**Worksheet 17 Voucher Codes**

Many online shops give out voucher codes to customers that can be redeemed for discounts on their purchases. A voucher code is usually valid for a specific time frame. The code can be redeemed one or multiple times.

We are going to create a voucher code system for our shop. Our voucher codes will be valid for clients that enter the voucher in a specific time frame. The voucher codes will not have any limitations in terms of the number of times they can be redeemed, and they will be applied to the total value of the shopping cart. For this functionality, we will need to create a model to store the voucher code, a valid time frame, and the discount to apply.

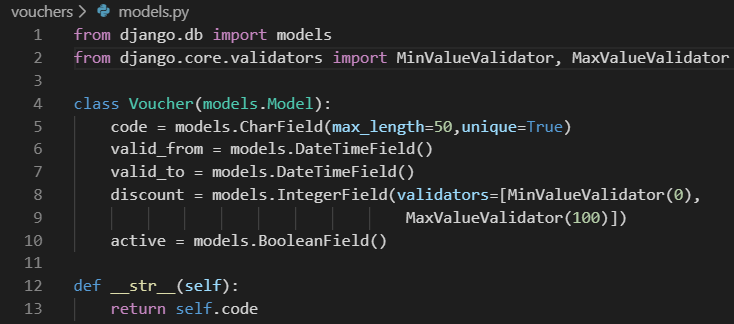
Create a new application inside the myshop project using the following command:



Edit the **settings.py** file of shop\_project and add the application to the **INSTALLED\_APPS** setting as follows:



Let's start by creating the Voucher model. Edit the **models.py** file of the vouchers application and add the following code to it:



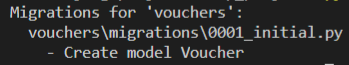
This is the model that we are going to use to store voucher codes. The Voucher model contains the following fields:

* code: The code that users must enter in order to apply the voucher code to their purchase.
* valid\_from: The datetime value that indicates when the voucher code becomes valid.
* valid\_to: The datetime value that indicates when the voucher code becomes invalid.
* discount: The discount rate to apply (this is a percentage, so it takes values from 0 to 100). We use validators for this field to limit the minimum and maximum accepted values.
* active: A Boolean that indicates whether the voucher code is active.

Run the following command to generate the initial migration for the vouchers application:



The output should include the following lines:



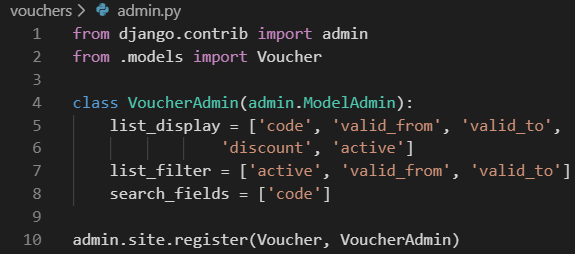
Then, we execute the next command to apply migrations:



You should see an output that includes the following line:

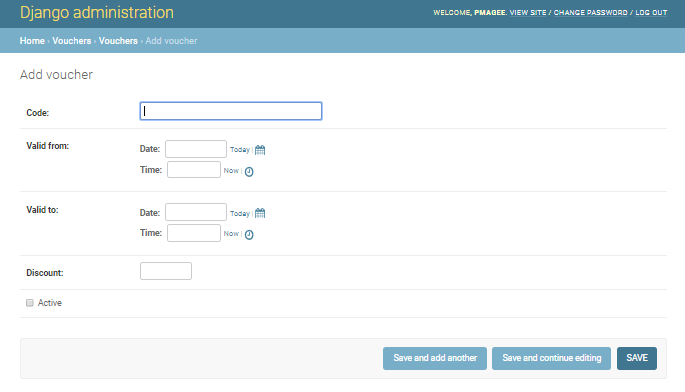


The migrations are now applied in the database. Let's add the Voucher model to the administration site. Edit the **admin.py** file of the **vouchers** application and add the following code to it:



The Voucher model is now registered in the administration site. Ensure that your local server is running with the command python manage.py runserver.

