Merchiston Castle School

E-Commerce Shop

Candidate Number: 1836
Artemii Khrystych

Contents page

Ecommerce Website

Analysis

Introduction

My current school Merchiston Castle has a school shop where students and parents can buy school supplies. The current system operates via email. It means that every student or parent who wants to buy something from the school shop has to email ‘schoolshop@merchiston.co.uk’ listing all the items they would like to purchase. However, the disadvantage of this system is that neither parents nor students know the price and they don’t know what the item they buy looks like. In addition to that, I assume it is hard to manage orders from emails and send them to the right house and person.

I want to make a site that would enable users to purchase items, see them and see what the price is. Therefore, the new system will be more user-friendly. Also, it will reduce the amount of work needed to manage the orders, because they will be already sorted in the admin panel of the site. As a result, the orders will be sent more efficiently.

Third-Party

My client is Mrs Cordingley, she is our school shop manager. My end users are the students and parents who will use the site to purchase items.

Interview

- **Me:** Please describe the order procedure, as well as any issues you're having with it.

- **Mrs Cordingley:** The current system is presented in a way that a student writes on a piece of paper his order and then I have to manually put it into the system to keep track of the student's expenses. This is not very convenient to rewrite information from a piece of paper into the system because there is one more step.

- **Me:** Ok, there is a simpler way I could implement which would exclude the step where students have to write their orders on a piece of paper. Instead, the orders would come to the admin panel where you would be able to see the name of the student who ordered the item, which item and the house the student is in. Would it make it any easier?

- **Mrs Cordingley:** Yes, that will be easier that way and also I won't have to store these papers as proof of purchase.

- **Me:** What do you do with the returned items?

- **Mrs Cordingley:** We have an accounting package. We have a stock list(everything that shop has), and a full accounting package. For orders, we have one table and for the returned items we have a different table

- **Me:** How do you count the amount each student has spent?

- **Mrs Cordingley:** Parents are given a form to say how much they want them to spend. We set the limit for their account that is stored in a table. We can see if the student is in the red zone or not. Also, we have a table with all the transactions they have made

- **Me:** What happens if the student has gone over the limit?

- **Mrs Cordingley:** Transactions can still go through, it just flashes as red on the system.

- **Me:** What is the best way to approach the order details?

- **Mrs Cordingley:** We could have a collection time so that the students will come to the door at certain times. However, for the younger students, they are not supposed to take items without having names on them, there has to be a teacher supervising them. I suppose it could be different for different year groups.

- **Me:** In the admin panel, you would see what orders the students have made, so I can make the form for order details where the students have to fill in their age groups and houses they are in. Hence there will be two different tables for senior and junior students with their orders. Would it be suitable for you?

- **Mrs Cordingley:** Yes, I think so. The only requirements are that we need to agree with the housemaster from which year group the students will be able to collect the items by themselves.

- **Me:** Do you need specifically the names of items each student has purchased?

- **Mrs Cordingley:** Yes, we do, because of the nature of it, everything has accounted for so, everything needs to come up to the stock list.

- **Me:** How many sizes would one item require?

- **Mrs Cordingley:** For example, PE t-shirts have sizes for junior students and senior students. For junior students, they are XS boy, S boy, M boy, L boy. For the senior, they are XS man, M man, L man, XL man

- **Me:** Okay, I will do 1 item and a selection of sizes. What would be a better way to make it possible for students to see what size they are?

- **Mrs Cordingley:** There would need to be a sizing guide, for different things, trousers, for instance, have two measurements, vast and the leg length. Each blazer has only a chest measurement. So, at the end of each item, we could have a sizing guide. Or it might be separate.

- **Me:** How would it be suitable to manage returned items? For example, if I want to return an item, what would I do?

- **Mrs Cordingley:** We need a precise record for returned items for accountability purposes. Probably creating another link for returned items so that each student will be able to write what he wants to return

- **Mrs Cordingley:** Would it be possible to make a registration for each student, it would require them to enter the house they are in, their surname and name, and the password, then after they have logged in, the site will display the limits for spendings that their parents have entered. And another registration for parents

- **Me:** Then we will have to make another order table for parents as well

- **Me:** One option I see is that there will be two registration for students and for parents

- **Mrs Cordingley:** Yes, that's fine. Every boy has a code for the school so you can use these codes which are generated for each student.

Project Proposal

My initial idea for this project is to build a site with items that would take orders, sort them and save them into the database. The website allows the school shop manager to digitize all the data about the orders and sort them. The website allows students and parents to see all the school shop items and to see the prices. In my system, it would be possible to add/delete items from the admin panel. Also, students would be able to see how much money they have left according to the limits that were set by their parents. Each student will have to be logged in with their username, password, and boarding house. They would also be able to order, and return the items. This program is designed for the school shop manager, students, and parents. The School shop manager will have access to the admin panel of the website and will be able to delete/add items and change their prices or images.

Stakeholders / Prospective users

For my project, the problem that I am going to be tackling is coordinated to minimize the amount of work that needs to be done to order the item or return the item. Also, it will minimise the number of steps for our school shop manager to sort the orders. On my website, there will be 3 groups of people who will have access to the system. They are the admin of the site, parents and students. This is to prevent people from using the system from outside the school community. Admin will be able to add/edit/delete the items and also have access to the orders that students and parents have made.

Objectives - Customers

1. The website will include a registration page
    a. Details required for registration:
        i. username, password, password verification, email address

2. My system must include a store page
    a. Panel with item groups and a return option:
        i. Item groups
            1. Names of the item categories (t-shirts, trousers, PE kit, etc.)

        ii. Return option
            1. Specification of items that need to be returned

    b. Displayed items of a chosen item group
        i. Each item will have an image and price
            1. Each item will have an item page that will include a little description and a table with sizes

3. The website will have a basket
    a. Products that is desired to be purchased will be displayed
    b. Shipping Address
        i. Address, town, postcode, confirmation of postcode, state

Objectives - Admin

1. My system will have an admin panel

    a. Admin user will be able to add items option:
        i. Here the admin user will be able to add items to certain categories. This will include price, images, sizes grid, and name of item

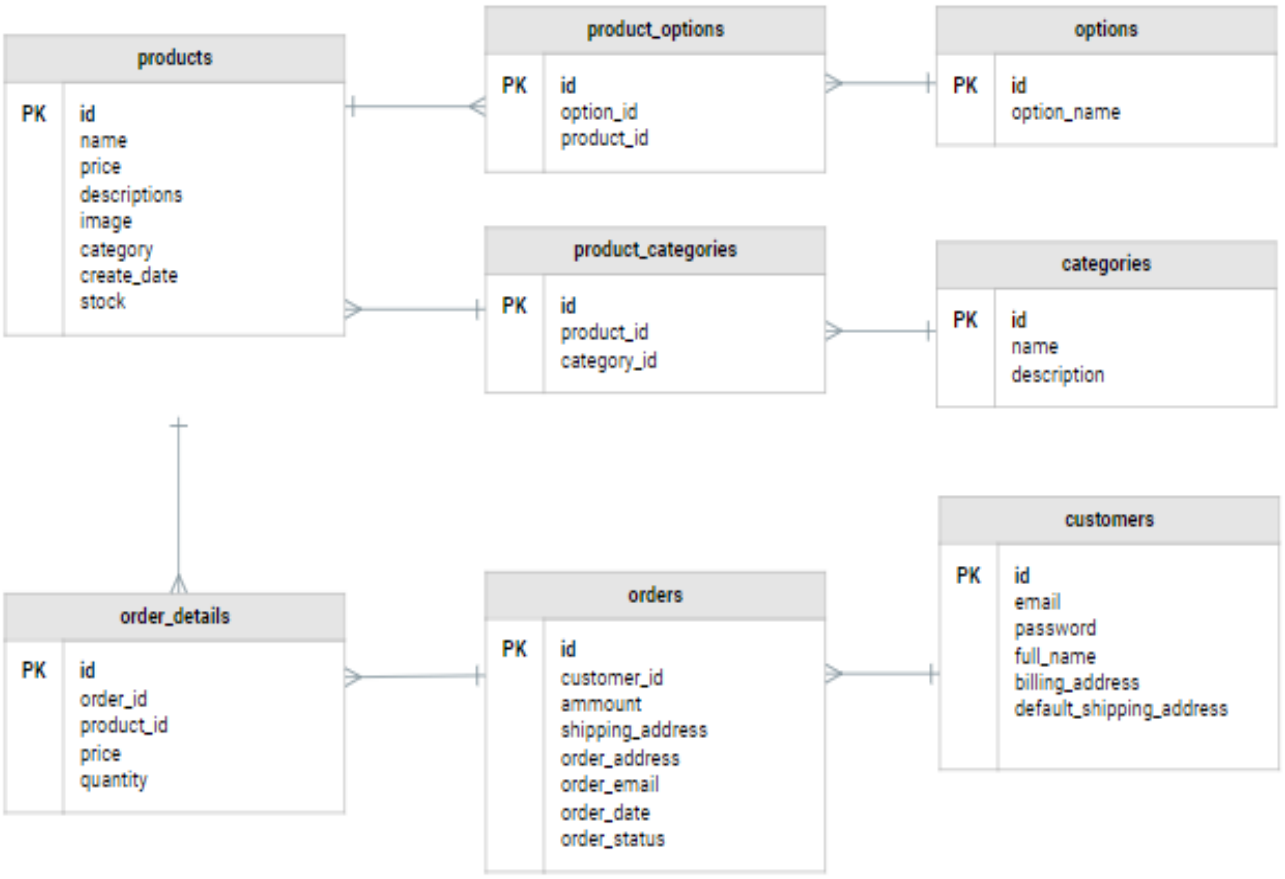
    b. My system will include deleting items option:
        i. Here the admin user will be able to delete items from the school shop

    c. My project provides an orders option:
        i. Page with orders
            1. Shipping address
                a. Adress, city, postcode, country
            2. Product information
                a. Quantity
                b. Product name

    d. Returned items option will be included:
        i. Database with returned items

Modelling

Here is my prototype of an E-R model of my database that I will be using. This model gives a general idea of how my database will be constructed. My final model might not look exactly the same. Possibly I will add some changes during the development process

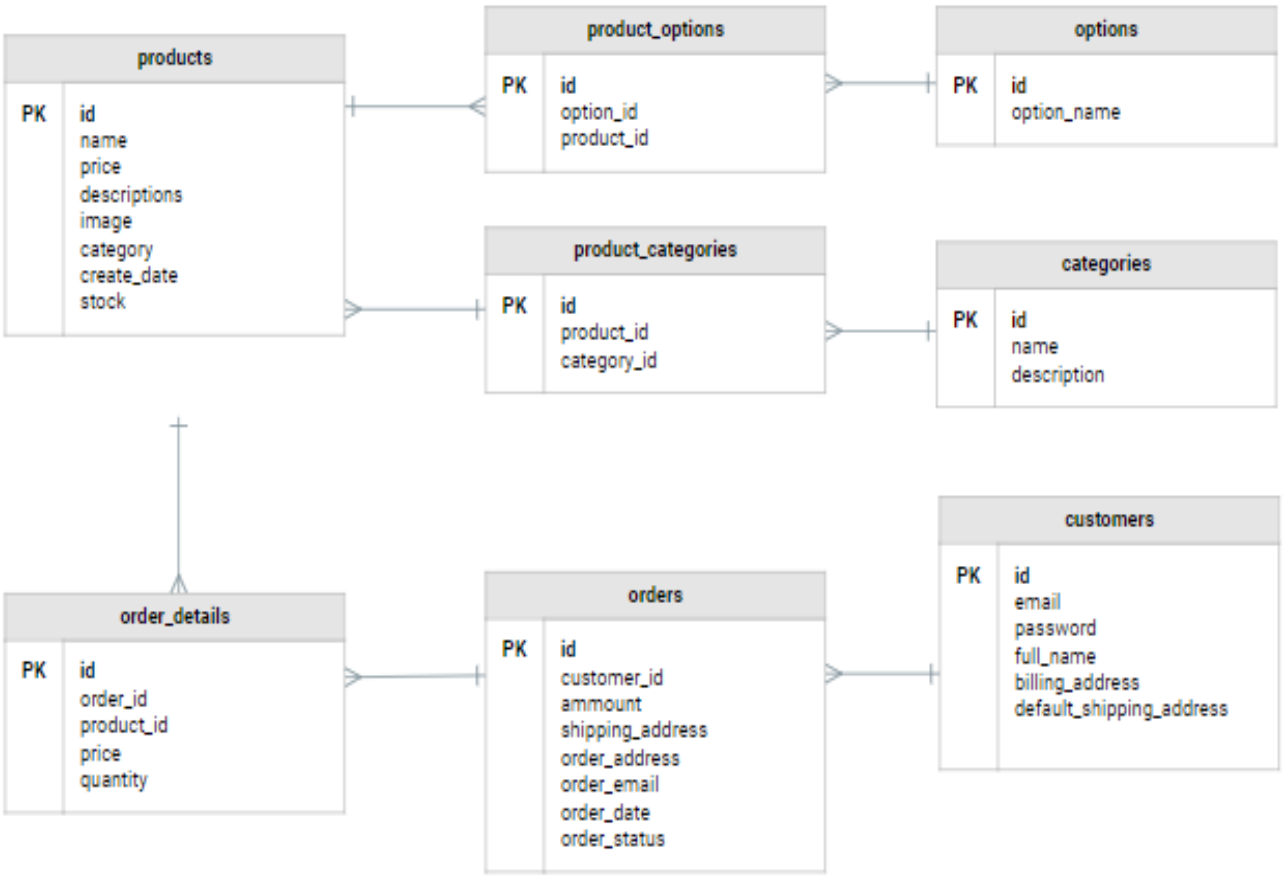


Payment Method

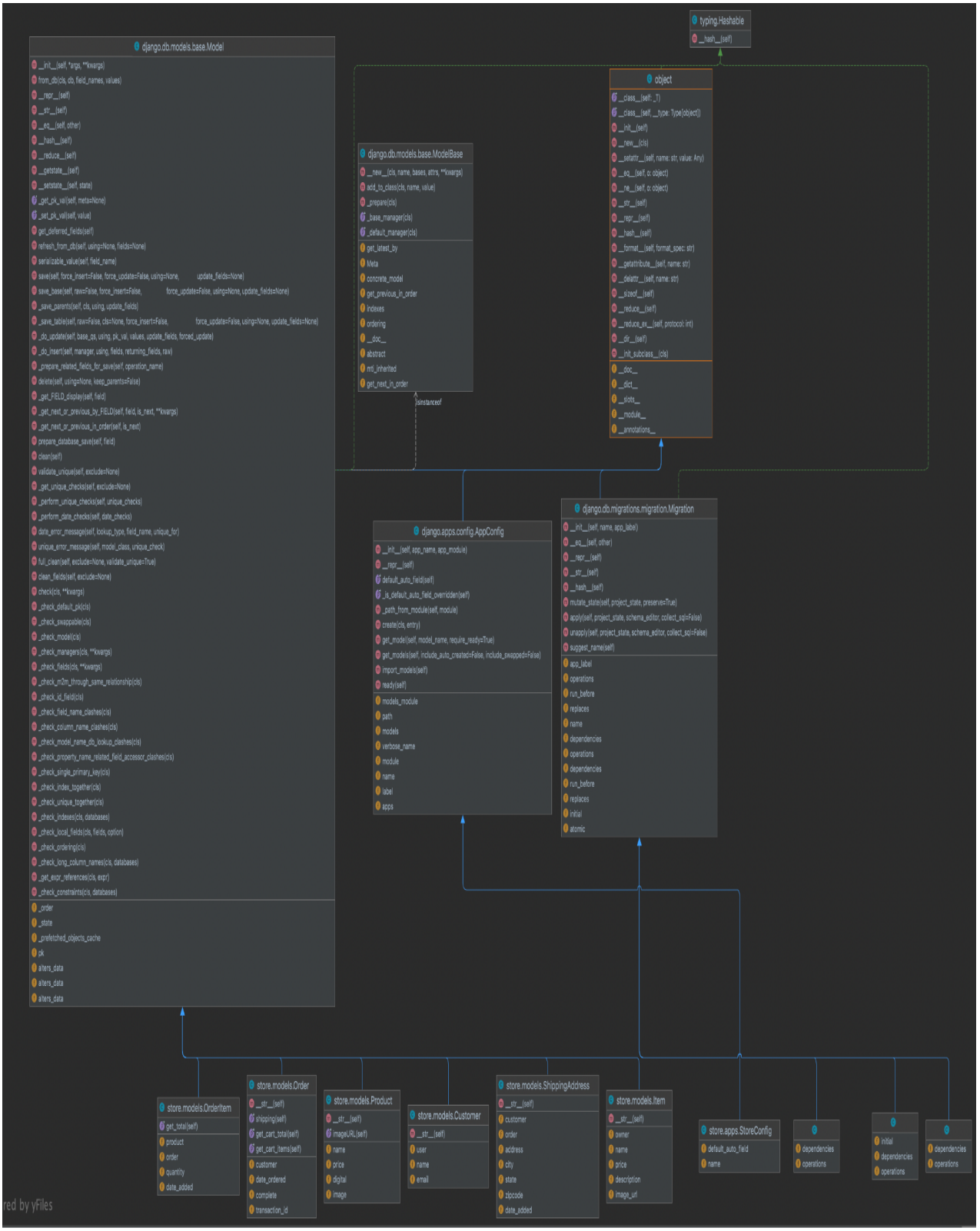
My project is not going to have a payment method. Instead, the price for the orders will be stored in a database for each individual student(customer_id). The orders will be seen in the admin panel, so the school shop manager will be able to see all orders information such as price, name of the student., and date. If parents want to buy something, it will be possible for them to do it as unregistered users and then go to the physical school shop and collect the items.

