# client code  Address.html

This code appears to be a template written in Django's template language. Django is a Python web framework, and templates like this are used to generate HTML content dynamically, typically within the context of a web application.

Let's break down the code step by step:

1. {% extends 'app/base.html' %}:
   - This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template.
2. {% load static %}:
   - This line is used to load the static files, such as CSS and JavaScript, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.
3. {% block title %}Address{% endblock title %}:
   - This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Address."
4. {% block main-content %} and {% endblock main-content %}:
   - These lines define a block named main-content, which will contain the main content of the page. Any content placed between these two block tags will replace the main-content block in the base template.
5. :
   - This starts an HTML div element with class attributes container and my-5. These classes are typically used for styling purposes in web development frameworks like Bootstrap.
6. Inside the container, there's a div element with the class row, which is part of the Bootstrap grid system.
7. <h3>Welcome {{request.user}}</h3>:
   - This line displays a welcome message with the username of the currently logged-in user. The {{request.user}} part fetches and displays the username.
8. :
   - This starts a Bootstrap column div with a width of 2 units on small screens (col-sm-2). It also adds a border-end class, which likely adds a right border to the column.
9. Inside this column div, there's an unordered list (<ul>) with two list items (<li>). Each list item contains a link (<a>) with an href attribute pointing to different URLs. These links are typically used for navigation within the web application.
10. :
    - This starts another Bootstrap column div with a width of 9 units on small screens (col-sm-9) and an offset of 1 unit (offset-sm-1). This means it's wider than the previous column and is indented by one column unit.
11. Inside this column div, there's a nested row ( ) for organizing content.
12. {% for ad in add %} and {% endfor %}:
    - This is a Django template for loop. It iterates over a list or queryset named add. For each item in add, it generates HTML content.
13. Inside the loop, it creates a Bootstrap card ( ) for each address (ad). The address details are displayed within the card's body.
14. The details of each address (ad) are displayed using HTML elements such as headings (<h3>) and paragraphs (<p>). The information is likely pulled from a database or context variable.

In summary, this Django template extends a base template, defines blocks for the title and main content, and displays a list of addresses in a structured format within a Bootstrap-based layout. The template is designed for a user profile page where the user can view their address information.

# AddtoCart.html

**This code is a Django template used to render a shopping cart page for an e-commerce website. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, the title block will be replaced with "Cart," and the main content block contains the content specific to the shopping cart page.**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Cart{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Cart."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the page. Any content placed between these two block tags will replace the main-content block in the base template.**
**{% comment %} Below Code will be used by add_to_cart view {% endcomment %}:**

**This is a comment in the template code. It's meant for documentation and is not rendered in the final HTML. It explains that the following code will be used by the add_to_cart view.**
**{% if messages %} and {% for message in messages %}:**

**These lines check if there are any messages to display. In Django, messages can be used to provide feedback or notifications to users. The code iterates through the messages and displays them with appropriate styling based on their tags (e.g., success, warning, error).**
**Inside the row div, there's a h1 element with the text "Shopping Cart," which is the main heading of the page.**

**The next div element has a class col-sm-8, indicating it occupies 8 out of 12 columns on small screens (Bootstrap grid system). This column contains the shopping cart items.**

**Inside the shopping cart column, a for loop iterates through the carts (presumably shopping cart items) and displays details about each item. It includes the product image, title, description, quantity, and price. The code also provides buttons to adjust the quantity or remove items from the cart.**

**The right-hand column (col-sm-4) displays the total amount, shipping cost, and the grand total of the items in the shopping cart. It includes a "Place Order" button that likely leads to the checkout page.**

**The template also includes a section at the bottom for displaying accepted payment methods using an image.**

**Overall, this template is used to render a shopping cart page, showing the items in the cart, their details, and the total amount. It also handles displaying messages to the user and provides options to adjust the cart contents and proceed to checkout.**

This code is an HTML template for a web page, and it appears to be used in a Django web application. Let's break down the code step by step:

# Base.html

{% load static %}:

This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files. This is commonly used in Django templates to ensure that static files are included in the rendered HTML.

<html lang="en">:

This is the opening tag for the HTML document. It sets the language attribute to "en" (English).

<head> Section:

The <head> section contains metadata and links to external resources used by the web page.

<meta charset="utf-8">: Specifies the character encoding for the document as UTF-8.

<meta name="viewport" content="width=device-width, initial-scale=1">: This meta tag is used for responsive web design. It ensures that the web page content is scaled appropriately on different devices.

Bootstrap CSS and Other Stylesheets:

Several <link> tags are used to include external CSS files. These CSS files are for styling the web page. They include Bootstrap CSS, Owl Carousel CSS, FontAwesome CSS, and a custom CSS file called "style.css."

<title>Fixer | {% block title %} {% endblock title %} </title>:

This sets the title of the web page to "Fixer" with a dynamic part enclosed in {% block title %}. The dynamic part can be overridden in templates that extend this one.

<body> Section:

The <body> section contains the main content of the web page.

<nav>: This is the navigation bar of the website. It uses Bootstrap classes for styling and includes links to various sections of the website.

User Authentication and Cart Links:

Depending on whether the user is authenticated (logged in) or not, different navigation links are displayed. If the user is authenticated, it shows links to the user's profile, orders, change password, and logout. If not, it shows links to login and registration.

{% block banner_slider %} {% endblock banner_slider %}:

These are block tags that allow extending templates to insert content in specific places. For example, the {% block banner_slider %} can be overridden in templates that extend this one to add a banner slider.

{% block livesale %} {% endblock livesale %}:

Similar to the previous block, this one can be overridden to add content related to live sales or promotions.

{% block main-content %} {% endblock main-content %}:

This is a block where the main content of the page will be placed. Content within this block can be overridden in templates that extend this one to provide specific page content.

{% block payment-gateway %} {% endblock payment-gateway %}:

Another block for potential content related to payment gateways.

Footer:

This section contains the website's footer. It includes a small copyright notice and possibly an image (the image source is currently empty).

JavaScript Libraries:

Several JavaScript libraries are included at the bottom of the page. These include jQuery, Bootstrap, Owl Carousel, FontAwesome, and a custom JavaScript file called "myscript.js." These scripts are used to enhance the functionality and interactivity of the website.

Overall, this HTML template provides the basic structure and styling for a web page. The actual content for specific sections (e.g., banner_slider, livesale, main-content) can be customized in templates that extend this one, making it a flexible and reusable template for a Django-based web application.