Open <http://127.0.0.1:8000/admin/vouchers/voucher/add/> in your browser. You should see the following form:



Fill in the form to create a new coupon that is valid for the current date and make sure that you check the Active checkbox and click the SAVE button.

**Applying a voucher to the shopping cart**

We can store new vouchers and make queries to retrieve existing vouchers. Now we need a way for customers to apply voucher codes to their purchases. The functionality to apply a voucher code would be as follows:

1. The user adds products to the shopping cart.

2. The user can enter a voucher code in a form displayed in the shopping cart detail page.

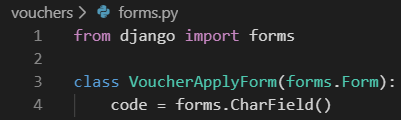
3. When a user enters a voucher code and submits the form, we look for an existing voucher with the given code that is currently valid. We must check that the voucher code matches the one entered by the user and that the active attribute is True, and that the current datetime is between the valid\_from and valid\_to values.

4. If a voucher is found, we save it in the user's session and display the cart, including the discount applied to it and the updated total amount.

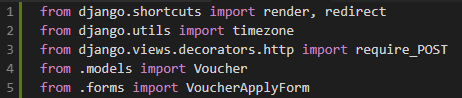
5. When the user places an order, we save the voucher to the given order.

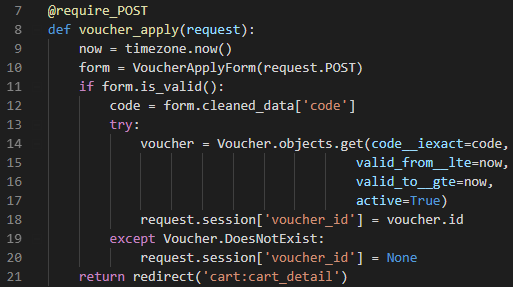
Create a new file inside the **vouchers** application directory and name it **forms.py**.

Add the following code to it:



This is the form that we are going to use for the user to enter a voucher code. Edit the **views.py** file inside the **vouchers** application and add the following code to it:





The **voucher\_apply** view validates the voucher and stores it in the user's session. We apply the require\_POST decorator to this view to restrict it to POST requests. In the view, we perform the following tasks:

1. We instantiate the VoucherApplyForm form using the posted data and we check that the form is valid.

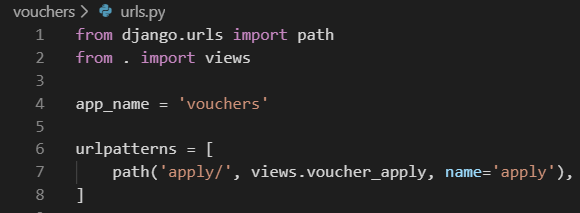
2. If the form is valid, we get the code entered by the user from the form's cleaned\_data dictionary. We try to retrieve the Voucher object with the given code. We use the iexact field lookup to perform a case-insensitive exact match. The coupon must be currently active (active=True) and valid for the current datetime. We use Django's timezone.now() function to get the current datetime and we compare it with the valid\_from and valid\_to fields performing lte (less than or equal to) and gte (greater than or equal to) field lookups, respectively.

3. We store the voucher ID in the user's session.

4. We redirect the user to the cart\_detail URL to display the cart with the voucher applied.

We need a URL pattern for the **voucher\_apply** view. Create a new file inside the **vouchers** application directory and name it **urls.py**.

Add the following code to it:



Then, edit the main urls.py of **shop\_project** and include the vouchers URL patterns as follows:



Remember to place this pattern before the shop.urls pattern.

Now, edit the **cart.py** file of the cart application.

Include the following import:



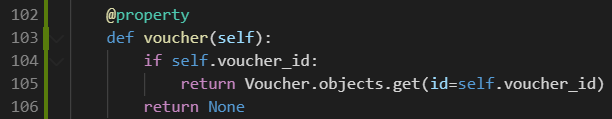
Add the following code to the end of the **\_\_init\_\_()** method of the **Cart** class to initialize the voucher from the current session:



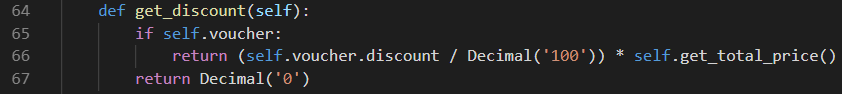
In this code, we try to get the **voucher\_id** session key from the current session and store its value in the **Cart** object.

