

National College of Ireland

BSHC4

Software Development

2021/2022

Sorin Nechifor

x18393913

x18393913@student.ncirl.ie

phoneplex – Mobile Phone E-Commerce
Website

Technical Report

Contents

Executive Summary	2
1.0 Introduction	2
1.1. Background	2
1.2. Aims	3
1.3. Technology	3
2.0 System	4
2.1. Requirements	4
2.1.1. Functional Requirements	4
2.1.1.1. Requirement 1: User Register	4
Description & Priority	4
Use Case	4
2.1.1.2. Requirement 2: User Login	6
Description & Priority	6
Use Case	6
2.1.1.3. Requirement 3: User Cart	9
Description & Priority	9
Use Case	9
2.1.1.4. Requirement 4: User Checkout	11
Description & Priority	11
Use Case	11
2.1.2. Non-Functional Requirements	14
2.2. Design & Architecture	14
UML Diagram	14
Packages	17
2.3 Implementation	18
2.4 Graphical User Interface (GUI)	44
2.5 Testing	48
2.6 Evaluation	57
3.0 Conclusion	59
4.0 Further Development or Research	59
5.0 References	60
6.0 Appendices	61
6.1 Project Proposal	61
6.1.1 Objectives	64
6.1.2 Background	64

6.1.3 State of the Art	64
6.1.4 Technical Approach	64
6.1.5 Technical Details	65
6.1.6 Special Resources Required	65
6.1.7 Project Plan	66
6.1.8 Testing	67
6.2 Reflective Journals	67
October Monthly Report	67
November Monthly Report	67
December Monthly Report	68
January Monthly Report	68
February Monthly Report	68
March Monthly Report	68
April Monthly Report	69

Executive Summary

This technical report from phoneplex outlines the premise of the E-Commerce website that sells mobile phones to customers with a basic artificial intelligence that recommends phones to users based on viewing history. Django is the web framework of choice for the application due to the rich features that are packaged within, as well as other libraries like Bootstrap and jQuery that help deal with the styling and manipulation of HTML elements in the DOM, sklearn for the K Nearest Neighbour algorithm and Selenium for testing. The advanced mobile store project idea originated from the second template in the ‘Undergraduate Project Ideas’ as the project had to be changed a week before the mid-point implementation due to technical issues.

The functionality currently incorporated into the application is a login page, a register page, adding items to cart, removing items from a cart, checkout validation, email scheduling, recommending phones based on viewing history and payment handling. Other screenshots of code snippets dealing with Models and powerful HTML templates have been provided as well as key GUI information. An extensive testing suite covers the majority of the application. The project is a success as all key requirements have been met and are functional. The response times of the application is also relatively speedy, ensuring that users have a smooth browsing experience.

1.0 Introduction

1.1. Background

With a big interest in the field of artificial intelligence and after viewing the many templates for project ideas, the idea to build an E-Commerce website with giving

recommendations based on user viewing and purchases was deemed the perfect project to undertake given my interest.

The company that I had done an internship with already used Django for their product and already having experience with Django, I believed that I could use my previous experience to use and further improve my knowledge with the framework as well as other web development skills. Web development is an area which I greatly want to improve on and by developing a web application that is an E-Commerce website would hopefully develop my skills in both front-end and back-end web development.

The template for this specific project was selected from the undergraduate faculty project proposals document, outlined in 'PROJECT 2 – Advanced Mobile Store'.

1.2. Aims

The project aims to build a new E-Commerce website that sells mobile phones and accessories for those phones, using an Agile methodology when developing the software. Deliverables should be attempted to be met every week as outlined in the Gantt Chart created. Users will begin to see recommendations for mobile phones and accessories based on their viewing history of the products available on the website and their purchasing history with the use of basic artificial intelligence. The website will allow for a user to add products into their cart and purchase those items with a checkout process. User activity can be logged through an admin account on the website which will track the items they viewed and when they've logged in or out. After a successful purchase, the user will receive an e-mail of the transaction that had just occurred.

1.3. Technology

HTML and CSS will be used for developing the appearance of the website as is the standard for any web application. Bootstrap, a CSS and JavaScript library, will help further streamline the front-end development of the website with pre-built HTML and CSS templates to choose from and help to organise the web pages in a powerful grid system (Bootstrap, 2022). The use of another JavaScript library, jQuery, is also being used for visually pleasing animations and allowing the web application to be dynamic with its handling of HTML elements (OpenJS Foundation, 2022). Django will be the web framework of choice, which will allow the use of Python as a programming language to handle the back-end functionality of the website which deals with the data a user can interact with and handling server requests and responses, as well as the basic artificial intelligence that will deal with phone recommendations based on user activity (Django Software Foundation, 2022). GitHub will be used as a means of version control of the project (GitHub Inc., 2022). Selenium will also be used to help with system testing and validating that all components of the application work seamlessly together (Muthukadan, 2022).

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

- User may register a new account.
- User may login using an existing account.
- User may add items to a cart.
- User may update item quantities in their cart.
- User may remove items from their cart.
- User can proceed to a checkout page from their cart.

Other functional requirements:

- User sees phone recommendations based on past viewing and purchasing history.
- User can enter all personal details and credit card information to confirm an order of items.
- User receives an email for a successful purchase.

2.1.1.1. Requirement 1: User Register

Description & Priority

A user must register a new account with the application in order to add desired products to a cart and go to the checkout. This is essential for the application and a high priority.

Use Case

Scope

The scope of this use case is for a user to register a new account with phoneplex.

Description

This use case describes the process of a user registering a new account and the possible exceptions that could come up if they enter invalid data during account creation.

Use Case Diagram

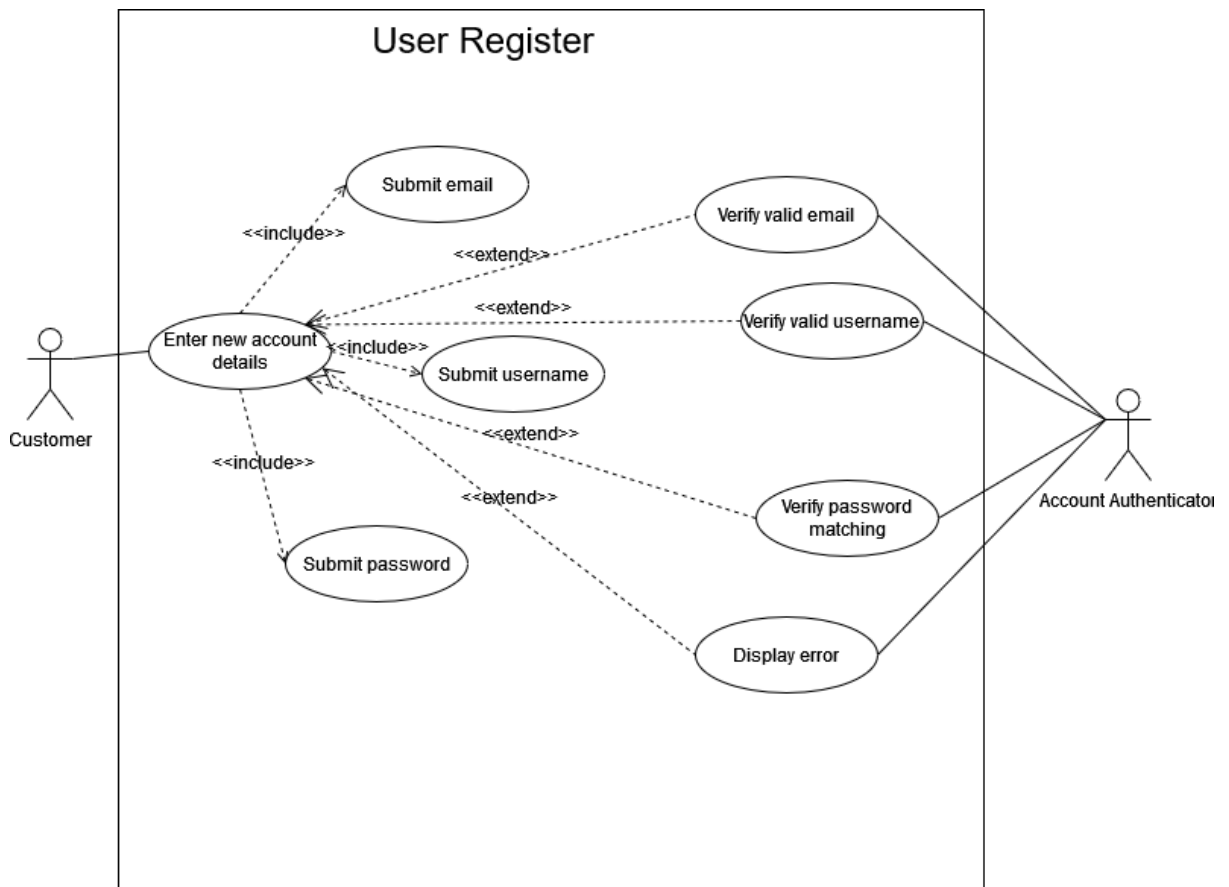


Figure 1 - User Register Use Case

Flow Description

Precondition

N/A

Activation

This use case starts when an unregistered user is on the sign up page of the web application.

Main flow

1. The unregistered user supplies an email.
2. The unregistered user a username.
3. The unregistered user supplies a password.
4. The unregistered user supplies the same password a second time.
5. The system verifies the user's email. [R1: Email is invalid]
6. The system verifies the user's username. [R2: Username is invalid]
7. The system verifies the user's passwords entered. [R3: Passwords entered do not match]
8. The system creates a new account based on the details submitted.

9. The system redirects the user to the login page.

Recovery Flow

R1 : Email is invalid

1. The system refreshes the register page.
2. The unregistered user is notified that the email they entered is invalid.
3. Use case begins at step 1 again.

R2 : Username is invalid

1. The system refreshes the register page.
2. The unregistered user is notified that the username they entered is invalid.
3. Use case begins at step 1 again.

R3 : Passwords entered do not match

1. The system refreshes the register page.
2. The unregistered user is notified that the passwords they entered do not match.
3. Use case begins at step 1 again.

Termination

The system creates a new account for the unregistered user.

Post condition

The system redirects to the login page.

A new account is created based on the details supplied by the user.

2.1.1.2. Requirement 2: User Login

Description & Priority

A user logging into the phoneplex web application. This is essential for the application and a high priority.

Use Case

Scope

The scope of this use case is for a user to login with a pre-existing account with phoneplex.

Description

This use case describes the process of a user registering a new account and the possible exceptions that could come up if they enter invalid data during account creation.

Use Case Diagram

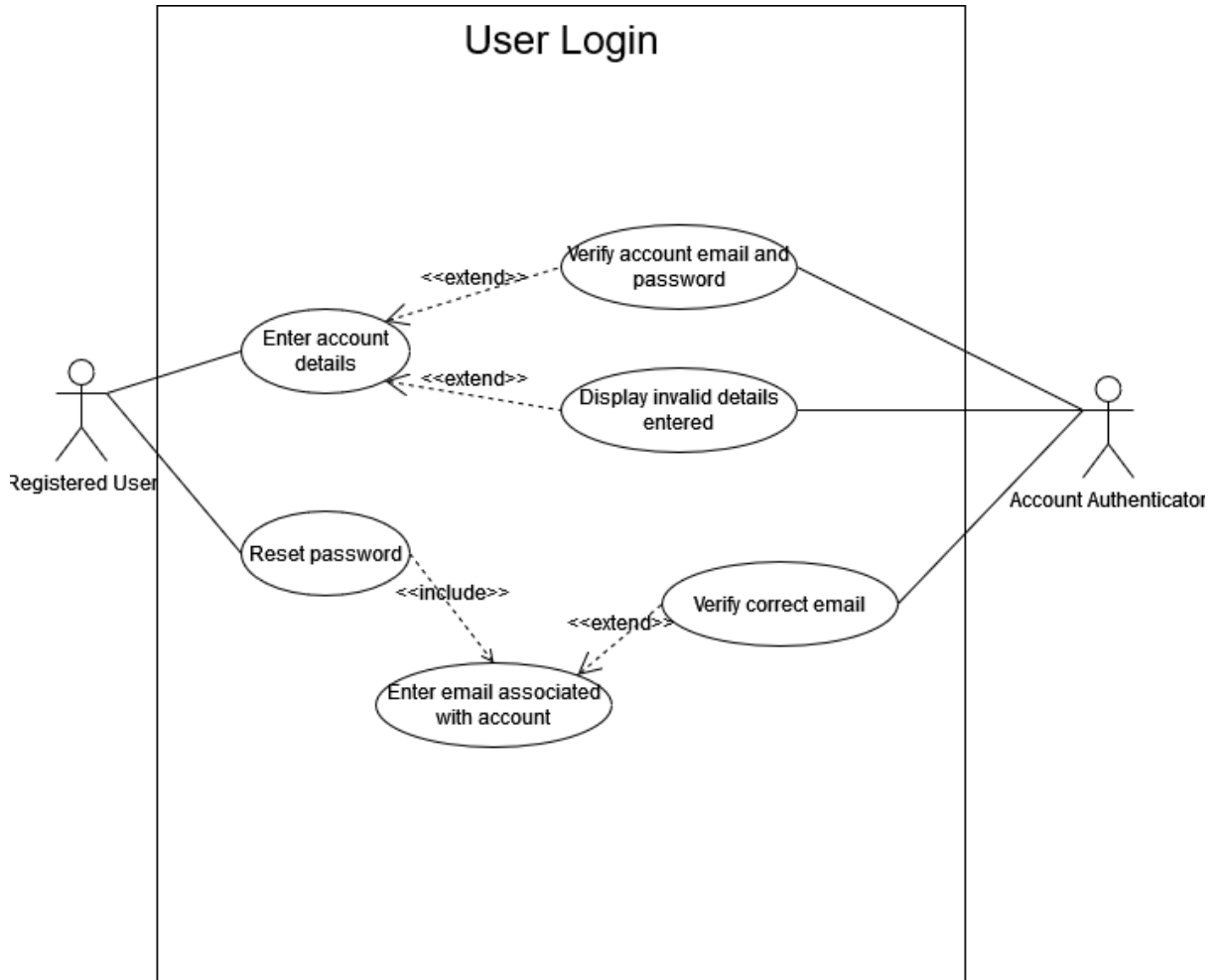


Figure 2 - User Login Use Case

Flow Description

Precondition

User must have an account created with phoneplex.
User is not already logged in.

Activation

This use case starts when a registered user is on the login page of the application.

Main flow

1. The registered user supplies a username.
2. The registered user supplies a password.
3. The system verifies the user's username. [R2: Username is invalid]
4. The system verifies the user's passwords entered. [R3: Passwords entered do not match, E1: User is locked out for a temporary amount of time]
5. The user presses the sign in button.
6. The system redirects the user to the home page.

Recovery Flow

R1 : Username is invalid

1. The system refreshes the login page.
2. The registered user is notified that the username they entered is invalid.
3. Use case begins at step 1 again.

R2 : Password is invalid

1. The system refreshes the login page.
2. The registered user is notified that the password they entered is invalid.
3. Use case begins at step 1 again.

Exceptional flow

E1 : User is locked out for a temporary amount of time

1. The system refreshes the login page.
2. The registered user is notified that their account has been locked for a temporary amount of time due to numerous unsuccessful attempts to login. invalid.
3. Use case ends in failure.

Termination

The system redirects the logged in user to the home page.

Post condition

The system redirects to the home page.

The registered user is logged in and authenticated.

2.1.1.3. Requirement 3: User Cart

Description & Priority

A user is interacting with the cart system on the web application. This is essential for the application and is a priority.

Use Case

Scope

The scope of this use case is for a user to add items to their cart, remove items and update item quantities in the cart.

Description

This use case describes the process of how a user adds and removes products from their carts during their use of the website.

