DB Pilot – Django Modular Project

DB Pilot is a lightweight, modular **Django** application demonstrating complete **CRUD** operations using **SQLite**. It is designed as a **step-by-step learning project**, showing how to build scalable Django apps with a clean, modular architecture.

The repository includes a **sample SQLite database (`db.sqlite3`)** with preloaded tables and test data.

Available login credentials:

- **User account:** `user / demo`
- **Admin account:** `admin / root`

You can **check the running project online** at: https://padiks.pythonanywhere.com ✅

This live demo lets you **explore all modules**, test CRUD operations, and see role-based access in action without needing to set up anything locally.

🧩 Project Structure (Modular Django)

```
project_folder/
├── manage.py
│
├── core/                         # Core Django settings & URL configurat
│   ├── settings.py
│   └── urls.py
│
├── apps/                         # Modular Django apps
│   ├── items/                    # Items app (CRUD + ForeignKey examples
│   │   ├── apps.py
│   │   ├── models.py
│   │   ├── urls.py
│   │   ├── views.py
│   │   ├── forms.py
│   │   └── templates/items/
│   │       ├── index.html
│   │       └── form.html
│   │
│   ├── <other-modules>/          # Placeholder for additional apps/modul
│   │
│   └── users/                    # Authentication & role-based access
│       ├── apps.py
│       ├── urls.py
│       ├── views.py
│       ├── templatetags/group_filters.py  # Custom template filters for user grou
│       └── templates/users/login.html
│
```

```
├── templates/                          # Project-wide templates
│   ├── base.html                       # Main layout for all pages
│   ├── admin/base_site.html            # Custom Django admin layout
│   └── includes/                       # Reusable partial templates
│       ├── _table_select.html
│       └── footer.html
│
├── static/                             # Static files (CSS, images)
│   ├── css/style.css
│   └── img/favicon.png
│
└── db.sqlite3                          # SQLite database
```

> *Some modules (items, users) are shown. Additional modules exist in earlier guide folders.*

📘 Included Learning Guides

The project includes **14 modular guides**, each a self-contained working project:

Guide	Description
01 — Base Template	Bootstrap layout, global includes.
02 — SQLite Database	Database config + first tables.
03 — Full CRUD (ORM)	Create, Read, Update, Delete.
04 — Multi-Table Includes	Rendering multiple tables modularly.
05 — Foreign Keys	Items linked to UOM (relationships).
06 — Authentication	Login & logout using Django Auth.
07 — User Management Admin Customization	Manage users and superusers, use Groups for roles, customize admin panel, control table visibility per role.
08 — Role-Based Admin	Admin panel with advanced role-based access.
09 — Role-Based CRUD Module	CRUD operations restricted by user roles.
10 — DataTables Integration	Dynamic tables with search, sort, and pagination.
11 — Stock Movements Module	Manages and tracks stock transactions, including the movement of items and supporting stock-related entries.
12 — Compute Module & Markdown Renderer	Handles basic calculations (e.g., summation) and renders static markdown content into HTML.
13 — Django Rest Framework (DRF)	Setting up a based API to manage data such as books.
14 — Rest API Consumer	Fetch and display data from an external REST API in Django.

Guide	Description
15 — Display Excel Data	Read and display Excel data in a table format.

Each guide is a **fully working project** and **continues from the previous guide**. This means **every guide folder already includes all features, files, and improvements from the earlier guides**, so you can download **any guide** and run it instantly.

🚀 How to Run

Minimal requirements (already tested on **Windows** and **Debian**):

```
asgiref==3.10.0
certifi==2025.11.12
charset-normalizer==3.4.4
Django==5.2.8
django-cors-headers==4.9.0
django-debug-toolbar==6.1.0
djangorestframework==3.16.1
djangorestframework_simplejwt==5.5.1
et_xmlfile==2.0.0
idna==3.11
Markdown==3.10
numpy==2.3.5
openpyxl==3.1.5
pandas==2.3.3
PyJWT==2.10.1
python-dateutil==2.9.0.post0
pytz==2025.2
PyYAML==6.0.3
requests==2.32.5
six==1.17.0
sqlparse==0.5.3
tzdata==2025.2
urllib3==2.5.0
```

Start the Server

Install dependencies and run the project:

```
pip install -r requirements.txt
python manage.py runserver
```

✔ **No migrations required** — the included `db.sqlite3` already contains the necessary data and schema.

✔ **Migrations might be required in the future** — If you modify models or the schema, run the following commands to apply changes:

```
python manage.py makemigrations
python manage.py migrate
```

Running the Project from GitHub

You can also download the project directly from GitHub and run it easily:

```
# Clone the repository
git clone https://github.com/padiks/django-modular-project.git

# Go into the project folder
cd django-modular-project
cd 14-rest-api-consumer

# Create a virtual environment
python3 -m venv venv
source venv/bin/activate       # Linux / macOS
# For Windows PowerShell use: venv\Scripts\Activate.ps1
# For Windows CMD use: venv\Scripts\activate.bat

# Upgrade pip and install required packages
pip install --upgrade pip
pip install Django django-debug-toolbar markdown djangorestframework djangorestfra

# Run the development server
python manage.py runserver
```