Add the following methods to the **Cart** class:

We define this method as property. If the cart contains a voucher\_id attribute, the **Voucher** object with the given ID is returned.



We define this method to check if the cart contains a voucher. If it does, we retrieve its discount rate and return the amount to be deducted from the total amount of the cart.



In the following method we return the total amount of the cart after deducting the amount returned by the get\_discount() method.

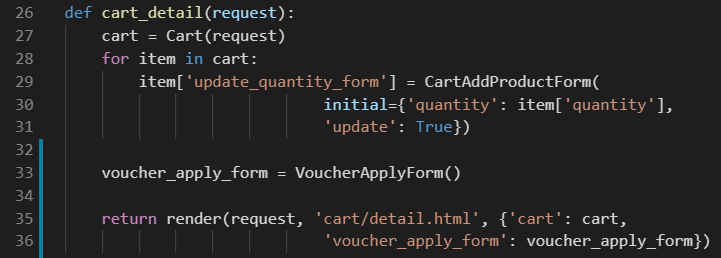


The **Cart** class is now prepared to handle a voucher applied to the current session and apply the corresponding discount.

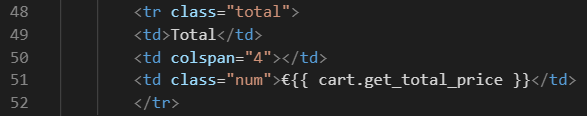
Let's include the voucher system in the cart's detail view. Edit the **views.py** file of the **cart** application and add the following import at the top of the file:



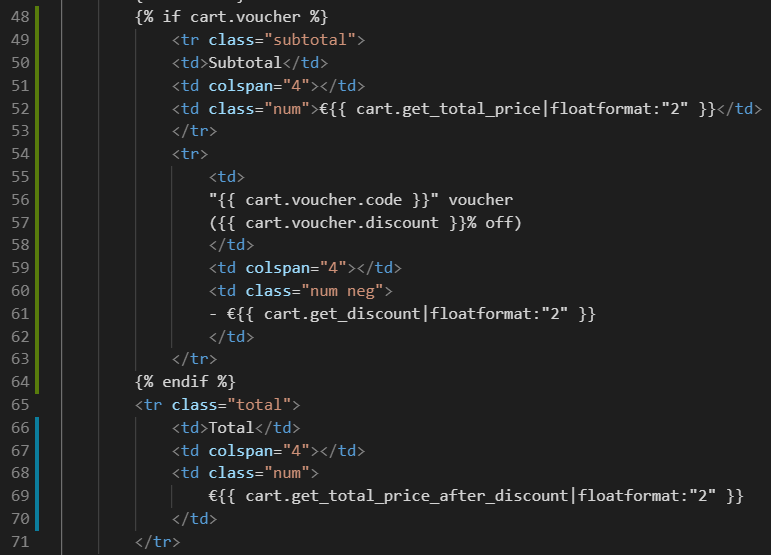
Further down, edit the **cart\_detail** view and add the new form to it as follows:



Edit the **cart/detail.html** template of the **cart** application and locate the following lines:



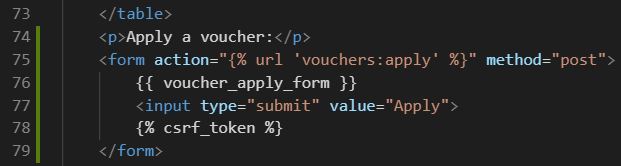
Replace them with the following:



This is the code for displaying an optional voucher and its discount rate. If the cart contains a voucher, we display a first row, including the total amount of the cart as the subtotal. Then we use a second row to display the current voucher applied to the cart.

Finally, we display the total price including any discount by calling the get\_total\_price\_after\_discount() method of the cart object.