Design

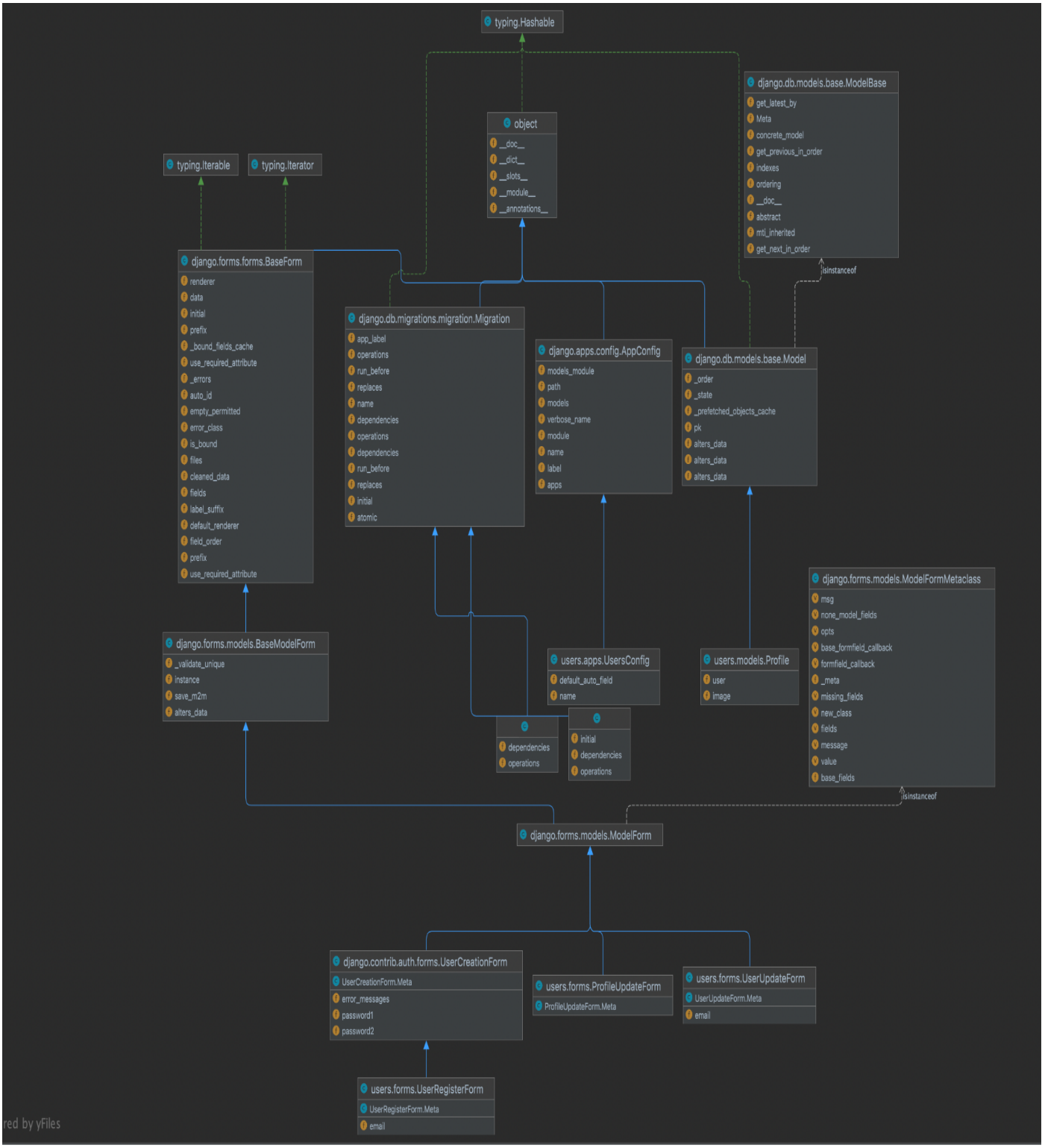
Here is my prototype of an E-R model of my database that I will be using. This model gives a general idea of how my database will be constructed. My final model might not look exactly the same. Possibly I will add some changes during the development process



Class Diagram for Django app: store (sub-module of a project)

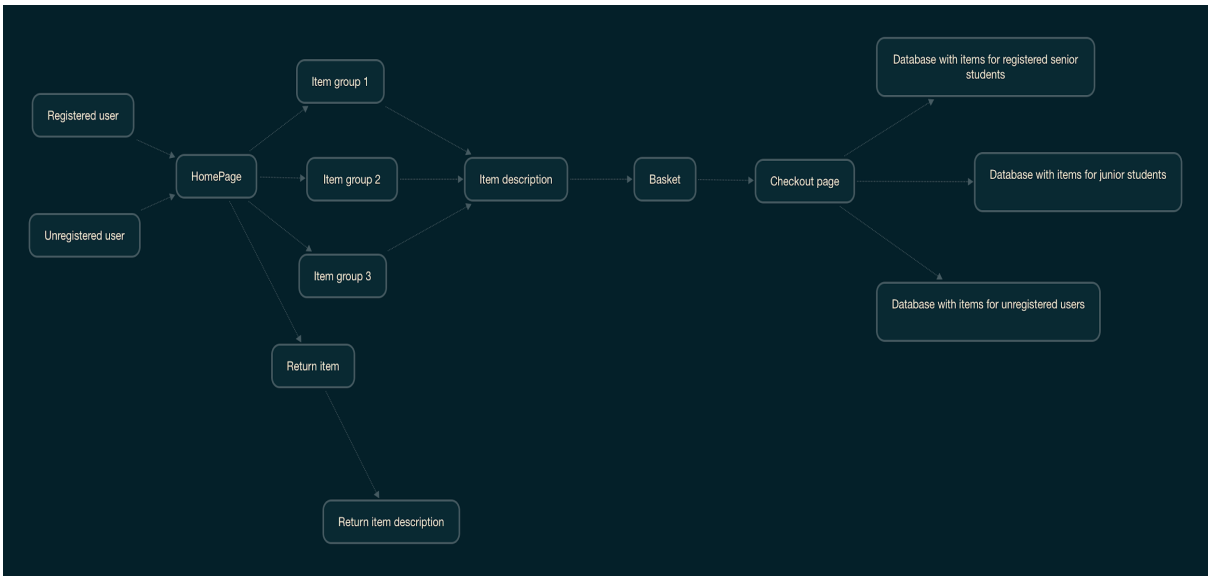


Class Diagram for Django app: users (sub-module of a project)


typing.Hashable
object
__doc__
__dict__
__slots__
__module__
__annotations__
typing.Iterable
typing.Iterator
django.db.models.base.ModelBase
get_latest_by
Meta
concrete_model
get_previous_in_order
indexes
ordering
__doc__
abstract
mti_inherited
get_next_in_order
isinstanceof
django.forms.forms.BaseForm
renderer
data
initial
prefix
_bound_fields_cache
use_required_attribute
_errors
auto_id
empty_permitted
error_class
is_bound
files
cleaned_data
fields
label_suffix
default_renderer
field_order
prefix
use_required_attribute
django.db.migrations.migration.Migration
app_label
operations
run_before
replaces
name
dependencies
operations
dependencies
run_before
replaces
initial
atomic
django.apps.config.AppConfig
models_module
path
models
verbose_name
module
name
label
apps
django.db.models.base.Model
_order
_state
_prefetched_objects_cache
pk
alters_data
alters_data
alters_data
django.forms.models.ModelFormMetaclass
msg
none_model_fields
opts
base_formfield_callback
formfield_callback
_meta
missing_fields
new_class
fields
message
value
base_fields
isinstanceof
django.forms.models.BaseModelForm
_validate_unique
instance
save_m2m
alters_data
users.apps.UsersConfig
default_auto_field
name
users.models.Profile
user
image
dependencies
operations
initial
dependencies
operations
django.forms.models.ModelForm
django.contrib.auth.forms.UserCreationForm
UserCreationForm.Meta
error_messages
password1
password2
users.forms.ProfileUpdateForm
ProfileUpdateForm.Meta
users.forms.UserUpdateForm
UserUpdateForm.Meta
email
users.forms.UserRegisterForm
UserRegisterForm.Meta
email
red by yFiles

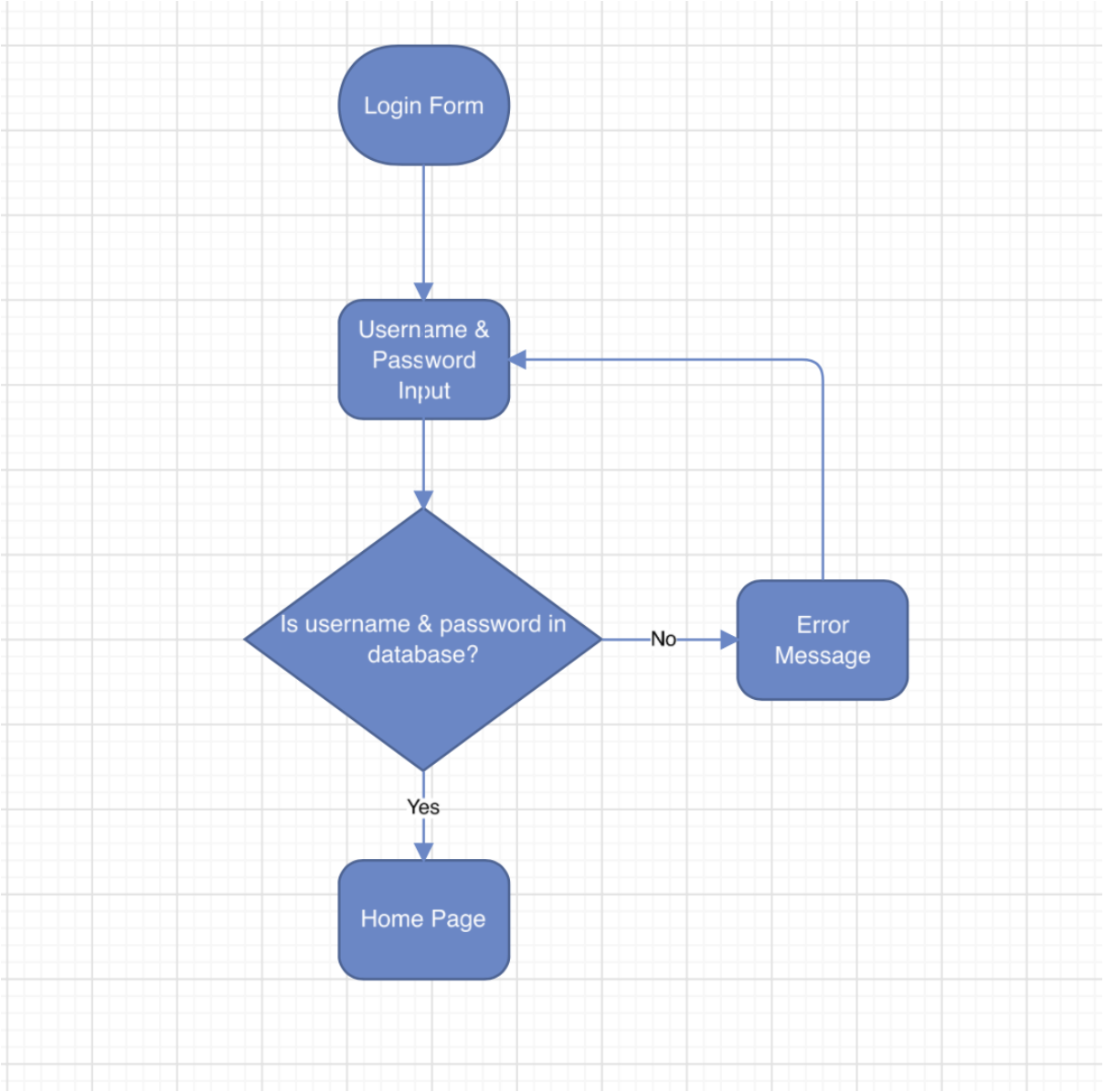
Hierarchy Chart

I have created a hierarchy chart that is an illustration of my web application. Each line indicates a connection between models in my program. This will help me to understand the shape of my web application.



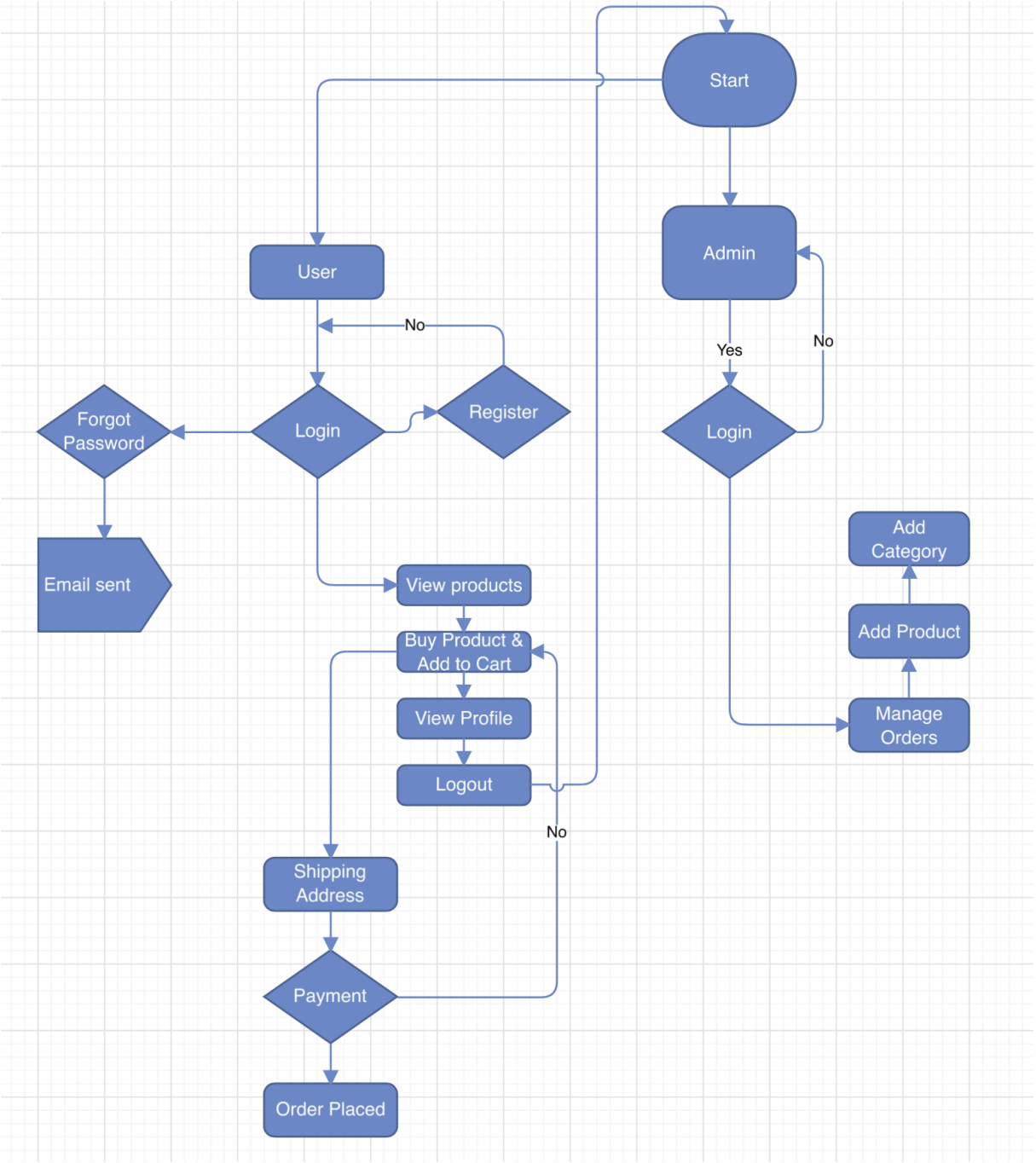
Login Form

The login form gets 2 inputs from the username which are the username field for email address and password. After clicking a button to log in, the program checks if the username and password are in the database. If these inputs are in the database, then the program redirects to the home page. If not, an alert message is displayed.