You should see output similar to:

```
System check identified no issues (0 silenced).
Django version 5.2.8, using settings 'core.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

🏗️ Features

- Modular Django Architecture (copy → rename → new app)
- Bootstrap UI with reusable includes
- Full CRUD using Django ORM
- Clean foreign key examples (Items → UOM)
- Login/Logout using Django Auth
- Preloaded sample data
- Debug toolbar **already integrated** (can be easily removed if needed)
- Easy to extend and scale with new modules

📄 License

This project is for **learning and educational use**. Feel free to explore, extend, and build upon it.

Django App Setup: `dbpilot` with `uom` and `base.html`

Project Structure

```
project_folder/
├── manage.py                   # Django management entrypoint
│
├── core/
│   ├── settings.py             # Centralized settings (DB, paths, debug, apps)
│   └── urls.py                 # Root URL router, includes app URLs
│
├── apps/
│   └── uom/
│       ├── urls.py             # App-specific routes
│       ├── views.py            # Views / controllers
│       ├── models.py           # Database models for CRUD
│       ├── forms.py            # Forms for CRUD (optional)
│       └── templates/
│           └── uom/
│               └── index.html # App-level template
│
├── templates/
│   └── base.html               # Global base layout
│
└── static/
    ├── css/
    │   └── style.css           # Global CSS
    └── img/
        └── favicon.png          # Global images
```

Install and setup Django, debug toolbar, folders, files

```
 $ cd /home/user/Public/web
$ mkdir <project-folder>
$ cd <project-folder>
$ python3 -m venv venv
$ source venv/bin/activate
(venv) $ pip install --upgrade pip
(venv) $ pip install Django django-debug-toolbar
(venv) $ django-admin startproject core .
(venv) $ mkdir apps
```

```
(venv) $ python manage.py startapp uom
(venv) $ mv uom apps
(venv) $ mkdir templates static
(venv) $ touch templates/base.html
(venv) $ mkdir static/css
(venv) $ touch static/css/style.css
(venv) $ mkdir apps/uom/templates
(venv) $ mkdir apps/uom/templates/uom
(venv) $ touch apps/uom/templates/uom/index.html
```

1. `core/urls.py` — Root URL Routing

```
# core/urls.py
from django.contrib import admin
from django.urls import path, include
from django.conf import settings  # For DEBUG

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('apps.uom.urls')),  # Home app
]

# Debug Toolbar URLs only if DEBUG=True
if settings.DEBUG:
    import debug_toolbar
    urlpatterns = [path('__debug__/', include(debug_toolbar.urls))] + urlpatterns
```

Note:

- `uom` is currently set as the **temporary home URL**.
- The **Django Debug Toolbar** is added for debugging. If it is not installed or configured, you can refer to the setup in the **previous guide/project** for instructions.

---

2. `apps/uom/urls.py` — App-specific Routes

```
from django.urls import path
from . import views

app_name = 'uom'

urlpatterns = [
    path('', views.index, name='index'),  # Home page for the app
]
```

---

3. `apps/uom/views.py` — Views / Controllers

```
from django.shortcuts import render

# Home page view
def index(request):
    context = {
        'title': 'UOM Home',  # Example context variable
        'welcome_message': 'Welcome to the UOM App!',
    }
    return render(request, 'uom/index.html', context)
```

---

4. `apps/uom/apps.py`

```
from django.apps import AppConfig

class UomConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'apps.uom'   # <-- must match with `core/settings.py`
```

---

5. Using `base.html` in `uom/index.html`

In your `uom/templates/uom/index.html`:

```
{% extends 'base.html' %}

{% block title %}{{title}} - Django & SQLite{% endblock %}

{% block content %}
        <div class="card shadow-lg p-4 mb-3">
    <h4 class="mb-4 text-center">Database db.sqlite3</h4>
    <div class="mb-3 d-flex flex-wrap align-items-center gap-2">
      <label for="tableSelect" class="form-label mb-0">Table:</label>
      <select class="form-select form-select-sm" id="tableSelect" style="width: au
        <option value="stock_items_uom">stock_items_uom</option>
      </select>
    </div>
  </div>
<div class="card shadow-lg p-4 mb-3">
<pre><code>CREATE TABLE "stock_items_uom" (
  "id"  INTEGER,
  "name"        TEXT NOT NULL UNIQUE,
  "description" TEXT,
  "status"      INTEGER DEFAULT 1,
  "created_at"  DATETIME DEFAULT CURRENT_TIMESTAMP,
  "updated_at"  DATETIME DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY("id" AUTOINCREMENT)
);</code></pre>

<h3>{{ title }}</h3>
```

```
<p>
{{ welcome_message }}.
This guide does not require creating a new database file. The SQLite database is a
along with sample data. This guide is only for displaying the schema and reading d
</p>
 </div>
{% endblock %}
```

> *`content` block must exist in `base.html` as the placeholder for page content.*

---

6. `templates/base.html` — Global Layout

Example skeleton:

```
 <!DOCTYPE html>
<html lang="en">
<head>{% load static %}
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>{% block title %}Django & SQLite{% endblock %}</title>
  <link href="{% static 'img/favicon.png' %}" rel="icon" type="image/x-icon">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.
  <link href="{% static 'css/style.css' %}" rel="stylesheet">
</head>
<body class="bg-light d-flex flex-column align-items-center">
<header><h1>Django DB Pilot Header</h1></header>
<main>
{% block content %}{% endblock %}
</main>
<footer class="attribution mt-auto py-3 bg-light text-center">
        <p>
                Template by <a href="https://getbootstrap.com/" target="_blank">Bo
                Favicon by <a href="https://www.freepik.com/" target="_blank">Free
                Powered by <a href="https://www.djangoproject.com/" target="_blank
        </p>
</footer>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle
</body>
</html>
```

---

7. Settings Reminder (`core/settings.py`)

Ensure:

```
 SECRET_KEY = 'Put-your-own-secret-key-here-!j08aqmj1dul*x5&(j$ltap'

DEBUG = True

ALLOWED_HOSTS = ['127.0.0.1', 'localhost']
```

```
INTERNAL_IPS = [
    "127.0.0.1",  # Localhost
    "localhost",  # Also allow localhost by name
]

INSTALLED_APPS = [
    # Default Django apps...
    'django.contrib.staticfiles',

    # Your apps
    'apps.uom',             # <-- must match with `apps/uom/apps.py`
    'debug_toolbar',            # <-- Debug Toolbar
]

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],  # Global templates
        'APP_DIRS': True,                  # Looks inside app templates
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'debug_toolbar.middleware.DebugToolbarMiddleware',  # <-- Debug Toolbar
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

STATIC_URL = '/static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
```

✅ 8. How it Works

1. Root URL (`''`) routes to `apps.uom.views.index` .
2. `index` view passes context variables to `uom/index.html` .
3. `index.html` extends `base.html` , injecting content into `{% block content %}` .
4. Global CSS and assets are loaded via `STATICFILES_DIRS` .

This is the **minimal working setup** for templating with `base.html` and using `uom` as the home app. This guide does not require creating a new database file. The SQLite database is already provided in the repository along with sample data.

📘 Django App Guide 2: dbpilot with uom, base.html, and SQLite3

(Using `models.py` but WITHOUT creating tables — reading existing SQLite tables)

This guide shows how to build a Django app that:

- ✅ Uses `base.html`
- ✅ Uses your existing SQLite database
- ✅ Connects to an existing table (`stock_items_uom`)
- ✅ Reads/dumps data using Django ORM
- ❌ Does NOT create or modify the table
- ❌ Does NOT run migrations for this table
- ❌ Does NOT create or modify the database. The SQLite database is already provided in the repository with sample data

---

📁 Project Structure

```
project_folder/
├── manage.py
│
├── core/
│   ├── settings.py
│   └── urls.py
│
├── apps/
│   └── uom/
│       ├── models.py
│       ├── views.py
│       ├── urls.py
│       └── templates/
│           └── uom/
│               └── index.html
├── templates/
│   └── base.html
└── static/
│   ├── css/
│   │   └── style.css
│   └── img/
│       └── favicon.png
└── db.sqlite3
```

⚙️ 1. Configure SQLite in `settings.py`

Make sure your database is set like this:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

✔ No migrations needed for existing tables.

🧩 2. Create a Model for the Existing Table

Django must know the schema so it can read rows. But we prevent Django from creating or modifying the table using:

```
managed = False
```

`apps/uom/models.py`

```
from django.db import models

class StockItemUOM(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=255, unique=True)
    description = models.TextField(null=True, blank=True)
    status = models.IntegerField(default=1)
    created_at = models.DateTimeField()
    updated_at = models.DateTimeField()
    class Meta:
        db_table = 'stock_items_uom'  # Use your exact table name
        managed = False               # <-- Django will NOT create or alter this t

    def __str__(self):
        return self.name
```

✔ What this does:

- Maps Django ORM → to an **existing** SQLite table
- No migration needed
- You can run `StockItemUOM.objects.all()`
- Django reads data normally
- Zero risk of Django editing the schema

💻 3. Dump Data Using Django ORM

This view reads from the existing table and sends the data to the template.

`apps/uom/views.py`

```
 from django.shortcuts import render
from .models import StockItemUOM

def index(request):
    records = StockItemUOM.objects.all()  # ORM reading real rows

    return render(request, 'uom/index.html', {
        'title': 'UOM Home',
        'welcome_message': 'Dumping Data via Django ORM',
        'records': records,
    })
```

✔ Super clean ✔ ORM-only ✔ No raw SQL

🌐 4. Add URL Route

`apps/uom/urls.py`

```
 from django.urls import path
from . import views
```

```
app_name = 'uom'

urlpatterns = [
    path('', views.index, name='index'),
]
```

Root router (`core/urls.py`):

```
path('', include('apps.uom.urls')),
```

🎨 5. Display the Data in the Template

Inside **`apps/uom/templates/uom/index.html`**:

```
{% extends 'base.html' %}

{% block title %}{{ title }}{% endblock %}

{% block content %}
<h4 class="mb-3">SQLite Data Dump (Using Model)</h4>

<table class="table table-bordered table-striped">
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Description</th>
      <th>Status</th>
      <th>Created</th>
    </tr>
  </thead>
  <tbody>
    {% for row in records %}
    <tr>
      <td>{{ row.id }}</td>
      <td>{{ row.name }}</td>
      <td>{{ row.description }}</td>
      <td>{{ row.status }}</td>
      <td>{{ row.created_at }}</td>
    </tr>
    {% empty %}
    <tr>
      <td colspan="5" class="text-center text-muted">No data found</td>
    </tr>
    {% endfor %}
  </tbody>
</table>
{% endblock %}
```

✔ Uses your current layout ✔ No design changes required ✔ Simple data dump

---

🎉 Done!

You now have:

- Django reading your existing SQLite DB
- Models that map to tables without migration
- Data dumping on a clean template
- Base template layout supported
- A minimal, professional, GitHub-ready project

Only 2 core steps:

✔ `models.py` → add model with `managed=False`

✔ `views.py` → dump using ORM with `.objects.all()`

📘 Django App Guide 3: Full CRUD with dbpilot & UOM

(Using `models.py`, `forms.py`, and existing SQLite tables — with add/update/delete and automatic timestamps)

This guide shows how to build a Django app that:

- ✅ Displays all `stock*` tables for reference (read-only)
- ✅ CRUD for `stock_items_uom`
- ✅ Uses separate routes for Add / Update
- ✅ Auto-saves `created_at` and `updated_at`
- ✅ Uses a `select` for Status
- ❌ Does NOT create or modify existing tables
- ❌ Does NOT create a new database — SQLite database is provided

In this guide, we use **Django's ORM** (Object-Relational Mapping) to interact with the database without writing raw SQL. It allows us to manage database records easily through Python code, making it safer and faster to work with data like adding, updating, and deleting records.

📁 Project Structure

```
project_folder/
├── manage.py
│
├── core/
│   ├── settings.py
│   └── urls.py
│
├── apps/
│   └── uom/
│       ├── models.py
│       ├── views.py
│       ├── urls.py
│       ├── forms.py
│       ├── admin.py   # Optional
│       ├── tests.py   # Optional
│       └── templates/
│           └── uom/
│               ├── index.html
│               └── form.html
├── templates/
│   └── base.html
└── static/
│   ├── css/
│   └── img/
└── db.sqlite3
```

⚙️ 1. Configure SQLite in `settings.py`

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

✔ No migrations needed for existing tables.

🧩 2. Model for `stock_items_uom`

`apps/uom/models.py`

```
from django.db import models
```

```
class StockItemUOM(models.Model):
    STATUS_CHOICES = [
        (1, 'Active'),
        (2, 'Inactive'),
    ]

    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=255, unique=True)
    description = models.TextField(null=True, blank=True)
    status = models.IntegerField(choices=STATUS_CHOICES, default=1)
    created_at = models.DateTimeField()
    updated_at = models.DateTimeField()

    class Meta:
        db_table = 'stock_items_uom'
        managed = False  # Django will NOT create or alter this table

    def __str__(self):
        return self.name
```

📝 3. Form for Add/Update

`apps/uom/forms.py`

```
 from django import forms
from .models import StockItemUOM

class StockItemUOMForm(forms.ModelForm):
    STATUS_CHOICES = [
        (1, 'Active'),
        (2, 'Inactive'),
    ]

    status = forms.ChoiceField(
        choices=STATUS_CHOICES,
        widget=forms.Select(attrs={'class': 'form-select'})
    )

    class Meta:
        model = StockItemUOM
        fields = ['name', 'description', 'status']

        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control'}),
            'description': forms.TextInput(attrs={'class': 'form-control'}),
        }
```

💻 4. Views

apps/uom/views.py

```
 import sqlite3
from django.conf import settings
from django.shortcuts import render, redirect, get_object_or_404
from django.utils import timezone
from .models import StockItemUOM
from .forms import StockItemUOMForm

DB_PATH = settings.BASE_DIR / 'db.sqlite3'

def index(request):
    # List all stock* tables (read-only)
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name LIK
    tables = [row[0] for row in cursor.fetchall()]
    conn.close()

    # Active table: stock_items_uom
    records = StockItemUOM.objects.all().order_by('id')

    return render(request, 'uom/index.html', {
        'title': 'UOM List',
        'records': records,
        'tables': tables,
    })

def add_record(request):
    if request.method == 'POST':
        form = StockItemUOMForm(request.POST)
        if form.is_valid():
            record = form.save(commit=False)
            record.created_at = timezone.now()
            record.updated_at = timezone.now()
            record.save()
            return redirect('uom:index')
    else:
        form = StockItemUOMForm()
    return render(request, 'uom/form.html', {'title': 'Add UOM', 'form': form})

def update_record(request, pk):
    record = get_object_or_404(StockItemUOM, pk=pk)
    if request.method == 'POST':
        form = StockItemUOMForm(request.POST, instance=record)
        if form.is_valid():
            updated = form.save(commit=False)
            updated.updated_at = timezone.now()
            updated.save()
            return redirect('uom:index')
    else:
        form = StockItemUOMForm(instance=record)
    return render(request, 'uom/form.html', {'title': f'Update UOM ID {record.id}'

def delete_record(request, pk):
    record = get_object_or_404(StockItemUOM, pk=pk)
    record.delete()
    return redirect('uom:index')
```

🌐 5. URLs

`apps/uom/urls.py`

```
from django.urls import path
from . import views

app_name = 'uom'

urlpatterns = [
    path('', views.index, name='index'),
    path('add/', views.add_record, name='add'),
    path('update/<int:pk>/', views.update_record, name='update'),
    path('delete/<int:pk>/', views.delete_record, name='delete'),
]
```

Root `core/urls.py`:

```
from django.urls import path, include

urlpatterns = [
    path('', include('apps.uom.urls')),
]
```

🖌️ 6. Templates

Index — `apps/uom/templates/uom/index.html`

```
{% extends 'base.html' %}
{% block title %}{{ title }}{% endblock %}

{% block content %}
<div class="card shadow-lg p-4 mb-3">
    <h4 class="mb-4 text-center">Database db.sqlite3 / SQLite Tables & UOM Records

    <!-- Table selector + Add button -->
    <div class="mb-3 d-flex flex-wrap align-items-center gap-2">
        <label for="tableSelect" class="form-label mb-0">Table:</label>
        <select class="form-select form-select-sm" id="tableSelect" style="width:a
            {% for table in tables %}
                <option value="{{ table }}" {% if table == 'stock_items_uom' %}sel
            {% endfor %}
        </select>
        <a href="{% url 'uom:add' %}" class="btn btn-success btn-sm">Add</a>
    </div>

    <!-- Records table -->
    <div class="table-responsive">
        <table class="table table-sm table-striped table-bordered">
            <thead>
                <tr>
                    <th>ID</th><th>Name</th><th>Description</th><th>Status</th>
                    <th>Created At</th><th>Updated At</th><th width="120">Actions<
                </tr>
            </thead>
            <tbody>
                {% for row in records %}
                <tr>
                    <td>{{ row.id }}</td>
                    <td>{{ row.name }}</td>
                    <td>{{ row.description }}</td>
                    <td>{{ row.get_status_display }}</td>
                    <td>{{ row.created_at }}</td>
                    <td>{{ row.updated_at }}</td>
                    <td class="text-nowrap d-flex gap-1">
                        <a href="{% url 'uom:update' row.id %}" class="btn btn-lig
                        <a href="{% url 'uom:delete' row.id %}"
                           onclick="return confirm('Delete this record?')"
                           class="btn btn-light btn-sm">Delete</a>
                    </td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
</div>
{% endblock %}
```

Form — `apps/uom/templates/uom/form.html`

```
 {% extends 'base.html' %}
{% block title %}{{ title }}{% endblock %}

{% block content %}
<div class="card shadow-lg p-4 mb-3">
    <h3 class="mb-4">{{ title }}</h3>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="btn btn-primary btn-sm">Save</button>
        <a href="{% url 'uom:index' %}" class="btn btn-secondary btn-sm">Cancel</a
    </form>
</div>
{% endblock %}
```

Got it! Here's how we can integrate that into the guide:

🎉 Done!

- Display all `stock*` tables
- Full CRUD for `stock_items_uom`
- Add / Update with separate forms
- Delete inline
- Automatic timestamps
- Status as a dropdown
- ORM-only — no raw SQL for CRUD
- Existing SQLite DB only — no migrations needed

Optional: Using **Django Admin Integration** to Manage `StockItemUOM`

Once you add the following code to your `admin.py` , you can log into the Django admin interface at `http://localhost:8000/admin` and easily manage your `StockItemUOM` records. The project already includes a sample SQLite database with tables and data, so you don't need to worry about setting up the database.

To access the Django admin interface, use one of the following accounts:

- **User account:**
  - **Username:** `user`
  - **Password:** `demo`
- **Admin account:**
  - **Username:** `admin`
  - **Password:** `root`

`apps/uom/admin.py`

```
 from django.contrib import admin
from .models import StockItemUOM

class StockItemUOMAdmin(admin.ModelAdmin):
    # Make all fields read-only
    readonly_fields = ('id', 'name', 'description', 'status', 'created_at', 'updat

    # Optionally, you can exclude certain fields from the admin form
    # exclude = ('field_to_exclude',)

    # If you want to make the model non-editable, you can define:
    # fields = ('name', 'description', 'status')  # and exclude other fields from

    # You can add more filters, ordering, or search fields here if needed
    search_fields = ['name']
    list_filter = ['status']
    list_display = ['name', 'status', 'created_at', 'updated_at']

admin.site.register(StockItemUOM, StockItemUOMAdmin)
```

Once you've added this to your `admin.py`, head over to `http://localhost:8000/admin` in your browser, log in with the provided credentials, and start managing your `StockItemUOM` records right from the admin panel.

This is entirely optional, but it can save time when working with large datasets!

📘 Django App Guide 4: Add 2nd Table (Categories) & Include Partial Template Globally

This guide shows how to **add a second table to your Django project** by copying the structure of an existing table (in this case, `UOM`) and modifying it to create a new, independent app (`Categories`). The process can be repeated as many times as needed to add more tables without affecting existing apps.

It also demonstrates how to **include a partial template globally** using Django's `{% include %}` tag, allowing you to reuse common interface elements like table selectors, buttons, or headers across multiple apps, ensuring a consistent look and reducing duplication in your templates.

By the end of this guide, you will be able to:

- ✅ Duplicate and adapt an existing CRUD app for a new table.
- ✅ Maintain independent routes, forms, and views for each table.
- ✅ Include reusable template parts for shared UI components.
- ✅ Handle automatic timestamps (`created_at and updated_at`) for new records.
- ✅ Keep full control over your database without creating new tables manually.

(Using `models.py`, `forms.py`, existing SQLite table `stock_items_categories`, and reusable partial templates)

This guide shows how to build a Django app that:

- ✅ Displays all `stock*` tables for reference (read-only)
- ✅ CRUD for `stock_items_categories`
- ✅ Uses separate routes for Add / Update
- ✅ Auto-saves `created_at and updated_at`

- ✅ Uses a `select` for Status
- ✅ Includes a reusable partial template (`_table_select.html`) globally
- ❌ Does NOT create or modify existing tables
- ❌ Does NOT create a new database — SQLite database is provided

This guide continues from **Guide 3** and focuses on **adding the Categories app** and using **partial templates** for code reuse.

📁 Project Structure

```
project_folder/
├── manage.py
│
├── core/
│   ├── settings.py
│   └── urls.py
│
├── apps/
│   ├── uom/
│   │   ├── models.py
│   │   ├── views.py
│   │   ├── urls.py
│   │   ├── forms.py
│   │   └── templates/uom/
│   │       ├── index.html
│   │       └── form.html
│   │
│   └── categories/
│       ├── models.py
│       ├── views.py
│       ├── urls.py
│       ├── forms.py
│       └── templates/categories/
│           ├── index.html
│           └── form.html
│
├── templates/
│   ├── base.html
│   └── includes/
│       └── _table_select.html      # Partial template for table selection
└── db.sqlite3
```

⚙️ 1. Configure SQLite in `settings.py`

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

✔ No migrations needed for existing tables.

🧩 2. Model for `stock_items_categories`

`apps/categories/models.py`

```
from django.db import models

class StockItemCategories(models.Model):
    STATUS_CHOICES = [
        (1, 'Active'),
        (2, 'Inactive'),
    ]

    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=255, unique=True)
    description = models.TextField(null=True, blank=True)
    status = models.IntegerField(choices=STATUS_CHOICES, default=1)
    created_at = models.DateTimeField()
    updated_at = models.DateTimeField()

    class Meta:
        db_table = 'stock_items_categories'
        managed = False  # Do NOT let Django alter the table

    def __str__(self):
        return self.name
```

📝 3. Form for Add/Update

`apps/categories/forms.py`

```
from django import forms
from .models import StockItemCategories

class StockItemCategoriesForm(forms.ModelForm):
    STATUS_CHOICES = [
        (1, 'Active'),
        (2, 'Inactive'),
    ]

    status = forms.ChoiceField(
        choices=STATUS_CHOICES,
        widget=forms.Select(attrs={'class': 'form-select'})
    )

    class Meta:
        model = StockItemCategories
        fields = ['name', 'description', 'status']  # created_at / updated_at excl

        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control'}),
            'description': forms.TextInput(attrs={'class': 'form-control'}),
        }
```

💻 4. Views

apps/categories/views.py

```
import sqlite3
from django.conf import settings
from django.shortcuts import render, redirect, get_object_or_404
from django.utils import timezone
from .models import StockItemCategories
from .forms import StockItemCategoriesForm

DB_PATH = settings.BASE_DIR / 'db.sqlite3'

def index(request):
    # List all stock* tables for display only
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name LIK
    tables = [row[0] for row in cursor.fetchall()]
    conn.close()

    records = StockItemCategories.objects.all().order_by('id')

    return render(request, 'categories/index.html', {
        'title': 'Categories List - Database db.sqlite3 / SQLite Tables',
        'records': records,
        'tables': tables,
        'current_table': 'stock_items_categories',
    })

def add_record(request):
    if request.method == 'POST':
        form = StockItemCategoriesForm(request.POST)
        if form.is_valid():
            record = form.save(commit=False)
            now = timezone.now()
            record.created_at = now
            record.updated_at = now
            record.save()
            return redirect('categories:index')
    else:
        form = StockItemCategoriesForm()

    return render(request, 'categories/form.html', {
        'title': 'Add Category',
        'form': form,
    })


def update_record(request, pk):
    record = get_object_or_404(StockItemCategories, pk=pk)

    if request.method == 'POST':
        form = StockItemCategoriesForm(request.POST, instance=record)
        if form.is_valid():
            updated = form.save(commit=False)
            updated.updated_at = timezone.now()
            updated.save()
            return redirect('categories:index')
    else:
        form = StockItemCategoriesForm(instance=record)

    return render(request, 'categories/form.html', {
        'title': f'Update Category ID {record.id}',
        'form': form,
    })


def delete_record(request, pk):
    record = get_object_or_404(StockItemCategories, pk=pk)
    record.delete()
    return redirect('categories:index')
```

🌐 5. URLs

`apps/categories/urls.py`

```
from django.urls import path
from . import views

app_name = 'categories'

urlpatterns = [
    path('', views.index, name='index'),
    path('add/', views.add_record, name='add'),
    path('update/<int:pk>/', views.update_record, name='update'),
    path('delete/<int:pk>/', views.delete_record, name='delete'),
]
```

Root core/urls.py:

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('apps.uom.urls')),            # Home app
    path('categories/', include('apps.categories.urls')),  # Categories app
]

if settings.DEBUG:
    import debug_toolbar
    urlpatterns = [path('__debug__/', include(debug_toolbar.urls))] + urlpatterns
```

🖌️ 6. Templates

Partial Template — `_table_select.html`

```
<select class="form-select form-select-sm" id="tableSelect" style="width:auto; mi
    {% for table in tables %}
        <option value="{{ table }}" {% if table == current_table %}selected{% endi
    {% endfor %}
</select>
```