Use Case Diagram

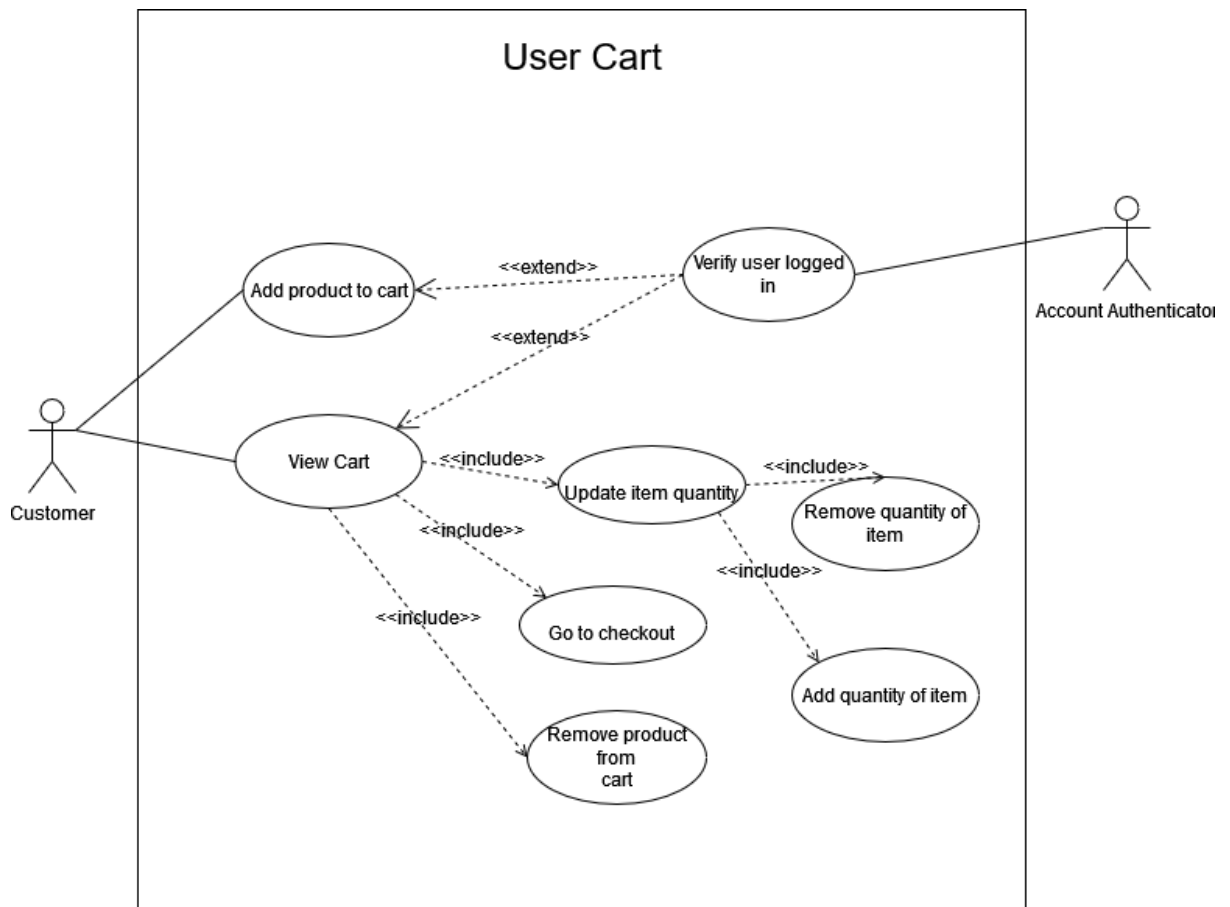


Figure 3 - User Cart Use Case

Flow Description

Precondition

User must have an account created with phoneplex.

User must be logged in.

Activation

This use case starts when a registered user is on any given product's information page.

Main flow

1. The registered user clicks the 'Add to Cart' button.
2. The system notifies the user that the product has been added to cart.
3. The registered user navigates to the cart page.
4. The system displays the cart to the user with the items added, their respective prices and total price altogether. [O1: User can add quantities of a product in their cart, O2: User can remove quantities of a product in their cart, O3: User can remove an item entirely from the cart]
5. The use case is successful.

Option flow

O1 : User can add quantities of a product in their cart

1. The user clicks the plus symbol in the item quantity column of the order table.
2. The system adds 1 additional quantity of the item to the user's order.
3. The use case continues at step 4.

O2 : User can remove quantities of a product in their cart

1. The user clicks the minus symbol in the item quantity column of the order table.
2. The system removes 1 quantity of the item to the user's order.
3. The use case continues at step 4.

O3 : : User can remove an item entirely from the cart The system refreshes the register page.

1. The user clicks the trash can symbol in the last column of the order table.
2. The system removes the item entirely from the user's order.
3. The use case continues at step 4.

Termination

The system adds an item to the cart, removes an item from the cart or updates the quantity of an item in the cart.

Post condition

The user's cart has an item in it, no longer has an item in it or the quantities of an item in a user's order has been updated.

2.1.1.4. Requirement 4: User Checkout

Description & Priority

A user is on the checkout page of the website and seeks to confirm their order. This is essential for the application and is a priority.

Use Case

Scope

The scope of this use case is for a user to enter their personal details including payment information and successfully purchase whatever their order entails.

Description

This use case describes the process of how a user fills out their personal information including payment option as well as any promotion codes to be applied. Afterwards, the system should display whether or not their submitted details are valid and if their purchase has been successful or not.

Use Case Diagram

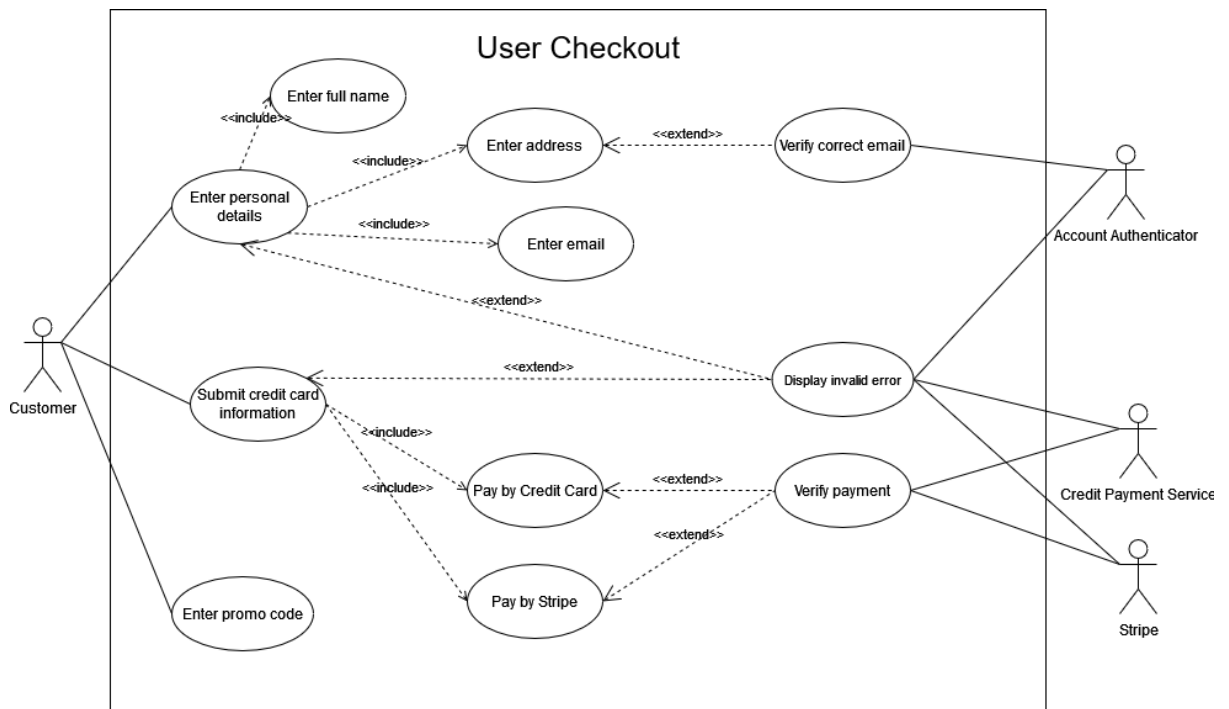


Figure 4 - User Checkout Use Case

Flow Description

Precondition

User must have an account created with phoneplex.

User must be logged in.

User must have at least one item inside their cart.

User is on the checkout page.

Activation

This use case starts when a registered user is at the checkout page of the website.

Main flow

1. The registered user enters their full name.
2. The registered user enters their email. [R1: User did not enter a valid email address]
3. The registered user enters their address.
4. The registered user enters their credit card information [A1: User pays with a third party application like Stripe, R1: User Credit card information is incorrect, R2: User's credit payment service denies authorization]
5. The registered user clicks the 'Confirm order' button. [O1: User wants to enter a promotion code for a discount]

6. The user's credit payment service validates the transaction.
7. The credit payment service notifies the phoneplex system that the transaction was deemed valid.
8. The system confirms and notifies the user of their order.
9. The system redirects the user to the home page.
10. The use case is successful.
- 11.

Recovery flow

R1 : User Credit card information is incorrect.

1. The system notifies the registered user that the credit card information they submitted is incorrect.
2. The use case begins again at step 4.

R2 : User's credit payment service denies authorization.

1. The credit payment system notifies the phoneplex system that the transaction was not authorized due to an error.
2. The phoneplex system notifies the user that the transaction was not authorized by the credit payment service.
3. The use case begins again at step 4.

Alternate flow

A1 : User pays with a third party application like Stripe

1. Registered user enters valid account details for Stripe.
2. Stripe's system authenticates the submitted details.
3. The credit payment service of the registered user validates the transaction.
4. Stripe notifies the phoneplex web application that the transaction is deemed valid.
5. The use case continues from step 8.

Option flow

O1 : User wants to enter a promotion code for a discount

1. The registered user enters a valid promotion code into the promotion code field.
2. The system applies a discount to the total price of the order.
3. The use case continues from step 5.

Termination

The system adds an item to the cart, removes an item from the cart or updates the quantity of an item in the cart.

Post condition

The registered user has a new order added to their purchase history.

2.1.2. Non-Functional Requirements

- Admin users can view regular users' viewing history, purchase history, login activity and logout activity.
- Users should be informed of what data is being monitored and collected in the application.
- The application should be easy to navigate with a well-made graphical user interface.
- Users must make an account first in order to add items to a cart and purchase them.
- Back-end and front-end testing should be part of the testing suite.
- Performance of the application should be fast, with the user not waiting a large amount of time for a page to load or process to finish.
- Data should be stored in a secure manner where activity will not be access by unauthorized users.
-

2.2. Design & Architecture

UML Diagram

Since this application is using Django as the web framework, by entering the command 'python manage.py graph_models --pydot -a -g -o my_project_visualized.png' into a terminal, Django will automatically generate the entire architecture of the project into a PNG file. The image generated is far too large for this document due to the other packages installed in Django that have not been utilised for this project. Only the relevant architecture will be described.

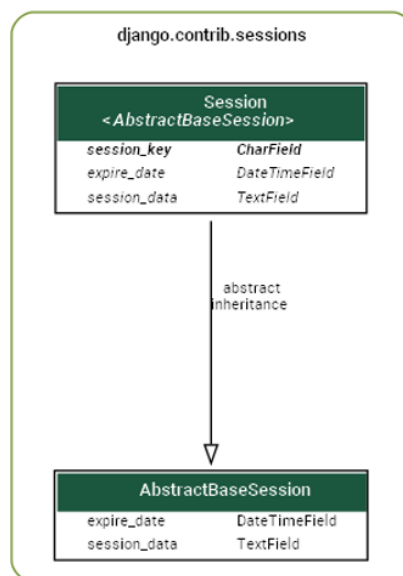
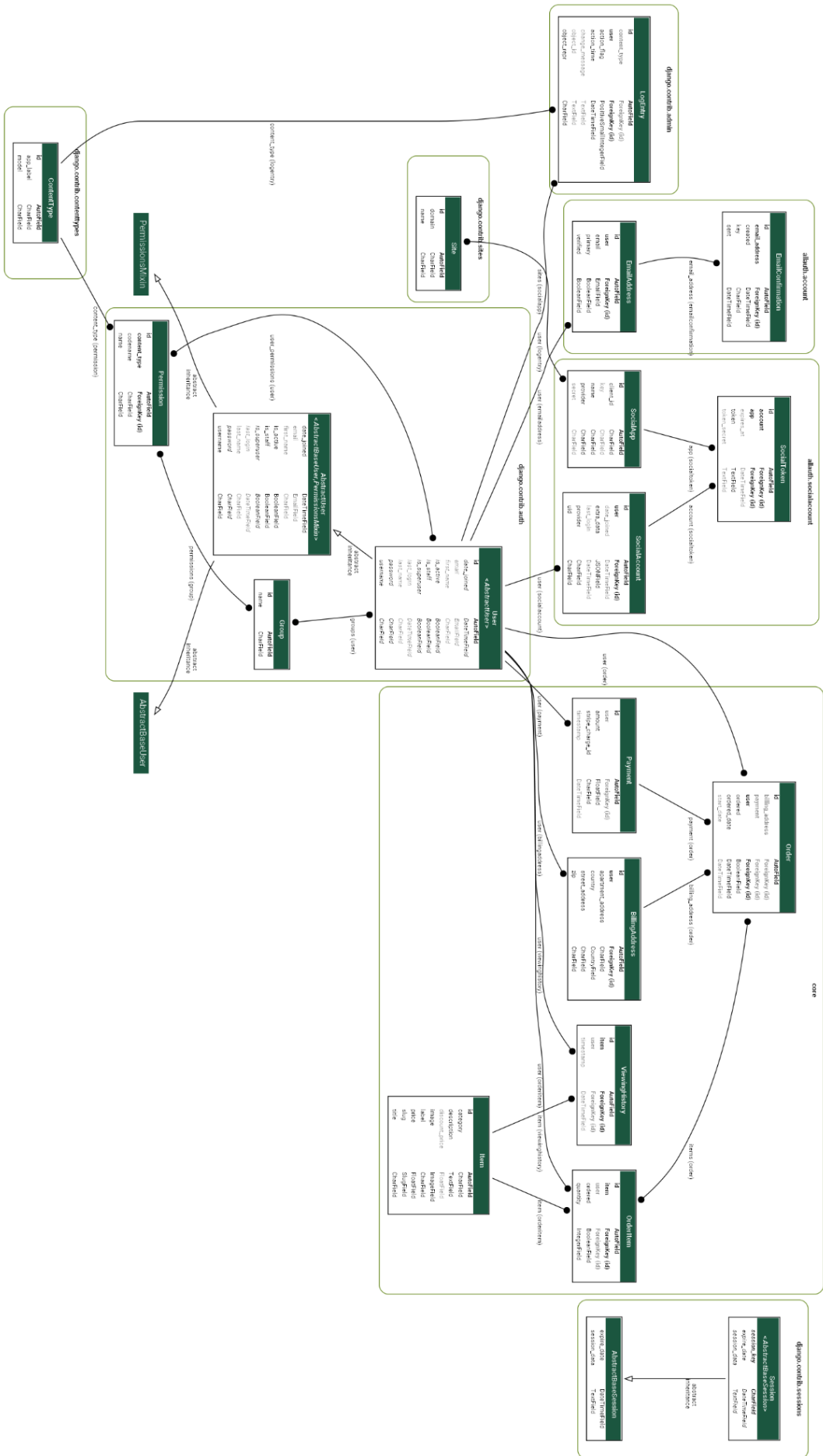


Figure 5 - `django.contrib.sessions` package diagram

Each user of the website creates session data as dictated in the sessions Model. This session cookie may expire after a certain period of time but no limit has been imposed on the application so far.



Packages

Django.contrib.contenttypes and Django.contrib.auth – Permission Model:

Django ContentTypes stores the information about the models in any given project (Models being Django’s method of handling data). Permissions are also closely tied into these models so that if your application required that only a certain user can access specific data, then there would be some authorization that would have to occur.

Django.contrib.auth – User Model:

Another generated object by the Django framework which helps with creating user objects, used for authentication in your web application. The permission model also present in the same package sets the permissions that these users have upon creation. Regular users like customers would have no staff or admin permissions. The attributes associated with a default user object are email, first_name, last_name, password and their ID - which is a primary key.

Django.contrib.auth – AbstractUser Model:

This model is specifically imported and not available by default when a Django project is created. The abstract user model contains useful a useful function for the application which is ‘PermissionsMixin’. Incorporating PermissionMixins into a Django web application for the Views (Resources) in the views.py file will enforce users to be logged into the website if requesting a specific resource that is using PermissionMixins as a parameter.

Django.contrib.auth – Group Model:

Allows for the setting of specific permissions for a group of users. This functionality isn’t essential for the web application but is bundled into a Django project upon initialization.

Django.contrib.admin – LogEntry:

Package generated by Django which logs when a new user makes changes to the database e.g. adding or deleting objects like items from their order. These logs are available to admins/superusers by logging in at the /admin resource when running the server.

allauth.account:

An external package that must be installed with the package installer for python (pip). Unlike using authorization by signing in with social accounts such as Google or Facebook, this package allows Django developers to use local authentication by creating and storing accounts within the Django application itself. For this specific application, users can sign up with an e-mail and also potentially have their account verified. Verification hasn’t been implemented as of yet, but remains a possibility without too much hassle to implement.

core:

The core package contains custom Models created during development of this web application. The models that have been created are Item, OrderItem, Order, Payment, BillingAddress and ViewingHistory.

Item objects created are listed on the home page of phoneplex and have a multitude of attributes associated with them:

ID: Generated ID of the item object itself in the database.

Category: Android, iPhone, Cover or Accessory

Description: Product description

Discount Price (Optional): New price of the phone that overrides the price attribute

Image: Location of the image file to be used when displaying the product

Label: New, Sold out or Hot

Price: Original price of the phone

Slug: Specific resource pathing e.g. product/iphone-7

Title: Name of the phone

OrderItem objects are Items that have been added to a user's cart and have not yet been ordered yet. Order items inherit all of the Item's previous attributes but also add a quantity attribute and an ordered attribute, which signifies if the item has been bought yet.

The Order model inherit OrderItem objects and have an additional attribute of ordered_date that notifies the system of when the order was successfully completed. These Order objects serve the role of displaying to the user what their order exactly entails before and after purchasing.

The BillingAddress model contains a user field, storing an object that directly correlates to the authenticated user of which the order belongs to. The street address, apartment address, country and zip code make up the other fields in the address.

The Payment model contains an id for the stripe charge transaction that would be made available after a successful purchase. The model also contains a field for the user object, the amount that was paid and the timestamp at which the payment was successful.

The ViewingHistory model contains a field for the user object of the currently logged in user, the item that was just viewed by the user and the timestamp at which the product was viewed at.