# Checkout.html

**This code is a Django template used for a checkout page in a web application. It extends a base template, which likely provides the overall structure and navigation elements for the website. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. Here, it sets the title block content to "Checkout" and provides the main content of the checkout page.**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Checkout{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Checkout."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the page. Any content placed between these two block tags will replace the main-content block in the base template. Inside the main-content block, the content for the checkout page is structured as follows:**

**There are two columns within a Bootstrap container. The first column (col-sm-6) contains the order summary, including details about the products in the cart, their quantities, and prices. It also displays the total cost, which includes a fixed delivery charge of Rs. 70.**

**The second column (col-sm-4 offset-sm-1) allows the user to select a shipping address from a list of addresses. It presents a form with radio buttons for each address. The user can choose the shipping address they want to use for this order. Below the address options, there's a "Continue" button to proceed with the checkout process.**

**Additionally, the script for rendering a PayPal payment button is included in this section. It sets up the PayPal button and handles the payment process when the user clicks the button.**

**{% block payment-gateway %}:**

**This is another block, but this time it is for content related to the payment gateway. It includes the JavaScript code for rendering a PayPal button for payment processing.**

**<script src="https://www.paypal.com/sdk/js?client-id=sb&currency=USD"></script>:**

**This line includes the PayPal JavaScript SDK, which is required to render the PayPal payment button.**
**The JavaScript code inside this block sets up the PayPal button using the SDK. It specifies how the transaction should be created and how to handle the approval of the transaction. When the user approves the transaction, it submits the form with the ID "myform."**

**Finally, there's a link that allows users to return to the home page after interacting with the payment gateway.**

**Overall, this template provides the structure and functionality for a checkout page in a web application. It allows users to review their order, select a shipping address, and make a payment using PayPal.**

# CustomerRegistration.html

**This code is a Django template used for a checkout page in a web application. It extends a base template, which likely provides the overall structure and navigation elements for the website. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. Here, it sets the title block content to "Checkout" and provides the main content of the checkout page.**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Checkout{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Checkout."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the page. Any content placed between these two block tags will replace the main-content block in the base template. Inside the main-content block, the content for the checkout page is structured as follows:**

**There are two columns within a Bootstrap container. The first column (col-sm-6) contains the order summary, including details about the products in the cart, their quantities, and prices. It also displays the total cost, which includes a fixed delivery charge of Rs. 70.**

**The second column (col-sm-4 offset-sm-1) allows the user to select a shipping address from a list of addresses. It presents a form with radio buttons for each address. The user can choose the shipping address they want to use for this order. Below the address options, there's a "Continue" button to proceed with the checkout process.**

**Additionally, the script for rendering a PayPal payment button is included in this section. It sets up the PayPal button and handles the payment process when the user clicks the button.**

**{% block payment-gateway %}:**

**This is another block, but this time it is for content related to the payment gateway. It includes the JavaScript code for rendering a PayPal button for payment processing.**

**<script src="https://www.paypal.com/sdk/js?client-id=sb&currency=USD"></script>:**

**This line includes the PayPal JavaScript SDK, which is required to render the PayPal payment button. The JavaScript code inside this block sets up the PayPal button using the SDK. It specifies how the transaction should be created and how to handle the approval of the transaction. When the user approves the transaction, it submits the form with the ID "myform."**

**Finally, there's a link that allows users to return to the home page after interacting with the payment gateway.**

**Overall, this template provides the structure and functionality for a checkout page in a web application. It allows users to review their order, select a shipping address, and make a payment using PayPal.**

# emptycart.html

**This code is a Django template used to render a page that informs the user that their shopping cart is empty. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Empty Cart" and provides the main content of the empty cart page.**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Empty Cart{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Empty Cart."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the page. Any content placed between these two block tags will replace the main-content block in the base template.**
**Inside the main-content block, the content for the empty cart page is structured as follows:**

**There's a Bootstrap container ( ) that centers the content vertically on the page.**

**Inside the container, there's a row ( ) that contains the main message for the user: "You have no Product in Your Cart."**

**Below the message, there's an image (<img>) displayed in the center of the page. The image source (src) is specified using the {% static 'app/images/emptycart.png' %} template tag, which generates the URL for the static image file.**

**The image has the classes img-fluid and img-thumbnail, which are Bootstrap classes used for responsive and bordered images, respectively. Additionally, it has w-25 class, which specifies its width as 25% of the container.**

**Overall, this template is used to display a simple message and image to inform the user that their shopping cart is empty. It inherits the overall structure and styling from the base template.**

# Home.html

**This code is a Django template used to render the home page of a web application. It extends a base template, meaning it inherits the overall structure and styling from the base.html template while allowing customization of specific content blocks. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Home."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Home{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Home."**
**{% block banner_slider %}:**

**This block is used to insert content for a banner slider. The content within this block will replace the banner_slider block in the base template.**

**Inside this block, a Bootstrap carousel ( ) is defined with multiple carousel items, each containing an image (<img>) sourced from static files. This creates a slideshow of images.**

**{% block livesale %}:**

**This block is used to insert content related to a live sale section. The content within this block will replace the livesale block in the base template.**

**Inside this block, there's a container with a red background and white text that displays a sale announcement along with terms and conditions.**

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

**These lines define a block named main-content, which will contain the main content of the home page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the home page is structured as follows:**

**There's a section for the 1st product slider, which displays fashion wear products. It uses an Owl Carousel to create a carousel of product items.**

**After the 1st product slider, there's a section for displaying payment information. It shows different payment methods with associated images.**

**Next, there's a section for the 2nd product slider, which displays more fashion wear products.**

**Finally, there's a section for the 3rd product slider, which displays mobile products.**

**Inside the product sliders, {% for ... %} loops are used to iterate through product data and generate dynamic content for each product item. Each product item includes an image, product title, and discounted price. The href attribute of the anchor tags (<a>) links to the product detail page.**

**Overall, this template provides the structure for the home page of a web application, including a banner slider, live sale section, and product sliders. It allows for dynamic content generation based on the available products in each category. The base template provides the overall structure and styling for the entire website, while this template fills in the specific content for the home page.**

# login.html

**This code is a Django template used to render a login page for a web application. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Login."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Login{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Login."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the login page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the login page is structured as follows:**

**There's a Bootstrap container ( ) that centers the content vertically on the page.**

**Inside the container, there's a row ( ) with a column ( ) that contains the login form.**

**The login form includes the following elements:**

**A heading "Login."**
**A horizontal line (<hr>).**
**A <form> element with the action attribute left empty (meaning it submits to the current URL), the method set to "post," and the novalidate attribute (which disables browser validation).**
**A {% csrf_token %} template tag to include a CSRF token for security.**
**A {% for fm in form %} loop that iterates through the form fields. For each form field, it displays the label, input field, and any associated validation errors. The {{fm.label_tag}} tag generates the label for the field, {{fm}} generates the input field, and {{fm.errors|striptags}} displays any validation errors in red text.**
**A "Forgot Password?" link that redirects to the password reset page using the {% url 'password_reset' %} template tag.**
**A "Login" button to submit the form.**
**A "New to Fixer?" message with a link to create a new account using the {% url 'customerregistration' %} template tag.**
**Finally, there's a conditional check for form.non_field_errors, which displays any non-field errors (e.g., authentication errors) as alert messages.**

**Overall, this template provides the structure for the login page, including the login form, links for password reset and account creation, and the use of Bootstrap for styling. It allows users to enter their login credentials and submit the form to log in to the application.**

# mobile.html

**This code is a Django template used to render a mobile products page for a web application. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Mobile."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Mobile{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Mobile."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the mobile products page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the mobile products page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with two columns ( and ) for navigation and product display.**