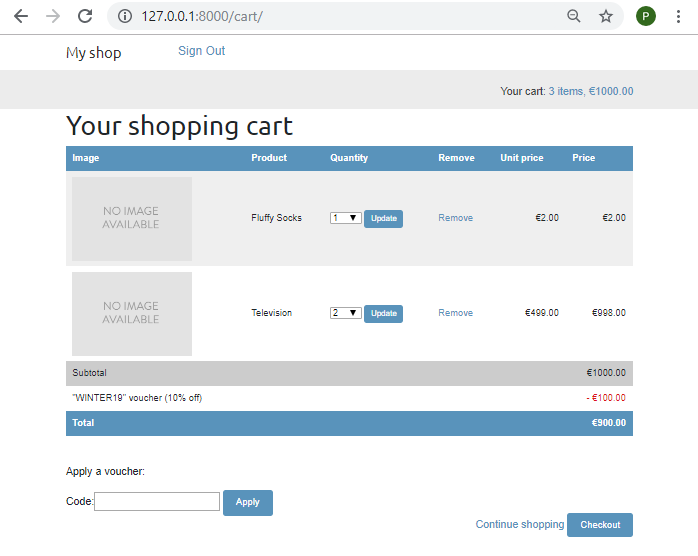
In the same file, include the following code after the </table> HTML tag:



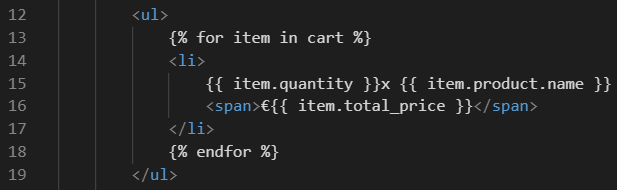
This will display the form to enter a voucher code and apply it to the current cart.

Open http://127.0.0.1:8000/ in your browser, add a product to the cart, and apply the voucher you created by entering its code in the form.

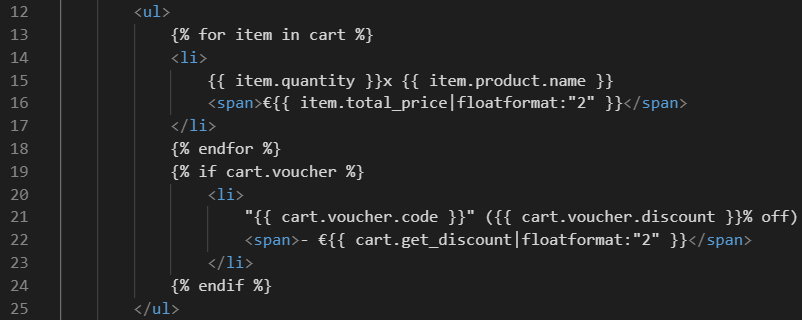
You should see that the cart displays the voucher discount as follows:



Let's add the voucher to the next step of the purchase process. Edit the **orders/order/create.html** template of the **orders** application and locate the following lines:



Replace them with the following code:



The order summary should now include the voucher applied, if there is one. Now find the following line:

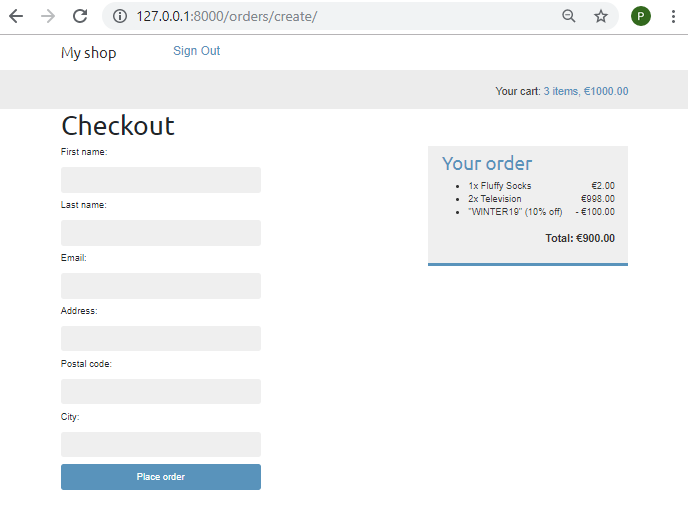


Replace it with the following:



By doing so, the total price will also be calculated by applying the discount of the voucher.

