**Exercise 2.4: Django Views and Templates - Answers**

**Reflection Questions**

**1. How Django Views Work (with Example)**

Django views are Python functions or classes that handle web requests and return responses. They act as the business logic layer between models (data) and templates (presentation).

**Example:**

python

```
# views.py
from django.shortcuts import render
from django.http import HttpResponse

def book_list(request):
  # Business logic: Get books from database
  books = Book.objects.all()

  # Render template with context data
  return render(request, 'books/list.html', {'books': books})
```

**How it works:**

1. A user visits a URL (e.g., /books/)
2. Django's URL dispatcher matches the URL pattern to the book_list view
3. The view executes business logic (querying the database)
4. The view renders a template with the retrieved data
5. Django returns the HTML response to the user's browser

**2. Function-Based Views vs Class-Based Views for Code Reuse**

For a scenario requiring extensive code reuse across various parts of the project, I would choose **Class-Based Views (CBVs)** because:

- **Inheritance**: CBVs allow creating base view classes with common functionality that can be inherited by multiple views

- **Mixins**: You can create reusable components (mixins) that can be combined in different views
- **Built-in Generic Views**: Django provides CBVs like ListView, DetailView, CreateView that handle common patterns
- **Method Overriding**: Easy to override specific methods (get(), post(), get_context_data()) while keeping the rest of the functionality
- **DRY Principle**: CBVs better adhere to "Don't Repeat Yourself" principle

Example of reusable CBV:

python

```
class LoggedInMixin:
    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated:
            return redirect('login')
        return super().dispatch(request, *args, **kwargs)

class MyView(LoggedInMixin, ListView):
    # This view automatically checks for authentication
    model = MyModel
```

**3. Django Template Language Basics**

**Key Features of Django Template Language:**

1. **Variables**: {{ variable_name }}
    a. Accesses and displays variable values
    b. Example: {{ book.title }}
2. **Tags**: {% tag_name %}
    a. Perform logic and control flow
    b. Examples: {% for %}, {% if %}, {% block %}, {% extends %}
3. **Filters**: {{ variable|filter }}
    a. Modify variable output
    b. Examples: {{ name|lower }}, {{ date|date:"Y-m-d" }}, {{ text|truncatewords:50 }}

4. **Template Inheritance**:

html

```
<!-- base.html -->
{% block content %}{% endblock %}

<!-- child.html -->
{% extends "base.html" %}
{% block content %}Child content{% endblock %}
```

5. **Includes**: {% include "template_name.html" %}
    a. Reuse template components
6. **Comments**: {# comment #} or {% comment %}multiline{% endcomment %}
7. **Auto-escaping**: Automatically escapes HTML for security
8. **URL Handling**: {% url 'view_name' %} for reverse URL lookups

**Task Implementation Summary**

I have successfully completed the task by creating a custom welcome page for my recipe application. Here's what I implemented:

**Files Created/Modified:**

1. recipes/views.py:

python

```
from django.shortcuts import render

def recipes_home(request):
   return render(request, 'recipes/recipes_home.html')
```

2. **Template Structure**:

text

```
recipes/
├── templates/
│   └── recipes/
│       └── recipes_home.html
```

3. recipes/urls.py (created):

python

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

app_name = 'recipes'

urlpatterns = [
    path('', recipes_home, name='home'),
]
```

4. **Project** urls.py (updated):

python

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('recipes.urls')),
]
```

**Custom Welcome Page Features:**

- Clean, visually appealing design
- Welcome message and brief description
- Navigation-ready structure for future features
- Responsive HTML layout

The application now displays my custom welcome page instead of the default Django page when accessing http://127.0.0.1:8000/.

**GitHub Links:**

- Exercise 2.4 Folder: [GitHub Link]
- Recipe App Repository: [GitHub Link]

The implementation follows Django best practices with proper separation of concerns, template organization, and URL routing.