**The left column (col-sm-3) contains a list group ( ) with navigation links:**

**"All Mobile" links to the page displaying all mobile products.**
**"Redmi" links to the page displaying Redmi mobile products.**
**"Samsung" links to the page displaying Samsung mobile products.**
**"Below 10000" links to the page displaying mobile products with prices below 10,000.**
**"Above 10000" links to the page displaying mobile products with prices above 10,000.**
**The right column (col-sm-8) contains a row ( ) with product items. Inside a {% for product in mobiles %} loop, it iterates through the mobile products and generates dynamic content for each product item.**

**Each product item is displayed as a Bootstrap card ( ) with an anchor (<a>) wrapping the product details.**
**The anchor links to the product detail page using the {% url 'product-detail' product.id %} template tag.**
**Inside each card, there's an image of the product, its title, discounted price, and the original selling price (with a line-through). This information is retrieved from the product object in the loop.**
**Overall, this template provides the structure for the mobile products page, including navigation links and a grid of mobile products. Users can click on the links to filter mobile products by different criteria, such as brand or price range.**

# orders.html

**This code is a Django template used to render the "Orders" page for a web application. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Orders."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Orders{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Orders."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the Orders page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the Orders page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with two columns ( and ) for navigation and displaying order details.**

**The left column (col-sm-2) contains a list of navigation links. In this example, there's only one link for "Orders," which is currently active (<a href="{% url 'orders' %}" class="btn btn-primary">Orders</a>). This link corresponds to the current page.**

**The right column (col-sm-9 offset-sm-1) contains a loop that iterates through the order_placed objects. This loop displays order details for each order placed by the user.**

**For each order (op), it displays the following information:**

**An image of the ordered product.**
**Product details such as title, quantity, and total cost.**
**Order status information (op.status) along with a progress bar. The progress bar style and width vary based on the order status. Four possible order statuses are considered: "Accepted," "Packed," "On The Way," and "Delivered." Depending on the status, the progress bar's appearance changes accordingly.**
**The template generates a list of orders with their details and corresponding progress bars to indicate the order's status. This provides users with an overview of their order history and the current status of each order.**

# passwordresetcomplete.html

**This code is a Django template used to render a page that confirms the completion of a password reset process. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Password Reset Complete."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Password Reset Complete{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Password Reset Complete."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the "Password Reset Complete" page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the "Password Reset Complete" page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with one column ( ) for displaying the password reset completion message and an animated GIF.**

**The content in this column includes:**

**A heading (<h4>Reset Password Completed!</h4>) that informs the user that the password reset process is complete.**

**An embedded GIF (animated image) that provides a visual element to indicate the completion of the process. The GIF is embedded using an <iframe> element with the source URL pointing to a Giphy animation.**

**The template's main purpose is to provide a confirmation message and a visual cue to the user that their password reset process has been successfully completed.**

# passwordresetconfirm.html

**This code is a Django template used to render a page for confirming a password reset. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Password Reset Confirm."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Password Reset Confirm{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Password Reset Confirm."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the "Password Reset Confirm" page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the "Password Reset Confirm" page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with one column ( ) for displaying the password reset confirmation form.**

**The content in this column includes:**

**A heading (<h3>Password Reset Confirm</h3>) that informs the user about the purpose of the page.**

**A horizontal line (<hr>) to separate the heading from the form.**

**A form element (<form>) with the following attributes:**

**action="": The form action is empty, which means it will submit the form to the current URL.**
**method="post": The form will use the HTTP POST method for submission.**
**novalidate: This attribute disables HTML5 form validation.**
**{% csrf_token %}: This template tag generates a hidden input field containing a CSRF token for security.**

**A for loop that iterates over the form fields ({% for fm in form %}) and generates form inputs based on the form's fields. Each form field is enclosed in a div with a class of form-group. It displays the field's label, the form input, and any validation errors in case of submission errors.**

**An "Submit" button (<input type="submit">) for submitting the form.**

**A conditional check that displays any non-field errors (e.g., form-level errors) as alert messages.**

**The template's main purpose is to display a form for the user to confirm their password reset request. Users can input the necessary information, and the form can be submitted for further processing.**

# passwordresetdone.html

**This code is a Django template used to render a page that informs the user that the password reset process has been completed successfully. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Password Reset Done."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Password Reset Done{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Password Reset Done."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the "Password Reset Done" page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the "Password Reset Done" page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with one column ( ) for displaying the success message.**

**The content in this column includes:**

**A heading (<h4 class="alert-heading">Well done!</h4>) that serves as a visual indicator of success.**

**A paragraph (<p>) that provides a message to the user, stating that an email has been sent with instructions to reset their password. The message also includes the instruction to check their email.**

**The template's main purpose is to provide feedback to the user that the password reset process has been successfully initiated and that they should check their email for further instructions on resetting their password. It's a simple confirmation page with a success message.**

# passwordreset.html

**This code is a Django template used to render a page for resetting a user's password. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Reset Password."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Reset Password{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Reset Password."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the "Reset Password" page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the "Reset Password" page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with one column ( ) for displaying the password reset form.**

**The content in this column includes:**

**A heading (<h3>Reset Password</h3>) that indicates the purpose of the page.**

**A horizontal line (<hr>) for visual separation.**

**A form (<form>) for resetting the password. The form includes:**

**A CSRF token ({% csrf_token %}) to protect against cross-site request forgery attacks.**

**A loop ({% for fm in form %}) that iterates through the form's fields. For each field, it displays the field's label and the field itself, along with any validation errors.**

**An input field (<input type="submit">) for submitting the form with the label "Submit."**

**A section for displaying non-field errors (e.g., form-level errors) using a loop ({% if form.non_field_errors %}) and displaying them as alert messages.**

**The template's main purpose is to render a password reset form that allows users to submit a new password. Users will see this page after clicking a password reset link received via email. They can enter their new password and submit the form to complete the password reset process.**

# passwordchange.html

**This code is a Django template used to render a page for changing a user's password. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Change Password."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Change Password{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Change Password."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the "Change Password" page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the "Change Password" page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with two columns:**

**The first column ( ) contains a list of links (in this case, just one link for changing the password). The link is wrapped in a button element.**

**The second column ( ) is for displaying the password change form.**

**In the second column, the content includes:**

**A heading (<h3>Change Password</h3>) that indicates the purpose of the page.**

**A horizontal line (<hr>) for visual separation.**

**A form (<form>) for changing the password. The form includes:**

**A CSRF token ({% csrf_token %}) to protect against cross-site request forgery attacks.**

**A loop ({% for fm in form %}) that iterates through the form's fields. For each field, it displays the field's label and the field itself, along with any validation errors.**

**An input field (<input type="submit">) for submitting the form with the label "Save."**

**A section for displaying non-field errors (e.g., form-level errors) using a loop ({% if form.non_field_errors %}) and displaying them as alert messages.**

**The template's main purpose is to render a password change form that allows users to submit a new password. Users will see this page when they want to update their password. They can enter their current password and the new password, then submit the form to save the changes.**