Open <http://127.0.0.1:8000/orders/create/> in your browser. You should see that the order summary includes the applied voucher as follows:



Users can now apply vouchers to their shopping cart. However, we still need to store voucher information in the order that is created when users check out the cart.

**Applying vouchers to orders**

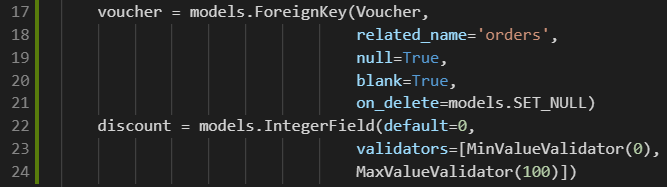
We are going to store the voucher that was applied to each order.

First, we need to modify the Order model to store the related **Voucher** object, if there is any.

Edit the **models.py** file of the **orders** application and add the following imports to it:



Then, add the following fields to the **Order** model:



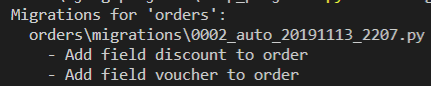
These fields allow us to store an optional voucher for the order and the discount percentage applied with the voucher. The discount is stored in the related Voucher object, but we include it in the Order model to preserve it if the voucher is modified or deleted.

We set on\_delete to models.SET\_NULL so that if the voucher gets deleted, the voucher field is set to Null.

We need to create a migration to include the new fields of the Order model. Run the following command from the command line:



You should see an output like the following:

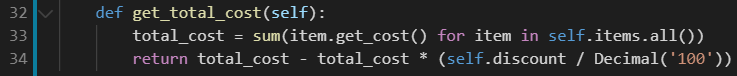


Apply the new migration with the following command:



You should see a confirmation indicating that the new migration has been applied. The **Order** model field changes are now synced with the database.

Go back to the **models.py** file and change the get\_total\_cost() method of the **Order** model as follows:

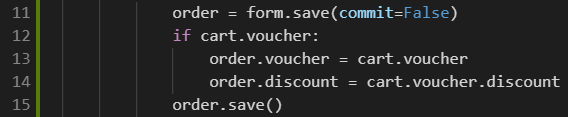


The get\_total\_cost() method of the **Order** model will now take into account the discount applied if there is one.

Edit the **views.py** file of the orders application and modify the **order\_create** view to save the related voucher and its discount when creating a new order. Find the following line:



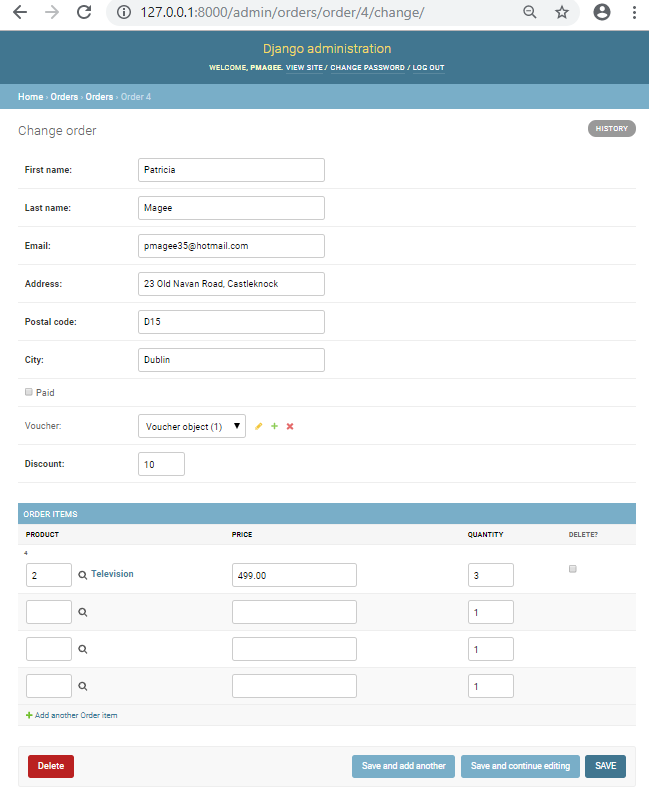
Replace it with the following:



In the new code, we create an **Order** object using the save() method of the OrderCreateForm form. We avoid saving it to the database yet by using commit=False. If the cart contains a voucher, we store the related voucher and the discount that was applied. Then we save the order object to the database.

