# Workshop: Fruitipedia App

This document defines the **exercise assignments** for the [Python ORM course @ Software University](https://softuni.bg/trainings/4253/python-orm-october-2023).

## Skeleton

You are provided with all the needed **HTML pages**, **images,** and **CSS** files for the project.

## Setup

### Creating the Django Project

Create a new **Django** project called "**fruitipediaApp**".

### Creating the App

After creating the **Django** project, we are ready to create the **app** we will work with. For instance, we can name it "**fruits"**.

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

For clarification, **move the created app inside** the project:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

### Configurations

We need to **add the apps** we just created in the **INSTALLED\_APPS** setting:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

### Adding the Templates

The next step is to **move the already-configured folders with HTML templates inside the templates folder.**

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

### Adding the Static Files

Add the provided folders ("**images**" and "**styles**") to the directory. Next, **Django** should find the static files when loading web pages, so write the setting in the **settings.py** file:

Картина, която съдържа текст, Шрифт, софтуер, номер

Описанието е генерирано автоматично

### Adding the URLs (paths)

We want to load each template in the browser using a concrete path - each app should load its templates.

To do that, we should add **urls.py** files in each app:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

Then, we can start including them in the main project **urls.py** file. We should import the **include()** function from the Django **urls** module, then we can use the **path()** function to **construct a path**, which will **lead to each app** **urlpatterns**:

Картина, която съдържа текст, Шрифт, линия, номер

Описанието е генерирано автоматично

## Database

You will need **2 models**:

### Category Model

* + **name**
    - Character field, **required.**
    - Each category is **unique**.

### Fruit Model

* + **name**
    - Character field, **required.**
    - It should consist of a maximum of **30 and** a **minimum of 2 characters**.
    - The name should contain **only letters**. Otherwise raise a **ValidationError** with the following message: "**Fruit name should contain only letters!**"
  + **Image\_url**
    - URL field, **required.**
  + **description**
    - Text field, **required.**
  + **nutrition**
    - Text field, **optional.**

## Routes

* <http://localhost:8000/> - index page
* <http://localhost:8000/dashboard/> - dashboard page
* <http://localhost:8000/create-fruit/> - fruit create page
* [http://localhost:8000/<fruitId>/details-fruit/](http://localhost:8000/%3cfruitId%3e/details-fruit/) - fruit details page
* [http://localhost:8000/<fruitId>/edit-fruit/](http://localhost:8000/%3cfruitId%3e/edit-fruit/) - fruit edit page
* [http://localhost:8000/<fruitId>/delete-fruit/](http://localhost:8000/%3cfruitId%3e/delete-fruit/) - fruit delete page
* <http://localhost:8000/create-category/> - category create page

## Views

In Django, a **view** is a Python function or class that defines the logic for processing and handling HTTP requests. Views are a fundamental part of the **Model-Template-View (MTV) architectural pattern used in Django.** They receive an **HTTP** **request** from a client (usually a web browser), perform some processing, and then return an HTTP response, which could be an **HTML page**, JSON data, or any other content type.

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

Описанието е генерирано автоматично

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

## Fruits App URLs

Inside the **fruits** app directory, create a Python file named **urls.py**. In this file, you define URL patterns that specify which **views** should be called for different URL **paths**. The **urls.py** file should contain a variable called **urlpatterns** that is a list of URL patterns. For example, your **fruits/urls.py** might look like this:

Картина, която съдържа текст, екранна снимка, номер, Шрифт

Описанието е генерирано автоматично

### Creating Template Inheritance

If we look closely at each template, we can see that there are many common parts. The head, the header with the navigation bar, and the footer are the same for all templates. We can export them in a separate **.html** file in the project's **template** directory.

Let us create a **base.html** template in the **common** directory. We will position it there because the code is common for the app:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8"/>  
 <meta http-equiv="X-UA-Compatible" content="IE=edge"/>  
 <meta name="viewport" content="width=\, initial-scale=1.0"/>  
 <link rel="stylesheet" href="/styles/styles.css"/>  
 <title>Fruitipedia</title>  
</head>  
<body>  
<div id="wrapper">  
 <header>  
 *<!-- Navigation -->* <a id="logo" href="">  
 <img id="logo-img" src="/images/logo.png" alt="logo"/>  
 </a>  
  
 <nav>  
 <div class="nav-items">  
 <a href="#">Create Category</a>  
 <a href="#">Create Fruit</a>  
 <a href="#">Dashboard</a>  
 </div>  
 </nav>  
 </header>  
  
 <main>  
  
 {% block content %}  
 {% endblock %}  
 </main>  
</div>  
*<!-- footer -->*<footer>  
 <p>@Fruitipedia - SoftUni Team 2023. All rights reserved.</p>  
</footer>  
</body>  
</html>

In Django templates, **{% block %}** and **{% endblock %}** are template tags used to define and delimit named content blocks within a template. These blocks are a fundamental part of the template inheritance system, which allows you to create a base template and override specific parts of it in child templates.

To inherit this base HTML code in every template we need to **extend** the **base.html** file (for this example we are going to work with the **index.html**):

{% extends 'common/base.html' %}  
{% block content %}  
  
 *<!-- Home page -->* <section id="home">  
 <h1>Learn more about your favorite fruits</h1>  
 <img  
 src="/images/pexels-pixabay-161559-dImkWBDHz-transformed (1).png"  
 alt="home"  
 />  
  
 </section>  
  
{% endblock %}

**You should apply this for every template.**

### Loading Static files

In Django, the **load** **static** template tag is used to load and render static files, such as CSS, JavaScript, images, and other assets, in a Django template. Static files are typically used for styling your web pages and adding client-side functionality to your website (here we are working in the **base.html**). **Apply these changes in every template.**

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

## Pages

## Index page

**Template file**: **"index.html"**

The page consists of:

* A **navigation bar** with:
* **"Fruitipedia"** link, which leads to the **index** page.
* **"Create Category"** link, which leads to **create category** page.
* **"Create Fruit"** link, which leads to the **create fruit** page.
* **"Dashboard"** link, which leads to the **dashboard** page.

Картина, която съдържа текст, оранжево, плод, грейпфрут

Описанието е генерирано автоматично

## Dashboard Page

**Template file**: **"dashboard.html"**

The **dashboard page shows all fruits that have been created.**

If there are **no fruits created yet**, the page should have the following:

* A navigation bar, as shown below.
* A heading **"No fruit info yet"**

Картина, която съдържа текст, екранна снимка, дизайн

Описанието е генерирано автоматично

If the **there are some** **fruits,** the page should have the following:

* A navigation bar, as shown below.
* A division for each fruit, showing:
* The fruit's **image.**
* The fruit's **name.**
* The fruit's **description**.
* A **button** **"More Info"** leading to the **details page** for the selected fruit.

Картина, която съдържа текст, плод, екранна снимка, ябълка

Описанието е генерирано автоматично

Here we create the functionality in the **views.py:**

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

We also need to display the need information as we modify the **dashboard.html**.

{% extends 'common/base.html' %}  
{% block content %}  
  
  
 *<!-- Dashboard Page -->* {% if fruits %}  
 <h2>Fruits</h2>  
 <section id="dashboard">  
 {% for fruit in fruits %}  
 <div class="fruit">  
 *<!-- fruit image -->* <img src="{{ fruit.image\_url }}" alt="example1"/>  
 *<!-- fruit name -->* <h3 class="title">Name: {{ fruit.name }}</h3>  
 *<!-- fruit description -->* <p class="description">Description: {{ fruit.description }}</p>  
 *<!-- fruit details button -->* <a class="details-btn" href="{% url 'details fruit' fruit\_id=fruit.pk %}">More Info</a>  
 </div>  
 {% endfor %}  
 </section>  
 {% else %}  
 <h2>No fruit info yet.</h2>  
 {% endif %}  
  
{% endblock %}

## Create Category Page

Template file: **"create-category.html"**

The page consists of:

* A **navigation bar,** as shown below.
* A **category** **creation form** consisting of:
* A **"name"** field.
* A button **"Add Category"**
* When you **click** on it **if** the fruit is **successfully created**, you should be **redirected** to the **dashboard page**.

Картина, която съдържа текст, екранна снимка, дизайн

Описанието е генерирано автоматично

First, we must create a **Django** **form** that will handle the inputs from the user side:

Картина, която съдържа текст, екранна снимка, софтуер, дисплей

Описанието е генерирано автоматично

Let's break the down the code:

**Import Statements**:

* The code imports necessary modules and the **Fruit** and **Category** models from the application.

**CategoryBaseForm Class**:

* **CategoryBaseForm** is a subclass of **forms.ModelForm**, which is a base class for creating forms based on Django models. In this case, it's associated with the **Category** model.
* The **Meta** class is nested inside **CategoryBaseForm** and is used to provide additional metadata about the form.
* **model = Category** specifies that this form is based on the **Category** model.
* **fields = '\_\_all\_\_'** means that the form should include all fields defined in the **Category** model. This is a shortcut to include all fields without explicitly listing them.

**CategoryCreateForm Class:**

* **CategoryCreateForm** is a subclass of **CategoryBaseForm**. It inherits all the fields and behavior from **CategoryBaseForm** without any additional customization.
* It's common to create form classes that inherit from a base form to reuse common form logic and fields.