# passwordchanged.html

**This code is a Django template used to render a page confirming that a user has successfully changed their password. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Change Password Done."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Change Password Done{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Change Password Done."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the "Change Password Done" page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the "Change Password Done" page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with two columns:**

**The first column ( ) contains a list of links (in this case, just one link for navigating back to the "Change Password" page). The link is wrapped in a button element.**

**The second column ( ) is for displaying a confirmation message and an embedded GIF.**

**In the second column, the content includes:**

**A heading (<h3>Password Changed Successfully !!</h3>) indicating that the password change was successful.**

**An embedded GIF (<iframe>) that displays a congratulatory animation. The GIF is sourced from an external website.**

**The template's main purpose is to provide feedback to the user that their password change was successful. It also includes a link/button for the user to navigate back to the "Change Password" page or any other relevant actions.**

# passworddetails.html

**This code is a Django template used to render a product detail page. It provides information about a specific product, such as its title, description, price, and available offers. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Product Detail."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Product Detail{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Product Detail."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the product detail page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the product detail page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with two columns:**

**The first column ( ) displays the product's image as an <img> element. The image source is obtained from the product.product_image.url attribute.**

**The second column ( ) contains detailed information about the product, including:**

**The product's title (<h2>{{product.title}}</h2>).**

**A horizontal line (<hr>) for visual separation.**

**The product's description ( {{product.description}} ).**

**The product's price, with the discounted price displayed in bold and the original selling price with a strikethrough style (<h4>Rs. {{product.discounted_price}} <small class="fw-light text-decoration-line-through">{{product.selling_price}}</small></h4>).**

**Conditional rendering of buttons based on whether the user is authenticated:**

**If the user is authenticated, it checks if the product is already in the cart (item_already_in_cart). If it is, it displays a "Go to Cart" button that links to the cart page. Otherwise, it shows an "Add to Cart" button that allows the user to add the product to the cart.**
**If the user is not authenticated, it displays an "Add to Cart" button without checking whether the product is already in the cart.**
**A "Buy Now" button that links to the checkout page (<a href="{% url 'checkout' %}" class="btn btn-danger shadow px-5 py-2 ms-4">Buy Now</a>).**

**A list of available offers displayed as an unordered list (<ul>) with several list items (<li>).**

**This template is designed to present detailed information about a product and provide actions for users to either add it to their cart or proceed with the purchase (Buy Now). The template handles conditional rendering of buttons based on user authentication status and whether the product is already in the cart.**

# profile.html

**This code is a Django template used to render a user profile page. It allows users to view and update their profile information. Let's break down the code step by step:**

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

**This line indicates that this template extends another template named 'app/base.html'. It means that this template will inherit the structure and blocks defined in the base.html template. Any content defined in this template will replace the corresponding blocks in the base template. In this case, it sets the title block content to "Profile."**
**{% load static %}:**

**This line is used to load the static files, such as CSS, JavaScript, and images, for this template. It allows you to use {% static 'filename' %} later in the template to generate URLs for static files.**
**{% block title %}Profile{% endblock title %}:**

**This code defines a block named title. Blocks are used to define sections of the template that can be overridden in templates that extend this one. Here, it sets the title block content to "Profile."**
**{% block main-content %} and {% endblock main-content %}:**

**These lines define a block named main-content, which will contain the main content of the user profile page. Any content placed between these two block tags will replace the main-content block in the base template.**

**Inside the main-content block, the content for the user profile page is structured as follows:**

**There's a Bootstrap container ( ) that adds spacing and margins around the content.**

**Inside the container, there's a row ( ) with two columns:**

**The first column ( ) contains a list of navigation buttons for different profile-related pages, such as "Profile" and "Address." These buttons are created using anchor (<a>) tags.**

**The second column ( ) contains a form that allows the user to update their profile information. The form's action attribute is empty, indicating that it will be submitted to the current URL (the same page). It uses the POST method to submit the form data securely.**

**Inside the form, there's a CSRF token ({% csrf_token %}) for security.**

**A for loop iterates over the form's fields ({% for fm in form %}) and displays each form field with its label and input element.**

**A "Submit" button allows the user to submit the form.**

**Below the form, there's a conditional check for Django messages ({% if messages %}). If there are any messages, such as success or error messages, they are displayed here. The loop iterates over the messages and displays them with appropriate styles based on their tags.**

**This template provides a user-friendly interface for viewing and updating user profile information and includes navigation to other profile-related pages.**

# code for database 0001_initial.py

**This code represents a Django database migration file. It's automatically generated by Django when you create a new model or modify an existing one. The purpose of this code is to define the database schema for your application, including tables and their fields, relationships between tables, and initial data migrations. Let's break down this code step by step:**

**from django.conf import settings:**

**This line imports the Django settings module, which allows access to various project settings, including the user model specified in the project settings.**
**from django.db import migrations, models:**

**These lines import the migrations and models modules from Django's database-related components.**
**class Migration(migrations.Migration)::**

**This class represents a database migration and inherits from migrations.Migration. A migration is a versioned set of changes to your database schema.**

**initial = True: This line indicates that this migration represents the initial state of the database schema. It's typically created when you first define your models.**

**dependencies = [migrations.swappable_dependency(settings.AUTH_USER_MODEL),]:**

**This list specifies the dependencies of this migration. In this case, it depends on the user model specified in the project settings. This means that this migration should be applied after the user model is created or modified.**
**operations = [...]:**

**This list contains a series of database schema operations that define the structure of the database tables. Each operation corresponds to creating a table, adding a field, creating a foreign key relationship, and so on.**

**Here's a breakdown of the operations:**

**migrations.CreateModel(name='Cart', ...):**

**This operation creates a database table named Cart. It defines two fields: id (an auto-incremented primary key) and quantity (an integer field with a default value of 1).**
**migrations.CreateModel(name='Product', ...):**

**This operation creates a database table named Product. It defines several fields such as title, selling_price, discounted_price, description, brand, category, and product_image.**
**migrations.CreateModel(name='OrderPlaced', ...):**

**This operation creates a database table named OrderPlaced. It defines fields for order-related information such as ordered_date, status, and foreign key relationships to the Cart and User models.**
**migrations.CreateModel(name='Customer', ...):**

**This operation creates a database table named Customer. It defines fields for customer-related information, including address, city, zipcode, state, and a one-to-one relationship with the User model.**
**migrations.AddField(model_name='cart', name='product', ...):**

**This operation adds a foreign key field named product to the Cart model. It establishes a relationship between a cart and a product.**
**migrations.AddField(model_name='cart', name='user', ...):**

**This operation adds a foreign key field named user to the Cart model, establishing a relationship between a cart and a user. These operations collectively define the database schema for your Django application, including the tables and their relationships. When you run migrations, Django will apply these changes to your database, creating or modifying tables as necessary to match the defined schema.**

# 0002_auto_20210113_1412.py

**This code represents another Django database migration file. It's automatically generated by Django when you create or modify a model. This specific migration appears to be an incremental migration to an existing schema. Let's break down this code:**