Make sure the development server is running with the command python manage.py runserver.

Open http://127.0.0.1:8000/ in your browser and complete a purchase using the voucher you created. When you finish a successful purchase, you can go to http://127.0.0.1:8000/admin/orders/order/ and check that the order object contains the voucher and the applied discount as follows:

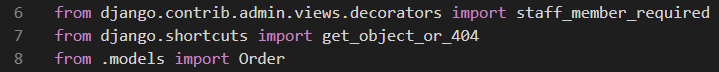


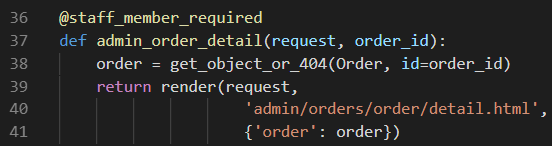
**Extending the admin site with custom views**

Sometimes, you may want to customize the administration site beyond what is possible through configuring ModelAdmin, creating admin actions, and overriding admin templates. If this is the case, you need to create a custom admin view. With a custom view, you can build any functionality you need. You must make sure that only staff users can access your view and that you maintain the admin look and feel by making your template extend an admin template.

Let's create a custom view to display information about an order.

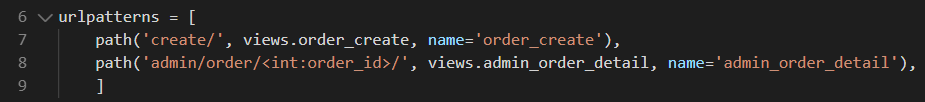
Edit the **views.py** file of the **orders** application and add the following code to it:



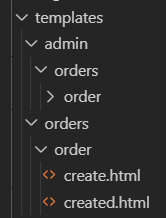


The **staff\_member\_required** decorator checks that both the **is\_active** and **is\_staff** fields of the user requesting the page are set to True. In this view, we get the Order object with the given ID and render a template to display the order.

Now, edit the **urls.py** file of the **orders** application and add the following URL pattern to it:



Create the following file structure inside the templates/ directory of the orders application:



Create a new file called detail.html inside the folder. Open the detail.html file on Moodle and copy the code in to this file and save it.

This is the template to display an order detail on the administration site. This template extends the admin/base\_site.html template of Django's administration site, which contains the main HTML structure and CSS styles of the admin. We load the custom static file css/admin.css.

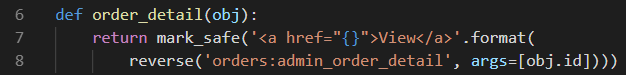
When you want to extend an admin template, you need to know its structure and identify existing blocks.

You can find all admin templates at

<https://github.com/django/django/tree/2.0/django/contrib/admin/templates/admin>

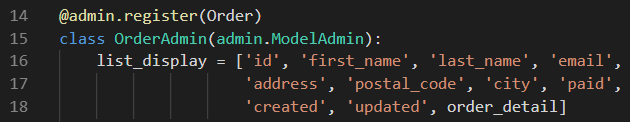
Finally, let's add a link to each Order object in the list display page of the administration site. Edit the **admin.py** file of the **orders** application and add the following code to it, above the OrderAdmin class:





This is a function that takes an Order object as an argument and returns an HTML link for the admin\_order\_detail URL. Django escapes HTML output by default. We must use the mark\_safe function to avoid auto-escaping.

Then, edit the **OrderAdmin** class to display the link:



Open <http://127.0.0.1:8000/admin/orders/order/> in your browser. Each row now includes a View link as follows:



Click on the View link for any order to load the custom order detail page. You should see a page like the following one:

