item details based on the `item_name`.

```
python
CopyEdit
def update_item(name, quantity, threshold, price, unit):
    connection = sqlite3.connect("inventory.db")
    cursor = connection.cursor()
    cursor.execute("""
        UPDATE inventory
        SET quantity = ?, threshold = ?, price = ?, unit = ?
        WHERE item_name = ?
    """, (quantity, threshold, price, unit, name))
    connection.commit()
    connection.close()
```

4. **Fetch Inventory**:
   - Retrieves all inventory records.

```
python
CopyEdit
def fetch_inventory():
    connection = sqlite3.connect("inventory.db")
    cursor = connection.cursor()
    cursor.execute("SELECT * FROM inventory")
    rows = cursor.fetchall()
    connection.close()
    return rows
```

5. **Export to CSV**:
   - Exports inventory data to a CSV file.

```
python
CopyEdit
def export_to_csv():
    inventory = fetch_inventory()
    with open("inventory.csv", "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(["ID", "Item Name", "Quantity", "Threshold", 
"Price", "Unit"])
        writer.writerows(inventory)
```

Testing

1. **Unit Testing**:
   - Test all core database functions (`add_item`, `update_item`, `fetch_inventory`, etc.) independently.
   - Ensure items are added, updated, and fetched correctly.
2. **GUI Testing**:
   - Verify all buttons and input fields work as expected.
   - Test edge cases (e.g., adding an item with a duplicate name or leaving fields blank).
3. **Error Handling**:

    - Validate user input to prevent database errors (e.g., non-numeric values in numeric fields).

Contributing

1. Fork the repository.
2. Create a new branch for your changes:

   ```
   bash
   CopyEdit
   git checkout -b feature-name
   ```

3. Commit your changes and submit a pull request.

Future Improvements

1. **Low Stock Alerts**:
   - Highlight items below their threshold in the GUI.
2. **Analytics**:
   - Add simple charts for stock trends.
3. **Multi-User Support**:
   - Enable multiple users to access the database.