Index — `apps/categories/templates/categories/index.html`

```
{% extends 'base.html' %}

{% block title %}{{ title }}{% endblock %}

{% block content %}
<div class="card shadow-lg p-4 mb-3">

    <h4 class="mb-4 text-center">{{ title }}</h4>

    <!-- Table selection + Add button -->
    <div class="mb-3 d-flex flex-wrap align-items-center gap-2">
        <label for="tableSelect" class="form-label mb-0">Table:</label>

        {% include 'includes/_table_select.html' %}
        <a href="{% url 'categories:add' %}" class="btn btn-success btn-sm" style=
    </div>

    <!-- Table with Categories records -->
    <div class="table-responsive">
        <table class="table table-sm table-striped table-bordered">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Name</th>
                    <th>Description</th>
                    <th>Status</th>
                    <th>Created At</th>
                    <th>Updated At</th>
                    <th width="120">Actions</th>
                </tr>
            </thead>
            <tbody>
                {% for row in records %}
                <tr>
                    <td>{{ row.id }}</td>
                    <td>{{ row.name }}</td>
                    <td>{{ row.description }}</td>
                    <td>{{ row.get_status_display }}</td>
                    <td>{{ row.created_at }}</td>
                    <td>{{ row.updated_at }}</td>
                    <td class="text-nowrap d-flex gap-1">
                        <a href="{% url 'categories:update' row.id %}" class="btn
                        <a href="{% url 'categories:delete' row.id %}"
                           onclick="return confirm('Delete this record?')"
                           class="btn btn-light btn-sm">Delete</a>
                    </td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>

</div>
{% endblock %}
```

Form — `apps/categories/templates/categories/form.html`

```
{% extends 'base.html' %}

{% block title %}{{ title }}{% endblock %}

{% block content %}
<div class="card shadow-lg p-4 mb-3">
    <h3 class="mb-4">{{ title }}</h3>

    <form method="post">
```

```
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="btn btn-primary btn-sm">Save</button>
        <a href="{% url 'categories:index' %}" class="btn btn-secondary btn-sm">Ca
    </form>
</div>
{% endblock %}
```

🎉 Done!

- Full CRUD for `stock_items_categories`
- Timestamps `created_at` and `updated_at` displayed, automatically added on creation
- Status dropdown selection
- Global partial template `_table_select.html` reused for table selector
- Add / Update forms exclude timestamps
- Delete inline with confirmation
- Existing SQLite DB only — no migrations needed

📘 Django - Guide 5: Add `Items` Table with Foreign Keys

This guide demonstrates how to **create a new `Items` table** in your Django project and establish foreign key relationships to the **`Categories`** and **`UOM` (Units of Measure)** tables created in previous guides. It will cover the **CRUD functionality** for the `Items` table, including how to handle these foreign key relationships in your forms and templates.

By the end of this guide, you will be able to:

- ✅ **Create a new `Items` table** with foreign keys to `Categories` and `UOM`.
- ✅ Handle **foreign key relationships** to display category and UOM names instead of IDs.
- ✅ Build **CRUD views** for the `Items` table.
- ✅ Use **ModelForms** for adding and updating `Items` records.
- ✅ Reuse the **partial template** for table selection globally.

📁 Project Structure

```
project_folder/
```

```
├── manage.py
│
├── core/
│   ├── settings.py
│   └── urls.py
│
├── apps/
│   ├── uom/
│   │   ├── models.py
│   │   ├── views.py
│   │   ├── urls.py
│   │   ├── forms.py
│   │   └── templates/uom/
│   │       ├── index.html
│   │       └── form.html
│   │
│   ├── categories/
│   │   ├── models.py
│   │   ├── views.py
│   │   ├── urls.py
│   │   ├── forms.py
│   │   └── templates/categories/
│   │       ├── index.html
│   │       └── form.html
│   │
│   └── items/
│       ├── models.py
│       ├── views.py
│       ├── urls.py
│       ├── forms.py
│       └── templates/items/
│           ├── index.html
│           └── form.html
│
├── templates/
│   ├── base.html
│   └── includes/
│       └── _table_select.html      # Partial template for table selection
└── db.sqlite3
```

⚙️ 1. Configure SQLite in `settings.py`

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

✔ **No migrations needed** for existing tables. We're working with a pre-existing database.

🧩 2. Model for `Items` Table

`apps/items/models.py`

```
from django.db import models

class StockItems(models.Model):
    STATUS_CHOICES = [
        (1, 'Active'),
        (0, 'Inactive'),
    ]

    id = models.AutoField(primary_key=True)
    code = models.CharField(max_length=255, unique=True)
    description = models.TextField(null=True, blank=True)
    category = models.ForeignKey('categories.StockItemCategories', on_delete=model
    uom = models.ForeignKey('uom.StockItemUOM', on_delete=models.SET_NULL, null=Tr
    status = models.IntegerField(choices=STATUS_CHOICES, default=1)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        db_table = 'stock_items'
        managed = False  # Do NOT let Django alter the table

    def __str__(self):
        return self.code
```

