Moneyworth Web App Project Report

Kazibwe David Nelson (2400724054)
Sseruwagi Don Marvin (2400724775)
Mila Samantha Likiya (2400724201)
Mayanja Joel Stephen (2400724184)

May 15, 2025

1 Problem Statement

The current lack of an integrated sales and inventory management system has led to several operational inefficiencies. Managers are spending up to 6 hours per week on manual processes such as tracking sales, managing stock levels, and following up with customers. This manual approach has negatively affected customer satisfaction, due to delayed responses and frequent stock issues. Additionally, poor inventory visibility has contributed to an approximate 10% increase in operational costs, driven by overstocking, stockouts, and mismanaged resources.**Moneyworth** is the django wep app presented for filling the gaps and it has the following features:

1.1 Home

The Home page features a welcoming jumbotron section with summary cards. Each card includes a call-to-action button linking to either the "Create Sale" or "Browse Sales" page. This design helps users quickly access the main functions of the app.

Figure 1: The Home page of the application, featuring navigation and summary cards.

1.2 Dashboard

The Dashboard page displays summary cards that provide an overview of sales performance:

- **Total sales:** The total number of sales recorded in the system.
- **Active sales:** The number of sales that are fully paid and currently in progress.
- **Pending sales:** The number of sales with partial payments.
- **Completed sales:** The number of sales that are fully paid and completed.

The Dashboard also includes a table of recent sales and call-to-action (CTA) buttons that link to the "Create Sale" and "Browse Sales" pages for quick access.

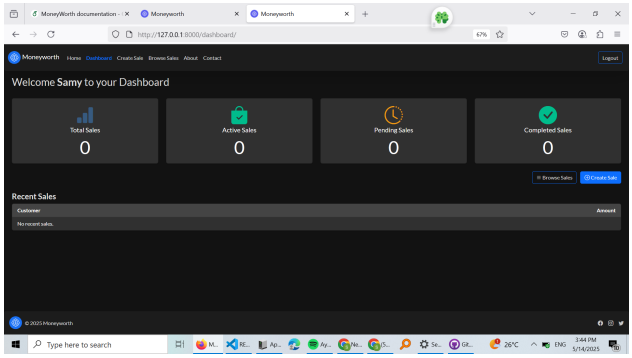

Figure 2: The Dashboard page displaying user statistics and performance charts.

1.3 Browse Sales

The Browse Sales page presents a table of all sales records, each identified by a unique ID. It also provides a search bar for filtering results and a CTA button that directs the user to the Create Sale page. This page allows users to quickly find and review existing sales entries.

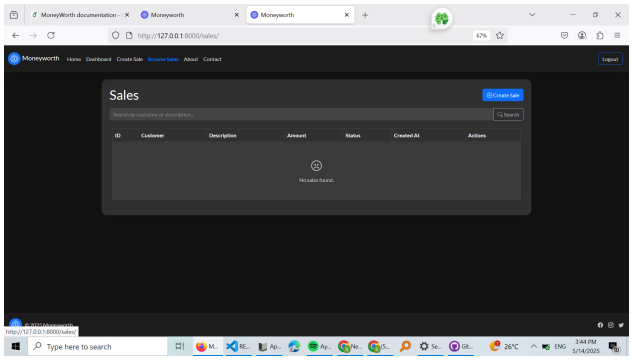

Figure 3: The Browse Sales page showing a table of all sales records.

1.4 Create Sale

The Create Sale page contains a form with fields for entering details of a new sale, such as the product name, quantity, and price. This page enables users to add new sales records to the system.

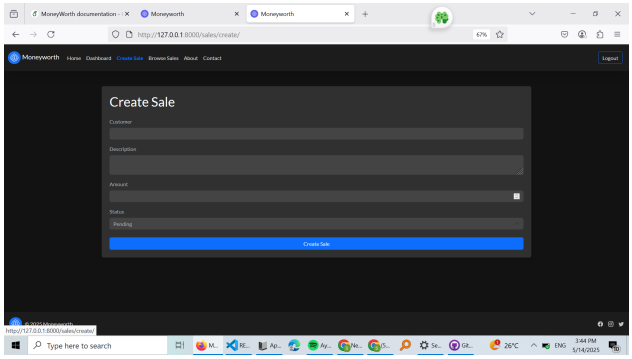

Figure 4: The Create Sale page with a form for adding a new sales record.

1.5 About

The About page provides a brief introduction to the Moneyworth sales app. It highlights the team behind the development of the application and the vision and mission of the project.