2.3 Implementation

The technology stack of the web application mainly consists of the Django web framework, Bootstrap, jQuery, Selenium, scikit-learn, GitHub and Stripe. Django was chosen as the web framework to base the application around as Python is a powerful language to use with minimal syntax and very robust functionality built-into it. Django also provided an admin interface which would automatically meet some of the requirements that was necessary to incorporate into the application. Bootstrap helped with the styling of the website and was chosen for its easy to use grid system to lay out certain html elements neatly. jQuery was used for the fade-in animations that can sometimes be seen in the website. Selenium would be the module used for creating the system tests of the application I had experience with it before during my internship and current work. GitHub is a version control I was most familiar with and the GitHub desktop application allowed for easy pushing and pulling of changes without writing commands. scikit-learn was used to import the K Nearest Neighbour algorithm for the recommended phone system.

The full list of packages/dependencies can be found in the requirements.txt file in the root of the folder on GitHub.

```

{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>{% block page_title %}{% endblock %}</title>
  {% block extra_head %}
  {% endblock %}
  <!-- Font Awesome -->
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.11.2/css/all.css">
  <!-- Bootstrap core CSS -->
  <link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet">
  <!-- Material Design Bootstrap -->
  <link href="{% static 'css/mdb.min.css' %}" rel="stylesheet">
  <!-- Custom stylesheet -->
  <link href="{% static 'css/style.css' %}" rel="stylesheet">
</head>
<body>
  {% if messages %}
    {% for message in messages %}
      <div style="margin-top: 5rem !important" class="alert alert-{{ message.tags }} alert-dismissible fade show" role="alert">
        {{ message }}
        <button type="button" class="close" data-dismiss="alert" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
    {% endfor %}
  {% endif %}

  {% include "navbar.html" %}

  {% block content %}
  {% endblock content %}

  {% block extra_body %}
  {% endblock extra_body %}

  {% include "footer.html" %}
  <!-- SCRIPTS -->

```

Figure 7 - templates/base.html

With the use of Django’s template functionality, HTML files can be used and created efficiently without the use of replicated code. This base.html file is the base file that all other html files inherit from; inheriting the scripts, navbar HTML elements, system messages to the user, CSS styling and other characteristics.

The main body of the other HTML pages are included in Djangos ‘block’ functionality, and one such block which takes the name ‘content’ is where the other HTML elements associated with specific pages in their body are declared.

```

{% extends "base.html" %}
{% load static %}
{% block page_title %}Phones at Bargain Prices 24/7! | phoneplex{% endblock %}
{% block content %}
<!--Main layout-->
<main>
  <div class="container">

    <!--Category Navbar-->
    <nav class="navbar navbar-expand-lg navbar-dark mdb-color lighten-3 mt-3 mb-5" id="category-navbar">

      <!-- Navbar brand -->
      <span class="navbar-brand">Categories:</span>

      <!-- Collapse button -->
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#basicExampleNav"
        aria-controls="basicExampleNav" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
    </div>
  </main>
</block content %}

```

Figure 8 - templates/home-page.html

Looking at a snippet of the home page HTML file which inherits from the base (defined at the top with `{% extends "base.html" %}`), we can see that inside the block content tag, other HTML elements associated with the home page HTML file will be displayed to the user.

```

{% else %}
<p>{% blocktrans %}If you have not created an account yet, then please
<a href="{{ signup_url }}">sign up</a> first.{% endblocktrans %}</p>
{% endif %}

<form class="login" method="POST" action="{% url 'account_login' %}">
  {% csrf_token %}
  {{ form|crispy }}
  {% if redirect_field_value %}
  <input type="hidden" name="{{ redirect_field_name }}" value="{{ redirect_field_value }}" />
  {% endif %}
  <a class="btn btn-default" href="{% url 'account_reset_password' %}">{% trans "Forgot Password?" %}</a>
  <button class="btn btn-primary" type="submit">{% trans "Sign In" %}</button>
</form>
</div>

```

Figure 9 – allauth Login template

The login page incorporates POST requests that contain user information to login, and Django easily handles the field logic with the module allauth, used for user authentication. URL paths defined in the `urls.py` file in the project folder are used here for redirecting the user after an account login, as seen with the attribute `'action="{% url 'account_login' %}"` in the form tag.

urls.py	urls.py — mobilestore	urls.py — core
<pre> 1 from django.conf import settings 2 from django.conf.urls.static import static 3 from django.contrib import admin 4 from django.urls import path, include 5 6 urlpatterns = [7 path('admin/', admin.site.urls), 8 path('accounts/', include('allauth.urls')), 9 path('', include('core.urls', namespace='core')), 10 path('product/', include('core.urls', namespace='core')), 11 path('checkout/', include('core.urls', namespace='core')), 12 path('recommended/', include('core.urls', namespace='core')) 13] 14 15 if settings.DEBUG: 16 urlpatterns += static(settings.STATIC_URL, 17 document_root=settings.STATIC_ROOT) 18 urlpatterns += static(settings.MEDIA_URL, 19 document_root=settings.MEDIA_ROOT) 20 </pre>	<pre> 1 from django.urls import path, include 2 from django.contrib import admin 3 from .views import (4 HomeView, 5 RecommendedView, 6 OrderSummaryView, 7 ProductDetailView, 8 CheckoutView, 9 PaymentView, 10 add_to_cart, 11 remove_from_cart, 12 remove_single_item_from_cart 13) 14 15 app_name = "core" 16 17 urlpatterns = [18 path('', HomeView.as_view(), name='home-page'), 19 path('product/<slug>', ProductDetailView.as_view(), name='product-page'), 20 path('checkout/', CheckoutView.as_view(), name='checkout-page'), 21 path('recommended/', RecommendedView.as_view(), name='recommended'), 22 path('order-summary/', OrderSummaryView.as_view(), name='order-summary'), 23 path('add-to-cart/<slug>', add_to_cart, name='add-to-cart'), 24 path('remove-from-cart/<slug>', remove_from_cart, 25 name='remove-from-cart'), 26 path('remove-item-from-cart/<slug>', remove_single_item_from_cart, 27 name='remove-single-item-from-cart'), 28 path('payment/<payment_option>', PaymentView.as_view(), name='payment') 29] 30 </pre>	

Figure 10 - core/urls.py & mobilestore/urls.py



Figure 11 - products/<slug>

As seen in both the project's urls.py file and the local 'application' urls.py to the right, resources are easy to access and read, without any .html being listed in the address of the website. Slugs are used for the different products available to be purchased.

```

{% extends "base.html" %}
{% load i18n %}
{% load account socialaccount %}
{% load crispy_forms_tags %}

{% block page_title %}{% trans "Sign Up | phoneplex" %}{% endblock %}

{% block content %}
<!--Main layout-->
<main>
  <div class="container">
    <!--Section: Products v.3-->
    <section class="text-center mb-4">
      <!--Grid row-->
      <div class="row wow fadeIn">
        <div class="col-lg-3">
        </div>
        <div class="col-lg-6">
          <h1 class="mt-5 pt-5">Sign Up</h1>

          <p>{% blocktrans %}Already have an account? Then please <a href="{{ login_url }}">sign in</a>.{% endblocktrans %}</p>

          <form class="signup" id="signup_form" method="post" action="{% url 'account_signup' %}">
            {% csrf_token %}
            {{ form|crispy}}
            {% if redirect_field_value %}
            <input type="hidden" name="{{ redirect_field_name }}" value="{{ redirect_field_value }}" />
            {% endif %}
            <button class="btn btn-primary" type="submit">{% trans "Sign Up" %} &raquo;</button>
          </form>

        </div>
        <div class="col-lg-3">
        </div>
      </div>
    </section>
    <!--Section: Products v.3 -->
  </div>
</main>
<!--Main layout-->
{% endblock %}

```

Figure 12 - allauth Sign Up page

This snippet encapsulates all the HTML elements for the sign up page, also inheriting the base.html and navbar HTML elements and only adding its own body in the block content tag. Again, the allauth library helps provide a lot of the logic with validating an account being created. The CSRF (cross-site forgery request) token tag is one of Django's many methods of increasing the security of a web application.

```
{% for item in object_list %}
<!--Grid column-->
<div class="col-lg-3 col-md-6 mb-4">

  <!--Card-->
  <div class="card h-100">

    <!--Card image-->
    <div class="view overlay">
      
      <a href="{{ item.get_absolute_url }}">
        <div class="mask rgba-white-slight"></div>
      </a>
    </div>
    <!--Card image-->

    <!--Card content-->
    <div class="card-body text-center">
      <!--Category & Title-->
      <a class="grey-text">
        <h5>{{ item.get_category_display }}</h5>
      </a>
      <h5>
        <strong>
          <a href="{{ item.get_absolute_url }}" class="dark-grey-text">{{ item.title }}
            <span class="badge badge-pill {{ item.get_label_display }}-color">NEW</span>
          </a>
        </strong>
      </h5>

      <h4 class="font-weight-bold blue-text">
        <strong>&euro;
        {% if item.discount_price %}
        {{ item.discount_price }}
        {% else %}
        {{ item.price }}
        {% endif %}
      </strong>
    </h4>
  </div>
</div>
```

Figure 13 - Looping through items (templates/home-page.html)

On the home page HTML file, when displaying product information, Django provides a developer the ability to loop through objects/data passed in when returning a resource to an end user. Since the user has requested the home page and products are listed on the home page, objects created through the Item model of the web application will be looped through and have each of their attributes displayed as values in the appropriate HTML tags; displaying the product's information. This leads to less code repetition and no hard-coding of the variables, which is an efficient way to quickly make changes to the application and save time during development.

```

class Item(models.Model):
    title = models.CharField(max_length=100)
    description = models.TextField()
    price = models.FloatField()
    discount_price = models.FloatField(blank=True, null=True)
    category = models.CharField(choices=CATEGORY_CHOICES, max_length=
    label = models.CharField(choices=LABEL_CHOICES, max_length=1)
    slug = models.SlugField()
    image = models.ImageField()

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse("core:product-page", kwargs={
            'slug': self.slug
        })

    def get_add_to_cart_url(self):
        return reverse("core:add-to-cart", kwargs={
            "slug": self.slug
        })

    def get_remove_from_cart_url(self):
        return reverse("core:remove-from-cart", kwargs={
            "slug": self.slug
        })

class OrderItem(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE, blank=True, null=True)
    ordered = models.BooleanField(default=False)
    item = models.ForeignKey(Item, on_delete=models.CASCADE)
    quantity = models.IntegerField(default=1)

    def __str__(self):
        return f"{self.quantity} of {self.item.title}"

    def get_total_item_price(self):
        return self.quantity * self.item.price

    def get_total_discount_item_price(self):
        return self.quantity * self.item.discount_price

    def get_amount_saved(self):
        return self.get_total_item_price() - self.get_total_discount_item_price()

    def get_final_price(self):
        if self.item.discount_price:
            return self.get_total_discount_item_price()
        return self.get_total_item_price()

class Order(models.Model):

```

Figure 14 - Item and OrderItem Models (core/models.py)

```

class Order(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE)
    items = models.ManyToManyField(OrderItem)
    start_date = models.DateTimeField(auto_now_add=True)
    ordered_date = models.DateTimeField()
    ordered = models.BooleanField(default=False)

    def __str__(self):
        return self.user.username

    def get_total(self):
        total = 0
        for order_item in self.items.all():
            total += order_item.get_final_price()
        return total

```

Figure 15 - Order Model (core/models.py)

Three data models have been created for handling products listed on the site and adding them into a user's order: Item, OrderItem and Order.

Effectively, these classes are creating the model/table in the SQL database that comes with Django with their respective attributes and what data they take e.g. price taking in float values. Coupled with these data attributes, these models can also have methods associated with them such as `get_total` for an Order, which runs a for loop through all the order items in the order and calls their `get_final_price` (created in the OrderItem class) and returns the sum to the application. Other methods created for these models are returning the url for a specific product slug, returning the item name as a string and calculating the amount saved from a discount price.

Django provides functionality for admin/superusers immediately upon installation. An admin page can be accessed by hitting the base url followed by /admin. After logging in as an admin, we can see the created models and add/edit/delete data appropriately.

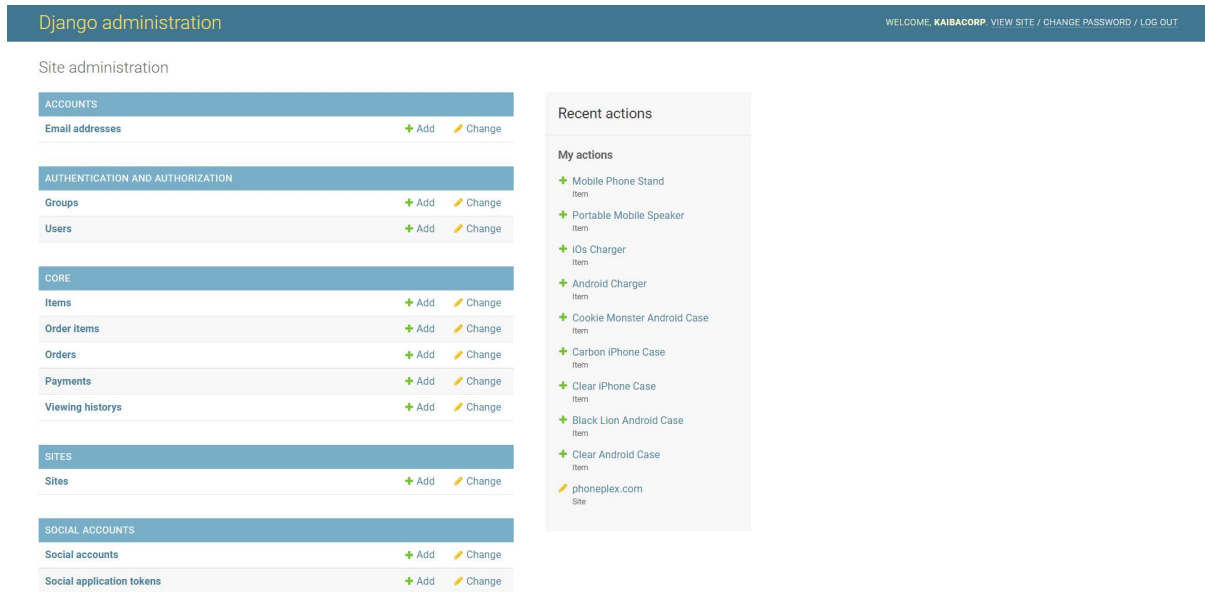


Figure 16 - Django Admin Site Homepage

If we view the Items page on the admin site, we can add items and also edit their attributes after creating them.

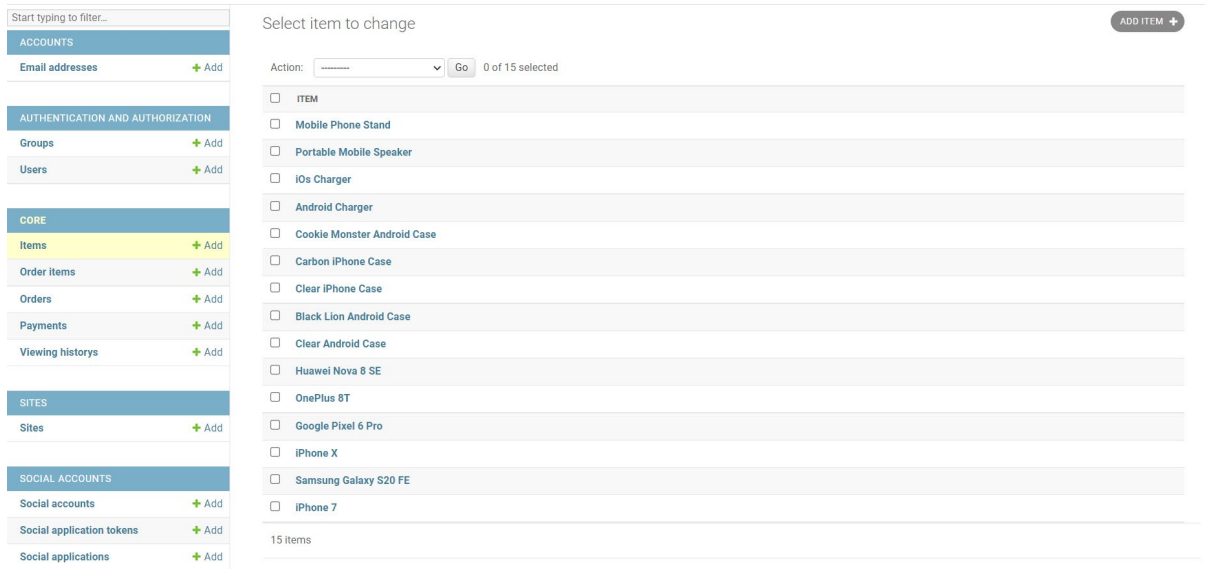


Figure 17 - Django Admin Site Items Page

The screenshot shows the Django Admin interface for editing an item. On the left is a sidebar menu with categories like ACCOUNTS, AUTHENTICATION AND AUTHORIZATION, CORE, SITES, and SOCIAL ACCOUNTS. The 'CORE' section is expanded, and 'Items' is selected. The main area is titled 'Change item' and shows the details for 'Mobile Phone Stand'. The form includes fields for Title, Description, Price, Discount price, Category, Label, Slug, and Image. The 'Image' field shows the current image URL and a 'Choose File' button. At the bottom, there are buttons for 'Delete', 'Save and add another', 'Save and continue editing', and 'SAVE'.