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

Let's analyze this code also:

**HTTP Method Check:**

* This code checks if the HTTP request method is a GET request. A GET request is typically used for retrieving information or displaying forms.

**Form Initialization:**

* If the request method is GET, a new instance of a **CategoryCreateForm** is created. This form is used to collect data for creating a new category.

**Form Submission Handling:**

* If the request method is not GET (presumably POST), it means the user has submitted the form. In this case, a new instance of **CategoryCreateForm** is created, and it is initialized with the data from the **request.POST** dictionary. This allows the form to capture the data submitted by the user.

**Form Validation:**

* The code checks if the submitted form is valid. In Django, a form can contain various fields and validation rules. **form.is\_valid()** checks if all the data entered by the user complies with the validation rules defined in the **CategoryCreateForm**.

**Saving the Form Data:**

* If the form is valid, the code proceeds to save the data to the database. The **form.save()** method is responsible for creating a new category object and saving it to the database. This assumes that the **CategoryCreateForm** is a Django ModelForm associated with the Category model.

**Redirecting:**

* After successfully saving the form data, the code redirects the user to a page named 'dashboard'. This is often used to show a success message or display the updated list of categories.

**Context Dictionary:**

* A context dictionary is created, which includes the form. The context is used to pass data from the view to the template.

**Rendering the Template**:

* Finally, the view renders an HTML template named 'create-category.html' and includes the context data (including the form). This rendered page is then sent back to the user's browser as a response to the HTTP request.

Furthermore, we need to modify our **create-category.html** file:

{% extends 'common/base.html' %}  
{% block content %}  
  
 *<!—Add Category Form -->* <section id="create">  
 <div class="form">  
 <h2>Add Category</h2>  
 <form class="create-form" method="post" action="{% url 'create category' %}">  
  
 {{ form }}  
 <button type="submit">Add Category</button>  
 {% csrf\_token %}  
  
 </form>  
 </div>  
 </section>  
  
{% endblock %}

**{% extends 'common/base.html' %}:**

* This line of code tells Django that this template should inherit the structure and content from the 'common/base.html' template. In other words, it establishes a parent-child relationship between this template and the base template. The base template usually contains the common structure, layout, and elements shared across multiple pages of the website, such as the header, navigation menu, and footer.

**{% block content %}** and **{% endblock %}**:

* These template tags define a content block named 'content.' The 'content' block is a placeholder that allows you to override or insert content specific to this page. In this case, it's used to replace the content section of the 'common/base.html' template with custom content for this page.

**HTML Content**:

* Inside the 'content' block, the HTML content for this specific page is defined. It appears to be a form for adding a new category. Here's a breakdown of the content:
  + **<section id="create">**:
    - This defines a section with the ID 'create,' which can be used for styling or JavaScript interactions.
  + **<div class="form">**:
    - A div element with the class 'form' is used to style the form container.
  + **<h2>Add Category</h2>**:
    - A heading that displays "Add Category."
  + **<form class="create-form" method="post" action="{% url 'create category' %}">**:
    - This is an HTML form element. It specifies that the form should use the POST method to submit data. The 'action' attribute is set to **{% url 'create category' %}**, which dynamically generates the URL for form submission using the named URL pattern 'create category.'
  + **{{ form }}**:
    - This is a Django template variable. It is likely to be a form variable passed from the view using the context data. This variable renders the form fields and their associated widgets, allowing users to input data for the 'Category' model. The form rendering is generated automatically based on the form definition and Django's form rendering system.
  + **<button type="submit">Add Category</button>**:
    - This is a submit button for the form. When clicked, it will trigger the form submission.
  + **{% csrf\_token %}**:
    - This template tag generates a hidden input field that contains a CSRF token. This is essential for security to prevent cross-site request forgery (**CSRF**) attacks.

## Create Fruit Page

Template file: **"create-fruit.html"**

The page consists of:

* A **navigation bar,** as shown below.
* A **fruit** **creation form** consisting of:
* A **"name"** field.
* An **"image\_url"** field.
* A **"description"** field.
* A **"nutrition"** field.
* A **"category"** field.
* A button **"Add Fruit"**
* When you **click** on it **if** the fruit is **successfully created**, you should be **redirected** to the **dashboard page**.
* Otherwise, the form should show the **appropriate validation errors**.

Картина, която съдържа текст, екранна снимка, дизайн

Описанието е генерирано автоматично

**Form**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

**View**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

**Template**:

{% extends 'common/base.html' %}  
{% block content %}  
  
 *<!-- Add Fruit Form -->* <section id="create">  
 <div class="form">  
 <h2>Add Fruit</h2>  
 <form class="create-form" method="post" action="{% url 'create fruit' %}">  
  
 {{ form }}  
 <button type="submit">Add Fruit</button>  
 {% csrf\_token %}  
  
 </form>  
 </div>  
 </section>  
  
{% endblock %}

## Fruit Details Page

Template file: **"details-fruit.html"**

This page contains a fruit’s information. It should have the following:

* The **fruit's image.**
* The **fruit's name.**
* The **fruit's category.**
* The **fruit's description.**
* The **fruit's nutrition info**, starting with the **paragraph "Nutrition"** (**visible** even if there is **no** nutrition info).
* An **"Edit"** button that leads to the **edit fruit page.**
* A **"Delete"** button that leads to the **delete fruit page.**

Картина, която съдържа текст, плод, екранна снимка, банан

Описанието е генерирано автоматично

**View**:

Картина, която съдържа текст, Шрифт, екранна снимка

Описанието е генерирано автоматично

**Template**:

In the code provided below, the **reason** for passing the **ID** of the **fruit** (**fruit\_id**) when generating URLs for the "**Edit**" and "**Delete**" buttons is to **identify** which specific **fruit** the user wants to **edit** or **delete**. This is **typically** used in a web application, especially when dealing with data in a database.

{% extends 'common/base.html' %}  
{% block content %}  
  
  
 *<!-- Fruit Details -->* <section id="details">  
 <div id="details-wrapper">  
 *<!-- fruit image -->* <img id="details-img" src="{{ fruit.image\_url }}" alt="example1"/>  
 *<!-- fruit name -->* <p id="details-title">Name: {{ fruit.name }}</p>  
 <div id="info-wrapper">  
 <div id="details-description">  
 *<!-- fruit category -->* <p>  
 Category: {{ fruit.category }}  
 </p>  
 *<!-- fruit description -->* <p>  
 Description: {{ fruit.description }}  
 </p>  
 *<!-- fruit nutrition -->* <p id="nutrition">Nutrition</p>  
 *<!-- fruit nutrition details -->* <p id="details-nutrition">  
 {{ fruit.nutrition }}  
 </p>  
 </div>  
  
 *<!--Edit and Delete buttons -->* <div id="action-buttons">  
 <a href="{% url 'edit fruit' fruit\_id=fruit.pk %}" id="edit-btn">Edit</a>  
 <a href="{% url 'delete fruit' fruit\_id=fruit.pk %}" id="delete-btn">Delete</a>  
 </div>  
 </div>  
 </div>  
 </section>  
  
{% endblock %}

## Edit Fruit Page

Template file: **"edit-fruit.html"**

On the page, the form must be **filled** with **information** about the **fruit** we want **to edit.** Each field has **a label: "Name:"**, **"Image URL:"**, **"Description:"**, **"Nutrition:",** and **"Category:"**.

When you click on the **"Edit"** button:

* **If** the fruit is **successfully edited**,you should be redirected to the **dashboard** **page**.
* **Otherwise**, the form should show the **appropriate validation errors**.

Картина, която съдържа текст, екранна снимка, дизайн

Описанието е генерирано автоматично

**View**:

In the following code, the **instance** parameter is used to pre-fill the form fields with the data of the specific fruit that is being edited. This is especially useful when you want to show the existing data to the user for reference and make it easier for them to modify only the necessary fields during the editing process.

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

**Template**:

{% extends 'common/base.html' %}  
{% block content %}  
  
 *<!-- Edit Fruit Form -->* <section id="edit">  
 <div class="form">  
 <h2>Edit Fruit</h2>  
 <form class="edit-form" method="post" action="{% url 'edit fruit' fruit\_id=fruit.pk %}">  
  
 {{ form }}  
 <button type="submit">Edit</button>  
 {% csrf\_token %}  
  
 </form>  
 </div>  
 </section>  
  
{% endblock %}

## Delete Fruit Page

Template file: **"delete-fruit.html"**

On the page, the form must be **filled** with the **fruit's information.** Select only the fields with **labels: "Name:"**, **"Image URL:"** and **"Description:"**.

When you click on the **"Delete"** button, the **fruit** is **deleted from the database**, and you should be redirected to the **dashboard page**.

The deleted fruit should be **no longer visible in the app**.

Картина, която съдържа текст, екранна снимка, дизайн

Описанието е генерирано автоматично

**View**:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

**Template**:

{% extends 'common/base.html' %}  
{% block content %}  
  
  
 *<!-- Delete Fruit Form -->* <section id="edit">  
  
 <div class="form">  
 <h2>Delete Fruit</h2>  
 <form class="edit-form" method="post" action="{% url 'delete fruit' fruit\_id=fruit.pk %}">  
  
 {{ form }}  
 <button type="submit">Delete</button>  
 {% csrf\_token %}  
  
 </form>  
 </div>  
  
 </section>  
  
{% endblock %}