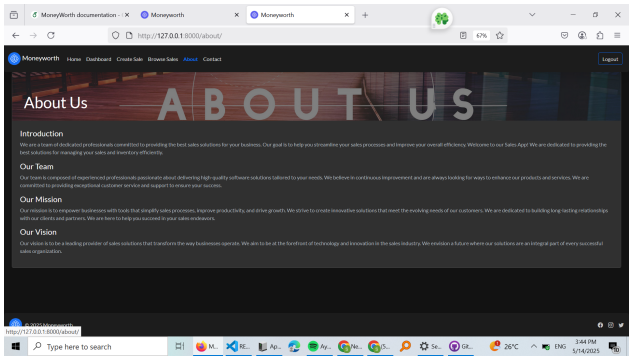

Figure 5: The About page of the application, describing its purpose and development team.

1.6 Contact

The Contact page provides users with a form to reach out to the development team or support staff. It includes fields for the user's name, email, and message, facilitating communication between users and developers.

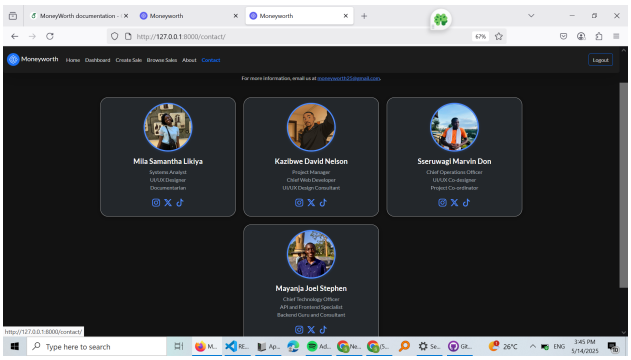

Figure 6: The Contact page offering a form to contact the development team.

2 HTML and Django Template Code Implementation

This section presents the HTML and Django template code used to implement the project's layout.

2.1 Base Template Explanation

The base template, `base.html`, defines the overall structure of the site. It includes a navigation bar, a hero (jumbotron) section on the Home page, and a footer. Content blocks are defined using Django's template tags, allowing child templates to override specific sections. For example:

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Django Sales App{% endblock %}</title>
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/bootstrap-icons.min.css' %}">
    {% block extra_head %}{% endblock %}
</head>
<body>
    <nav>...navbar code...</nav>
    {% block content %}{% endblock %}
    <footer>...footer code...</footer>
    <script src="{% static 'js/bootstrap.bundle.min.js' %}"></script>
</body>
</html>
```

2.2 Child Template Example

A child template (for instance, `home.html`) extends the base template and fills in the designated content blocks:

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

{% block title %}Home - Django Sales App{% endblock %}

{% block content %}
<div class="jumbotron">
    <h1>Welcome to the Django Sales App!</h1>
    <p>Browse and manage sales listings easily.</p>
</div>
{% endblock %}
```

2.3 Use of Template Tags

Key Django template tags used in this project include:

- `{% extends %}`: Allows a template to inherit from a base template.
- `{% block %}`: Defines sections of content that child templates can override.
- `{% static %}`: Resolves the URL for static files (CSS, JavaScript, images).

These tags help organize the templates, promote code reuse, and simplify static file management.

2.4 URL and View Structure

The `urls.py` and `views.py` files connect URLs to view functions and templates. For example:

```
# core/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('dashboard/', views.dashboard, name='dashboard'),
    path('sales/', views.browse_sales, name='browse_sales'),
    path('sales/create/', views.create_sale, name='create_sale'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]

# core/views.py
from django.shortcuts import render

def home(request):
    return render(request, 'core/home.html')

# ...other view functions...
```

3 Benefits of Django for Responsive Web Apps

Django is ideal for responsive web apps due to its clear separation of logic and presentation (MTV architecture), easy integration with Bootstrap for mobile-friendly design, and built-in support for security, scalability, and fast development.

4 Linking Static CSS and JavaScript in Django

To include static files (CSS and JavaScript) in Django:

1. Set `STATIC_URL` in `settings.py` (e.g., `STATIC_URL = '/static/'`).
2. Add `{% load static %}` at the top of each template.
3. Reference static files using `{% static 'path/to/file' %}` (for example, `{% static 'css/bootstra`

For example:

```
{% load static %}
<link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
```

5 Components of MTV Architecture in Django

Django follows the Model-Template-View (MTV) architecture:

- **Model:** Defines the data structure. For example, the `Sales` model in `core/models.py` manages the sales data.
- **View:** Handles requests and returns responses. In this project, functions in `core/views.py` render templates for each page based on the request.
- **Template:** Renders the HTML presentation. Template files (e.g., in `templates/core/`) define what the user sees in the browser.