**from django.db import migrations, models:**

**These lines import the migrations and models modules from Django's database-related components.**
**class Migration(migrations.Migration)::**

**This class represents a database migration and inherits from migrations.Migration. It's a part of a sequence of migrations that modify the database schema incrementally.**
**dependencies = [('app', '0001_initial'),]:**

**This list specifies the dependencies of this migration. In this case, it depends on a previous migration named '0001_initial' from an app named 'app'. This means that this migration should be applied after the initial migration for the 'app' app.**
**operations = [...]:**

**This list contains a series of database schema operations that modify the structure of the database tables. Each operation corresponds to renaming a field and adding a new field to an existing model.**

**Here's a breakdown of the operations:**

**migrations.RenameField(model_name='customer', old_name='address', new_name='locality'):**

**This operation renames a field in the Customer model. It renames the field from address to locality. This is a common operation when you want to change the name of a field in your database schema.**
**migrations.AddField(model_name='customer', name='name', ...):**

**This operation adds a new field named name to the Customer model. The field is of type CharField with a default value of 'Sonam' and a maximum length of 200 characters. The preserve_default parameter is set to False, indicating that the default value should not be preserved for existing records.**
**These operations are applied sequentially to the existing database schema when you run the migration. The first operation renames a field, and the second operation adds a new field to the Customer model in the database.**

# 0003_auto_20210113_1412.py

```
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('app', '0002_auto_20210113_1412'),
    ]

    operations = [
        migrations.AlterField(
            model_name='customer',
            name='city',
            field=models.CharField(blank=True, max_length=50),
        ),
        migrations.AlterField(
            model_name='customer',
            name='locality',
            field=models.CharField(blank=True, max_length=200),
        ),
        migrations.AlterField(
            model_name='customer',
            name='name',
            field=models.CharField(blank=True, max_length=200),
        ),
        migrations.AlterField(
            model_name='customer',
            name='state',
            field=models.CharField(blank=True, choices=[('Andaman & Nicobar Islands', 'Andaman &
Nicobar Islands'), ('Andhra Pradesh', 'Andhra Pradesh'), ('Arunachal Pradesh', 'Arunachal Pradesh'),
('Assam', 'Assam'), ('Bihar', 'Bihar'), ('Chandigarh', 'Chandigarh'), ('Chhattisgarh', 'Chhattisgarh'),
('Dadra & Nagar Haveli', 'Dadra & Nagar Haveli'), ('Daman and Diu', 'Daman and Diu'), ('Delhi', 'Delhi'),
('Goa', 'Goa'), ('Gujarat', 'Gujarat'), ('Haryana', 'Haryana'), ('Himachal Pradesh', 'Himachal Pradesh'),
('Jammu & Kashmir', 'Jammu & Kashmir'), ('Jharkhand', 'Jharkhand'), ('Karnataka', 'Karnataka'),
('Kerala', 'Kerala'), ('Lakshadweep', 'Lakshadweep'), ('Madhya Pradesh', 'Madhya Pradesh'),
('Maharashtra', 'Maharashtra'), ('Manipur', 'Manipur'), ('Meghalaya', 'Meghalaya'), ('Mizoram',
'Mizoram'), ('Nagaland', 'Nagaland'), ('Odisha', 'Odisha'), ('Puducherry', 'Puducherry'), ('Punjab',
'Punjab'), ('Rajasthan', 'Rajasthan'), ('Sikkim', 'Sikkim'), ('Tamil Nadu', 'Tamil Nadu'), ('Telangana',
'Telangana'), ('Tripura', 'Tripura'), ('Uttarakhand', 'Uttarakhand'), ('Uttar Pradesh', 'Uttar Pradesh'),
('West Bengal', 'West Bengal')], max_length=50),
        ),
        migrations.AlterField(
            model_name='customer',
            name='zipcode',
            field=models.IntegerField(blank=True),
        ),
    ]
```

# 0004_auto_20210113_1412.py

```
# Generated by Django 3.1.4 on 2021-01-13 10:17

from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('app', '0003_auto_20210113_1545'),
    ]

    operations = [
        migrations.AlterField(
            model_name='customer',
            name='city',
            field=models.CharField(blank=True, default='', max_length=50),
        ),
        migrations.AlterField(
            model_name='customer',
            name='locality',
            field=models.CharField(blank=True, default='', max_length=200),
        ),
        migrations.AlterField(
            model_name='customer',
            name='name',
            field=models.CharField(blank=True, default='', max_length=200),
        ),
        migrations.AlterField(
            model_name='customer',
            name='state',
            field=models.CharField(blank=True, choices=[('Andaman & Nicobar Islands', 'Andaman & Nicobar Islands'), ('Andhra Pradesh', 'Andhra Pradesh'), ('Arunachal Pradesh', 'Arunachal Pradesh'), ('Assam', 'Assam'), ('Bihar', 'Bihar'), ('Chandigarh', 'Chandigarh'), ('Chhattisgarh', 'Chhattisgarh'), ('Dadra & Nagar Haveli', 'Dadra & Nagar Haveli'), ('Daman and Diu', 'Daman and Diu'), ('Delhi', 'Delhi'), ('Goa', 'Goa'), ('Gujarat', 'Gujarat'), ('Haryana', 'Haryana'), ('Himachal Pradesh', 'Himachal Pradesh'), ('Jammu & Kashmir', 'Jammu & Kashmir'), ('Jharkhand', 'Jharkhand'), ('Karnataka', 'Karnataka'), ('Kerala', 'Kerala'), ('Lakshadweep', 'Lakshadweep'), ('Madhya Pradesh', 'Madhya Pradesh'), ('Maharashtra', 'Maharashtra'), ('Manipur', 'Manipur'), ('Meghalaya', 'Meghalaya'), ('Mizoram', 'Mizoram'), ('Nagaland', 'Nagaland'), ('Odisha', 'Odisha'), ('Puducherry', 'Puducherry'), ('Punjab', 'Punjab'), ('Rajasthan', 'Rajasthan'), ('Sikkim', 'Sikkim'), ('Tamil Nadu', 'Tamil Nadu'), ('Telangana', 'Telangana'), ('Tripura', 'Tripura'), ('Uttarakhand', 'Uttarakhand'), ('Uttar Pradesh', 'Uttar Pradesh'), ('West Bengal', 'West Bengal')], default='', max_length=50),
        ),
        migrations.AlterField(
            model_name='customer',
            name='zipcode',
            field=models.IntegerField(blank=True, default=''),
        ),
    ]
```

# 0005_auto_20210113_1412.py

```
# Generated by Django 3.1.4 on 2021-01-13 10:39

from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('app', '0004_auto_20210113_1547'),
    ]

    operations = [
        migrations.AlterField(
            model_name='customer',
            name='city',
            field=models.CharField(max_length=50),
        ),
        migrations.AlterField(
            model_name='customer',
            name='locality',
            field=models.CharField(max_length=200),
        ),
        migrations.AlterField(
            model_name='customer',
            name='name',
            field=models.CharField(max_length=200),
        ),
        migrations.AlterField(
            model_name='customer',
            name='state',
            field=models.CharField(choices=[('Andaman & Nicobar Islands', 'Andaman & Nicobar Islands'),
('Andhra Pradesh', 'Andhra Pradesh'), ('Arunachal Pradesh', 'Arunachal Pradesh'), ('Assam', 'Assam'),
('Bihar', 'Bihar'), ('Chandigarh', 'Chandigarh'), ('Chhattisgarh', 'Chhattisgarh'), ('Dadra & Nagar
Haveli', 'Dadra & Nagar Haveli'), ('Daman and Diu', 'Daman and Diu'), ('Delhi', 'Delhi'), ('Goa', 'Goa'),
('Gujarat', 'Gujarat'), ('Haryana', 'Haryana'), ('Himachal Pradesh', 'Himachal Pradesh'), ('Jammu &
Kashmir', 'Jammu & Kashmir'), ('Jharkhand', 'Jharkhand'), ('Karnataka', 'Karnataka'), ('Kerala',
'Kerala'), ('Lakshadweep', 'Lakshadweep'), ('Madhya Pradesh', 'Madhya Pradesh'), ('Maharashtra',
'Maharashtra'), ('Manipur', 'Manipur'), ('Meghalaya', 'Meghalaya'), ('Mizoram', 'Mizoram'), ('Nagaland',
'Nagaland'), ('Odisha', 'Odisha'), ('Puducherry', 'Puducherry'), ('Punjab', 'Punjab'), ('Rajasthan',
'Rajasthan'), ('Sikkim', 'Sikkim'), ('Tamil Nadu', 'Tamil Nadu'), ('Telangana', 'Telangana'), ('Tripura',
'Tripura'), ('Uttarakhand', 'Uttarakhand'), ('Uttar Pradesh', 'Uttar Pradesh'), ('West Bengal', 'West
Bengal')], max_length=50),
        ),
        migrations.AlterField(
            model_name='customer',
            name='zipcode',
            field=models.IntegerField(),
        ),
    ]
```

# 0006_auto_20210113_1412.py

**This is another Django database migration file that appears to be generated as part of the database schema evolution process. Let's break down the code:**

**from django.conf import settings:**

**This line imports the settings module from Django. The settings module allows access to various Django settings and configurations.**
**from django.db import migrations, models:**

**These lines import the migrations and models modules from Django's database-related components.**
**import django.db.models.deletion:**

**This line imports a specific module, deletion, from Django's database models package. This module is often used when defining relationships between models, especially for handling what happens when related objects are deleted.**
**class Migration(migrations.Migration)::**

**This class represents a database migration and inherits from migrations.Migration. It indicates that this file is a part of a sequence of migrations that modify the database schema incrementally.**
**dependencies = [...]:**

**This list specifies the dependencies of this migration. It depends on two things:**
**settings.AUTH_USER_MODEL: This represents the user model used in the Django project, which can be customized in the project settings.**
**'app', '0005_auto_20210113_1609': This migration depends on another migration named '0005_auto_20210113_1609' from an app named 'app'. This means that this migration should be applied after the specified migration from the 'app' app.**
**operations = [...]:**

**This list contains a single database schema operation.**

**migrations.AlterField(model_name='customer', name='user', field=...):**

**This operation alters an existing field in the Customer model. It changes the user field by modifying its ForeignKey relationship. Specifically, it sets the on_delete parameter to django.db.models.deletion.CASCADE, which means that when a related user is deleted, the corresponding Customer record will also be deleted (CASCADE).**
**This migration ensures that the relationship between the Customer model and the user model (defined in the user field) is configured to perform a cascading delete when a user is deleted, ensuring data consistency.**

# 0007_auto_20210113_1412.py

**This Django migration file appears to be generated as part of the database schema evolution process. Let's break down the code:**

**import django.core.validators:**

**This line imports the validators module from Django's core, which provides a set of pre-defined validators that can be used to validate model fields.**
**from django.db import migrations, models:**

**These lines import the migrations and models modules from Django's database-related components.**
**class Migration(migrations.Migration)::**

**This class represents a database migration and inherits from migrations.Migration. It indicates that this file is a part of a sequence of migrations that modify the database schema incrementally.**
**dependencies = [...]:**

**This list specifies the dependencies of this migration. It depends on another migration named '0006_auto_20210113_2006' from the 'app' app. This means that this migration should be applied after the specified migration.**
**operations = [...]:**

**This list contains a single database schema operation.**

**migrations.AlterField(model_name='cart', name='quantity', field=...):**

**This operation alters an existing field in the Cart model. It specifically modifies the quantity field by altering its attributes.**

**models.PositiveIntegerField(default=1, validators=[django.core.validators.MinValueValidator(1)]):**

**This part of the operation changes the quantity field to be a PositiveIntegerField, which means it should only store positive integer values.**

**default=1: It sets a default value of 1 for the quantity field. This means that if a Cart object is created without explicitly specifying a quantity, it will default to 1.**

**validators=[django.core.validators.MinValueValidator(1)]: It adds a validator to the field. This validator ensures that the value of the quantity field is at least 1. It prevents negative or zero values for the quantity field.**

**This migration is altering the Cart model to enforce constraints on the quantity field, ensuring that it always contains positive integer values with a minimum value of 1.**

# 0008_auto_20210113_1412.py

**This Django migration file appears to be generated as part of the database schema evolution process. Let's break down the code:**

**from django.db import migrations, models:**

**These lines import the migrations and models modules from Django's database-related components.**
**class Migration(migrations.Migration)::**

**This class represents a database migration and inherits from migrations.Migration. It indicates that this file is a part of a sequence of migrations that modify the database schema incrementally.**
**dependencies = [...]:**

**This list specifies the dependencies of this migration. It depends on another migration named '0007_auto_20210117_2100' from the 'app' app. This means that this migration should be applied after the specified migration.**
**operations = [...]:**

**This list contains a single database schema operation.**

**migrations.AlterField(model_name='cart', name='quantity', field=...):**

**This operation alters an existing field in the Cart model. It specifically modifies the quantity field by altering its attributes.**

**models.PositiveIntegerField(default=1): This part of the operation changes the quantity field to be a PositiveIntegerField, which means it should only store positive integer values.**

**default=1: It sets a default value of 1 for the quantity field. This means that if a Cart object is created without explicitly specifying a quantity, it will default to 1.**
**This migration is altering the Cart model by changing the quantity field to enforce constraints that ensure it always contains a positive integer value with a default value of 1. It seems similar to the previous migration, but it does not include a validator for a minimum value of 1. Instead, it relies on the positive integer field's inherent constraint.**

# 0009_auto_20210113_1412.py

**This Django migration file represents changes to the database schema, and it appears to be related to the OrderPlaced model. Let's break down the code:**

**from django.db import migrations, models:**

**These lines import the migrations and models modules from Django's database-related components.**
**class Migration(migrations.Migration)::**

**This class represents a database migration and inherits from migrations.Migration. It indicates that this file is part of a sequence of migrations that modify the database schema incrementally.**
**dependencies = [...]:**

**This list specifies the dependencies of this migration. It depends on another migration named '0008_auto_20210117_2101' from the 'app' app. This means that this migration should be applied after the specified migration.**
**operations = [...]:**

**This list contains several database schema operations:**

**migrations.RemoveField(model_name='orderplaced', name='cart'):**

**This operation removes the cart field from the OrderPlaced model.**
**migrations.AddField(model_name='orderplaced', name='product', field=...):**

**This operation adds a new field named product to the OrderPlaced model.**
**models.ForeignKey(default=1, on_delete=django.db.models.deletion.CASCADE, to='app.product'): This part of the operation specifies that the new product field will be a foreign key to the 'app.product' model. It also sets a default value of 1 for the product field. The on_delete argument specifies what should happen if the referenced product is deleted; in this case, it will be deleted as well.**
**migrations.AddField(model_name='orderplaced', name='quantity', field=...):**

**This operation adds a new field named quantity to the OrderPlaced model.**
**models.PositiveIntegerField(default=1): This part of the operation specifies that the new quantity field will be a positive integer field with a default value of 1.**
**In summary, this migration is making changes to the OrderPlaced model:**

**It removes the existing cart field.**
**It adds a new product field as a foreign key to the Product model.**
**It adds a new quantity field to store the quantity of products ordered with a default value of 1.**

# 0010_orderplaced_customer.py

**This Django migration file continues to represent changes to the database schema, specifically related to the OrderPlaced model. Let's break down the code:**

**from django.db import migrations, models:**

**These lines import the migrations and models modules from Django's database-related components.**
**class Migration(migrations.Migration)::**

**This class represents a database migration and inherits from migrations.Migration. It indicates that this file is part of a sequence of migrations that modify the database schema incrementally.**
**dependencies = [...]:**

**This list specifies the dependencies of this migration. It depends on another migration named '0009_auto_20210118_1245' from the 'app' app. This means that this migration should be applied after the specified migration.**
**operations = [...]:**

**This list contains a single database schema operation:**

**migrations.AddField(model_name='orderplaced', name='customer', field=...):**

**This operation adds a new field named customer to the OrderPlaced model.**
**models.ForeignKey(default=1, on_delete=django.db.models.deletion.CASCADE, to='app.customer'):**
**This part of the operation specifies that the new customer field will be a foreign key to the 'app.customer' model.**
**It sets a default value of 1 for the customer field.**
**The on_delete argument specifies what should happen if the referenced customer is deleted; in this case, it will be deleted as well.**
**In summary, this migration is making changes to the OrderPlaced model by adding a new field named customer that references the Customer model. This field allows each order placed to be associated with a specific customer.**

# code for backend py --admin.py

**This is a Django admin customization code that defines how the admin interface should display and interact with your models. Let's break down the code:**

**from django.contrib import admin:**

**This line imports the Django admin module.**
**from django.utils.html import format_html:**

**This line imports the format_html function, which is used to format HTML content safely.**
**from django.urls import reverse:**

**This line imports the reverse function, which is used to generate URLs for the admin interface.**
**from .models import (...):**

**These lines import the models from your Django application that you want to manage using the admin interface.**
**@admin.register(Customer) (and similar decorators for other models):**

**These decorators register your models with the Django admin. This allows you to manage these models through the admin interface.**
**class CustomerModelAdmin(admin.ModelAdmin): (and similar classes):**

**These classes define how the admin interface should display and interact with each of your models.**
**list_display = [...]:**

**These lists specify the fields that should be displayed as columns in the admin list view for each model.**
**def product_info(self, obj): (and similar methods):**

**These methods are defined to customize the display of related fields in the admin list view.**
**product_info generates a link to the admin change page for the related product.**
**customer_info generates a link to the admin change page for the related customer.**
**These methods use the reverse function to generate URLs for the admin change pages and format_html to format HTML links.**
**In summary, this code customizes the Django admin interface for your models (Customer, Product, Cart, and OrderPlaced) by specifying the fields to display in the list view and providing custom methods to format related field information as clickable links to their respective admin change pages.**

# forms.py

**This code defines several Django forms used for user registration, login, password management, and customer profile updates. Let's break down each form and its purpose:**

**CustomerRegistrationForm:**

**This form is used for user registration.**
**Inherits from UserCreationForm, which is a built-in Django form for user registration.**
**Customizes the appearance of form fields using the attrs parameter.**
**Adds an email field for user registration.**
**Meta class specifies the model, fields, labels, and widgets to be used in the form.**
**LoginForm:**

**This form is used for user login.**
**Inherits from AuthenticationForm, a built-in Django form for authentication.**
**Customizes the appearance of form fields using the attrs parameter.**
**MyPasswordChangeForm:**

**This form is used for changing the user's password.**
**Inherits from PasswordChangeForm, a built-in Django form for password change.**
**Customizes the appearance of form fields using the attrs parameter.**
**Adds the old password field.**
**MyPasswordResetForm:**

**This form is used for initiating a password reset process.**
**Inherits from PasswordResetForm, a built-in Django form for password reset.**
**Customizes the appearance of form fields using the attrs parameter.**
**MySetPasswordForm:**

**This form is used for setting a new password during the password reset process.**
**Inherits from SetPasswordForm, a built-in Django form for setting a new password.**
**Customizes the appearance of form fields using the attrs parameter.**
**CustomerProfileForm:**

**This form is used for updating customer profile information.**
**Inherits from forms.ModelForm and is based on the Customer model.**
**Specifies the fields that can be updated in the customer profile.**
**Customizes the appearance of form fields using the attrs parameter.**
**Each form class defines the appearance and behavior of its respective form, including labels, input types, and widget attributes, to control how the forms are rendered in HTML templates. These forms can be used in Django views to handle user registration, login, password management, and profile updates.**

# models.py

**This code defines several Django forms used for user registration, login, password management, and customer profile updates. Let's break down each form and its purpose:**

**CustomerRegistrationForm:**

**This form is used for user registration.**
**Inherits from UserCreationForm, which is a built-in Django form for user registration.**
**Customizes the appearance of form fields using the attrs parameter.**
**Adds an email field for user registration.**
**Meta class specifies the model, fields, labels, and widgets to be used in the form.**
**LoginForm:**

**This form is used for user login.**
**Inherits from AuthenticationForm, a built-in Django form for authentication.**
**Customizes the appearance of form fields using the attrs parameter.**
**MyPasswordChangeForm:**

**This form is used for changing the user's password.**
**Inherits from PasswordChangeForm, a built-in Django form for password change.**
**Customizes the appearance of form fields using the attrs parameter.**
**Adds the old password field.**
**MyPasswordResetForm:**

**This form is used for initiating a password reset process.**
**Inherits from PasswordResetForm, a built-in Django form for password reset.**
**Customizes the appearance of form fields using the attrs parameter.**
**MySetPasswordForm:**

**This form is used for setting a new password during the password reset process.**
**Inherits from SetPasswordForm, a built-in Django form for setting a new password.**
**Customizes the appearance of form fields using the attrs parameter.**
**CustomerProfileForm:**

**This form is used for updating customer profile information.**
**Inherits from forms.ModelForm and is based on the Customer model.**
**Specifies the fields that can be updated in the customer profile.**
**Customizes the appearance of form fields using the attrs parameter.**
**Each form class defines the appearance and behavior of its respective form, including labels, input types, and widget attributes, to control how the forms are rendered in HTML templates. These forms can be used in Django views to handle user registration, login, password management, and profile updates.**

# url.py

```
from django.urls import path
from app import views
from django.conf import settings
from django.conf.urls.static import static
from django.contrib.auth import views as auth_views
from .forms import LoginForm, MyPasswordChangeForm, MyPasswordResetForm, MySetPasswordForm
urlpatterns = [
   # path('', views.home),
   path('', views.ProductView.as_view(), name="home"),
   # path('product-detail', views.product_detail, name='product-detail'),
   path('product-detail/<int:pk>/', views.ProductDetailView.as_view(), name='product-detail'),
   path('add-to-cart/', views.add_to_cart, name='add-to-cart'),
   path('cart/', views.show_cart, name='showcart'),
   path('pluscart/', views.plus_cart),
   path('minuscart/', views.minus_cart),
   path('removecart/', views.remove_cart),
   path('checkout/', views.checkout, name='checkout'),
   path('address/', views.address, name='address'),
   path('orders/', views.orders, name='orders'),
   path('paymentdone/', views.payment_done, name='paymentdone'),

   path('mobile/', views.mobile, name='mobile'),
   path('mobile/<slug:data>', views.mobile, name='mobiledata'),

   path('accounts/login/', auth_views.LoginView.as_view(template_name='app/login.html',
authentication_form=LoginForm), name='login'),
   # path('profile/', views.profile, name='profile'),
   path('profile/', views.ProfileView.as_view(), name='profile'),
   path('logout/', auth_views.LogoutView.as_view(next_page='login'), name='logout'),
   path('passwordchange/',
auth_views.PasswordChangeView.as_view(template_name='app/passwordchange.html',
form_class=MyPasswordChangeForm, success_url='/passwordchangedone/'), name='passwordchange'),
   path('passwordchangedone/',
auth_views.PasswordChangeDoneView.as_view(template_name='app/passwordchangedone.html'),
name='passwordchangedone'),

   path("password-reset/",
auth_views.PasswordResetView.as_view(template_name='app/password_reset.html',
form_class=MyPasswordResetForm), name="password_reset"),
   path("password-reset/done/",
auth_views.PasswordResetDoneView.as_view(template_name='app/password_reset_done.html'),
name="password_reset_done"),
   path("password-reset-confirm/<uidb64>/<token>/",
auth_views.PasswordResetConfirmView.as_view(template_name='app/password_reset_confirm.html',
form_class=MySetPasswordForm), name="password_reset_confirm"),
   path("password-reset-complete/",
auth_views.PasswordResetCompleteView.as_view(template_name='app/password_reset_complete.html'),
name="password_reset_complete"),

   path('registration/', views.CustomerRegistrationView.as_view(), name='customerregistration')
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

# views.py

**This Django code defines views and functions for handling various actions related to user registration, product handling, shopping cart management, orders, and profile updates. Let's break down the code by its major components:**

**Views and Class-Based Views:**

**Django views are used to handle different HTTP requests and render HTML templates or perform other actions.**
**Class-based views are used to organize and encapsulate related functionality.**
**ProductView:**

**Renders the home page with product categories (top wear, bottom wear, and mobiles).**
**Counts the number of items in the user's shopping cart (totalitem) if the user is authenticated.**
**ProductDetailView:**

**Displays detailed information about a specific product.**
**Checks if the product is already in the user's shopping cart (item_already_in_cart).**
**Counts the number of items in the user's shopping cart (totalitem) if the user is authenticated.**
**add_to_cart:**

**Handles the addition of a product to the shopping cart.**
**Checks if the product is already in the user's cart.**
**Uses messages to display success messages.**
**show_cart:**

**Renders the shopping cart page.**
**Calculates the total amount and total items in the cart.**
**Handles both empty and non-empty cart scenarios.**
**plus_cart and minus_cart:**

**Handle increasing and decreasing the quantity of items in the cart via AJAX requests.**
**Update the cart's quantity and recalculate the total amount.**
**checkout:**

**Renders the checkout page.**
**Retrieves the user's address and cart items.**
**Calculates the total cost of items in the cart.**
**payment_done:**

**Handles payment completion and order placement.**
**Moves items from the cart to the order history.**
**remove_cart:**

**Handles the removal of a product from the cart via AJAX request.**
**Recalculates the total amount after removal.**
**address:**

**Renders the address page where users can manage their addresses.**
**orders:**

**Displays a list of orders placed by the user.**
**mobile:**

**Renders the mobile product page with filtering options based on brand and price range.**
**CustomerRegistrationView:**

**Handles user registration.**
**Uses the CustomerRegistrationForm to validate and save user registration data.**
**Displays a success message upon successful registration.**
**ProfileView:**

**Handles user profile updates.**
**Allows users to update their profile information.**
**Uses the CustomerProfileForm for updating profile details.**
**The code also includes decorators like login_required to restrict access to certain views to authenticated users only.**

**Overall, this code provides a foundation for building an e-commerce website with user registration, product management, shopping cart functionality, and order placement.**

# manage.py

**This is the main entry point script for running Django management commands using the command-line interface. It is typically named manage.py and is generated automatically when you create a Django project. Let's break down the code:**

**Shebang Line (#!/usr/bin/env python):**

**This line specifies the interpreter that should be used to run the script. In this case, it's using the system's default Python interpreter located in /usr/bin/env python. This ensures that the script is executed with Python.**
**Django Settings Configuration:**

**os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'shoppinglyx.settings') sets the Django project's settings module. In this example, it's set to 'shoppinglyx.settings'. This module contains project-specific settings, including database configuration, installed apps, and more.**
**Import Django's execute_from_command_line Function:**

**The script attempts to import the execute_from_command_line function from django.core.management. This function is the core of Django's command-line management system.**
**Error Handling:**

**If there is an ImportError when trying to import Django, it raises an error message providing instructions on what could be wrong. This helps users troubleshoot common issues, such as Django not being installed or a virtual environment not being activated.**
**Execute the Management Command:**

**execute_from_command_line(sys.argv) is the main entry point to execute Django management commands. It takes the command-line arguments passed to the script (sys.argv) and processes them, running the appropriate management command. For example, you can run commands like python manage.py runserver, python manage.py migrate, and many others.**
**if __name__ == '__main__'::**

**This conditional statement ensures that the script is only executed if it is the main entry point (i.e., it's not imported as a module into another script). When you run python manage.py from the command line, this condition is met, and the main() function is called.**
**In summary, this script serves as the main interface for executing Django management commands, allowing you to interact with and manage your Django project from the command line.**