Figure 18 - Django Admin Site Edit Item Page

Three other models have been created for the application which are BillingAddress, Payment and ViewingHistory:

```

class BillingAddress(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    street_address = models.CharField(max_length=100)
    apartment_address = models.CharField(max_length=100)
    country = CountryField(multiple=False)
    zip = models.CharField(max_length=100)

    def __str__(self):
        return self.user.username

class Payment(models.Model):
    stripe_charge_id = models.CharField(max_length=50)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.SET_NULL, blank=True, null=True)
    amount = models.FloatField()
    timestamp = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.user.username

class ViewingHistory(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.SET_NULL, blank=True, null=True)
    item = models.ForeignKey(Item, on_delete=models.CASCADE)
    timestamp = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'{self.item.title}' at {self.timestamp} - {self.user.username}

```

Figure 19 - BillingAddress, Payment and ViewingHistory Models (core/models.py)

BillingAddress inherits information that relate to a user's billing address, so the appropriate fields have been created as such under the BillingAddress class.

Payment stores the Stripe charge ID after a transaction has taken place and also tracks the user, the amount paid and the timestamp of the transaction and stores it into the database.

ViewingHistory is used for creating logs of whenever a user visits a product information page and tracks the user that visited the product, the specific item/product page viewed and the time of when they viewed it.

Each time a logged in user visits a product information page, a new entry is stored to the database using the ViewingHistory model:

```
class ProductDetailView(DetailView):
    model = Item
    template_name = "product-page.html"

    def get_context_data(self, *args, **kwargs):
        context = super(ProductDetailView,
            self).get_context_data(*args, **kwargs)
        if self.request.user.is_authenticated:
            # Creating the view log of the product for the authenticated user
            view_log = ViewingHistory.objects.create(user=self.request.user, item=context['item'])
            # Saving it to the database
            view_log.save()

        return context
```

Figure 20 - View log functionality in ProductDetailView (core/views.py)

These logs can also be viewed by an admin user on the admin site:

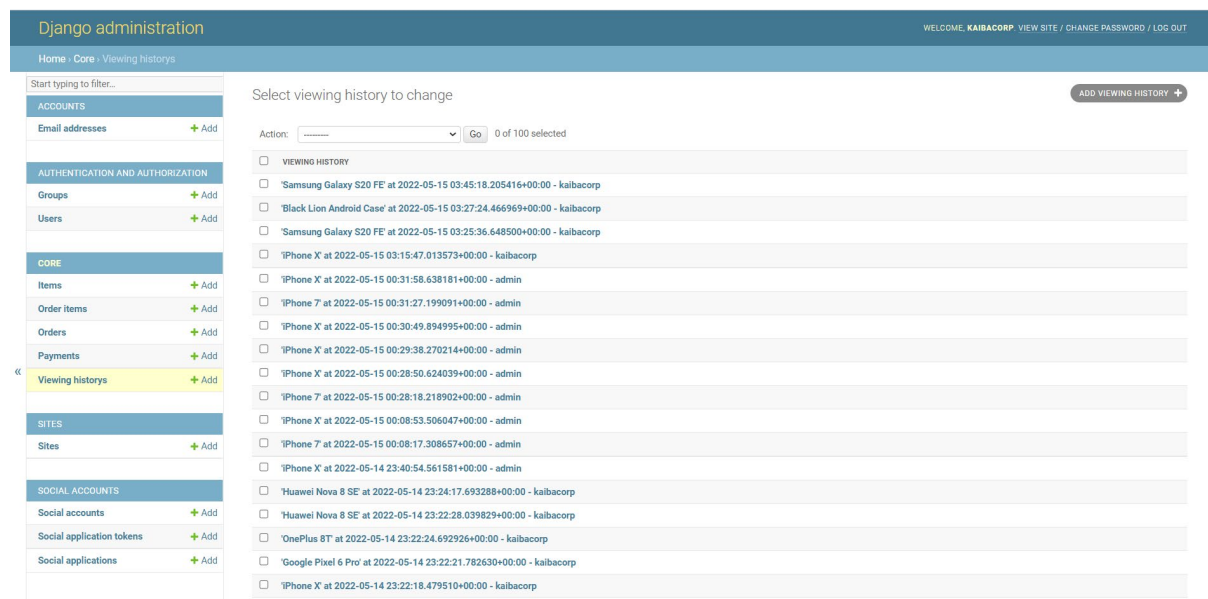


Figure 21 - View logs on Django Admin Site

Each product listed on the website has their own product information page which can be accessed by the slug associated with that product e.g. an iPhone 7 listed on the website would be able to be viewed by visiting the url '/products/iphone-7'. As seen with the django template tags above ({% %}), the product's information is able to be accessed and easily displayed to the user. This is achieved by passing in the specific Item model once the user requests the product information page for it:

```
class ProductDetailView(DetailView):
    model = Item
    template_name = "product-page.html"
```

Figure 22 - Item Model passed into product page template (core/views.py)

Below is the HTML template for the products listed on the website:

```
<!--Grid column-->
<div class="col-md-6 mb-4">

</div>
<!--Grid column-->

<!--Grid column-->
<div class="col-md-6 mb-4">

    <!--Content-->
    <div class="p-4">

        <div class="mb-3">
            <a href="#">
                <span class="badge purple mr-1">{{ object.get_category_display }}</span>
            </a>
        </div>

        <p class="lead">
            {% if object.discount_price %}
            <span class="mr-1">
                <del>&euro;{{ object.price }}</del>
            </span>
            <span>&euro;{{ object.discount_price }}</span>
            {% else %}
            <span>&euro;{{ object.price }}</span>
            {% endif %}
        </p>

        <p class="lead font-weight-bold">Description</p>

        <p>{{ object.description }}</p>

        <a href="{{ object.get_add_to_cart_url }}" class="btn btn-primary btn-md my-0 p">
            Add to cart
            <i class="fas fa-shopping-cart ml-1"></i>
        </a>
        <a href="{{ object.get_remove_from_cart_url }}" class="btn btn-danger btn-md my-0 p">
            Remove from cart
        </a>
    </div>
    <!--Content-->

</div>
<!--Grid column-->
```

Figure 23 - Product Page Information for item (templates/product-page.html)

Cart System:

Similar to the products being listed on the home page, order items that are using the OrderItem model can be looped through and have the items information listed out to the user. This snippet is the order summary/cart HTML file displaying a user's order back to them. Some conditional statements are being used, with djangos if block tags e.g. If the item has a discount price, display that discount price – otherwise display the normal price. As well as that, certain elements won't be loaded if the cart has no items in it as dictated by Djangos {% empty %} tag.

```

{% for order_item in object.items.all %}
<tr>
  <th scope="row">{{ forloop.counter }}</th>
  <td>{{ order_item.item.title }}</td>
  <td>{{ order_item.item.price }}</td>
  <td>
    <a href="{% url 'core:remove-single-item-from-cart' order_item.item.slug %}">
      <i class="fas fa-minus mr-2"></i>
    </a>
    {{ order_item.quantity }}
    <a href="{% url 'core:add-to-cart' order_item.item.slug %}">
      <i class="fas fa-plus ml-2"></i>
    </a>
  </td>
  <td>
    {% if order_item.item.discount_price %}
      &euro;{{ order_item.get_total_discount_item_price }}
      [&euro;{{ order_item.get_amount_saved }} SAVED]
    {% else %}
      &euro;{{ order_item.get_total_item_price }}
    {% endif %}
    <a href="{% url 'core:remove-from-cart' order_item.item.slug %}">
      <i class="fas fa-trash float-right"></i>
    </a>
  </td>
</tr>
{% empty %}
<tr>
  <td colspan="5">Your cart is empty.</td>
  <td colspan="5">
    <a class="btn btn-primary float-right " href="/">Continue shopping</a>
  </td>
</tr>
{% endfor %}
{% if object.get_total %}
<tr>
  <td colspan="4">Order Total</td>
  <td>&euro;{{ object.get_total }}</td>
</tr>
<tr>
  <td colspan="5">
    <a class="btn btn-warning float-right ml-2" href="/checkout/">Proceed to checkout</a>
    <a class="btn btn-primary float-right " href="/">Continue shopping</a>
  </td>
</tr>
{% endif %}

```


Figure 24 - Order Summary template/Cart (templates/order-summary.html)

When a user visits a product information page and adds an item to their cart, the application redirects them to the order summary page, displaying the user's current order information.


Order Summary

#	Item title	Price	Quantity	Total Item Price
1	iPhone X	750.0	- 1 +	€699.0 [€51.0 SAVED]


Why not add a case for your phone?




Cover
Clear Android Case LABEL
€ 10.0




Cover
Black Lion Android Case LABEL
€ 20.0



Cover
Clear iPhone Case LABEL
€ 15.0



Cover
Carbon iPhone Case LABEL
€ 30.0



Cover
Cookie Monster Android Case LABEL
€ 20.0

Order Total €699.0

CONTINUE SHOPPING PROCEED TO CHECKOUT

Figure 25 - Cover Suggestions in Cart

If the user did not have the item previously in their cart, the quantity is set to one.

If the user did have the same item previously in their cart, the quantity is updated by one.

```
order_qs = Order.objects.filter(user=request.user, ordered=False)
if order_qs.exists():
    order = order_qs[0]
    # If the user already has the item in their cart, add 1
    if order.items.filter(item_slug=item.slug).exists():
        order_item.quantity += 1
        order_item.save()

    messages.info(request, "This item's quantity was updated.")
    return redirect("core:order-summary")
else:
    # Add the item to the cart
    messages.info(request, "This item was added to your cart.")
    order.items.add(order_item)
    return redirect("core:order-summary")
```

Figure 26 - Order View Logic

If the user did not have a previous order, i.e. no item in their cart, the system also creates a new order object for the user:

```

else:
    # Create a new order if the user does not have an order
    ordered_date = timezone.now()
    order = Order.objects.create(user=request.user, ordered_date=ordered_date)
    order.items.add(order_item)

    messages.info(request, "This item was added to your cart.")
    return redirect("core:order-summary")

```

Figure 27 - Create new Order Logic

If the user clicks the minus button on an item in their cart, it will reduce the quantity of said item, however two different results may occur. Should the user only have 1 quantity of the item, when they click the minus button to remove a single quantity, the entire item will be removed from the order as such:

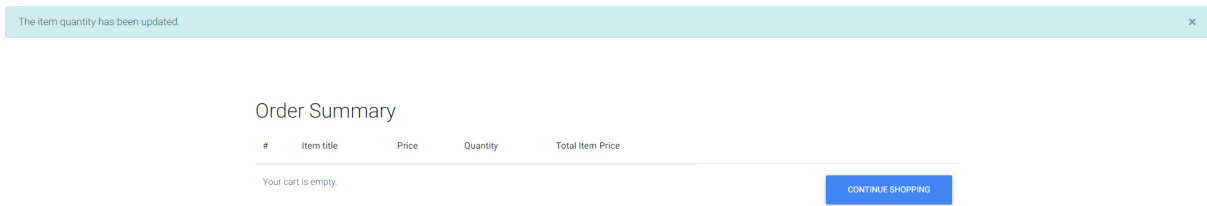


Figure 28 - Item removed by single item removal

If the user clicks the minus button when they have more than 1 quantity of said item in their cart, then the quantity will only be reduced by one:

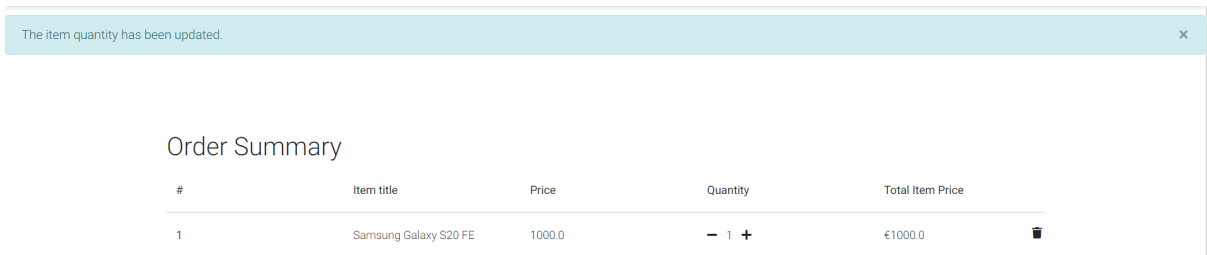


Figure 29 - Item quantity decreased by 1

If the user does not have a phone cover in their cart, the system will recommend phone covers to the user to add into their order. Once a cover has been added to the cart, the phone cover recommendations will disappear from the order summary page like so:

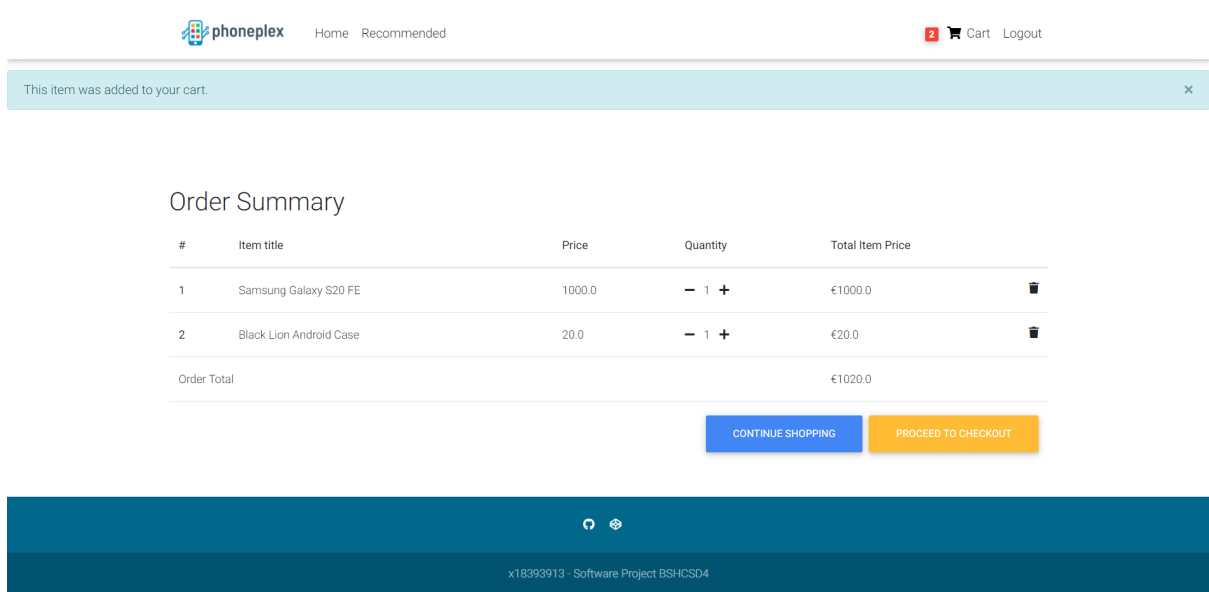


Figure 30 - Case Suggestions disappear after adding a phone case to the cart

The logic for this functionality is incorporated here:


```

class OrderSummaryView(LoginRequiredMixin, View):
    def get(self, *args, **kwargs):
        try:
            # Query the current Order that a user has
            order = Order.objects.get(user=self.request.user, ordered=False)
            # Query all existing covers in the database
            covers = Item.objects.filter(Q(category__startswith="Cov"))
            # Boolean for if a cover is already in the cart, default is False
            cover_in_cart = False

            # Checks if a phone is in the cart
            phone_in_cart = Order.objects.filter(Q(user=self.request.user,
            ordered=False, items__item__category__startswith="AND") |
            Q(user=self.request.user, ordered=False,
            items__item__category__startswith="IOS"))

            # print(phone_in_cart)

            # If a phone is in a cart and there is a cover, set cover_in_cart
            # to true so that covers ARE NOT suggested to users
            if phone_in_cart:
                cover = Order.objects.filter(Q(user=self.request.user,
                ordered=False, items__item__title__contains="Case"))
                if not cover:
                    cover_in_cart = True

            # Passing in the previous data collected
            context = {
                'object': order,
                'covers': covers,
                'cover_in_cart': cover_in_cart
            }
            return render(self.request, "order-summary.html", context)
        except ObjectDoesNotExist:
            messages.error(self.request, "You do not have an active order.")
            return redirect("/")

```

Figure 31 - Cart Logic in core/views.py

The order of the user is checked if they have a phone in the cart. If they do, an additional query is made to see if they have a cover inside the cart. If they do not, the boolean `cover_in_cart` is remains as False and is passed into the HTML template and thus, displays the cover information like so:

```

    {% if cover_in_cart %}
<tr>
  <td class="h2 text-center" colspan="5">Why not add a case for your phone?</td>
</tr>
{% for item in covers %}
<td>
  <!--Grid row-->
  <div class="row">
    <!--Grid column-->
    <div class="col-md-auto mb-4">

      <!--Card-->
      <div class="card" style="height: 400px; width: 200px;">

        <!--Card image-->
        <div class="view overlay">
          
          <a href="{{ item.get_absolute_url }}">
            <div class="mask rgba-white-slight"></div>
          </a>
        </div>
        <!--Card image-->

        <!--Card content-->
        <div class="card-body text-center">
          <!--Category & Title-->
          <a class="grey-text">
            <h5>{{ item.get_category_display }}</h5>
          </a>
          <h5>
            <strong>|
              <a href="{{ item.get_absolute_url }}" class="dark-grey-text">{{ item.title }}
                <span class="badge badge-pill {{ item.get_label_display }}-color">LABEL</span>
              </a>
            </strong>
          </h5>

          <h4 class="font-weight-bold blue-text">
            <strong>€euro;
              {% if item.discount_price %}
                {{ item.discount_price }}
              {% else %}
                {{ item.price }}
              {% endif %}
            </strong>
          </h4>

        </div>
        <!--Card content-->

      </div>
    </div>
  </div>
</td>
<!--Grid column-->
</td>
<!--Grid row-->
{% endfor %}
{% endif %}

```

Figure 32 - Django template tags for checking if the boolean has been passed in

If the user clicks on the trash icon beside their item on the order summary page, the entire item is removed from the order:

This item was removed from your cart. ×

Order Summary

#	Item title	Price	Quantity	Total Item Price
---	------------	-------	----------	------------------

Your cart is empty.