📝 3. Form for Add/Update

`apps/items/forms.py`

```
from django import forms
from .models import StockItems
from categories.models import StockItemCategories
from uom.models import StockItemUOM

class StockItemsForm(forms.ModelForm):
    STATUS_CHOICES = [
        (1, 'Active'),
        (0, 'Inactive'),
    ]

    # ForeignKey to StockItemCategories (Categories)
    category = forms.ModelChoiceField(
        queryset=StockItemCategories.objects.filter(status=1),  # Only show active
        required=True,
        widget=forms.Select(attrs={'class': 'form-select'}),
        empty_label='Select a category'
    )

    # ForeignKey to StockItemUOM (Units of Measure)
    uom = forms.ModelChoiceField(
        queryset=StockItemUOM.objects.all(),  # All UOMs are available
        required=True,
        widget=forms.Select(attrs={'class': 'form-select'}),
        empty_label='Select a UOM'
    )

    # Status dropdown (active or inactive)
    status = forms.ChoiceField(
        choices=STATUS_CHOICES,
        widget=forms.Select(attrs={'class': 'form-select'})
    )

    class Meta:
        model = StockItems
        fields = ['code', 'description', 'category', 'uom', 'status']
        widgets = {
            'code': forms.TextInput(attrs={'class': 'form-control'}),
            'description': forms.Textarea(attrs={'class': 'form-control'}),
        }
```

💻 4. Views

`apps/items/views.py`

```
 from django.shortcuts import render, redirect, get_object_or_404
from django.utils import timezone
from .models import StockItems
from .forms import StockItemsForm

def index(request):
    records = StockItems.objects.all().order_by('id')
    return render(request, 'items/index.html', {
        'title': 'Items List',
        'records': records,
    })

def add_record(request):
    if request.method == 'POST':
        form = StockItemsForm(request.POST)
        if form.is_valid():
            record = form.save(commit=False)
            record.created_at = timezone.now()
            record.updated_at = timezone.now()
            record.save()
            return redirect('items:index')
    else:
        form = StockItemsForm()

    return render(request, 'items/form.html', {
        'title': 'Add Item',
        'form': form,
    })

def update_record(request, pk):
    record = get_object_or_404(StockItems, pk=pk)
    if request.method == 'POST':
        form = StockItemsForm(request.POST, instance=record)
        if form.is_valid():
            updated = form.save(commit=False)
            updated.updated_at = timezone.now()
            updated.save()
            return redirect('items:index')
    else:
        form = StockItemsForm(instance=record)

    return render(request, 'items/form.html', {
        'title': f'Update Item ID {record.id}',
        'form': form,
    })

def delete_record(request, pk):
    record = get_object_or_404(StockItems, pk=pk)
    record.delete()
    return redirect('items:index')
```

🌐 5. URLs

`apps/items/urls.py`

```
 from django.urls import path
from . import views

app_name = 'items'

urlpatterns = [
    path('', views.index, name='index'),
    path('add/', views.add_record, name='add'),
    path('update/<int:pk>/', views.update_record, name='update'),
    path('delete/<int:pk>/', views.delete_record, name='delete'),
]
```

🖌️ 6. Templates

Index — `apps/items/templates/items/index.html`

```
 {% extends 'base.html' %}

{% block title %}{{ title }}{% endblock %}

{% block content %}
<div class="card shadow-lg p-4 mb-3">

    <h4 class="mb-4 text-center">{{ title }}</h4>

    <!-- Table selection + Add button -->
    <div class="mb-3 d-flex flex-wrap align-items-center gap-2">
        <label for="tableSelect" class="form-label mb-0">Table:</label>

        {% include 'includes/_table_select.html' %}

        <a href="{% url 'items:add' %}" class="btn btn-success btn-sm" style="posi
    </div>

    <!-- Table with Items records -->
    <div class="table-responsive">
        <table class="table table-sm table-striped table-bordered">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Code</th>
                    <th>Description</th>
                    <th>Category</th>
                    <th>UOM</th>
                    <th>Status</th>
                    <th>Created At</th>
                    <th>Updated At</th>
                    <th width="120">Actions</th>
                </tr>
            </thead>
            <tbody>
                {% for row in records %}
                <tr>
                    <td>{{ row.id }}</td>
                    <td>{{ row.code }}</td>
                    <td>{{ row.description }}</td>
                    <td>{{ row.category.name }}</td> <!-- Display category name --
                    <td>{{ row.uom.name }}</td> <!-- Display UOM name -->
                    <td>{{ row.get_status_display }}</td>
                    <td>{{ row.created_at }}</td>
```

{{ row.updated_at }} Update Delete {% endfor %} {% endblock %} ```

Form — `apps/items/templates/items/form.html`

```
 {% extends 'base.html' %}

{% block title %}{{ title }}{% endblock %}

{% block content %}
<div class="card shadow-lg p-4 mb-3">
    <h3 class="mb-4">{{ title }}</h3>

    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="btn btn-primary btn-sm">Save</button>
        <a href="{% url 'items:index' %}" class="btn btn-secondary btn-sm">Cancel<
    </form>
</div>
{% endblock %}
```

🎉 Done!

- Full CRUD for `Items` table with **foreign keys** to `Categories` and `UOM`
- **Timestamps** `created_at` and `updated_at` automatically set
- **Category** and **UOM** names displayed in the table
- Reused the **partial template** `_table_select.html` for table selection
- **No migrations required**—working with an existing SQLite database