CONTINUE SHOPPING

Figure 33 - Removing entire order item from cart

Recommendation System:

For the phone recommendation system of the application, a basic implementation of the K-Nearest Neighbour algorithm was used. This is because the algorithm does not need a large dataset to begin with and does not make any assumptions/needs a classification of the data being supplied (pawangfg, 2022).

Since the algorithm is based on positive ratings that are attributed to an item/product, the ratings will be retrieved from the analysing the viewing history of a user. Each time a user has viewed a product on the application, a score of '1' is added to that product. If a user would view the product a total of 5 times during their entire browsing history on the website, then their overall rating for that phone is a positive integer of 5.

```

# Tracking each time a phone appears in the viewing History
phone_score = {"item_id": {}}

for record in view_values:
    # Getting id of the item, user and name of the product
    item_id = record['item_id']

    # Create key in dictionary if item has not shown up before
    # in the viewing history
    if str(item_id) not in phone_score['item_id']:
        phone_score['item_id'].update({f"{item_id}": []})

    # Adds '1' as a rating to the phone
    counter = phone_score['item_id'][str(item_id)]
    counter.append(1)

# List of lists that contain: User IDs, Phone IDs and Phone Ratings
rating_list = []

for phone_id in phone_score['item_id']:
    # Grab the list of scores for a particular phone
    score_list = phone_score['item_id'][str(phone_id)]
    # Add up all the 1's found that could be present in the list
    score_sum = sum(score_list)

    # Accessing ID of the user from the user currently logged in
    user_id = self.request.user.id
    # Adding the list of [User ID, Phone ID and Phone Rating] to
    # the list of ratings
    rating_list.append([user_id, int(phone_id), score_sum])

```

Figure 34 - Creating lists of phone ratings to be used in matrix

A matrix is created where the user, product and the rating are mapped together. With the different phones and the potential for varied ratings between those phones, the application can suggest a mobile phone that is closest to their most viewed phone (the highest rated phone). Similarity scores can then be constructed by subtracting the mean of the ratings against the original ratings. The implementation of this algorithm could produce inaccurate results as the user is only compared to themselves instead of other users, so the recommendation of other phones heavily relies on the own user's browsing patterns and the small dataset involved during their minimal usage of the application.

If the user does not have at least four phones viewed on the website, they are informed to browse further and look at additional phones to gather more data.

```

# If there are less than 4 phones browsed on the website, redirect
# the user and inform them that the app needs more data before
# recommending a phone
if len(rating_list) < 4:
    messages.info(self.request, f"Please browse more products before seeing recommendations.")
    return redirect("/")

```

Figure 35 - Logic for informing the user that they need to browse more phones

After creating the matrix by mapping the user IDs, phone IDs and rating,, they are passed to the `find_similar_phones` method as well as the highest rated, lowest rated and mean scores of all the phones.

```
# Creating the dataframe for the phone ratings
ratings = pd.DataFrame(rating_list, columns=['User ID', 'Phone ID', "Phone Rating"])

# Number of ratings, phones used and users
n_ratings = len(ratings)
n_phones = len(ratings['Phone ID'].unique())
n_users = len(ratings['User ID'].unique())

# Tracking the amount of ratings a user has
user_freq = ratings[['User ID', 'Phone ID']].groupby('User ID').count() \
    .reset_index()
user_freq.columns = ['User ID', 'n_ratings']

# Grabbing the average, lowest and highest rated phones
mean_rating = ratings.groupby('Phone ID')[['Phone Rating']].mean()
lowest_rated = mean_rating['Phone Rating'].idxmin()
highest_rated = mean_rating['Phone Rating'].idxmax()

# Bayesian average is used since this is small dataset
phone_stats = ratings.groupby('Phone ID')[['Phone Rating']].agg(
    ['count', 'mean'])
phone_stats.columns = phone_stats.columns.droplevel()

# Creating the matrix with the phone ratings and mapping them onto the user and the phone IDs
X, user_mapper, phone_mapper, user_inv_mapper, phone_inv_mapper = create_matrix(ratings)

similar_ids = find_similar_phones(highest_rated, X, phone_mapper, phone_inv_mapper, k=3)
```

Figure 36 - Creating the matrix by passing in the User ID, Phone IDs and Ratings

The `find_similar_phones` method then passes the matrix (X being the variable storing the matrix) into the K Nearest Neighbour method found in the `sklearn` module, a machine learning Python library.

```
def find_similar_phones(phone_id, X, phone_mapper, phone_inv_mapper, k, metric='cosine', show_distance=False):
    """
    Finding similar phones using K-Nearest Neighbour
    """
    neighbour_ids = []
    phone_ind = phone_mapper[phone_id]
    phone_vector = X[phone_ind]

    k += 1
    k_nn = NearestNeighbors(n_neighbors=k,
                           algorithm="brute",
                           metric=metric)
    k_nn.fit(X)

    phone_vector = phone_vector.reshape(1,-1)
    neighbour = k_nn.kneighbors(phone_vector, return_distance=show_distance)

    for i in range(0, k):
        n = neighbour.item(i)
        neighbour_ids.append(phone_inv_mapper[n])
    neighbour_ids.pop(0)

    return neighbour_ids
```

Figure 37 - Using the K Nearest Neighbour algorithm to find similar phones to the most viewed phone by the user

Since the dataset is small, a brute algorithm is applied which does a brute-force calculation between the distances of the ratings and predicts the most similar items to the highest rated phone. The highest

rated phone is removed as a potential neighbour so that it is not recommended to the user again (removing the first index in neighbour_ids) (scikit-learn developers, 2022).

Checkout:

The checkout page can be accessed on the order summary/cart page by clicking the checkout button. Once redirected, the user has the ability to enter information related to their billing address (directly related to the BillingAddress model). These forms are created in the forms.py file:

```
from django import forms
from django_countries.fields import CountryField
from django_countries.widgets import CountrySelectWidget

PAYMENT_CHOICES = (
    ('S', 'Stripe'),
    ('P', 'PayPal')
)

class CheckoutForm(forms.Form):
    street_address = forms.CharField(widget=forms.TextInput(attrs={
        'placeholder': '1234 Main St.',
        'class': 'form-control'
    }))

    apartment_address = forms.CharField(required=False, widget=forms.TextInput(
        attrs={
            'placeholder': 'Apartment or Suite',
            'class': 'form-control'
        }))

    country = CountryField(blank_label='(select country)').formfield(widget=CountrySelectWidget(attrs={
        'class': 'custom-select d-block w-100',
        'id': 'zip'
    }))

    zip = forms.CharField(widget=forms.TextInput(attrs={
        'class': 'form-control'
    }))

    same_shipping_address = forms.BooleanField(required=False)
    save_info = forms.BooleanField(required=False)

    payment_option = forms.ChoiceField(widget=forms.RadioSelect(),
        choices=PAYMENT_CHOICES)
```

Figure 38 - Forms that are used in the Checkout process (core/forms.py)

The forms are then passed to the CheckoutView and this view handles both GET and POST requests a user could make. If the user POSTS their billing address information correctly and does not leave a

mandatory field blank, their billing address is saved and their order is updated:

```
class CheckoutView(View):
    def get(self, *args, **kwargs):
        form = CheckoutForm()
        context = {
            "form": form
        }
        return render(self.request, "checkout-page.html", context)

    def post(self, *args, **kwargs):
        form = CheckoutForm(self.request.POST or None)
        try:
            # Get the order of the user
            order = Order.objects.get(user=self.request.user, ordered=False)
            if form.is_valid():
                # Clean all form data submitted
                street_address = form.cleaned_data.get('street_address')
                apartment_address = form.cleaned_data.get('apartment_address')
                country = form.cleaned_data.get('country')
                zip = form.cleaned_data.get('zip')
                payment_option = form.cleaned_data.get('payment_option')
                billing_address = BillingAddress(
                    user = self.request.user,
                    street_address = street_address,
                    apartment_address = apartment_address,
                    country = country,
                    zip = zip
                )
                # Save the billing address to the database
                billing_address.save()
                order.billing_address = billing_address
                # Save the edited order to the database
                order.save()

                if payment_option == 'S':
                    return redirect('core:payment', payment_option='stripe')
                elif payment_option == 'P':
                    return redirect('core:payment', payment_option='paypal')
                else:
                    messages.warning(self.request, "Invalid payment option.")
                    return redirect('core:checkout-page')
            except ObjectDoesNotExist:
                messages.error(self.request, "You do not have an active order. Please add an item to the cart to get started.")
                return redirect("core:order-summary")
```

Figure 39 - Checkout Logic for handling billing address information

However, if a mandatory field is left blank, the user is prompted to enter that information before proceeding, acting as a security measure and ensuring no oversight in the payment process:

The screenshot shows the 'Checkout form' on the phoneplex website. The form includes the following fields and elements:

- Address:** A text input field containing '1234 Main St.'
- Address 2 (optional):** A text input field for 'Apartment or Suite'. A red error message 'Please fill out this field.' is displayed above this field.
- Country:** A dropdown menu currently set to 'Austria'.
- Zip:** A text input field containing '4334343'.
- Payment Option:** Two radio buttons: 'Stripe' (selected) and 'PayPal'.
- Buttons:** A blue 'CONTINUE TO CHECKOUT' button at the bottom of the form.

Figure 40 - Checkout form validation

Once the user clicks the 'continue to checkout' button, they are redirected to the payment page.

Payment Process:

phoneplex uses the Stripe API to help with payments. A test API key has been provided by Stripe for developers to use in test applications. The Stripe python module is incorporated with the website, as well as custom JavaScript and CSS from Stripe's documentation that helps displaying and handling the payment information a user enters into the forms (Stripe, 2022).

```
<style>
  #stripeBtnLabel {
    font-family: "Helvetica Neue", Helvetica, sans-serif;
    font-size: 16px;
    font-variant: normal;
    padding: 0;
    margin: 0;
    -webkit-font-smoothing: antialiased;
    font-weight: 500;
    display: block;
  }

  #stripeBtn {
    border: none;
    border-radius: 4px;
    outline: none;
    text-decoration: none;
    color: #fff;
    background: #32325d;
    white-space: nowrap;
    display: inline-block;
    height: 40px;
    line-height: 40px;
    box-shadow: 0 4px 6px rgba(50, 50, 93, .11), 0 1px 3px rgba(0, 0, 0, .08);
    border-radius: 4px;
    font-size: 15px;
    font-weight: 600;
    letter-spacing: 0.025em;
    text-decoration: none;
    -webkit-transition: all 150ms ease;
    transition: all 150ms ease;
    float: left;
    width: 100%
  }

  button:hover {
    transform: translateY(-1px);
    box-shadow: 0 7px 14px rgba(50, 50, 93, .10), 0 3px 6px rgba(0, 0, 0, .08);
    background-color: #43458b;
  }

  .stripe-form {
    padding: 5px 30px;
  }

  #card-errors {
    height: 20px;
    padding: 4px 0;
    color: #fa755a;
  }
}
```

Figure 41 - Stripe Styling from their documentation


```

<script src="https://js.stripe.com/v3/"></script>
<script>
  var stripe = Stripe('pk_test_qblFNYngBkEdjEZ16jxxowSM');
  var elements = stripe.elements();

  var style = {
    base: {
      color: "#32325d",
    }
  };

  var card = elements.create("card", {
    style: style
  });
  card.mount("#card-element");

  card.on('change', function(event) {
    var displayError = document.getElementById('card-errors');
    if (event.error) {
      displayError.textContent = event.error.message;
    } else {
      displayError.textContent = '';
    }
  });

  // Handle form submission.
  var form = document.getElementById('stripe-form');
  form.addEventListener('submit', function(event) {
    event.preventDefault();

    stripe.createToken(card).then(function(result) {
      if (result.error) {
        // Inform the user if there was an error.
        var errorElement = document.getElementById('card-errors');
        errorElement.textContent = result.error.message;
      } else {
        // Send the token to your server.
        stripeTokenHandler(result.token);
      }
    });
  });

  // Submit the form with the token ID.
  function stripeTokenHandler(token) {
    // Insert the token ID into the form so it gets submitted to the server
    var form = document.getElementById('stripe-form');
    var hiddenInput = document.createElement('input');
    hiddenInput.setAttribute('type', 'hidden');
    hiddenInput.setAttribute('name', 'stripeToken');
    hiddenInput.setAttribute('value', token.id);
    form.appendChild(hiddenInput);

    // Submit the form
    form.submit();
  }
</script>

```

Figure 42 - Stripe imported JavaScript for card form handling

The payment information page also displays key order details (total, quantity etc.) to the user for which they are about to pay for:

```

<main class="mt-5 pt-4">
  <div class="container wow fadeIn">

    <h2 class="my-5 h2 text-center">Payment</h2>

    <div class="row">

      <div class="col-md-12 mb-4">
        <div class="card">
          <form action="." method="post" id="stripe-form">
            {% csrf_token %}
            <div class="stripe-form-row">
              <div id="card-element">
                <!-- Elements will create input elements here -->
              </div>

              <div id="card-errors" class="mb-2" role="alert"></div>

              <button id="stripeBtn">Submit Payment</button>
            </div>
          </form>
        </div>
      </div>
      <div class="col-md-12 mb-4">

        <!-- Heading -->
        <h4 class="d-flex justify-content-between align-items-center mb-3">
          <span class="text-muted">Your cart</span>
          <span class="badge badge-secondary badge-pill">{{ order.items.count }}</span>
        </h4>

        <!-- Cart -->
        <ul class="list-group mb-3 z-depth-1">
          {% for order_item in order.items.all %}
          <li class="list-group-item d-flex justify-content-between lh-condensed">
            <div>
              <h6 class="my-0">{{ order_item.quantity }} x {{ order_item.item.title }}</h6>
              <small class="text-muted">{{ order_item.item.description }}</small>
            </div>
            <span class="text-muted">{{ order_item.get_final_price }}</span>
          </li>
          {% endfor %}
          <li class="list-group-item d-flex justify-content-between">
            <span>Total (EUR)</span>
            <strong>&euro;{{ order.get_total }}</strong>
          </li>
        </ul>
        <!-- Cart -->

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

```

Figure 43- Payment HTML page (templates/payment.html)

There are many forms of valid credit cards that the Stripe API provides for developers to use (Stripe, 2022) and the most commonly used one is the ‘4242 4242 4242 4242’ VISA card number. After the user sends a POST request to the application with the valid credit card information, a Stripe charge object is created storing this information:

```

# `source` is obtained with Stripe.js; see https://stripe.com/docs/payments/accept-a-payment#web-create-token
charge = stripe.Charge.create(
    amount=amount,
    currency="usd",
    source="tok_visa",
    description="My First Test Charge (created for API docs)"
)

```

Figure 44 - Creating a charge object with Stripe

If successful, the Payment model is then used to store the transaction to our own database as well as setting the ordered status to 'True' in the order object the user has used during the entire process:

```
# Creating the payment object and saving it to the database
payment = Payment()
payment.stripe_charge_id = charge['id']
payment.user = self.request.user
payment.amount = amount
payment.save()

# Setting the ordered field to True
order.ordered = True
# Saving the payment information to the order
order.payment = payment
# Saving the finished order to the database
order.save()

messages.success(self.request, "Your order was successful!")
```

Figure 45 - Saving the payment information and new order status after successful payment

The user is then redirected to the home page and notified that their order has been successful. The user's order is then reset and their cart is emptied:

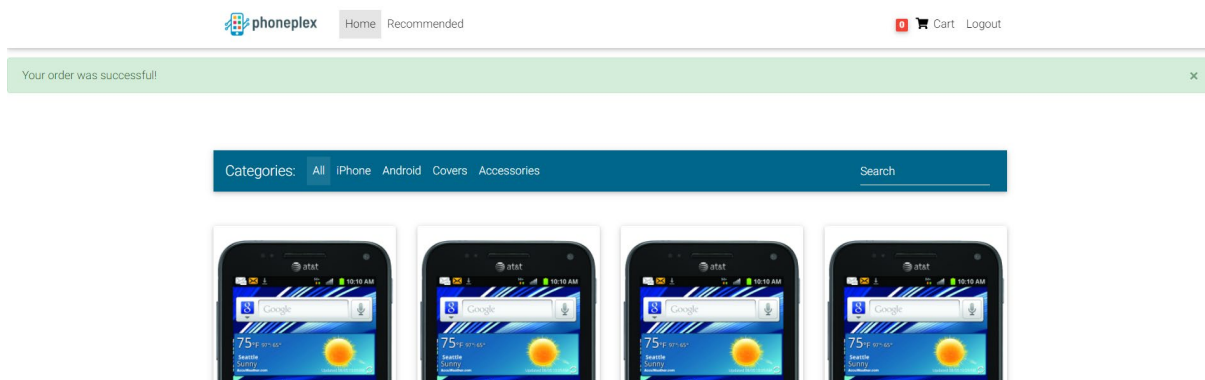


Figure 46 - User notified of their successful purchase

After a successful purchase, an email is sent to the associated email of the user. For this application, a free Simple Mail Transfer Protocol server was used (WPOven, 2022) and the port and host were specified in the django settings:

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.freesmtpservers.com'
EMAIL_PORT = 25
```

Figure 47 - Simple Mail Transfer Protocol Host and Port

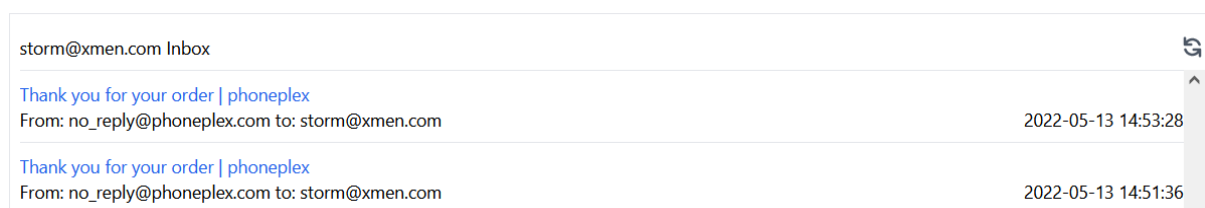


Figure 48 - Inbox for user that purchased items successfully

The contents of the email can be seen on <https://www.wpoven.com/tools/free-smtp-server-for-testing> and by accessing the inbox of the email a user signed up with:

Outgoing or Incoming email address Inbox

Figure 49 - Check inbox for email used to sign up for phoneplex

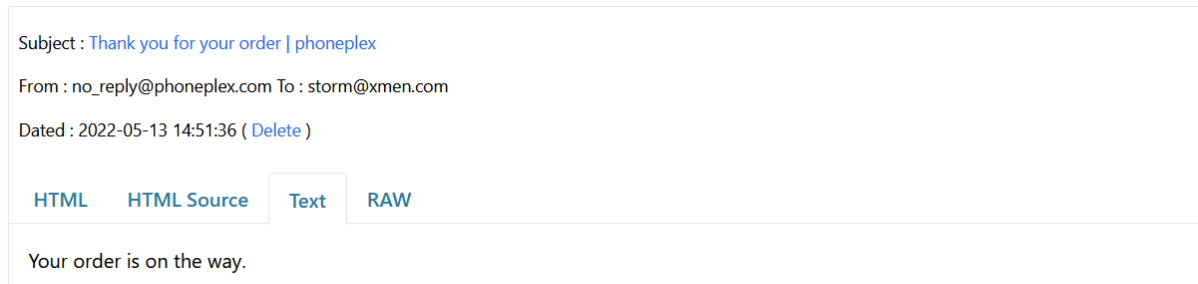


Figure 50 – Order confirmation in the email

Django provides email functionality by importing from the module ‘django.core.mail import send_mail’. When the payment and order objects are saved to the database, the snippet below is the only code that is used to send the email to the server:

```
# Sending the mail to the SMTP server
send_mail(
    'Thank you for your order | phoneplex',
    f'Your order is on the way.',
    'no_reply@phoneplex.com',
    [self.request.user.email],
    fail_silently=False,
)
print('Email sent!')
return redirect("/")
```

Figure 51 - Calling the django send_mail method to send an email

2.4 Graphical User Interface (GUI)

phoneplex Home SIMS FAQ Login Sign Up

Sign Up

Already have an account? Then please [sign in](#).

Username*

Email (optional)

Password*

Password (again)*

Password (again)

SIGN UP >

© 2019 Copyright: MGBBootstrap.com

Figure 52 - Sign Up Page

The Sign Up page for phoneplex, with forms for the username, email and password as well as a sign up button.

phoneplex Home SIMS FAQ Login Sign Up

Sign in

If you have not created an account yet, then please [sign up](#) first.

Username*

Password*

Remember Me

FORGOT PASSWORD? SIGN IN

© 2019 Copyright: MGBBootstrap.com

Figure 53 - Sign in Page

The Login page for phoneplex, with the field for the username and password.

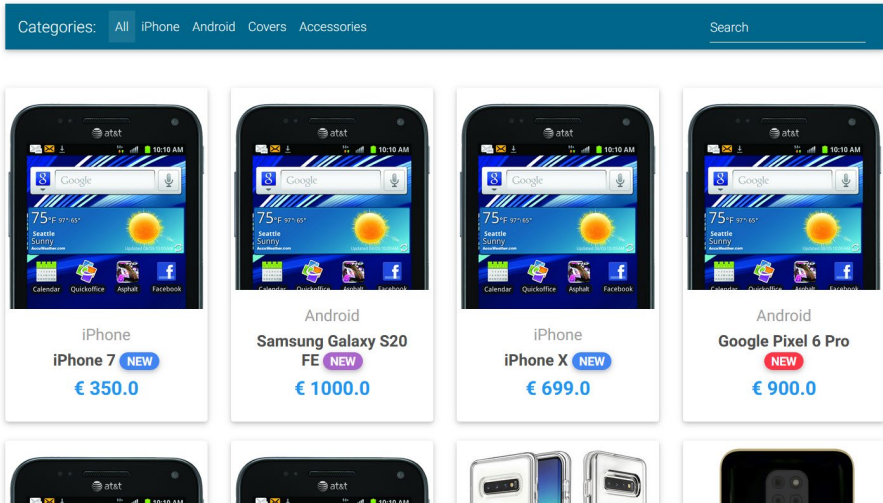


Figure 54 - phoneplex Home Page

Home page of phoneplex, with phone products all organised on the page. Images are set to each Item individually. The name of the phone, price and label are all present on the product listings.

Other tabs have been created for future development such as filtering by iPhones, Androids, phone covers and accessories.

Since a user has not been logged in, only the login and sign up navbar items are present, as users have to be logged in in order for them to add products to their cart.

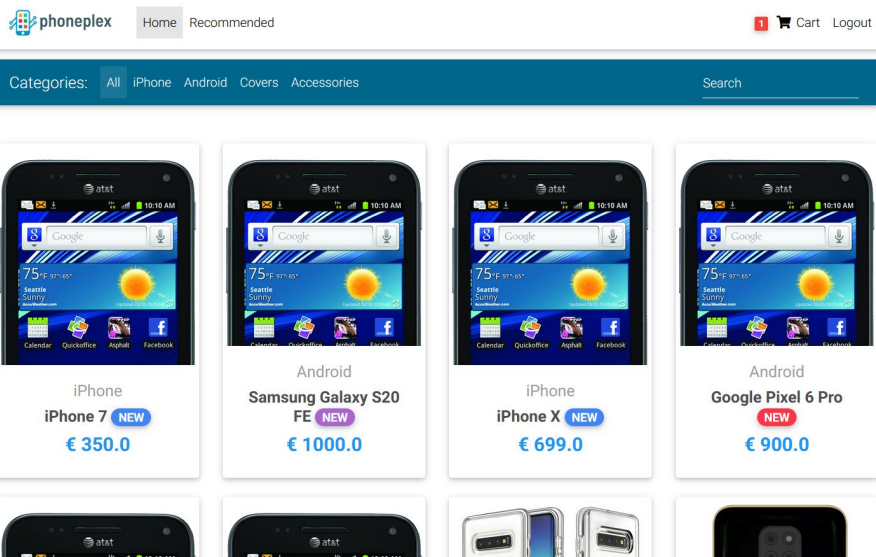


Figure 55 - phoneplex Home Page (Logged in)

Once a user is logged in, the cart icon and logout navbar item appears.



Cover

€10.0

Description

This Android clear case is made from plant-based material and is the world's first compostable, transparent Android case!

ADD TO CART

REMOVE FROM CART

Figure 56 - Product information page

Clicking on a product on the home page will lead a user to the specific product information of that page. The price of the item, its category and the description are displayed for the product.


If an item has a discount, that new price will be displayed rather than the original price. Buttons for adding an item to the cart and removing an item from the cart are available to the user.

All products listed on the website and created from the Item model have their own slug in the URL, as seen with the current example '/product/iphone-7'.


Order Summary

#	Item title	Price	Quantity	Total Item Price
1	iPhone 7	500.0	- 1 +	€350.0 (€150.0 SAVED)


Why not add a case for your phone?




Cover
Clear Android Case LABEL
€ 10.0




Cover
Black Lion Android Case LABEL
€ 20.0



Cover
Clear iPhone Case LABEL
€ 15.0



Cover
Carbon iPhone Case LABEL
€ 30.0



Cover
Cookie Monster Android Case LABEL
€ 20.0

Order Total

€350.0

CONTINUE SHOPPING

PROCEED TO CHECKOUT

Figure 57 - Cart Page

Users that have items in a cart will be greeted with an order summary that displays the items they are purchasing, their quantities, individual prices and total prices including discounts. Users will have the option to continue shopping or proceed to the checkout. If there is no phone case in the cart, suggestions for phone covers will be displayed to the user so that they may add them to the cart.

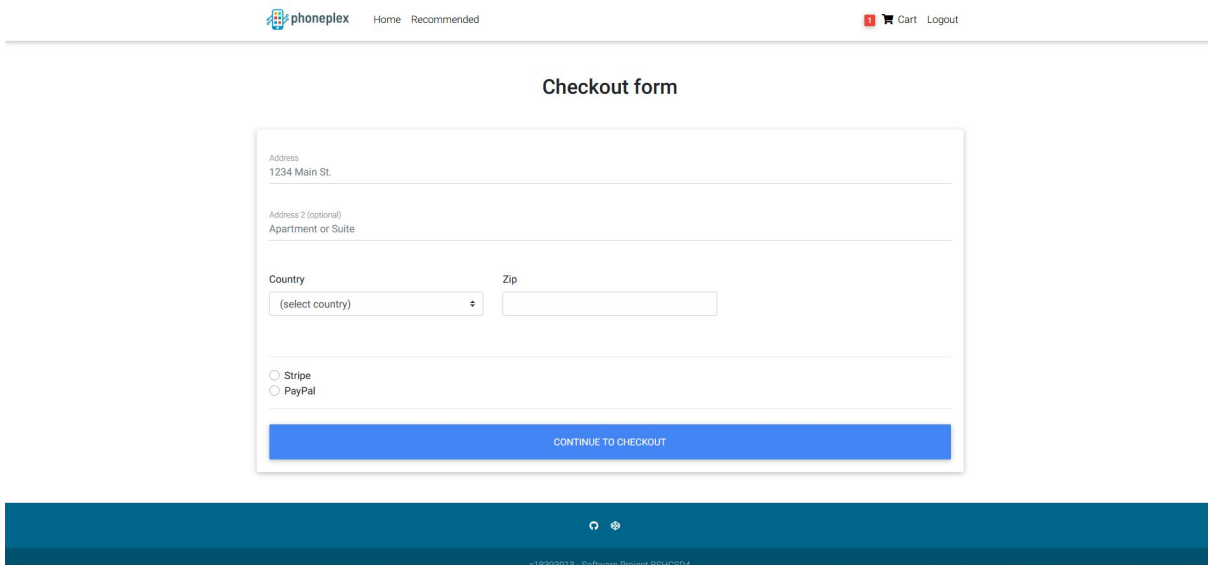


Figure 58 - Checkout page

The checkout page contains forms for users to fill out their billing address for the order, as well as their method of payment.

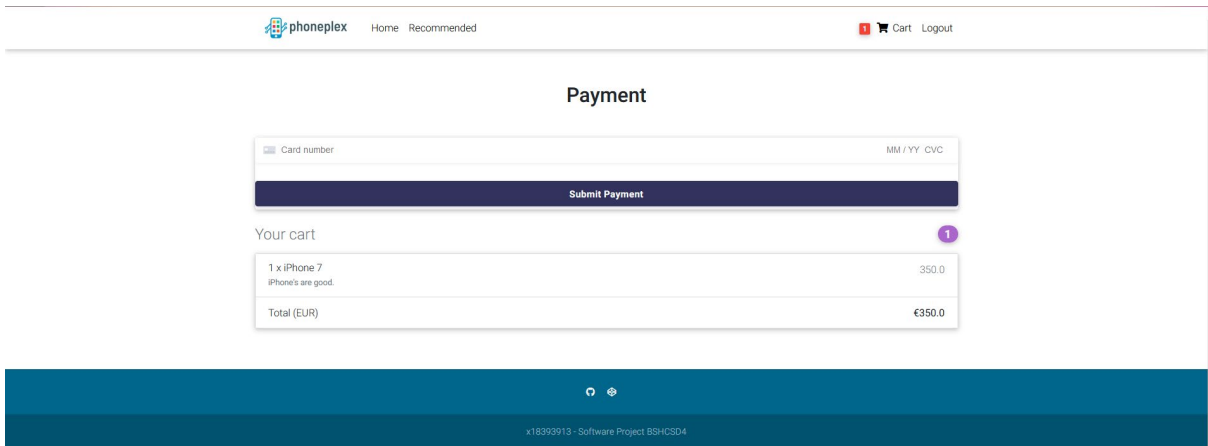


Figure 59 - Payment Page

The payment page displays the current order to the user and its total amount. Forms for entering their credit card details exist and are incorporated with Stripe's own styling. Clicking the 'Submit Payment' button will process the order and redirect the user to the home page if successful.

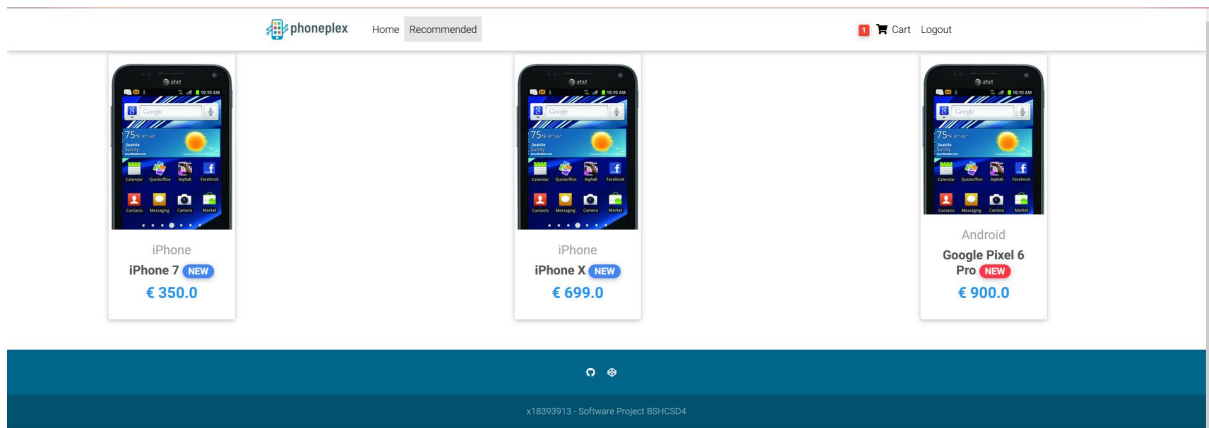


Figure 60 - Recommendation Page

The recommended page displays the 3 nearest neighbours (most similar phone) to the user's most viewed phone during their entire browsing history of the application.

2.5 Testing

Django provides functionality for testing the web application with their django.test module. To supplement these unit and integration tests, systems tests were incorporated using the Selenium library to automate the browser and go through the entire process of adding/removing items from a cart and going through the flow of starting and finishing an order.

Test Case ID:	1
Test Case Title:	Test Similar Phones
Test Type:	Unit Test
Test Requirement:	Recommend Similar Phones to highest rated phone
Test Case Description:	Based on the highest rated phone of any given user, the K Nearest Neighbour algorithm should return similar phones that are also rated similarly.
Test Priority:	High
Pre-Conditions:	A Phone ratings list has been created.
Execution Steps:	<ol style="list-style-type: none"> 1. Map the phone, user, and ratings into a matrix object. 2. Pass the matrix into the sklearn KNN method as parameters. 3. Return the similar phone IDs.
Post-Conditions:	System waits for the next test to execute.
Test Status:	Success
Test Case ID:	2
Test Case Title:	Test Item Model Content
Test Type:	Unit Test

Test Requirement: Test that the Item model stores product information correctly

Test Case Description: Asserts that the Item created during the setup of the test executions is storing the correct information.

Test Priority: High

Pre-Conditions: Set up dummy Item data

Execution Steps:

1. Assert the expected item title
2. Assert the expected item description
3. Assert the expected item price
4. Assert the expected item discount price
5. Asserts the expected item category
6. Assert the expected item label
7. Assert the expected item slug
8. Assert the expected item image

Post-Conditions: System waits for the next test to run

Test Status: Success

Test Case ID: 3

Test Case Title: Test Item URL exists at correct location

Test Type: Unit Test

Test Requirement: Ensure that the URL for the item is correct

Test Case Description: Ensures that the URL for the item is correct.

Test Priority: Medium

Pre-Conditions: Set up dummy Item data

Execution Steps:

1. Send GET request to product/{item.slug}
2. Assert status code 200 in the response

Post-Conditions: System waits to run the next test

Test Status: Success

Test Case ID: 4

Test Case Title: Test Product Homepage

Test Type: Unit Test

Test Requirement: Ensure that the product template page is correct

Test Case Description: Ensures that the product page is displayed correctly.

Test Priority:	Medium
Pre-Conditions:	Set up dummy Item data
Execution Steps:	<ol style="list-style-type: none"> 1. Visit item's product page information 2. Assert status code 3. Assert correct template used 4. Assert product description is on the page
Post-Conditions:	System waits to run additional tests
Test Status:	Success
Test Case ID:	5
Test Case Title:	Test Order Item Unordered Model Content
Test Type:	Integration
Test Requirement:	Ensures that the order item successfully uses the Item model
Test Case Description:	Ensures that an order item is inheriting the correct information from the Item passed in.
Test Priority:	Medium
Pre-Conditions:	Set up dummy Item data
Execution Steps:	<ol style="list-style-type: none"> 1. Assert the expected item title 2. Assert the expected item description 3. Assert the expected item price 4. Assert the expected item discount price 5. Asserts the expected item category 6. Assert the expected item label 7. Assert the expected item slug 8. Assert the expected item image 9. Assert the expected ordered status 10. Assert the expected quantity 11. Assert the expected user
Post-Conditions:	System waits to run additional tests
Test Status:	Success
Test Case ID:	6
Test Case Title:	Test Order Model Content

Test Type: Integration
Test Requirement: Ensure the Order has been created successfully
Test Case Description: Ensures that the Order has inherited information from the OrderItem used
Test Priority: High
Pre-Conditions: Set up dummy Item data
Set up dummy Order data
Execution Steps: 1. Assert the number of items in the order
2. Assert the user attached to the other
Post-Conditions: System waits to run additional tests
Test Status: Success

Test Case ID: 7
Test Case Title: Test Adding Item to Order
Test Type: Integration
Test Requirement: Ensure that additional items can be added to an order
Test Case Description: Ensure that additional items can be added to an order
Test Priority: High
Pre-Conditions: Set up dummy Item data
Set up dummy OrderItem data
Set up dummy Order data
Execution Steps: 1. Assert the number of items in the order
2. Assert the title of the newly added item to the order
3. Assert the ordered status
Post-Conditions: System waits to run additional tests
Test Status: Success

Test Case ID: 8
Test Case Title: Test Login Page
Test Type: Integration
Test Requirement: Ensure that the login page can be accessed correctly
Test Case Description: Ensure that the login page can be accessed correctly
Test Priority: High

Pre-Conditions:	Set up user credentials
Execution Steps:	<ol style="list-style-type: none"> 1. Request the login page 2. Assert the status code 3. Assert the html template used in the DOM
Post-Conditions:	System waits to run additional tests
Test Status:	Success
Test Case ID:	9
Test Case Title:	Test Login using empty username and password
Test Type:	Integration
Test Requirement:	Ensure that a user can't login with an empty username and password
Test Case Description:	Ensure that a user can't login with an empty username and password
Test Priority:	Low
Pre-Conditions:	Set up user credentials
Execution Steps:	<ol style="list-style-type: none"> 1. Post login credentials to login page 2. Assert status response code 3. Assert error in password form 4. Assert error in username form
Post-Conditions:	System waits for additional tests to be run
Test Status:	Success
Test Case ID:	10
Test Case Title:	Test username not correct
Test Type:	Integration
Test Requirement:	Ensure that a user can't login with a wrong username and valid password
Test Case Description:	Ensure that a user can't login with a wrong username and valid password
Test Priority:	Low
Pre-Conditions:	Set up user credentials
Execution Steps:	<ol style="list-style-type: none"> 1. Post login credentials to login page 2. Assert status response code 3. Assert error in username form

Post-Conditions:	System waits for additional tests to be run
Test Status:	Success
Test Case ID:	11
Test Case Title:	Test password not correct
Test Type:	Integration
Test Requirement:	Ensure that a user can't login with a valid username and wrong password
Test Case Description:	Ensure that a user can't login with a valid username and wrong password
Test Priority:	Low
Pre-Conditions:	Set up user credentials
Execution Steps:	<ol style="list-style-type: none"> 1. Post login credentials to login page 2. Assert status response code 3. Assert error in password form
Post-Conditions:	System waits for additional tests to be run
Test Status:	Success
Test Case ID:	12
Test Case Title:	Test login success
Test Type:	Integration
Test Requirement:	Ensure that a user can login with a valid username and valid password
Test Case Description:	Ensure that a user can login with a valid username and valid password
Test Priority:	Low
Pre-Conditions:	Set up user credentials
Execution Steps:	<ol style="list-style-type: none"> 1. Post login credentials to login page 2. Assert status response code 3. Assert error not in response content
Post-Conditions:	System waits for additional tests to be run
Test Status:	Success
Test Case ID:	13

Test Case Title: Test Add Item to Cart
Test Type: System
Test Requirement: Be able to add items to a user's cart
Test Case Description: The driver visits a product information page, clicks the add to cart button and ensures that their item is in the cart.
Test Priority: High
Pre-Conditions: N/A
Execution Steps:
1. Create a new webdriver instance
2. Login with valid user credentials
3. Click the iPhone 7 card on the home page
4. Click the add to cart button
5. Ensure the product title is correct on the order summary page
6. Ensure the quantity is correct
Post-Conditions: System waits for additional tests to be run
Webdriver instance is torn down
Test Status: Success

Test Case ID: 14
Test Case Title: Test Add Item Quantity
Test Type: System
Test Requirement: Be able to add item quantities to an order item
Test Case Description: The driver visits the order summary page, clicks plus symbol to add 1 more quantity of an item and ensures that their item quantity is updated.
Test Priority: Medium
Pre-Conditions: N/A
Execution Steps:
1. Create a new webdriver instance
2. Login with valid user credentials
3. Click the cart nav link.
4. Assert the product title is correct
5. Ensure the quantity is correct
6. Ensure the plus icon is on the page
7. Click the plus icon
8. Check that the quantity has been updated

Post-Conditions: System waits for additional tests to be run
Webdriver instance is torn down

Test Status: Success

Test Case ID: 15

Test Case Title: Test Remove Item Quantity

Test Type: System

Test Requirement: Be able to remove item quantities to an order item

Test Case Description: The driver visits the order summary page, clicks minus symbol to remove 1 less quantity of an item and ensures that their item quantity is updated.

Test Priority: Medium

Pre-Conditions: N/A

Execution Steps:

1. Create a new webdriver instance
2. Login with valid user credentials
3. Click the cart nav link.
4. Assert the product title is correct
5. Ensure the quantity is correct
6. Ensure the minus icon is on the page
7. Click the minus icon
8. Check that the quantity has been updated

Post-Conditions: System waits for additional tests to be run
Webdriver instance is torn down

Test Status: Success

Test Case ID: 16

Test Case Title: Test Remove Item from Order

Test Type: System

Test Requirement: Be able to remove item from an order

Test Case Description: The driver visits the order summary page and clicks trash symbol to remove the item from an order.

Test Priority: Medium

Pre-Conditions: N/A

Execution Steps:

1. Create a new webdriver instance

	<ol style="list-style-type: none"> 2. Login with valid user credentials 3. Click the cart nav link. 4. Assert the product title is correct 5. Click the trash symbol next to the order item 6. Ensure the item has been removed from the order
Post-Conditions:	<p>System waits for additional tests to be run</p> <p>Webdriver instance is torn down</p>
Test Status:	Success
Test Case ID:	17
Test Case Title:	Test Buy Product
Test Type:	System
Test Requirement:	Ensure that the user can purchase a product
Test Case Description:	The driver will go through the whole flow of purchasing an item from phoneplex.
Test Priority:	High
Pre-Conditions:	N/A
Execution Steps:	<ol style="list-style-type: none"> 1. Create a new webdriver instance 2. Login with valid user credentials 3. Click the iPhone X card on the home page 4. Click the add to cart button 5. Click the checkout button 6. Assert that checkout is in the URL 7. Fill in the street address form 8. Select a country from the country dropdown 9. Fill in the zip code form 10. Click the stripe radio button 11. Click the checkout button 12. Assert payment/stripe is in the URL 13. Enter the card number 4242 repeatedly 14. Enter a valid expiry date 15. Enter a cvc 16. Enter a zip code

- 17. Click the 'Submit Payment' button
- 18. Assert that the order was successful

Post-Conditions:

Webdriver instance is torn down

Test Status:

Success

2.6 Evaluation

Since the application is not deployed to a hosted web service, the quantifiable measurements of how well it performs are limited to tracking local response times. All response times when visiting the specific pages were recorded without any of the site data being cached to the browser.

Home page: 504ms

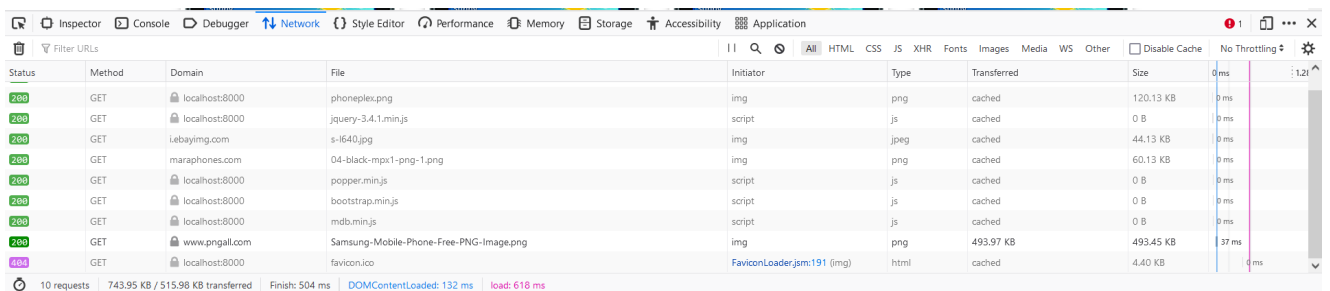


Figure 61 - Home Page response time in milliseconds

Recommended: 993ms

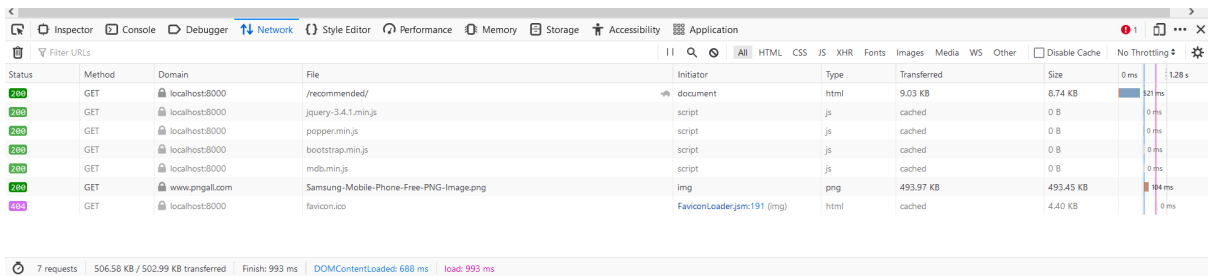


Figure 62 - Recommended page response time in milliseconds

Order Summary/Cart: 261ms

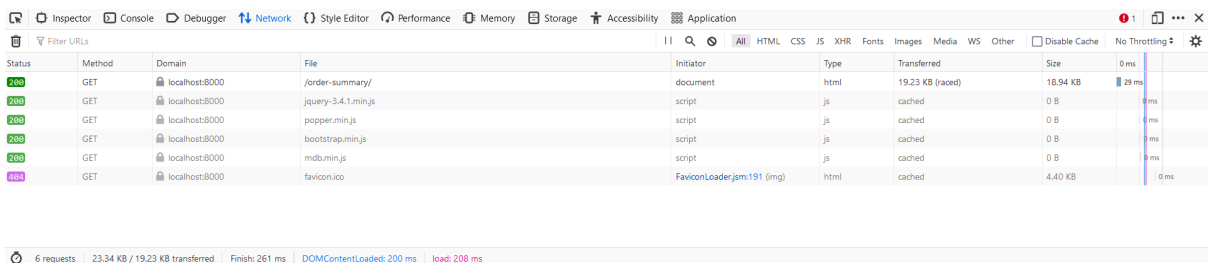


Figure 63 - Order summary/Cart page response time in milliseconds

Login: 436ms

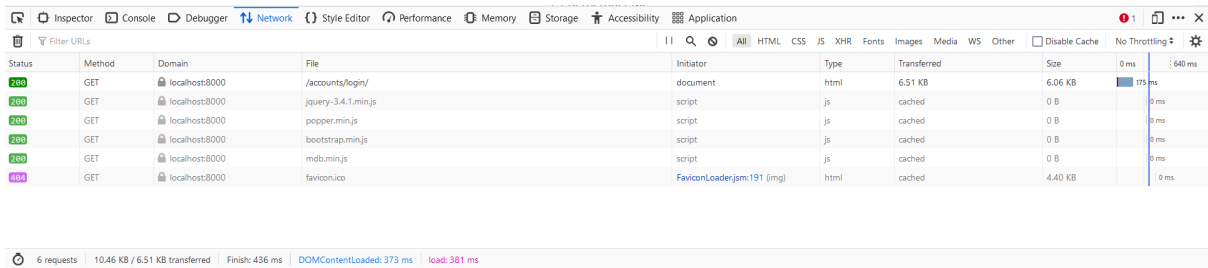


Figure 64 - Login page response time in milliseconds

Sign Up: 261ms

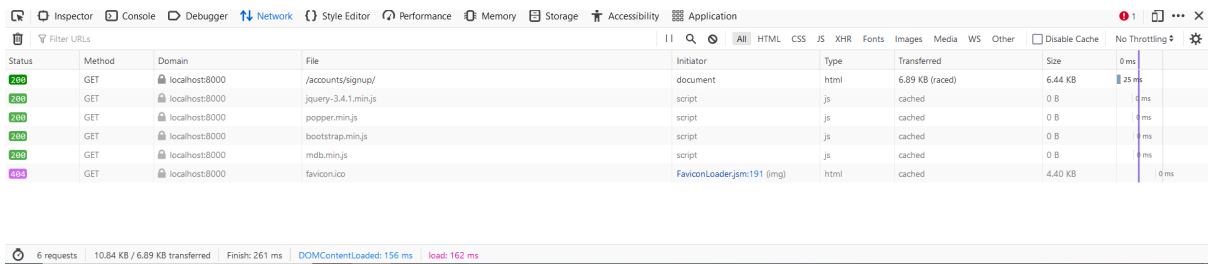


Figure 65 - Sign up page response times in milliseconds

Checkout: 324ms

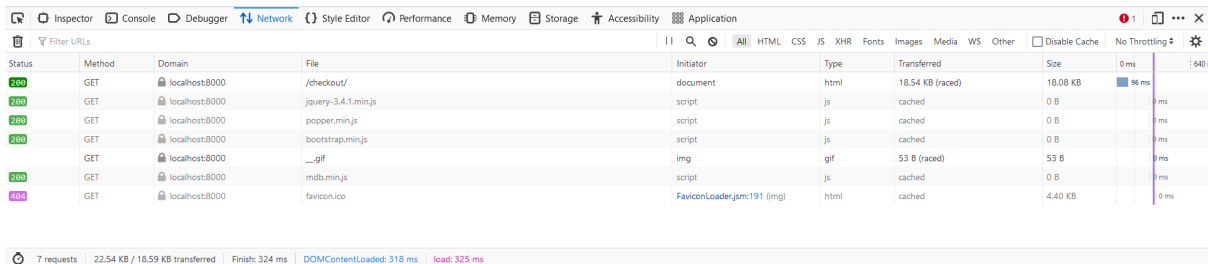


Figure 66 - Checkout page response times in milliseconds

Payment: 2.5s

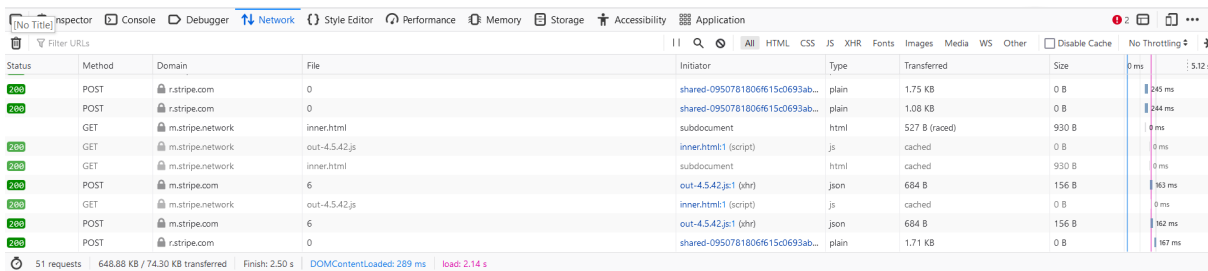


Figure 67 - Payment page response times in seconds

An extensive **test coverage** for the entire application exists, with 17 tests making up all the unit, integration and system tests that ensure the stability of the code. Although no CI/CD has been integrated, before any change to the codebase is made on GitHub, the test suite is run which acts as the regression tests for the application. The test suite takes under a minute to finish executing.

```
-----  
Ran 17 tests in 55.282s  
OK  
/Destroying test database for alias 'default'...  
  
(venv) D:\College\Final-Year\Software-Project\Mobile-Store>
```

Figure 68 - Test suite execution time in seconds

3.0 Conclusion

Overall, the project was a success. The user is able to browse various products on the mobile store and add/remove items to their cart. The system can adequately recommend phones based on their phone viewing history. The checkout is fully functional and the user can enter valid payment information to purchase their order.

The only requirements that could not be met from the template proposal were tracking the user's logout timestamp, promotional codes for discounts and using a user's past orders and not only their browsing history to influence the recommended phones. There are plenty of improvements that can be made, such as improving the recommendation system, adding more payment options, asynchronous requests when adding items to a cart and many more.

However, after having to change the project at the last minute to a default template when the original project could not be properly developed, the application was still able to meet most of the requirements with less development time being allocated towards it.

4.0 Further Development or Research

There are additional features that would have been ideal to implement into the project.

On the home page, there is a search feature that was never developed due to time constraints. Being able to filter the products by their categories could provide convenience to a user for when they are specifically looking for a single type of item to purchase.



Figure 69 - Product Filters on home page

More payment options such as PayPal or a direct credit card form handled by the applications own form logic would be a good feature to implement, rather than solely relying on the Stripe API for payments.

Further refinement on the artificial intelligence would go a long way in improving the recommendation system of the website. Perhaps using other data from users on the site rather than just the single user would provide much more accurate readings on which phones to recommend to the user. Adjusting the rating system on top of that would provide less skewed results, as setting an arbitrary rating of 1 to a phone whenever a user views the product information page does not hold up as one would expect.

Asynchronous item removal/adding to an order would improve the site's overall feel as each time an item is added or removed, the page must refresh, which leaves room for the possibility of a slow response time if the server has a bottleneck. More jQuery/AJAX calls would handle the user requests in the background without having the browser make a whole new request and redirect the user as a result.

Deploying the application to a hosted server could lead to even better response times than running the application locally. Server hardware can handle requests far better due to improved hardware and being dedicated to running the application rather than a single local machine hosting the app, which is constantly running other processes in the background and taking up precious resources.

Incorporating a much more intricate email system rather than using a free mail server is another possibility that could be explored, as it is quite limited with the fake email accounts that can be used and insecure since there is no authentication involved with sending or receiving emails. These fake email accounts only serve for testing purposes and perhaps being able to test emails with real email accounts would be ideal.

Multiple login systems such as Google, Facebook, Twitter etc. could be easily added into the website as Django is able to provide this seamless integration already with the documentation that is available to developers, rather than solely relying on its own allauth module.

5.0 References

- Bootstrap, 2022. *Introduction - Bootstrap*. [online] Getbootstrap.com. Available at: <<https://getbootstrap.com/docs/4.1/getting-started/introduction/>> [Accessed 1 January 2022].
- Django Software Foundation, 2022. *Django documentation | Django documentation | Django*. [online] Docs.djangoproject.com. Available at: <<https://docs.djangoproject.com/en/4.0/>> [Accessed 1 January 2022].
- Django Software Foundation, 2022. *Testing in Django | Django documentation | Django*. [online] Docs.djangoproject.com. Available at: <<https://docs.djangoproject.com/en/4.0/topics/testing/>> [Accessed 15 May 2022].
- GitHub Inc., 2022. *Getting started with GitHub - GitHub Docs*. [online] GitHub Docs. Available at: <<https://docs.github.com/en/get-started>> [Accessed 1 January 2022].
- Muthukadan, B., 2022. *Selenium with Python — Selenium Python Bindings 2 documentation*. [online] Selenium-python.readthedocs.io. Available at: <<https://selenium-python.readthedocs.io/>> [Accessed 15 May 2022].
- OpenJS Foundation, 2022. *jQuery*. [online] Jquery.com. Available at: <<https://jquery.com/>> [Accessed 1 January 2022].
- pawangfg, 2022. *Recommendation System in Python - GeeksforGeeks*. [online] GeeksforGeeks. Available at: <<https://www.geeksforgeeks.org/recommendation-system-in-python/>> [Accessed 15 May 2022].
- scikit-learn developers, 2022. *1.6. Nearest Neighbors*. [online] scikit-learn. Available at: <<https://scikit-learn.org/stable/modules/neighbors.html>> [Accessed 15 May 2022].
- Stripe, 2022. *Card payments on the Charges API*. [online] Stripe.com. Available at: <<https://stripe.com/docs/payments/charges-api>> [Accessed 15 May 2022].
- Stripe, 2022. *Testing*. [online] Stripe.com. Available at: <<https://stripe.com/docs/testing>> [Accessed 15 May 2022].
- WPOven, 2022. *Free SMTP Server for Testing*. [online] Wpoven.com. Available at: <<https://www.wpoven.com/tools/free-smtp-server-for-testing#>> [Accessed 15 May 2022].

6.0 Appendices

6.1 Project Proposal

National College of Ireland

Project Proposal phoneplex – Mobile Phone E-Commerce Website

05/11/2021

BSHC4

Software Development

Academic Year 2021/2022

Sorin Nechifor

x18393913

x18393913@student.ncirl.ie

Contents

6.1 Project Proposal	61
6.1.1 Objectives	64
6.1.2 Background	64
6.1.3 State of the Art	64
6.1.4 Technical Approach	64
6.1.5 Technical Details	65
6.1.6 Special Resources Required	65
6.1.7 Project Plan	66
6.1.8 Testing	67
6.2 Reflective Journals	67
October Monthly Report	67
November Monthly Report	67
December Monthly Report	68
January Monthly Report	68
February Monthly Report	68
March Monthly Report	68
April Monthly Report	69

6.1.1 Objectives

The main goal that phoneplex is attempting to achieve is to build a fully functional and advanced mobile store that will also incorporate A.I. that helps users in their decision-making when purchasing a mobile phone. As expected out of any modern E-Commerce website, users will be able to purchase a wide variety of mobile phones on the website and will also have phone recommendations displayed to them based on their browsing activity. Once the user selects a phone they wish to buy, the application will store that information for future use in case they wish to buy another phone.

When purchasing a phone, the application will also offer appropriate accessories to go along with that phone should the user wish to purchase them. After the user completes an order by submitting the correct credit card details, an email of the transaction will be sent to their email address that they provided to the application.

Lastly, the application will offer superuser permissions that will allow an admin to track when a user has visited the website, the products they viewed, their orders and when they've logged out.

6.1.2 Background

I decided to undertake this project as I have a huge interest in artificial intelligence and I've always had the desire to build a fully functional E-Commerce website with the Django framework. Combining these two interests will hopefully build up my skills with both Django and other web development frameworks as well as expanding my knowledge with artificial intelligence for web applications.

The company that I had my internship placement and am currently employed with also use Django for their product and hopefully by undertaking this project, I can hone my skills in the framework even further.

I intend to accomplish these objectives by improving my Django knowledge with the aid of their documentation and a plethora of Django tutorials that exist on YouTube.

6.1.3 State of the Art

Most applications that specialize in just a singular type of product should expect to see some competition from companies that hold monopolies in the industry already like Amazon. Amazon sell a wide range of products, not just mobile phones, which would be this application's biggest competitor.

Additionally, companies like Currys, Harvey Normans and Carphone Warehouse are essentially the same type of company that phoneplex is offering to be with an even wider range of electronics and other products available for purchase.

Phoneplex will try to emulate the online store that these companies would have on the internet, potentially being more visually pleasing to the consumer and easy to navigate.

6.1.4 Technical Approach

I hope to elicit some of the requirements of the project by talking to my supervisor and refining the them so that they are within scope to execute. As well as that, I will also seek to provide updated prototypes during development to my colleagues in an attempt to get feedback for my work in progress and make any necessary changes or additions based on their feedback.

I will attempt to adopt an agile approach to the development of the project by having weekly sprints to accomplish for a given functionality or feature of the software. Breaking up the development of the project into manageable chunks of work will be far easier to manage overtime than trying to program it as a whole. These deliverables will be based off of the requirements established.

The various milestones that will need to be fulfilled for an application like phoneplex will be based on a single requirement that the application needs. With said requirement, the various components that make up an activity e.g. shopping cart functionality, will have to be broken down into tasks such as creating a cart page, adding back-end functionality for storing order items, calculating order totals and removing said items from a cart.

Tracking the progress of the deliverables will be done by using a Gantt Chart, so that tasks can easily be entered onto the chart with their associated time estimate for completion.

6.1.5 Technical Details

Since this project creating a web application, a variety of languages and libraries are being used to build this E-Commerce store.

HTML, and CSS will be used for all the elements that will appear on the pages of the site as well as their styling. A CSS and JavaScript library known as Bootstrap is being used for the front-end of the website to aid with organizing HTML elements in a grid system, template styles and some custom JavaScript they have built. jQuery is another JavaScript library that is being utilised to enable phoneplex to be a dynamic web-page with the powerful manipulation of HTML elements. Python will be the programming language of choice for the back-end of the project, especially through the use of Django as the web framework of choice.

Django offers an abundance of features that will be instrumental to the application, such as built-in database functionality, great documentation on testing and a web-design architecture similar to having a Model, View and Controller but instead, referred to as Model, View and Template.

For the recommendation of phones based on user viewing and selection, Python will again be the main language used for the implementation of this basic A.I. Predominantly, if a user browses an Apple phone more often than an Android phone, they should expect to see more iPhone recommendations after prolonged use of the website. The opposite should be true for a user whose browsing history heavily lies in favour with Android phones.

A potential hosting site like Heroku may be used in the latter stages of development if deciding to deploy the application, and a database service like MongoDB or Azure are available options for a database when working with a Django application.

GitHub will be used as version control of the web application.

6.1.6 Special Resources Required

I do not believe any special resources is needed for this project, but if there are, they will be addressed immediately.

6.1.7 Project Plan

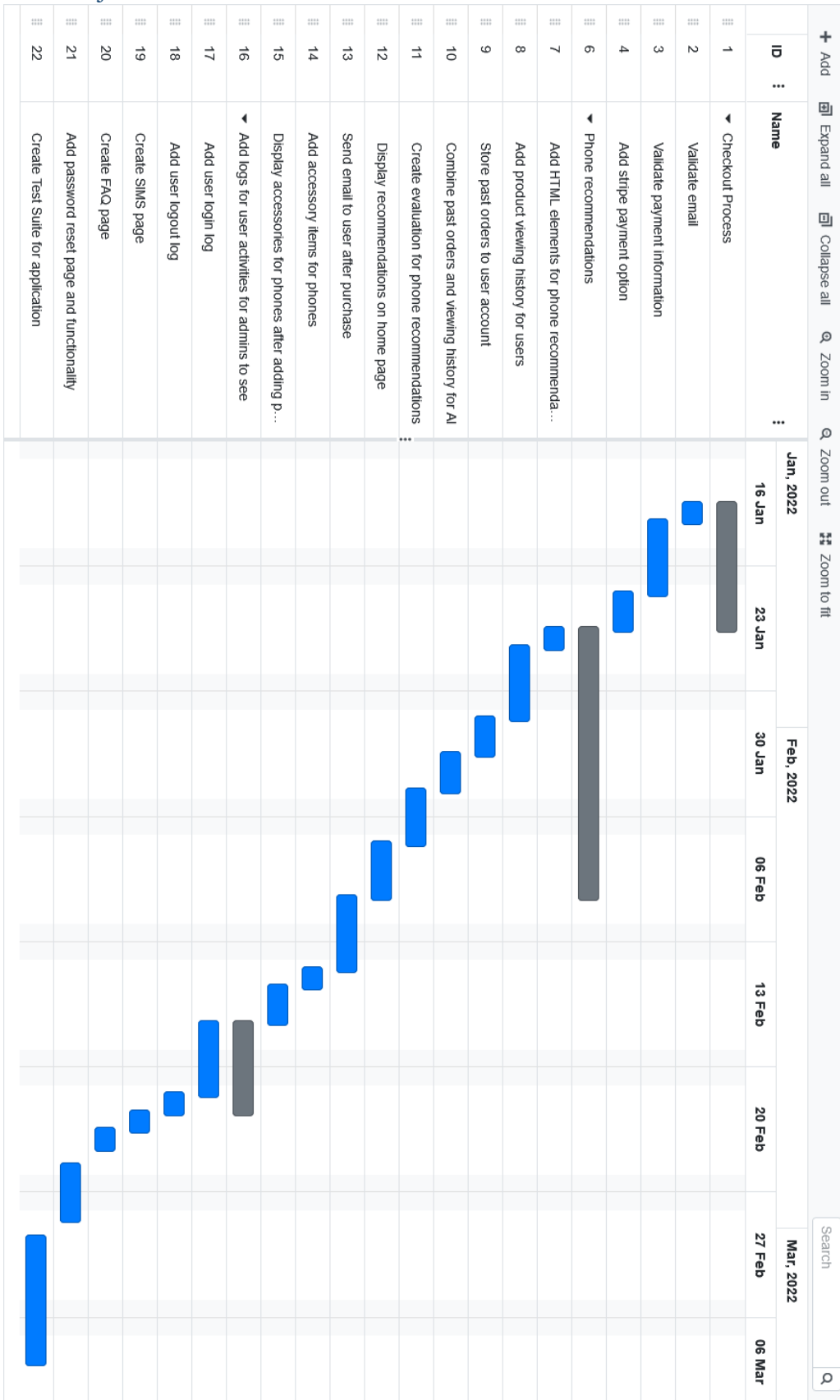


Figure 70 - Gantt Chart for tasks after mid-point implementation

6.1.8 Testing

Django provides developers the ability to write unit tests for their application which incorporates the pre-existing Python library called unittest. With the testing library provided by Django, a developer is able to test their models, views, templates, requests and forms used in their web application. This library will aid in allowing a developer like myself to test small components of the application, otherwise known as unit testing, to ensure that simple functions of the software is working as intended before adding more complexity and creating new integration tests using those units.

Models in Django contain information about the data that the web application is handling, and mock-up databases are able to be created during testing, which is great in separating the actual database used in the application so that no real damage is done in case a test fails. Using this mock database, models can be tested for extensively to ensure that objects are being created and that they are as expected. Views are what are typically regarded as the ‘controller’ for Django, and deal with the requests a user makes to the application and in turn, gives out responses. These requests and responses can be tested, ensuring that correct status codes are returned to the user and that the correct page/resource is delivered.

Using another library specifically designed for automated testing, such as Selenium, would further solidify the web application with the ability to run ‘regression tests’ that help to ensure that the web application hasn’t been broken during any update to the source code. This would ensure that the website is also as expected on the front-end by asserting certain web elements are present on the page.

Unit tests using Django’s library will form the core back-end testing of the web application by testing the Models and Views written during development, and with enough development time, Selenium will ensure the stability of the front-end.

All tests will be run by myself and my personal computer, without the use of an external user.

6.2 Reflective Journals

October Monthly Report

This month I finalized my idea which was to build an AI that can utilize machine learning to teach itself how to play a Pokémon game. I’ve investigated past projects and codebases that other people have coded, in order to get a better understanding of how this would be executed. The Pokémon game I have chosen for this specific project is Pokémon Emerald. Pokémon Emerald was developed for the Gameboy Advance consoles and the game is available in a ROM (read-only memory) format that can softwares can use to replicate playing the game and also accessing the internal memory of the game. Being able to readily access the memory will be a crucial aspect of this project, as this will most definitely tie into teaching the AI how to navigate through the world by reading obstacles and understanding which Pokémon have been sent out for battle and various other metrics used when deciding your next move in a Pokémon battle.

The project pitch video was also recorded this month and submitted. The next steps once I get approval is to immediately begin writing the proposal with the necessary requirements as well as drafting the use cases.

November Monthly Report

Not a whole lot of progress was made during the month of November. Juggling between the various assignments and my responsibilities outside of academia was very challenging and is the reason why I was not able to make great strides in developing the project. The draft project proposal was created this month and submitted to my supervisor, however, there is still more detail that needs to be added into the proposal to paint a clearer image of how the project will be executed and what exactly its deliverables are. Some light research regarding the application interface that will be implemented between the code written and the emulator was conducted, and a proof-of-concept is currently being developed with Python. Going forward, there should be more time to dedicate in December towards fulfilling a lot of the deliverables required for the Mid-Point submission.

December Monthly Report

A substantial amount of progress was done during the month of December. After realizing that my initial project idea of an A.I. for a Pokemon game was not remotely possible with the tools and time at my disposal, I had to switch to a template project idea on the provided by the lecturers. I chose a new project template which are approved and the template chosen was number 2, 'Advanced Mobile Store', which is a mobile phone E-Commerce website that used basic artificial intelligence to also recommend phones to its users.

Development speedily began on the application, with a ton of progress made thanks to an extension granted by the college. The login, register and cart functionality were all completed on time and the next phase in development is to begin working on the checkout for the web application. A lot of documentation was done for phoneplex, including a complete revamp of the project proposal and the mid-point technical report. Videos were recorded for the slides presentation and demonstration of the prototype. I have yet to speak to my supervisor about ethics concerns but resubmissions are available at different stages, leaving for flexibility.

January Monthly Report

This month, a short while after the end of semester one, I had begun working on the checkout process and that is being closed to finished. After the checkout process is complete, I'll set out to work on the discount coupons for the website that a user can apply to an order. Many of the deliverables have been refined and the Gantt chart for the project has been updated in order to reflect this, which paints a far more brighter development schedule and list of tasks to be completed. A lot of the visual work still remains to be done with updating product descriptions and their respective images but that will be closer to the end of development from my deliverable schedule.

February Monthly Report

This month, little progress has been made with the Software Project. Other assignments and responsibilities were given priority rather than development of the software. The goal of finishing the checkout process is still the next task that needs to be completed, and hopefully with the reading week in March, more progress can be made then.

March Monthly Report

Over the month, I've made little progress with the Software Project. Other assignments were prioritized over the project once again. The goal of completing the checkout process is still

the next task to be developed and I hope to after the last two remaining assignments to be submitted in April is complete, development can resume once more.

April Monthly Report

The checkout process has been completed for the project. What's left to complete is the phone recommendations based on purchasing history, the full test suite to be developed and user logs to be made available so that admins can see their browsing experience on the application. The documentation will also be worked on after the completion of the website.