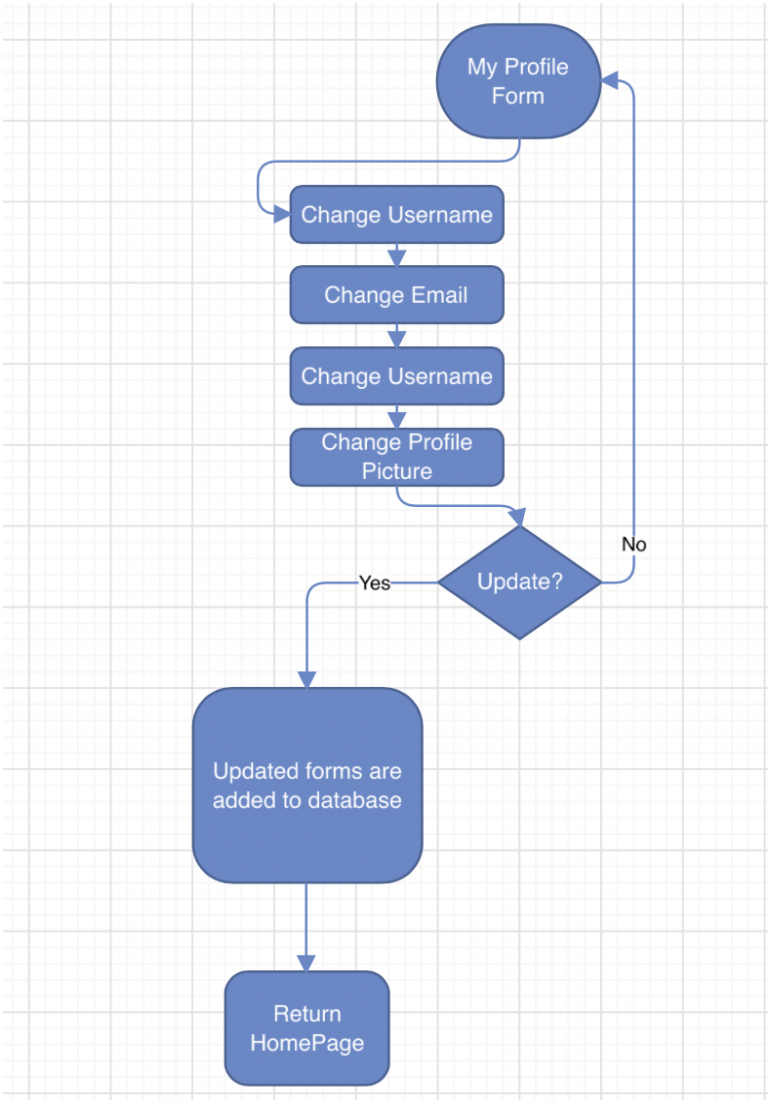
Home Page:

This chart shows how the home page is designed.



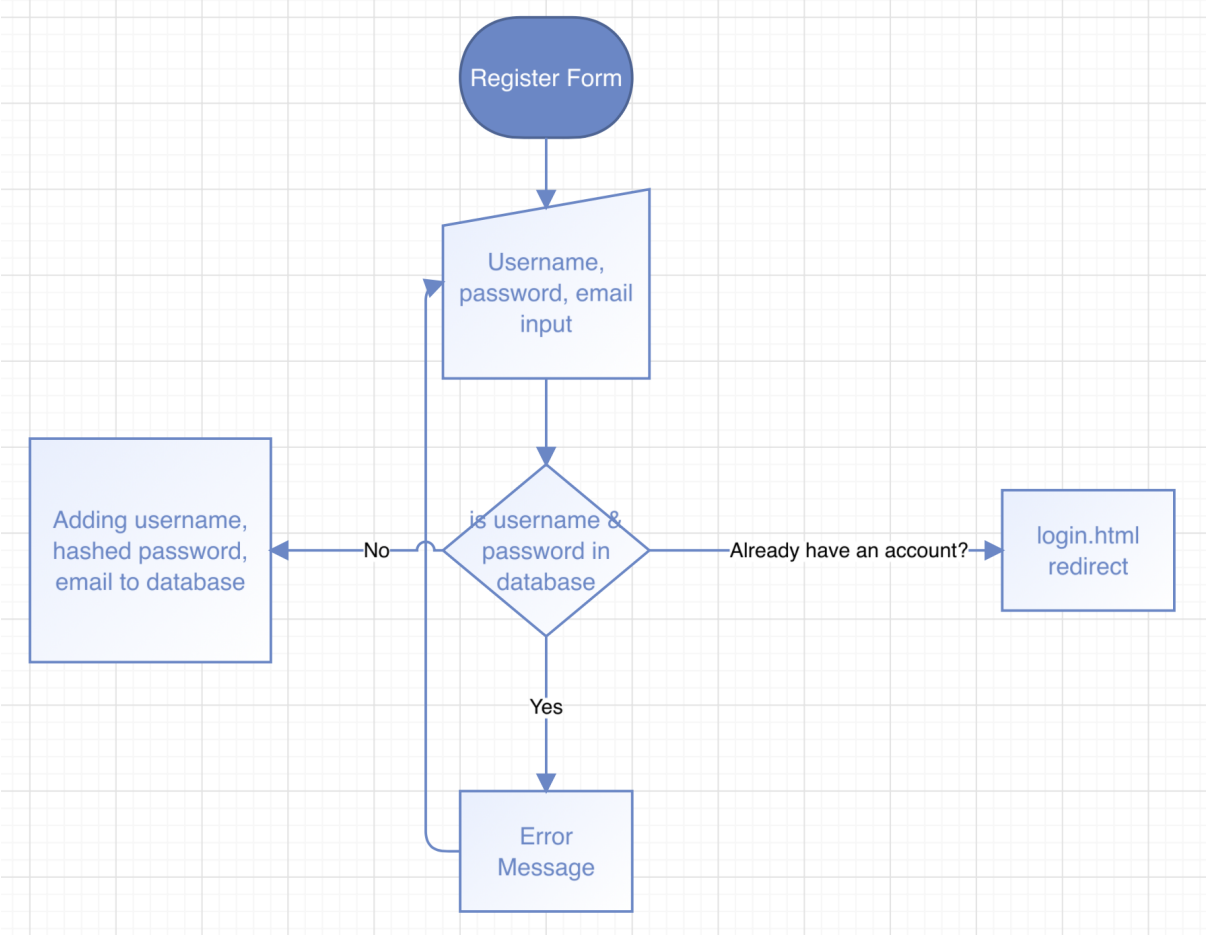
Profile Form:

The profile form has 5 different features that are changing username button, change email button, changing username button, change profile picture button. Update button is required for saving the filled information in the text fields on the page to the database.



Register Form:

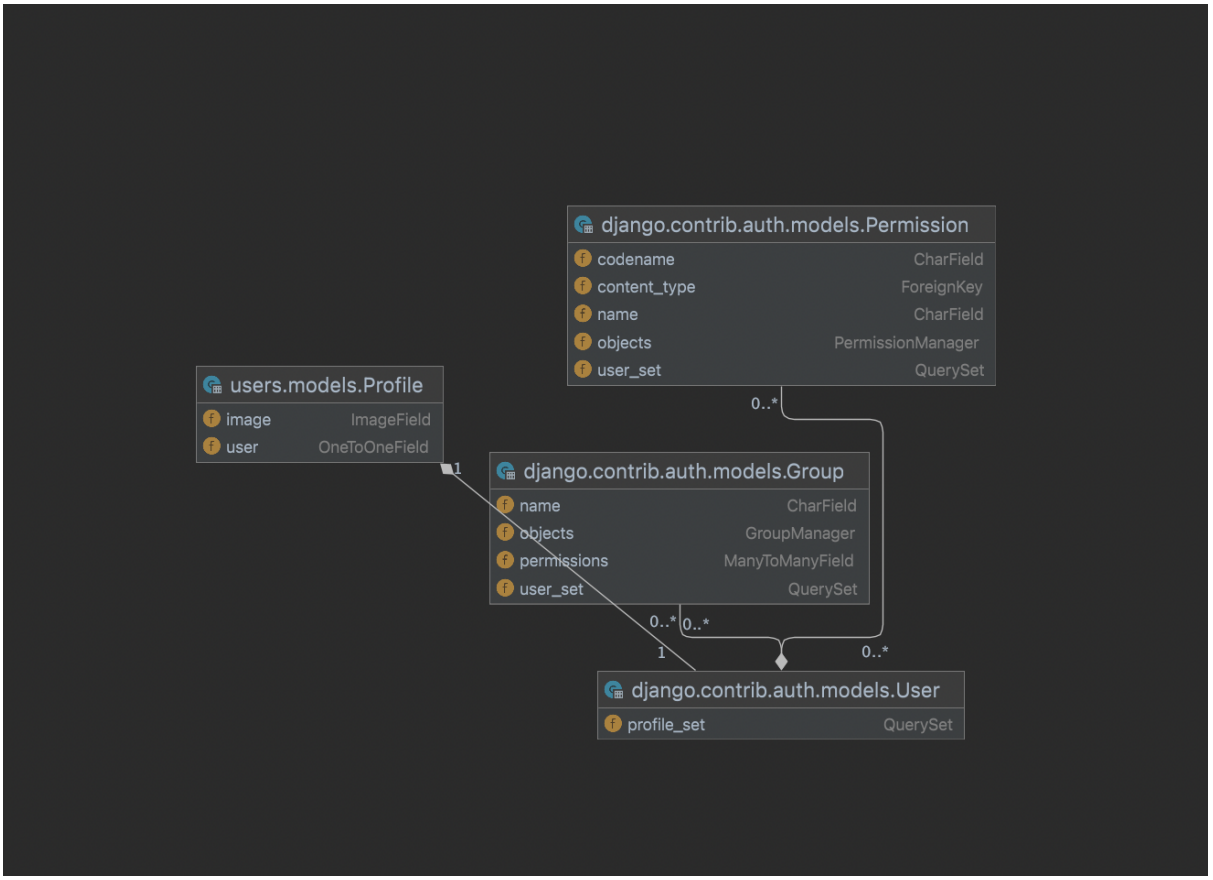
The registration form has 4 different inputs. Email, username, password and password confirmation. If username is already registered, the program will show the error and option to redirect to the login page. If the user's registration input is not in the database, then it will allow a new account to be created.



Data Flow Diagram

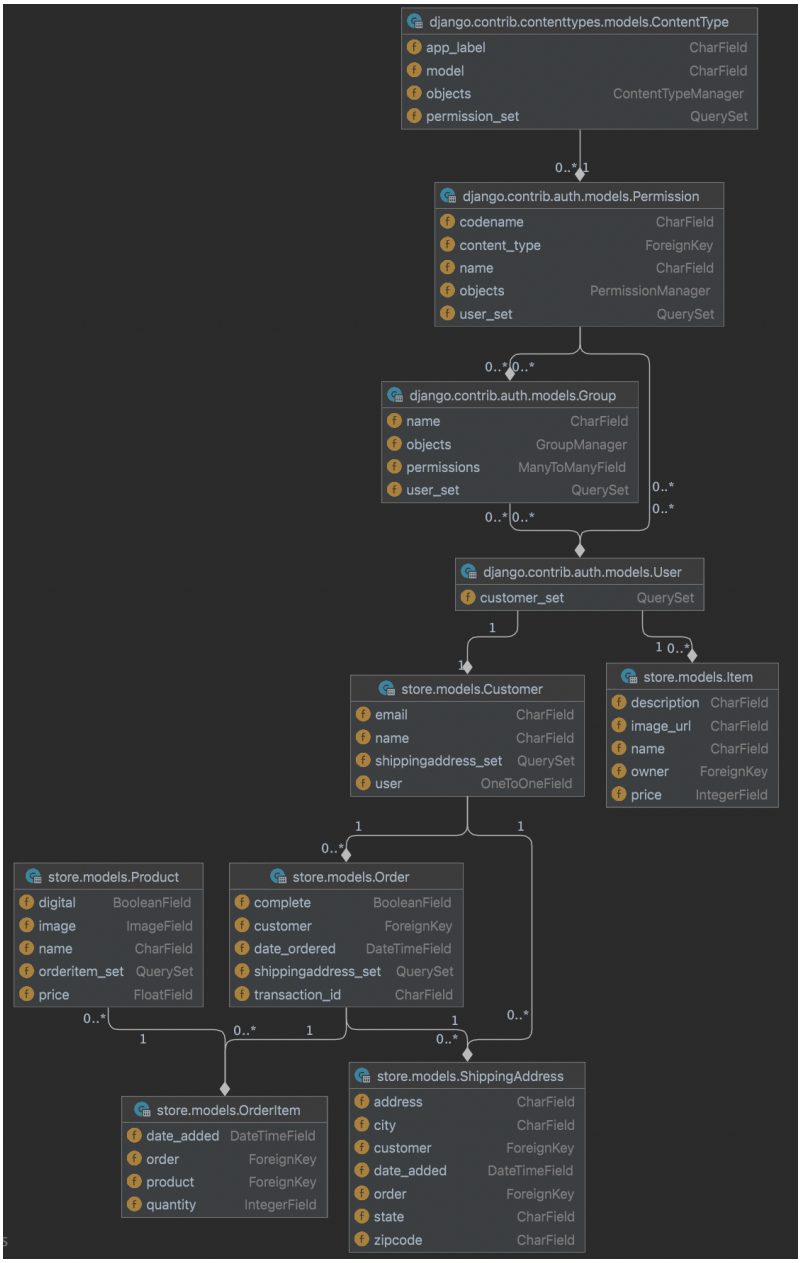
Database of Django app users

This database represents how data is being stored for Django app users. For example, the table users.model.Profile is connected to the django.contrib.auth.User to enable the creation of the profile after registration. When a user is registered, the program gives the created profile a value of a user. The user has one to one field with a profile. It means that the value user has certain features for instance preloading a default image in place of a profile image.



Database of Django apps for store, static, e-commerce

This database represents how the data is being stored and connected with the rest of the Django applications. In this database can be seen how the user's information is connected with store items. For instance, the table models.ContentType has a permission_set row which stores information about permissions that the user is allowed to access. If the user has permission from the superuser, then he will be able to access the admin panel. Another example is in models. Order that has a row customer which is a foreign key. It is done for initiating a unique code as a current date to seconds and stores it in a hash table. Therefore, orders can go to the user who ordered them.



Queries

store/views.py	store/templates/store/main.html

def get_queryset(self):
  query = self.request.GET.get('search')
  products = Product.objects.filter(Q(name__icontains=query))
  return products
	<form method="get" action="/search">
	
	  <input type="text" placeholder="Search" name="search" required>
	
	…
	</form>

I have implemented a search for items on the main page. Shown above is a query that I have used in my base html file. In order to make it work, after the user enters the desired search name and clicks search, the program passes the string entered by the user from main.html to views.py. This is done by using the following variable: **`query = self.request.GET.get('search')`**
In order to display the result, the program needs to scan the existing list of items and find the most relevant ones. This is done by using **`Product.objects.filter(Q(name__icontains=query))`** Essentially this line scans through the list of items and displays the most relevant items. SQL equivalent to **SELECT ... WHERE headline ILIKE '%query%'.**

store/views.py	store/templates/store/store.html

def category_detail(request, slug,):
  category = get_object_or_404(Category, slug=slug)
 	
  context = {
  'category': category,
  }
 	
  return render(request, 'store/category_detail.html', context)
	View

The query above is used to display more details about the product. In order to do this, I have used **slug** which is a unique identifier for every product. The **slug** is generated for every product after creating it from the admin panel. The relationship between the product and category is many to one because there can be many products under one category.

```
category = get_object_or_404(Category, slug=slug)
```

The line above gets a category with all registered objects in it. If the category is not registered, django raises a 404 error which is not allowing the site to crash.

The connection between front end and back end is done by registering a url path for product-detail

```
path('<slug:category_slug>/<slug:slug>/', product_detail,
name='product-detail')
```

Therefore, when the user clicks on the button ‘View’ on the page, the store.html file redirects to the product.detail view and from there, the program gets the **slug** of the category and slug of items that are in it. Finally the program gets information about the product from the database that has a corresponding slug and redirects the user to the product-detail page.

Data Dictionary

Table: Product

Field Name	Data Type	Description
category	String	Foreign key to Category, identifies the name of the category
name	String	Identifies the name of the product
price	Float	Identifies the price of the product in float to 2 decimal places
description	String	Identifies the description for product and can be viewed in product-details page
digital	Boolean	Identifies if the product is digital. If it is, then after making a purchase, no shipping address needed to be specified
image	URL	Identifies the image for the product, stored in local files
slug	AutoURL	A unique identifier of a product. It is created after a product is being initiated

Table: UserRegisterForm

Field Name	Data Type	Description
username	String	Used to log into a web application
email	Email Field	User's email address
password1	Short Text	Password to get into an account on a web application
password2	Short Text	Password verification

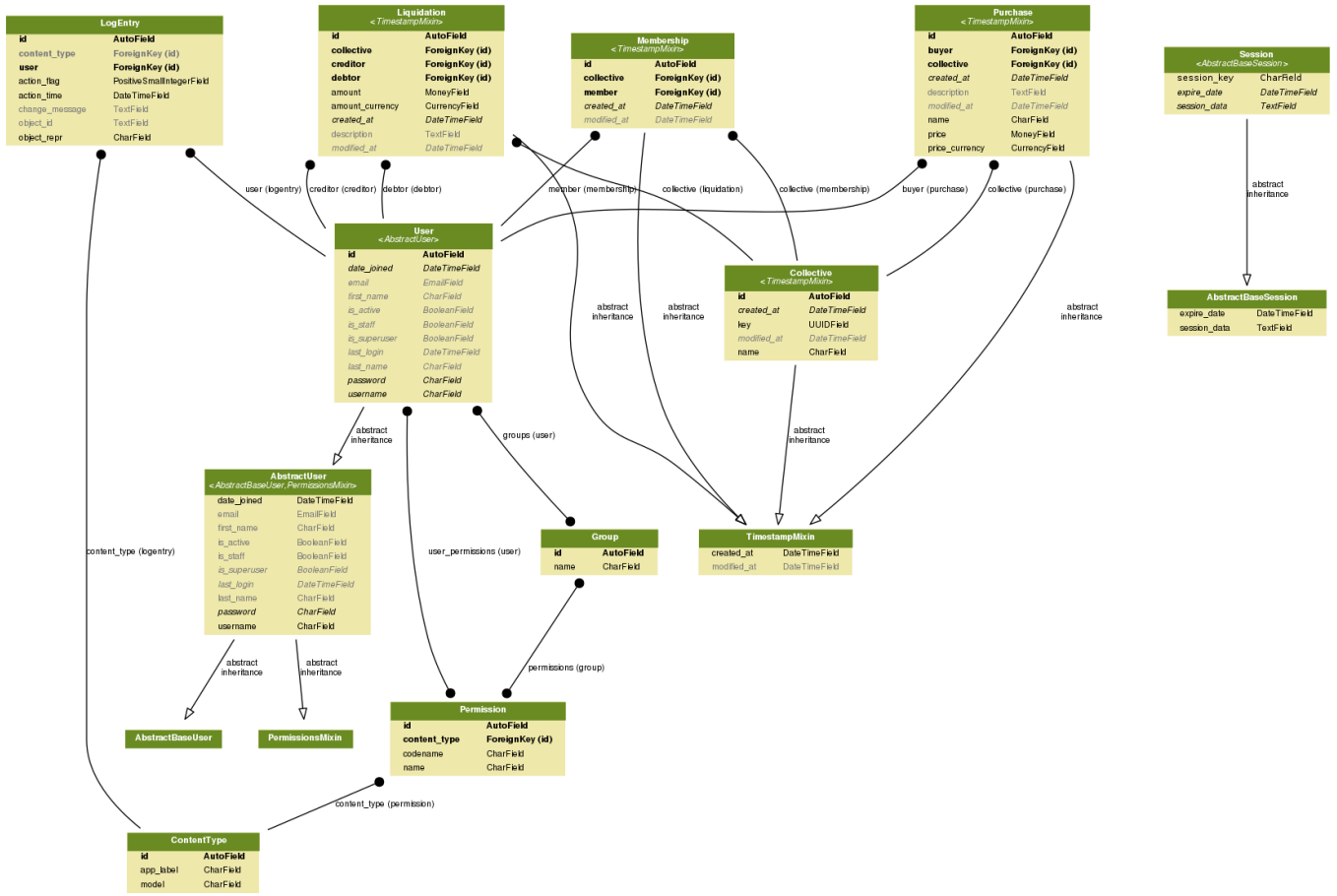
Table: ShippingAddress

Field Name	Data Type	Description
customer	String	Foreign key to User, used to identify name of the username who placed an order
order	String	Foreign key to Order, used to identify all required details for an item such as name, quantity, total price and transaction id
address	String	Identifies the address to be shipped
city	String	Identifies the city to be shipped
state	String	Identifies the state to be shipped
zipcode	String	Identifies the zip code to be shipped
date_added	Date/Time	Identifies when the time when the order was made

Table: Order

Field Name	Data Type	Description
customer	String	Foreign key to User, used to identify the name of the user who placed the order
date_ordered	Date/Time	This field is filled automatically when the user made an order and is used to identify the time when the order was placed
complete	Boolean	A boolean field, it is displayed as a checkbox in admin panel to identify which order has been completed
transaction_id	String	Transaction id is used to identify the id of the transaction, just like a receipt

Data Flow Diagram



Algorithms

Email Validation

For validation of an email address when the user registers, I decided to use regular expressions. Regular Expressions are widely used for pattern-matching, and various programming languages have interfaces for representing them, as well as interacting with the matched results.

Here is the format of email addresses we are looking for:

`(username)@(domain name).(top-level domain)`

As a result, we can reduce it to a pattern of the '@' symbol dividing the prefix from the domain segment.

The **prefix** is the recipient's name - a string that may contain uppercase and lowercase letters, numbers, and some special characters like the. (dot), -(hyphen), and _ (underscore).

The **domain** consists of its name and a top-level domain divided by a . (dot) symbol. The domain name can have uppercase and lowercase letters, numbers, and - (hyphen) symbols. Additionally, the top-level domain name must be at least 2 characters long (either all uppercase or lowercase letters) but can be longer.

`([A-Za-z0-9]+[.-_])*[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-z]{2,})+`

A special character in the prefix cannot be just before the @ symbol, nor can the prefix start with it, so I made sure that there is at least one alphanumeric character before and after every special character.
As for the domain, an email can contain a few top-level domains divided by a dot.

Here is my implementation for email validation in Python:

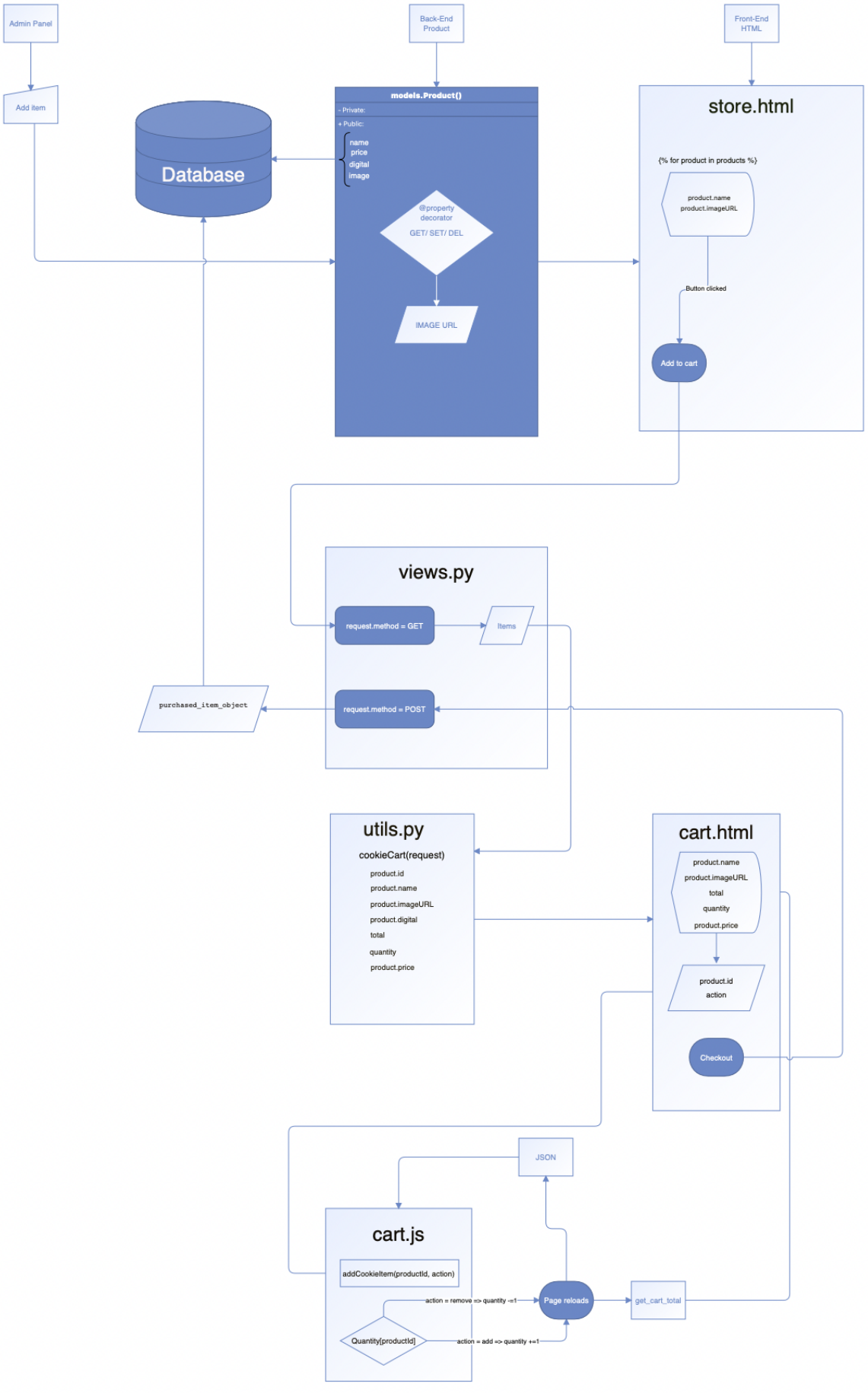
```
import re

regex =
re.compile(r'([A-Za-z0-9]+[.-_])*[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-z]{2,}
)+')

def isValid(email):
   if re.fullmatch(regex, email):
       return True
   else:
       return False
```

The re.compile() method compiles a regex pattern into a regex object. It's mostly used for efficiency reasons when matching the pattern more than once.

Generating items and cart logic

I have created a flow chart of an algorithm for rendering products on a web page and cart logic. In the admin panel, the user enters the required information to render an item and then the data is transferred to the file models.py. This file creates the database tables from the class. In models.py there is a class called Product() with required instances for an item to be represented on a web page. This class also has a method that returns the image URL that the admin has chosen while creating an item. This image as well as the product.name is being passed to the store.html to represent the image and product name to the user. After the button ‘Add to cart’ is clicked, the program sets request.method to GET in views.py, properties of the selected item is being moved to utils.py. This file prepares the data to be passed into JSON in the next steps and to represent the selected item in the user’s cart via cookies. In addition, the method cookieCart has a list quantity where the items are being passed according to the count of the for a loop. Then the program checks product.price by the unique product id and multiplies the quantity by the price. Data about a unique id of a product and action(button selected) are passed to the cart.js. The file cart.js is responsible for putting action and product.id to the cookie file. It is done to enable the functionality of the cart such as when the user presses to add a certain item to the cart, the number of items in the cart would be displayed. Every time the user adds an item to the cart, the number of items changes automatically after the page has been reloaded. The current information about the unique id and quantity of the item is stored in JSON. After items have been added to the cart, the user can purchase the items by clicking the checkout button which will change request.method to POST in views.py. Finally, the object of purchased items has been created and it is passed into a database. The user can view ordered items in the admin panel.

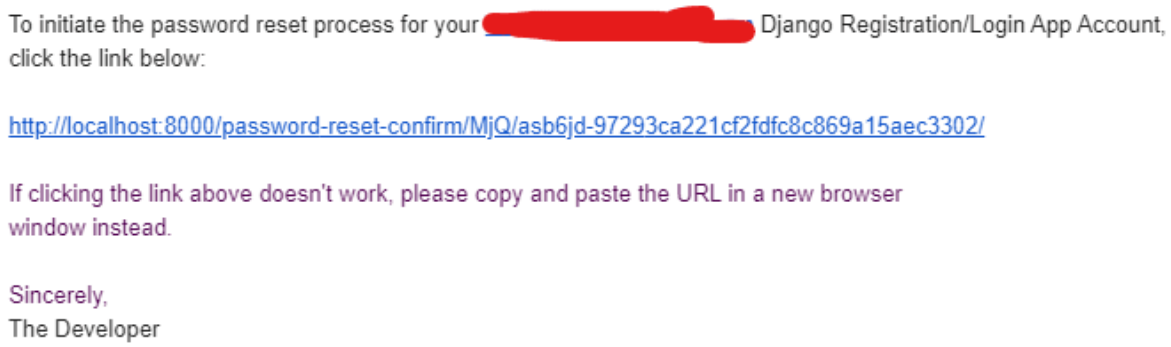
To protect the user from the CSRF attacks, I have implemented a CSRF token into my HTML page. CSRF is when the attacker will exploit the website by identifying the session cookie of the session and using that to send his payload to run on the application. A CSRF token is a secret, unique and unpredictable value a server-side application generates to protect CSRF vulnerable resources. If the token is missing or does not match the value within the user session, the request is rejected, the user session terminated and the event logged as a potential CSRF attack


Admin Panel
Add item
Back-End Product
Front-End HTML
Database
models.Product()
- Private:
+ Public:
name
price
digital
image
@property decorator
GET/ SET/ DEL
IMAGE URL
store.html
{% for product in products %}
product.name
product.imageURL
Button clicked
Add to cart
views.py
request.method = GET
Items
purchased_item_object
request.method = POST
utils.py
cookieCart(request)
product.id
product.name
product.imageURL
product.digital
total
quantity
product.price
cart.html
product.name
product.imageURL
total
quantity
product.price
product.id
action
Checkout
JSON
cart.js
addCookieItem(productId, action)
action = remove => quantity -=1
Page reloads
get_cart_total
Quantity[productId]
action = add => quantity +=1

Password Reset

1. User clicks on "forgot password" link - Which will take them to a page where they will be prompted to enter their email address. This view is going to be handled by extending from the built in PasswordResetView.
   PasswordResetView - This view displays the form where the user will submit his/her email address -> checks if a user with the provided email address exists in the database or not -> generates a password reset link that is going to be used only once -> and finally sends that link to the user's email address.
   - Note - Django doesn't throw in an error if the provided email address isn't associated with any user but won't send the email. The reason is to "prevent information leaking to potential attackers".

2. Okay after the password reset request is done, the user will be redirected to the home page with a message informing him/her to go and check their email address. The message will be displayed even if a user requesting the password reset doesn't exist.
3. User goes to his/her email and checks for a message. Let's say all went well and he/she has instruction for setting their password. It should look something like this.

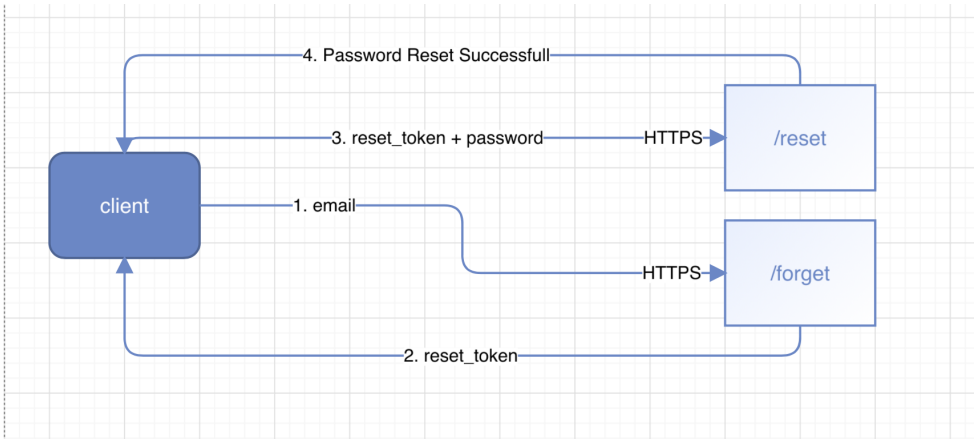

To initiate the password reset process for your [redacted] Django Registration/Login App Account, click the link below:

http://localhost:8000/password-reset-confirm/MjQ/asb6jd-97293ca221cf2fdfc8c869a15aec3302/

If clicking the link above doesn't work, please copy and paste the URL in a new browser window instead.

Sincerely,
The Developer

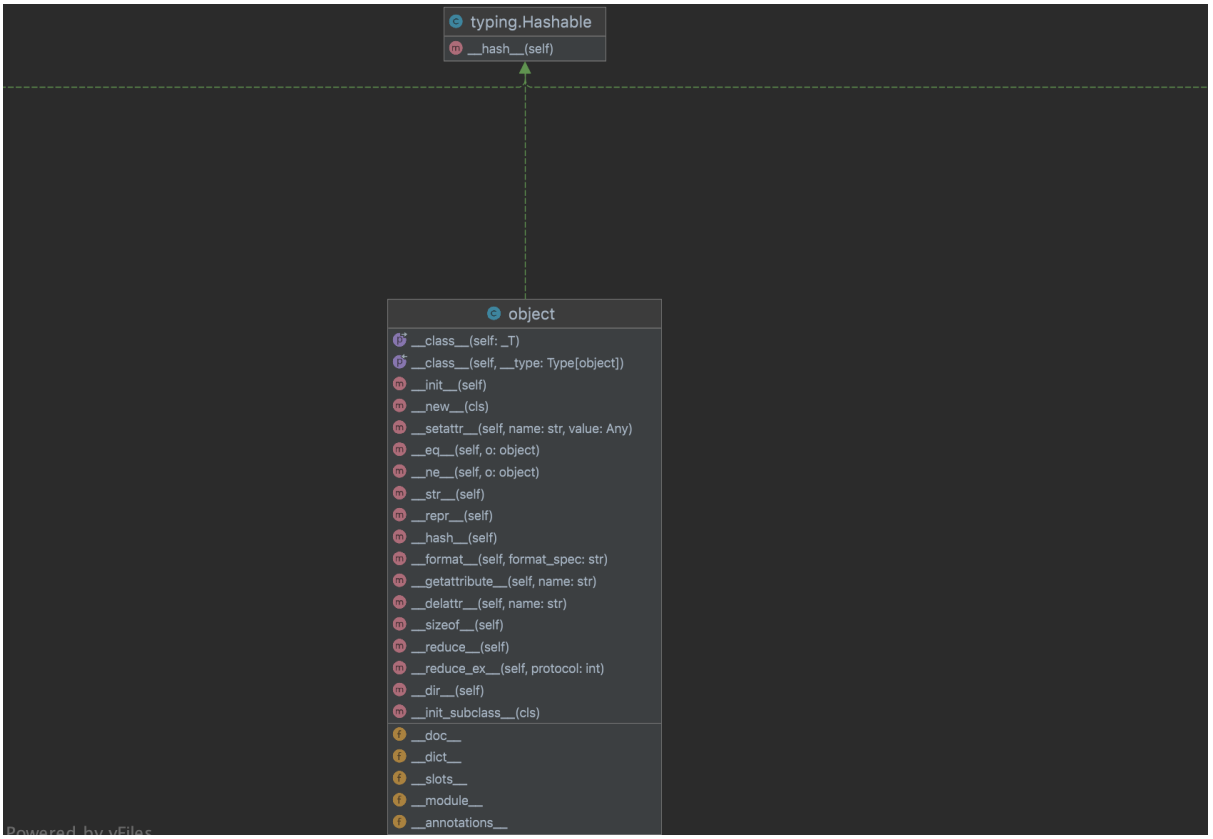
4) User clicks on the generated link and he/she will be provided with a form for entering a new password.



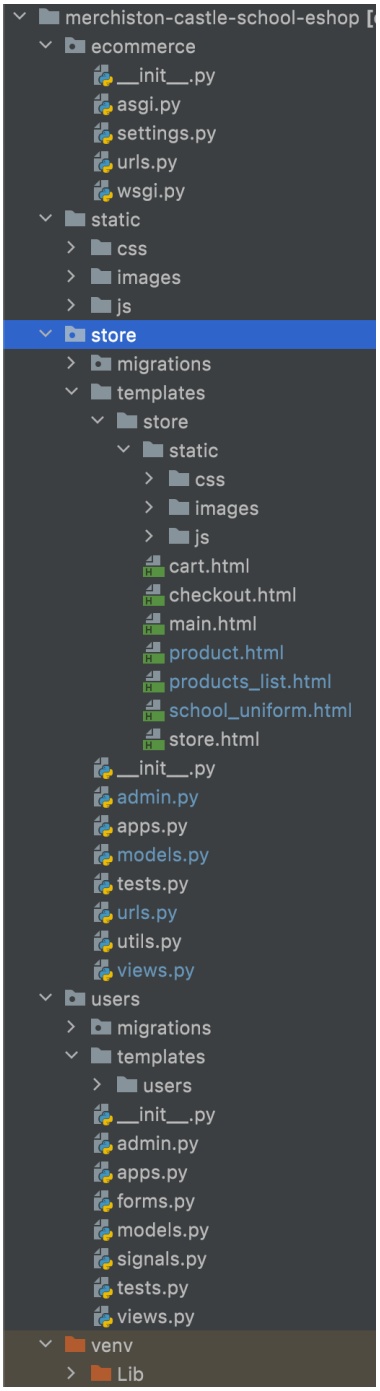
Data Structures

Hashing of passwords

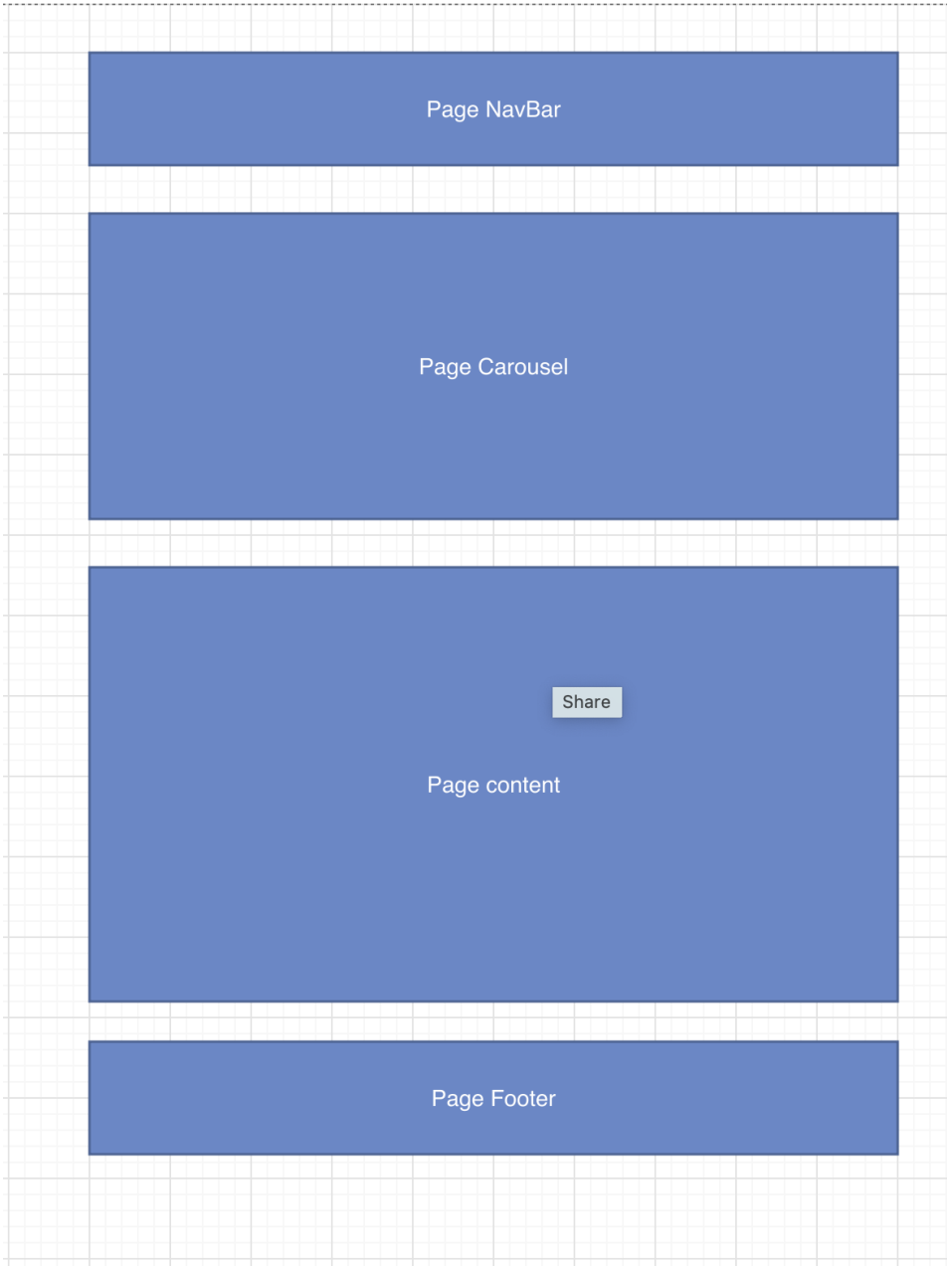
I have implemented hashing of passwords to add more security into the system. When the password is hashed, it is stored in a hash table, therefore, it will not be possible for an admin to see all the passwords of the users.



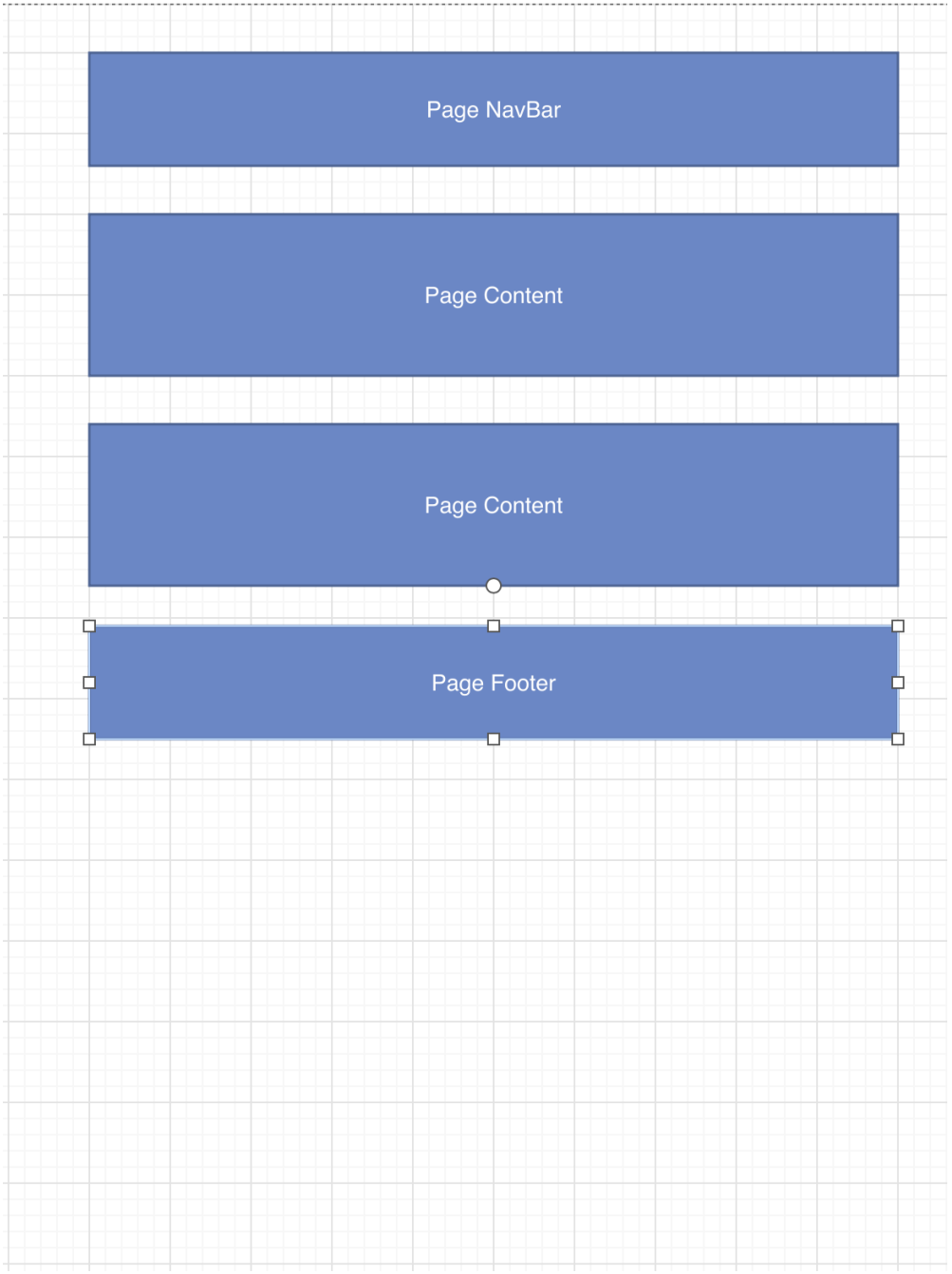
File Structure



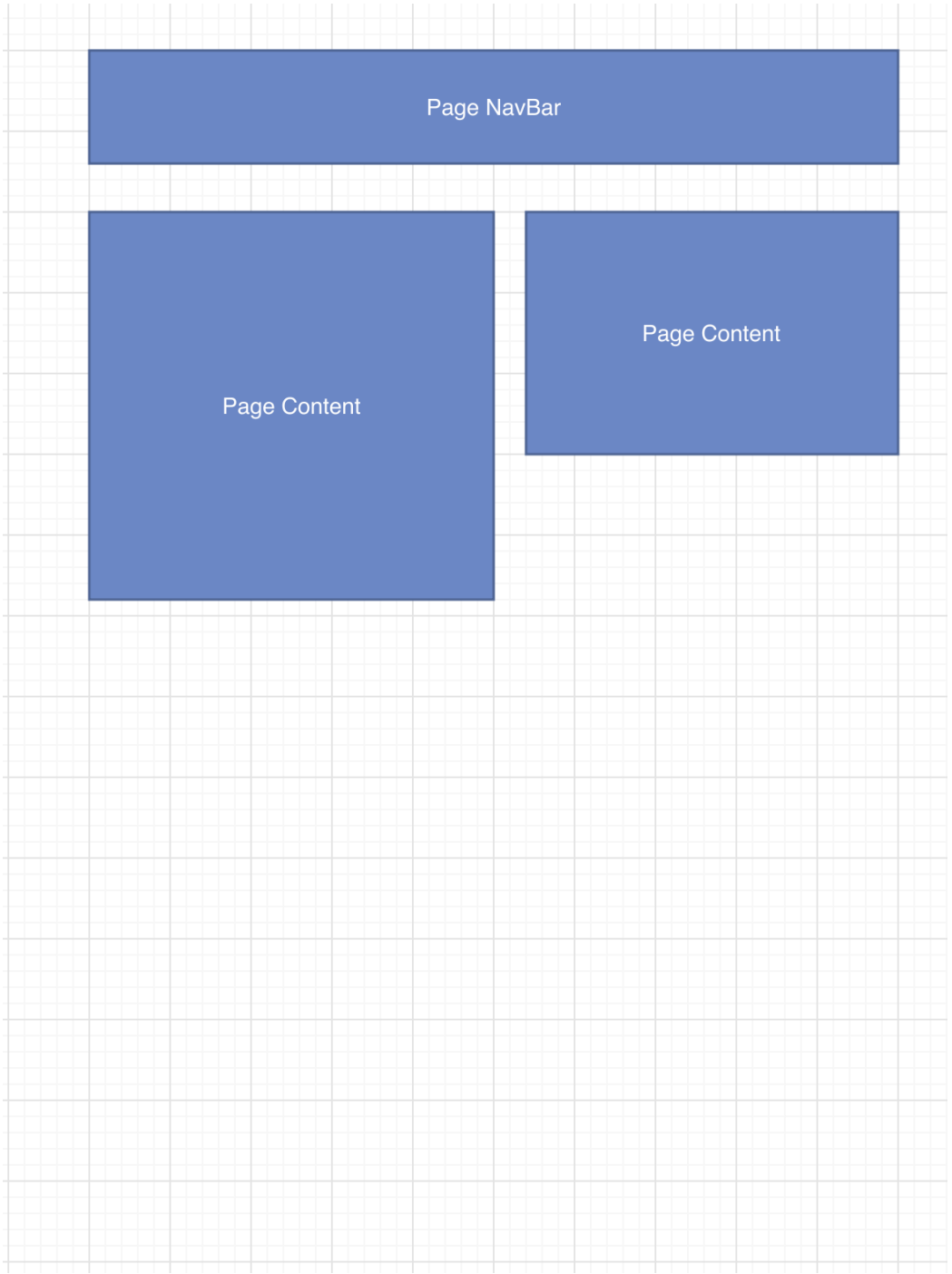
HCI Screen for the main page



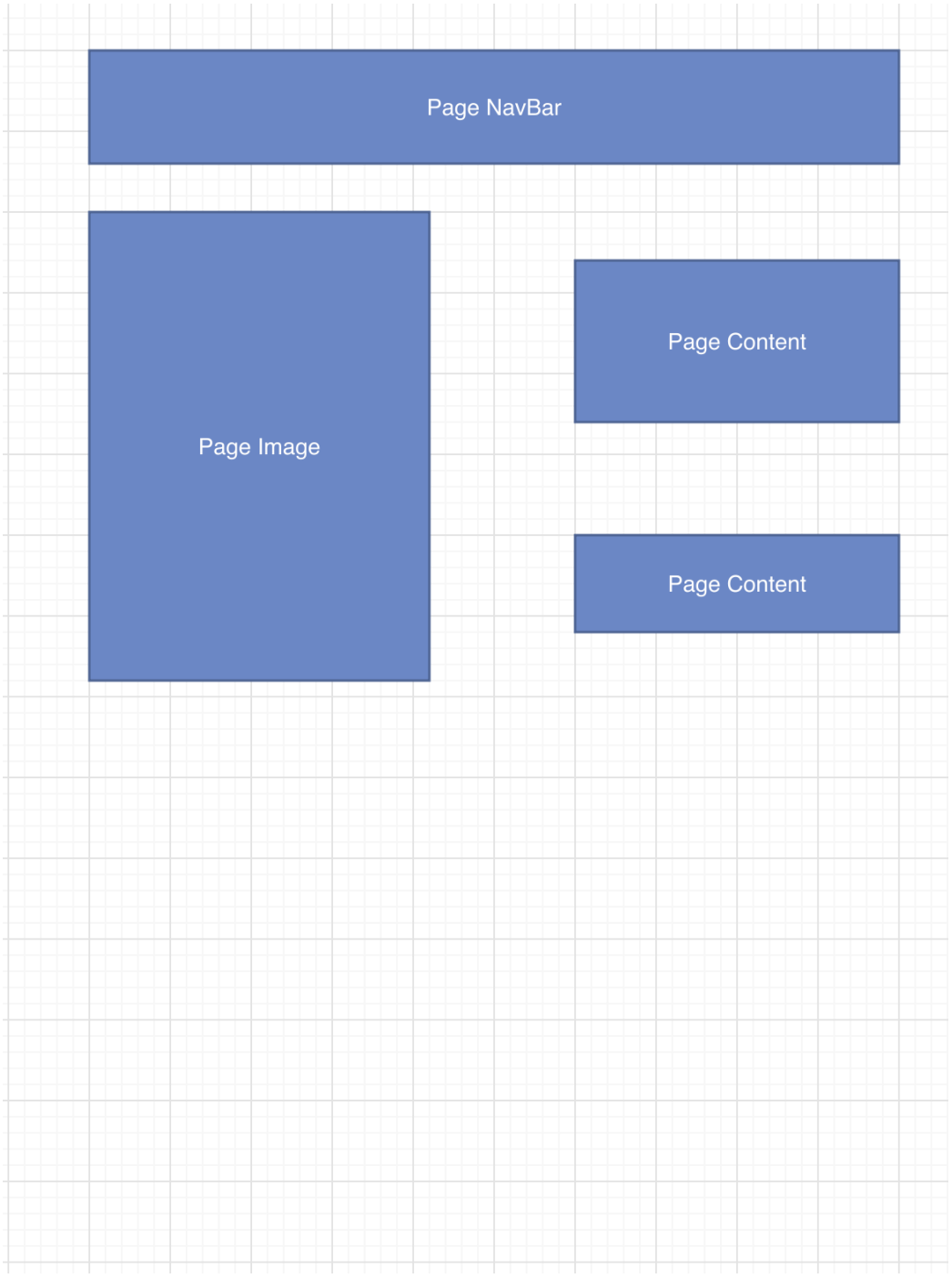
HCI Screen for the cart page



HCI Screen for the checkout page



HCI Screen for the product-details page



Technical Solution

Citations

(“https://stackoverflow.com/questions/24234296/google-calender-vr3-how-do-i-add-a-reminder-
to-an-event”) on the website stack overflow which helped me to fix some bugs

ecommerce

- asgi.py

```
import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'ecommerce.settings')

application = get_asgi_application()
```

- settings.py

```
import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY =
'django-insecure-vdq)2_f@#uz4m6%3ff21@vmma)blot=y&s)2z2+zb_s8zboc*j'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'users.apps.UsersConfig',
    'crispy_forms',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'store.apps.StoreConfig',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
```

```
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'ecommerce.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'ecommerce.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
#
https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validator
s

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
```

```
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static')
]

MEDIA_URL = '/images/'

MEDIA_ROOT = os.path.join(BASE_DIR, 'static/images')


DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

CRISPY_TEMPLATE_PACK = 'bootstrap4'

LOGIN_REDIRECT_URL = 'store'

LOGIN_URL = 'login'

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 2525
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'merchiston.store@gmail.com'
EMAIL_HOST_PASSWORD = 'Merchiston2022'
```

- urls.py

```
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import path, include
from django.conf.urls.static import static
from django.conf import settings

from users import views as user_views

urlpatterns = [
   path('admin/', admin.site.urls),
   path('register/', user_views.register, name='register'),
   path('profile/', user_views.profile, name='profile'),

   path('login/',
auth_views.LoginView.as_view(template_name='users/login.html'),
name='login'),
   path('logout/',
auth_views.LogoutView.as_view(template_name='users/logout.html'),
name='logout'),

   path('', include('store.urls'), name='store'),

   path('password-reset/',
auth_views.PasswordResetView.as_view(template_name='users/password_reset.html
'), name='password_reset'),
   path('password-reset/done/',
auth_views.PasswordResetDoneView.as_view(template_name='users/password_reset_
done.html'),
        name='password_reset_done'),
   path('password-reset-confirm/<uidb64>/<token>',
auth_views.PasswordResetConfirmView.as_view(template_name='users/password_res
et_confirm.html'),
       name='password_reset_confirm'),
   path('password-reset-complete/',
auth_views.PasswordResetCompleteView.as_view(template_name='users/password_re
set_complete.html'), name='password_reset_complete'),


]

if settings.DEBUG:
   urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

static

- main.css

```
body{
  background-color: hsl(0, 0%, 98%);
}

h1,h2,h3,h4,h5,h6{
  color:hsl(0, 0%, 30%);
}

.box-element{
  box-shadow:hsl(0, 0%, 80%) 0 0 16px;
  background-color: #fff;
  border-radius: 4px;
  padding: 10px;
}

.thumbnail{
  width: 100%;
  height: 200px;
  -webkit-box-shadow: -1px -3px 5px -2px rgba(214,214,214,1);
   -moz-box-shadow: -1px -3px 5px -2px rgba(214,214,214,1);
   box-shadow: -1px -3px 5px -2px rgba(214,214,214,1);
}

.product{
  border-radius: 0 0 4px 4px;
}

.bg-dark{
  background-color: #000080!important;
}

#cart-icon{
  width:25px;
  display: inline-block;
  margin-left: 15px;
}

#cart-total{
  display: block;
  text-align: center;
  color:#fff;
  background-color: red;
  width: 20px;
  height: 25px;
  border-radius: 50%;
  font-size: 14px;
}
```

```
.col-lg-4, .col-lg-6, .col-lg-8, .col-lg-12{
  margin-top: 10px;
}

.btn{
  border-radius: 0;
}

.row-image{
  width: 100px;
}

.form-field{
  width:250px;
  display: inline-block;
  padding: 5px;
}

.cart-row{
  display: flex;
   align-items: flex-stretch;
   padding-bottom: 10px;
   margin-bottom: 10px;
   border-bottom: 1px solid #ececec;

}

.quantity{
  display: inline-block;
  font-weight: 700;
  padding-right:10px;


}

.chg-quantity{
  width: 12px;
  cursor: pointer;
  display: block;
  margin-top: 5px;
  transition:.1s;
}

.chg-quantity:hover{
  opacity: .6;
}


.hidden{
  display: none!important;
}
```

- cart.js

```
var updateBtns = document.getElementsByClassName('update-cart')

for (i = 0; i < updateBtns.length; i++) {
  updateBtns[i].addEventListener('click', function(){
     var productId = this.dataset.product
     var action = this.dataset.action
     console.log('productId:', productId, 'Action:', action)
     console.log('USER:', user)

     if (user == 'AnonymousUser'){
        addCookieItem(productId, action)
     }else{
        updateUserOrder(productId, action)
     }
  })
}

function updateUserOrder(productId, action){
  console.log('User is authenticated, sending data...')

     var url = '/update_item/'

     fetch(url, {
        method:'POST',
        headers:{
           'Content-Type':'application/json',
           'X-CSRFToken':csrftoken,
        },
        body:JSON.stringify({'productId':productId, 'action':action})
     })
     .then((response) => {
        return response.json();
     })
     .then((data) => {
         location.reload()
     });
}

function addCookieItem(productId, action){
  console.log('User is not authenticated')

  if (action == 'add'){
     if (cart[productId] == undefined){
     cart[productId] = {'quantity':1}

     }else{
        cart[productId]['quantity'] += 1
     }
  }

  if (action == 'remove'){
```

```
    cart[productId]['quantity'] -= 1

    if (cart[productId]['quantity'] <= 0){
       console.log('Item should be deleted')
       delete cart[productId];
    }
  }
  console.log('CART:', cart)
  document.cookie ='cart=' + JSON.stringify(cart) + ";domain=;path=/"

  location.reload()
}
```

store

- templates

  - store

    cart.html

```
{% extends 'store/main.html' %}
{% load static %}
{% block content %}
  <div class="row">
     <div class="col-lg-12">
        <div class="box-element">

           <a  class="btn btn-outline-dark" href="{% url 'store' %}">&#x2190;
Continue Shopping</a>

           <br>
           <br>
           <table class="table">
              <tr>
                 <th><h5>Items:
<strong>{{order.get_cart_items}}</strong></h5></th>
                 <th><h5>Total:<strong>
${{order.get_cart_total|floatformat:2}}</strong></h5></th>
                 <th>
                    <a  style="float:right; margin:5px;" class="btn
btn-success" href="{% url 'checkout' %}">Checkout</a>
                 </th>
              </tr>
           </table>

        </div>
```

```
        <br>
        <div class="box-element">
           <div class="cart-row">
              <div style="flex:2"></div>
              <div style="flex:2"><strong>Item</strong></div>
              <div style="flex:1"><strong>Price</strong></div>
              <div style="flex:1"><strong>Quantity</strong></div>
              <div style="flex:1"><strong>Total</strong></div>
           </div>
           {% for item in items %}
           <div class="cart-row">
              <div style="flex:2"><img class="row-image"
src="{{item.product.imageURL}}"></div>
              <div style="flex:2"><p>{{item.product.name}}</p></div>
              <div
style="flex:1"><p>${{item.product.price|floatformat:2}}</p></div>
              <div style="flex:1">
                 <p class="quantity">{{item.quantity}}</p>
                 <div class="quantity">
                    <img data-product="{{item.product.id}}" data-action="add"
class="chg-quantity update-cart" src="{% static  'images/arrow-up.png' %}">

                    <img data-product="{{item.product.id}}"
data-action="remove" class="chg-quantity update-cart" src="{% static
'images/arrow-down.png' %}">
                 </div>
              </div>
              <div
style="flex:1"><p>${{item.get_total|floatformat:2}}</p></div>
           </div>
           {% endfor %}
        </div>
     </div>
  </div>
{% endblock content %}
```

checkout.html

```
{% extends 'store/main.html' %}
{% load static %}
{% block content %}
    <div class="row">
     <div class="col-lg-6">
        <div class="box-element" id="form-wrapper">
           <form id="form">
                   {% csrf_token %}
              <div id="user-info">
                 <div class="form-field">
                    <input required class="form-control" type="text"
name="name" placeholder="Name..">
                 </div>
                 <div class="form-field">
                    <input required class="form-control" type="email"
name="email" placeholder="Email..">
                 </div>
              </div>

              <div id="shipping-info">
                 <hr>
                 <p>Shipping Information:</p>
                 <hr>
                 <div class="form-field">
                    <input class="form-control" type="text" name="address"
placeholder="Address..">
                 </div>
                 <div class="form-field">
                    <input class="form-control" type="text" name="city"
placeholder="City..">
                 </div>
                 <div class="form-field">
                    <input class="form-control" type="text" name="state"
placeholder="State..">
                 </div>
                 <div class="form-field">
                    <input class="form-control" type="text" name="zipcode"
placeholder="Zip code..">
                 </div>
                 <div class="form-field">
                    <input class="form-control" type="text" name="country"
placeholder="Zip code..">
                 </div>
              </div>

              <hr>
              <input id="form-button" class="btn btn-success btn-block"
type="submit" value="Continue">
           </form>
        </div>
```

```
        <br>
        <div class="box-element hidden" id="payment-info">
           <small>Paypal Options</small>
           <button id="make-payment">Make payment</button>
        </div>

     </div>

     <div class="col-lg-6">
        <div class="box-element">
           <a  class="btn btn-outline-dark" href="{% url 'cart' %}">&#x2190;
Back to Cart</a>
           <hr>
           <h3>Order Summary</h3>
           <hr>
           {% for item in items %}
           <div class="cart-row">
              <div style="flex:2"><img class="row-image"
src="{{item.product.imageURL}}"></div>
              <div style="flex:2"><p>{{item.product.name}}</p></div>
              <div
style="flex:1"><p>${{item.product.price|floatformat:2}}</p></div>
              <div style="flex:1"><p>x{{item.quantity}}</p></div>
           </div>
           {% endfor %}
           <h5>Items:   {{order.get_cart_items}}</h5>
           <h5>Total:   ${{order.get_cart_total|floatformat:2}}</h5>
        </div>
     </div>
  </div>

  <script type="text/javascript">
     var shipping = '{{order.shipping}}'
     var total = '{{order.get_cart_total|floatformat:2}}'

     if (shipping == 'False'){
        document.getElementById('shipping-info').innerHTML = ''
     }

     if (user != 'AnonymousUser'){
        document.getElementById('user-info').innerHTML = ''
      }

     if (shipping == 'False' && user != 'AnonymousUser'){
        //Hide entire form if user is logged in and shipping is false
           document.getElementById('form-wrapper').classList.add("hidden");
           //Show payment if logged in user wants to buy an item that does
not require shipping

document.getElementById('payment-info').classList.remove("hidden");
     }

     var form = document.getElementById('form')
```

```
    form.addEventListener('submit', function(e){
        e.preventDefault()
        console.log('Form Submitted...')
        document.getElementById('form-button').classList.add("hidden");
        document.getElementById('payment-info').classList.remove("hidden");
    })

    document.getElementById('make-payment').addEventListener('click',
function(e){
        submitFormData()
    })

    function submitFormData(){
        console.log('Payment button clicked')

        var userFormData = {
          'name':null,
          'email':null,
          'total':total,
        }

        var shippingInfo = {
          'address':null,
          'city':null,
          'state':null,
          'zipcode':null,
        }

        if (shipping != 'False'){
            shippingInfo.address = form.address.value
            shippingInfo.city = form.city.value
            shippingInfo.state = form.state.value
            shippingInfo.zipcode = form.zipcode.value
        }

        if (user == 'AnonymousUser'){
            userFormData.name = form.name.value
            userFormData.email = form.email.value
        }

        console.log('Shipping Info:', shippingInfo)
        console.log('User Info:', userFormData)

        var url = "/process_order/"
        fetch(url, {
            method:'POST',
            headers:{
                'Content-Type':'applicaiton/json',
                'X-CSRFToken':csrftoken,
            },
            body:JSON.stringify({'form':userFormData,
'shipping':shippingInfo}),
```

```
          })
          .then((response) => response.json())
          .then((data) => {
            console.log('Success:', data);
            alert('Transaction completed');

            cart = {}
            document.cookie ='cart=' + JSON.stringify(cart) +
";domain=;path=/"

            window.location.href = "{% url 'store' %}"

            })
      }
  </script>
{% endblock content %}
```

main.html

```
<!DOCTYPE html>
{% load static %}
<html>
<head>
  <title>Eshop</title>

  <meta name="viewport" content="width=device-width, initial-scale=1,
maximum-scale=1, minimum-scale=1" />

  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.cs
s"
integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q
9Ifjh" crossorigin="anonymous">

  <link rel="stylesheet" type="text/css" href="{% static 'css/main.css' %}">

  <script type="text/javascript">
     var user = '{{request.user}}'

     function getToken(name) {
         var cookieValue = null;
         if (document.cookie && document.cookie !== '') {
             var cookies = document.cookie.split(';');
             for (var i = 0; i < cookies.length; i++) {
                 var cookie = cookies[i].trim();
                 // Does this cookie string begin with the name we want?
                  if (cookie.substring(0, name.length + 1) === (name + '='))
{
                      cookieValue =
decodeURIComponent(cookie.substring(name.length + 1));
                      break;
                 }
             }
         }
         return cookieValue;
     }
     var csrftoken = getToken('csrftoken');

     function getCookie(name) {
         // Split cookie string and get all individual name=value pairs in
an array
         var cookieArr = document.cookie.split(";");

         // Loop through the array elements
         for(var i = 0; i < cookieArr.length; i++) {
             var cookiePair = cookieArr[i].split("=");

             /* Removing whitespace at the beginning of the cookie name
              and compare it with the given string */
             if(name == cookiePair[0].trim()) {
```

```
                // Decode the cookie value and return
                return decodeURIComponent(cookiePair[1]);
            }
        }

        // Return null if not found
        return null;
    }
    var cart = JSON.parse(getCookie('cart'))

    if (cart == undefined){
       cart = {}
       console.log('Cart Created!', cart)
       document.cookie ='cart=' + JSON.stringify(cart) + ";domain=;path=/"
    }
    console.log('Cart:', cart)

  </script>

</head>
<body>

  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <a class="navbar-brand" href="{% url 'store' %}">Merchiston Eshop</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav mr-auto">
        <li class="nav-item active">
          <a class="nav-link" href="{% url 'store' %}">Store <span
class="sr-only">(current)</span></a>
        </li>
           <a class="nav-link" href="{% url 'school_uniform' %}">School
Uniform <span class="sr-only"></span></a>


      </ul>
      <div class="form-inline my-2 my-lg-0">
           {% if user.is_authenticated %}
               <a href="{% url 'profile' %}"class="btn
btn-warning">Profile</a>
            <a href="{% url 'logout' %}"class="btn btn-warning">Logout</a>

           {% else %}
             <a href="{% url 'login' %}"class="btn btn-warning">Login</a>
               <a href="{% url 'register' %}"class="btn
btn-warning">Register</a>
           {% endif %}
           <a href="{% url 'cart' %}">
```

```
            <img  id="cart-icon" src="{% static 'images/cart.png' %}">
          </a>
          <p id="cart-total">{{cartItems}}</p>

      </div>
    </div>
  </nav>

    <div class="container">
           <br>

           {% if messages %}
               {% for message in messages %}
                   <div class="alert alert-{{ message.tags }}">
                   {{ message }}
                   </div>
               {% endfor %}
           {% endif %}
           {% block content %}



           {% endblock content %}
        </div>


  <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRS
JoZ+n" crossorigin="anonymous"></script>

  <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxM
fooAo" crossorigin="anonymous"></script>

  <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8
ifwB6" crossorigin="anonymous"></script>

  <script type="text/javascript" src="{% static 'js/cart.js' %}"></script>
</body>
</html>
```

product.html

```
{% extends "store/main.html" %}



{% block content %}
{% load static %}


<link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}">

<!DOCTYPE html>
<html lang="">
 <head>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>Tutorial</title>
   <!-- Fonts -->
   <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500"
rel="stylesheet">
   <!-- CSS -->
   <link href="static/css/style.css" rel="stylesheet">
   <meta name="robots" content="noindex,follow" />

 </head>

 <body>
   <main class="container">

     <!-- Left Column / Headphones Image -->



     <!-- Right Column -->
     <div class="right-column">

       <!-- Product Description -->
       <div class="product-description">
         <span></span>
           <h1>{{product.name}}</h1>
                 <div class="left-column">
       <img data-image="black" src="{{ product.imageURL }}">


     </div>
           <p>"{{product.description}}"</p>
       </div>


       <!-- Product Configuration -->

       </div>
```

```
        <!-- Product Pricing -->
        <div class="product-price">
            <button data-product="{{product.pk}}" data-action="add" class="btn
btn-outline-secondary add-btn update-cart">Add to Cart</button>

            <div class="product-configuration">
              <a href="#">How to take the measurements</a>
            </div>
        </div>
      </div>
    </main>

    <!-- Scripts -->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.1/jquery.min.js"
charset="utf-8"></script>
    <script src="static/js/script.js" charset="utf-8"></script>
  </body>
</html>

{% endblock %}
```

products_list.html

```
{% extends "store/main.html" %}

{% block content %}
   <h2>Products</h2>
   <ul>
       {% for product in products %}
           <li><a href="{% url 'product-detail' product.id %}">{{
product.name }} </a></li>
       {% endfor %}

   </ul>
{% endblock %}
```

school_uniform.html

```
{% extends 'store/main.html' %}
{% load static %}
{% block content %}
  <div class="row">
     {% for product in products %}
     <div class="col-lg-4">
        <img class="thumbnail" src="{{ product.imageURL }}">
        <div class="box-element product">
           <h6><strong>{{product.name}}</strong></h6>
           <hr>

               <button data-product="{{product.pk}}" data-action="add"
class="btn btn-outline-secondary add-btn update-cart">Add to Cart</button>

               <a class="btn btn-outline-secondary" href="{% url
'product-detail' product.pk%}">View</a>
               <h4 style="display: inline-block; float:
right"><strong>${{product.price|floatformat:2}}</strong></h4>

        </div>
     </div>
     {% endfor %}
  </div>

{% endblock content %}
```

store.html

```
{% extends 'store/main.html' %}
{% load static %}
{% block content %}
  <div class="row">
     {% for product in products %}
     <div class="col-lg-4">
        <img class="thumbnail" src="{{ product.imageURL }}">
        <div class="box-element product">
           <h6><strong>{{product.name}}</strong></h6>
           <hr>

               <button data-product="{{product.id}}" data-action="add"
class="btn btn-outline-secondary add-btn update-cart">Add to Cart</button>

               <a class="btn btn-outline-secondary" href="{% url
'product-detail' product.id%}">View</a>          <h4 style="display:
inline-block; float:
right"><strong>${{product.price|floatformat:2}}</strong></h4>

        </div>
     </div>
     {% endfor %}
  </div>

{% endblock content %}
```

store

- templates

- store

- static

style.css

```
/* Basic Styling */
html, body {
 height: 100%;
 width: 100%;
 margin: 0;
 font-family: 'Roboto', sans-serif;
}

.container {
 max-width: 1200px;
 margin: 0 auto;
 padding: 15px;
 display: flex;
}

/* Columns */
.left-column {
 width: 65%;
 position: relative;
}

.right-column {
 width: 35%;
 margin-top: 60px;
}


/* Left Column */
.left-column img {
 width: 100%;
 position: absolute;
 left: 0;
 top: 0;
 opacity: 0;
 transition: all 0.3s ease;
}
```

```
.left-column img.active {
 opacity: 1;
}


/* Right Column */

/* Product Description */
.product-description {
 border-bottom: 1px solid #E1E8EE;
 margin-bottom: 20px;
}
.product-description span {
 font-size: 12px;
 color: #358ED7;
 letter-spacing: 1px;
 text-transform: uppercase;
 text-decoration: none;
}
.product-description h1 {
 font-weight: 300;
 font-size: 52px;
 color: #43484D;
 letter-spacing: -2px;
}
.product-description p {
 font-size: 16px;
 font-weight: 300;
 color: #86939E;
 line-height: 24px;
}

/* Product Configuration */
.product-color span,
.cable-config span {
 font-size: 14px;
 font-weight: 400;
 color: #86939E;
 margin-bottom: 20px;
 display: inline-block;
}

/* Product Color */
.product-color {
 margin-bottom: 30px;
}

.color-choose div {
 display: inline-block;
}

.color-choose input[type="radio"] {
```

```
 display: none;
}

.color-choose input[type="radio"] + label span {
 display: inline-block;
 width: 40px;
 height: 40px;
 margin: -1px 4px 0 0;
 vertical-align: middle;
 cursor: pointer;
 border-radius: 50%;
}

.color-choose input[type="radio"] + label span {
 border: 2px solid #FFFFFF;
 box-shadow: 0 1px 3px 0 rgba(0,0,0,0.33);
}

.color-choose input[type="radio"]#red + label span {
 background-color: #C91524;
}
.color-choose input[type="radio"]#blue + label span {
 background-color: #314780;
}
.color-choose input[type="radio"]#black + label span {
 background-color: #323232;
}

.color-choose input[type="radio"]:checked + label span {
 background-image: url(templates/store/static/);
 background-repeat: no-repeat;
 background-position: center;
}

/* Cable Configuration */
.cable-choose {
 margin-bottom: 20px;
}

.cable-choose button {
 border: 2px solid #E1E8EE;
 border-radius: 6px;
 padding: 13px 20px;
 font-size: 14px;
 color: #5E6977;
 background-color: #fff;
 cursor: pointer;
 transition: all .5s;
}

.cable-choose button:hover,
.cable-choose button:active,
.cable-choose button:focus {
```

```
 border: 2px solid #86939E;
 outline: none;
}

.cable-config {
 border-bottom: 1px solid #E1E8EE;
 margin-bottom: 20px;
}

.cable-config a {
 color: #358ED7;
 text-decoration: none;
 font-size: 12px;
 position: relative;
 margin: 10px 0;
 display: inline-block;
}
.cable-config a:before {
 content: "?";
 height: 15px;
 width: 15px;
 border-radius: 50%;
 border: 2px solid rgba(53, 142, 215, 0.5);
 display: inline-block;
 text-align: center;
 line-height: 16px;
 opacity: 0.5;
 margin-right: 5px;
}

/* Product Price */
.product-price {
 display: flex;
 align-items: center;
}

.product-price span {
 font-size: 26px;
 font-weight: 300;
 color: #43474D;
 margin-right: 20px;
}

.cart-btn {
 display: inline-block;
 background-color: #7DC855;
 border-radius: 6px;
 font-size: 16px;
 color: #FFFFFF;
 text-decoration: none;
 padding: 12px 30px;
 transition: all .5s;
}
```

```
.cart-btn:hover {
 background-color: #64af3d;
}

/* Responsive */
@media (max-width: 940px) {
 .container {
   flex-direction: column;
   margin-top: 60px;
 }

 .left-column,
 .right-column {
   width: 100%;
 }

 .left-column img {
   width: 300px;
   right: 0;
   top: -65px;
   left: initial;
 }
}

@media (max-width: 535px) {
 .left-column img {
   width: 220px;
   top: -85px;
 }
}
```

script.js

```
$(document).ready(function() {

 $('.color-choose input').on('click', function() {
     var htmlColor = $(this).attr('data-image');

     $('.active').removeClass('active');
     $('.left-column img[data-image = ' + htmlColor +
']').addClass('active');
     $(this).addClass('active');
 });

});
```

users

admin.py

```
from django.contrib import admin
from .models import Profile

admin.site.register(Profile)
```

apps.py

```
from django.apps import AppConfig


class UsersConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'users'

    def ready(self):
        import users.signals
```

forms.py

```
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from crispy_forms.helper import FormHelper
from .models import Profile


class UserRegisterForm(UserCreationForm):
    email = forms.EmailField()

    # keeps the configurations in one place.
    # Setting the required fields by default=True
    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']


class UserUpdateForm(forms.ModelForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email']


class ProfileUpdateForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ['image']
```

models.py

```
from django.db import models
from django.contrib.auth.models import User
from PIL import Image


class Profile(models.Model):
    user = models.OneToOneField(User, null=True, on_delete=models.CASCADE)
    image = models.ImageField(default='default.jpg', upload_to='profile_pics')

    def __str__(self):
        return f'{self.user.username} Profile'

    # This function resizes images to 300 x 300
    def save(self, *args, **kwargs):
        super(Profile, self).save(*args, **kwargs)

        img = Image.open(self.image.path)

        if img.height > 300 or img.width > 300:
            output_size = (300, 300)
            img.thumbnail(output_size)
            img.save(self.image.path)
```

signals.py

```
from django.db.models.signals import post_save
from django.contrib.auth.models import User
from django.dispatch import receiver
from .models import Profile


@receiver(post_save, sender=User)
def create_profile(sender, instance, created, **kwargs):
   if created:
       Profile.objects.create(user=instance)


@receiver(post_save, sender=User)
def save_profile(sender, instance, **kwargs):
   instance.profile.save()
```

views.py

```
import email

from django.contrib.auth import authenticate, login
from django.shortcuts import render, redirect
from django.contrib import messages

from store.models import Customer
from .forms import UserRegisterForm, UserUpdateForm, ProfileUpdateForm
from django.contrib.auth.decorators import login_required

# logic for registration page
def register(request):
   if request.method == 'POST':
       form = UserRegisterForm(request.POST)
       if form.is_valid():
           form.save()  # This line saves the user information and hashes
the password
           username = form.cleaned_data.get('username')
           raw_password = form.cleaned_data.get('password1')
           user = authenticate(username=username, password=raw_password)
           Customer.objects.create(user=user, name=username, email=email)
           messages.success(request, f'Your account has been updated! You are
now able to log in')
           login(request, user)

           return redirect('login')
   else:
       form = UserRegisterForm()
   return render(request, 'users/register.html', {'form': form})

@login_required
def profile(request):
   if request.method == 'POST':
       u_form = UserUpdateForm(request.POST, instance=request.user)
       p_form = ProfileUpdateForm(request.POST, request.FILES,
instance=request.user)

       if u_form.is_valid() and p_form.is_valid():
           u_form.save()
           p_form.save()
           messages.success(request, f'Your account has been updated!')
           return redirect('profile')
   else:
       u_form = UserUpdateForm(instance=request.user)
       p_form = ProfileUpdateForm(instance=request.user.profile)

   context = {
       'u_form': u_form,
       'p_form': p_form
   }
   return render(request, 'users/profile.html', context)
```

users

- templates
    - users

login.html

```
{% extends 'store/main.html' %}
{% load crispy_forms_filters %}
{% load crispy_forms_tags %}
{% block content %}
   <div class="content-section">
       <form method="POST">
           {% csrf_token %}
           <fieldset class="form-group">
               <legend class="border-bottom mb-4">Log In</legend>
               {{ form|crispy }}
           </fieldset>
           <div class="form-group">
               <button class="btn btn-outline-info"
type="submit">Login</button>
               <small class="text-muted ml-2">
                   <a href="{% url 'password_reset' %}">Forgot Password?</a>
               </small>
           </div>
       </form>
       <div class="border-top pt-3">
           <small class="text-muted">
               Need An Account? <a class="ml-2" href="{% url 'register'
%}">Sign Up Now</a>
           </small>
       </div>
   </div>
{% endblock content %}
```

logout.html

```
{% extends 'store/main.html' %}
{% block content %}

   <h2>You have been logged out</h2>
   <div class="border-top pt-3">
       <small class="text-muted">
           <a href="{% url 'login' %}">Log In Again</a>
       </small>
   </div>
{% endblock content %}
```

password_reset.html

```
{% extends 'store/main.html' %}
{% load crispy_forms_filters %}
{% load crispy_forms_tags %}
{% block content %}
   <div class="content-section">
       <form method="POST">
           {% csrf_token %}
           <fieldset class="form-group">
               <legend class="border-bottom mb-4">Reset Password</legend>
               {{ form|crispy }}
           </fieldset>
           <div class="form-group">
               <button class="btn btn-outline-info" type="submit">Request
Password Reset</button>
           </div>
       </form>

   </div>
{% endblock content %}
```

password_reset_complete.html

```
{% extends 'store/main.html' %}
{% block content %}
   <div class="alert alert-info">
       Your password has been set
   </div>
   <a href="{% url 'login' %}">Sign In Here</a>

{% endblock content %}
```

password_reset_confirm.html

```
{% extends 'store/main.html' %}
{% load crispy_forms_filters %}
{% load crispy_forms_tags %}
{% block content %}
   <div class="content-section">
       <form method="POST">
           {% csrf_token %}
           <fieldset class="form-group">
               <legend class="border-bottom mb-4">Reset
Password</legend>
               {{ form|crispy }}
           </fieldset>
           <div class="form-group">
               <button class="btn btn-outline-info"
type="submit">Reset Password</button>
           </div>
       </form>

   </div>
{% endblock content %}
```

password_reset_done.html

```
{% extends 'store/main.html' %}
{% block content %}
   <div class="alert alert-info">
       An email has been sent with instructions to reset your password
   </div>

{% endblock content %}
```

profile.html

```
{% extends 'store/main.html' %}
{% load crispy_forms_filters %}
{% load crispy_forms_tags %}
{% block content %}
   <div class="content-section">
       <div class="media">
           <img class="rounded-circle account-img" src="{{
user.profile.image.url }}">
           <div class="media-body">
               <h2 class="account-heading">{{ user.username }}</h2>
               <p class="text-secondary">{{ user.email}}</p>
           </div>
       </div>
       <form method="POST" enctype="multipart/form-data">
           {% csrf_token %}
           <fieldset class="form-group">
               <legend class="border-bottom mb-4">Profile Info</legend>
               {{ u_form|crispy }}
               {{ p_form|crispy }}
           </fieldset>
           <div class="form-group">
               <button class="btn btn-outline-info"
type="submit">Update</button>
           </div>
       </form>
   </div>
{% endblock content %}
```

register.html

```
{% extends 'store/main.html' %}
{% load crispy_forms_filters %}
{% load crispy_forms_tags %}
{% block content %}
   <div class="content-section">
       <form method="POST">
           {% csrf_token %}
           <fieldset class="form-group">
               <legend class="border-bottom mb-4">Join Today</legend>
               {{ form|crispy }}
           </fieldset>
           <div class="form-group">
               <button class="btn btn-outline-info" type="submit">Sign
Up</button>
           </div>
       </form>
       <div class="border-top pt-3">
           <small class="text-muted">
               Already Have An Account? <a class="ml-2" href="{% url 'login'
%}">Sign In</a>
           </small>
       </div>
   </div>
{% endblock content %}
```

Objectives

Objectives - Customers

4. The website will include a registration page
    a. Details required for registration:
        i. username, password, password verification, email address

5. My system must include a store page
    a. Panel with item groups and a return option:
        i. Item groups
            1. Names of the item categories (t-shirts, trousers, PE kit, etc.)

        ii. Return option
            1. Specification of items that need to be returned

    b. Displayed items of a chosen item group
        i. Each item will have an image and price
            1. Each item will have an item page that will include a little description and a table with sizes

6. The website will have a basket
    a. Product that are desired to be purchased will be displayed
    b. Shipping Address
        i. Address, town, post code, confirmation of post code, state

Objectives - Admin

2. My system will have an admin panel

    a. Admin user will be able to add items option:
        i. Here the admin user will be able to add items to certain categories. This will include price, images, sizes grid, name of the item

    b. My system will include deleting items option:
        i. Here the admin user will be able to delete items from the school shop

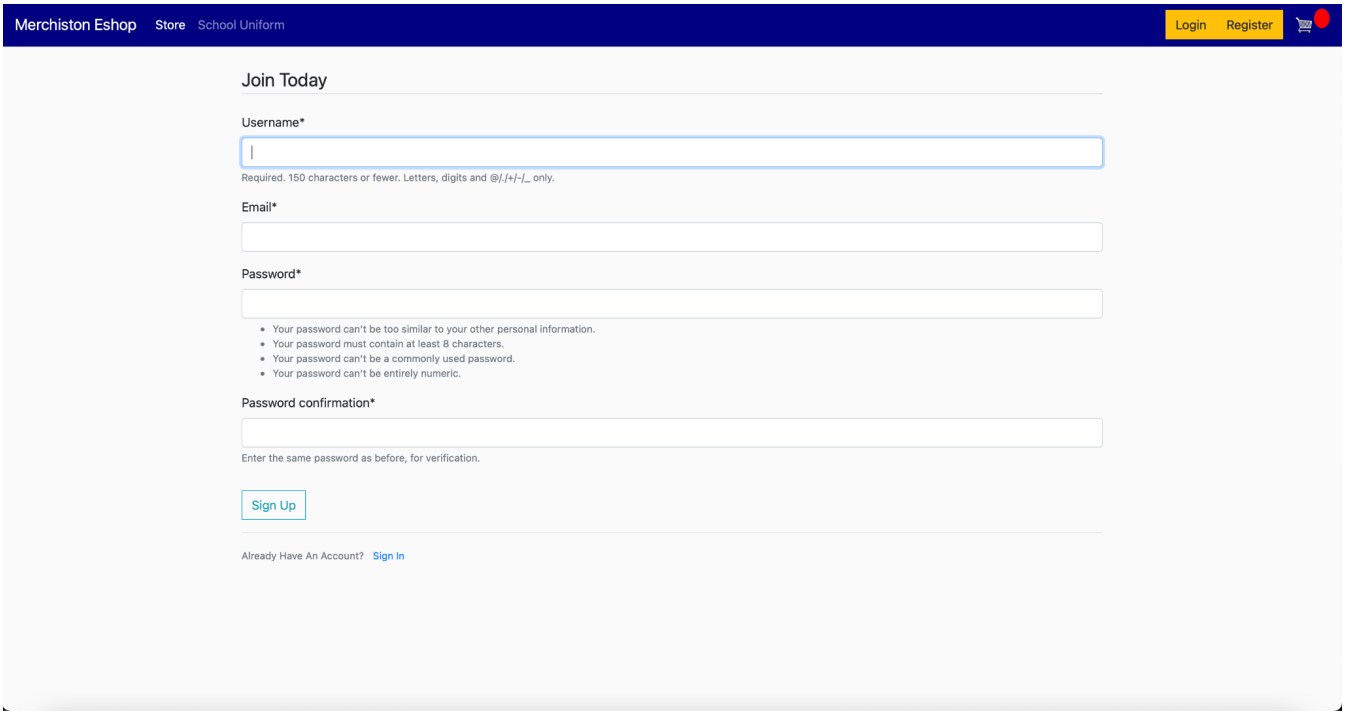
    c. My project provides an orders option:
        i. Page with orders
            1. Shipping address
                a. Adress, city, postcode, country
            2. Product information
                a. Quantity
                b. Product name

    d. Returned items option will be included:
        i. Database with returned items

Customer objectives - Result vs Expectations

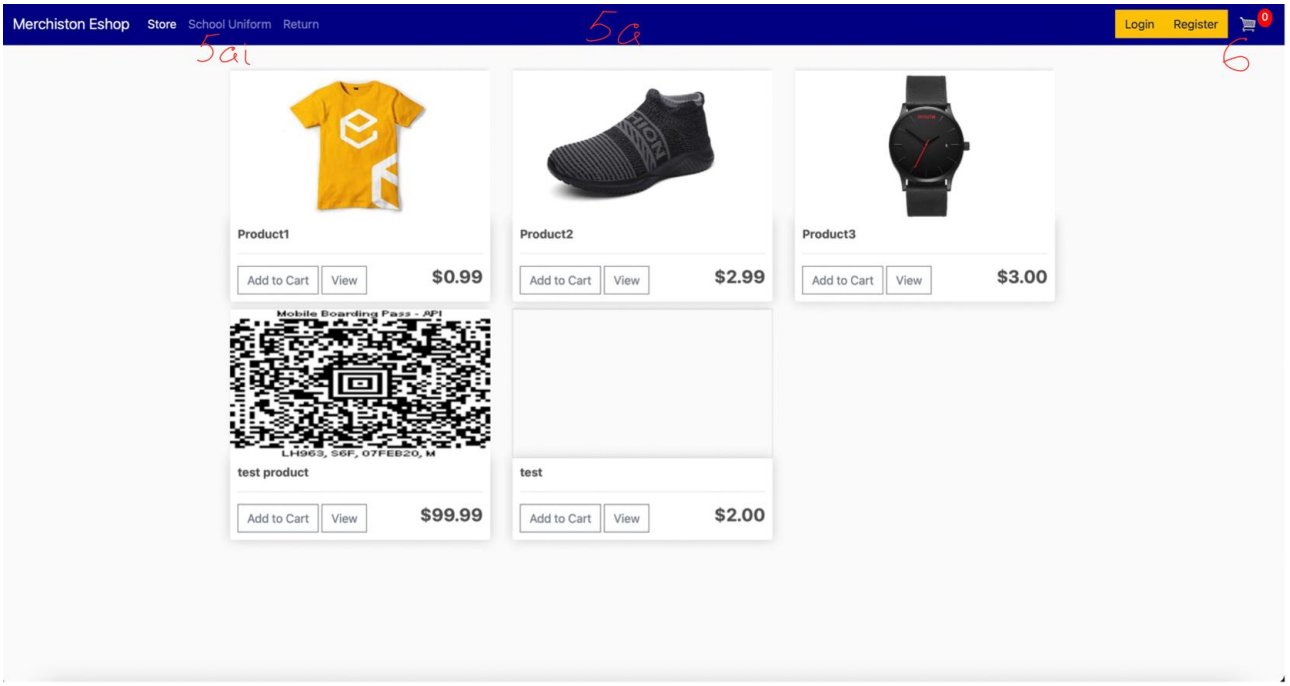
4.a i, 4 b i

I have implemented a different solution due to school policy limitations for the registration page. The registration page has fields for username, email and password so students and parents can log in in the same way.



5 a i, 5 a, 6

I created a navigation bar to represent the panel with item groups, return options and basket.

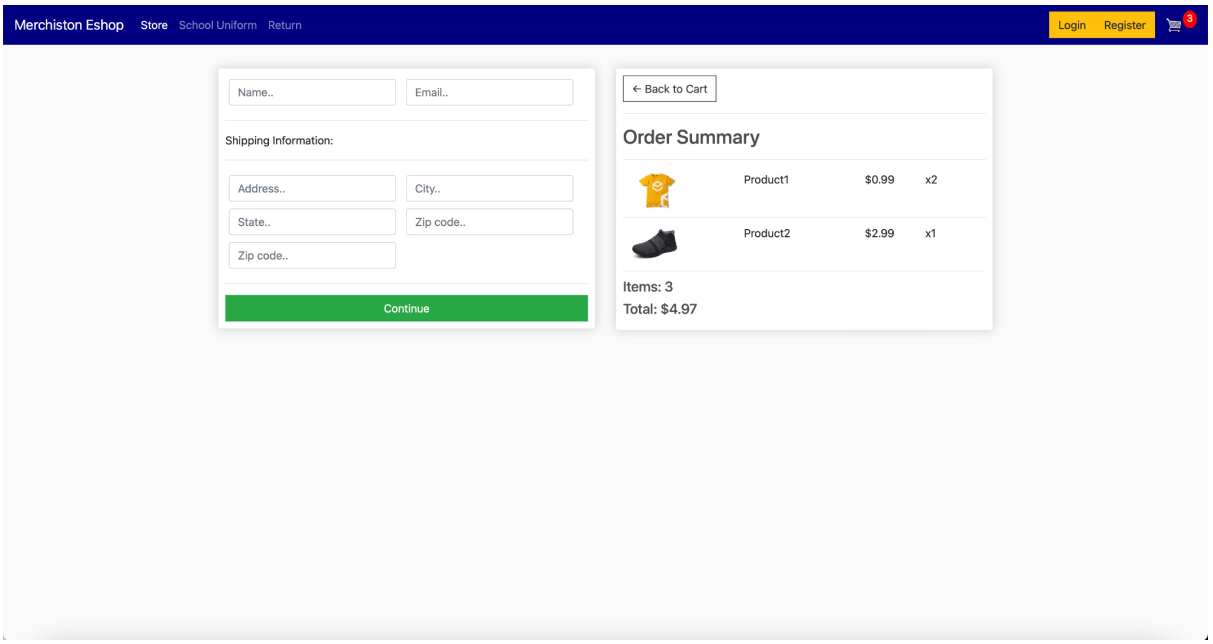


5 b i

Here is the product detail page. The name of the product, price and description will be unique for each product.

6 a i, 6 b i

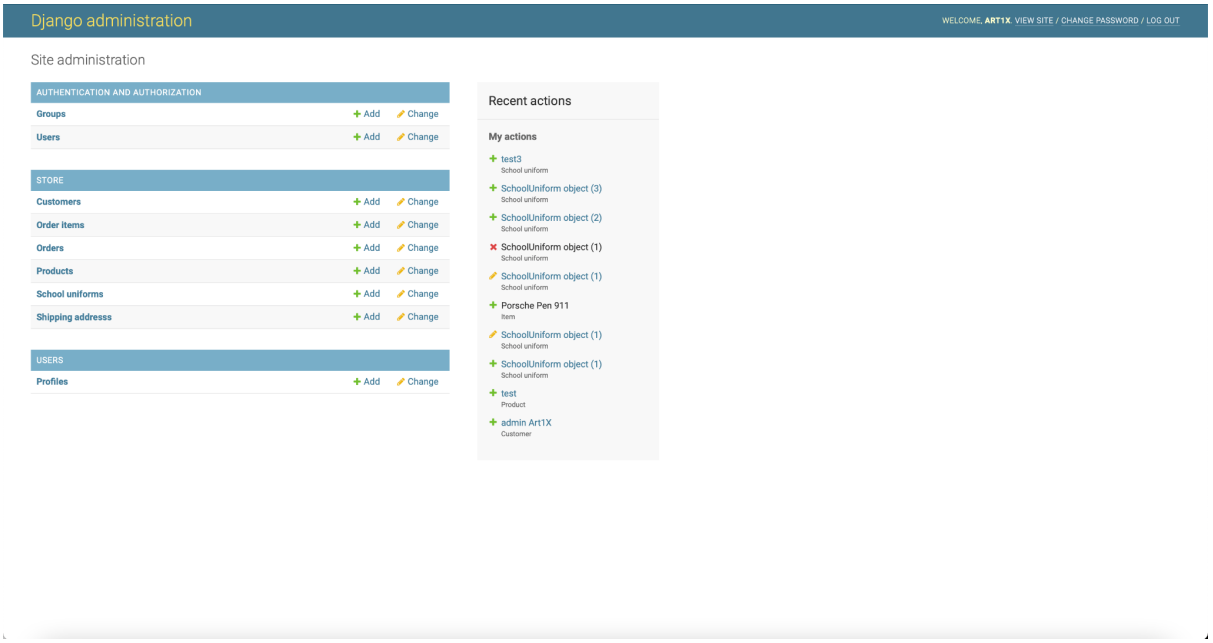
Since there is no separate login page for parents and students, I have implemented the required information for the international shipping.



Admin objectives - Result vs Expectations

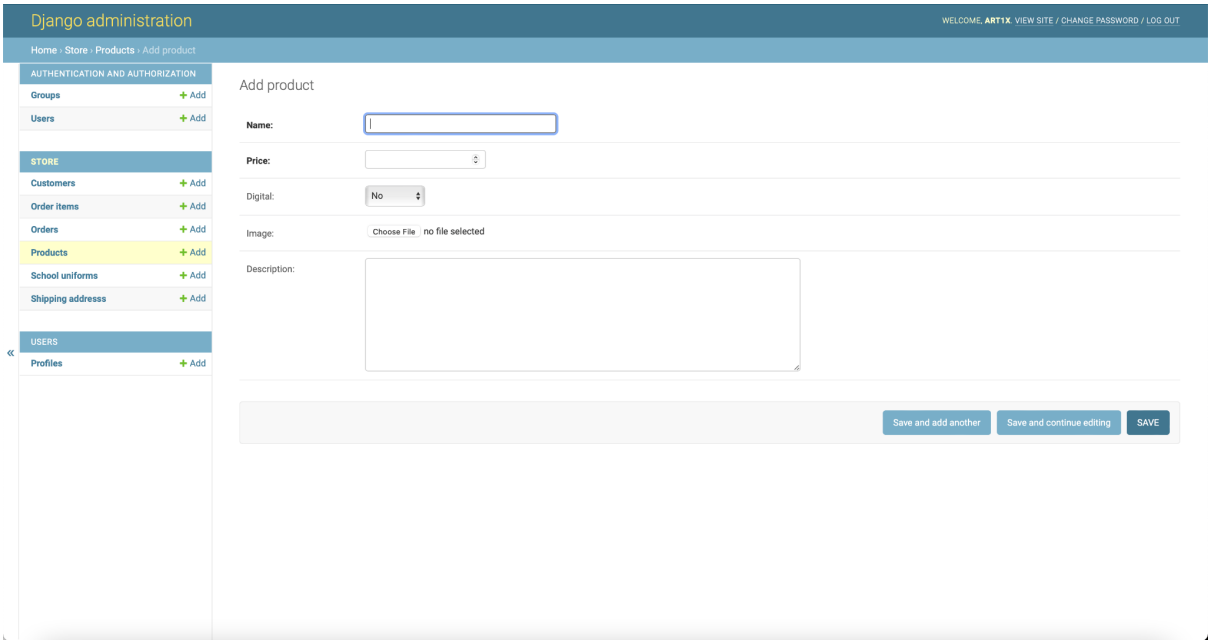
2

Here is the admin panel



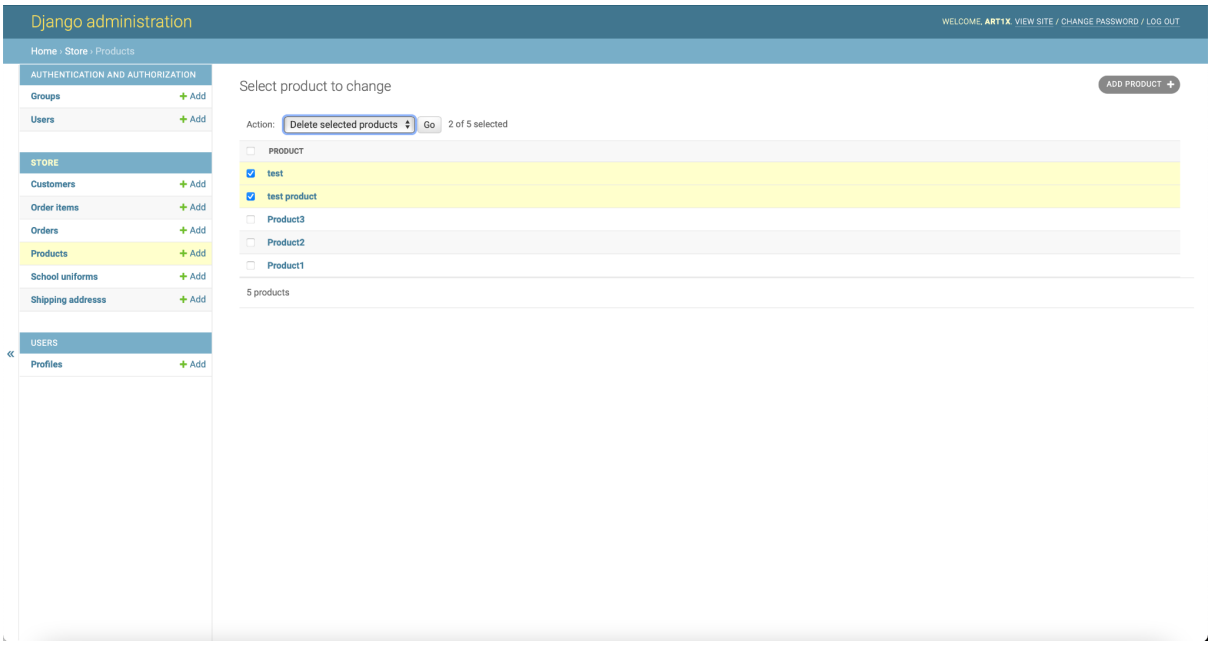
2 a i

Here I have created the option for the admin to add items to the store. The admin can change price, name, and description, and choose the image. If it is a digital item, it means that there will be no shipping address required.



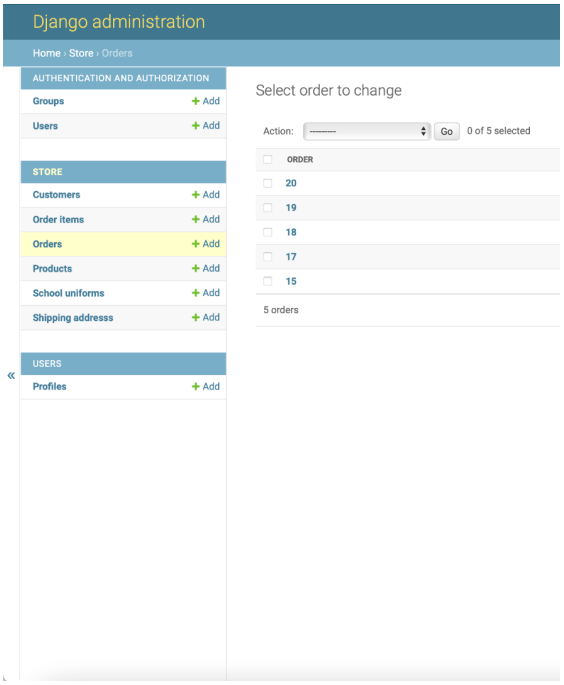
2 b i

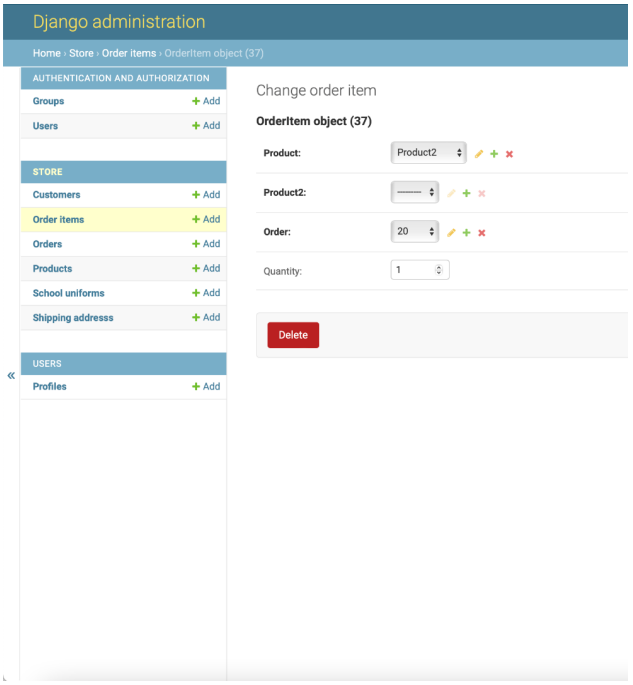
Here the admin can delete items.

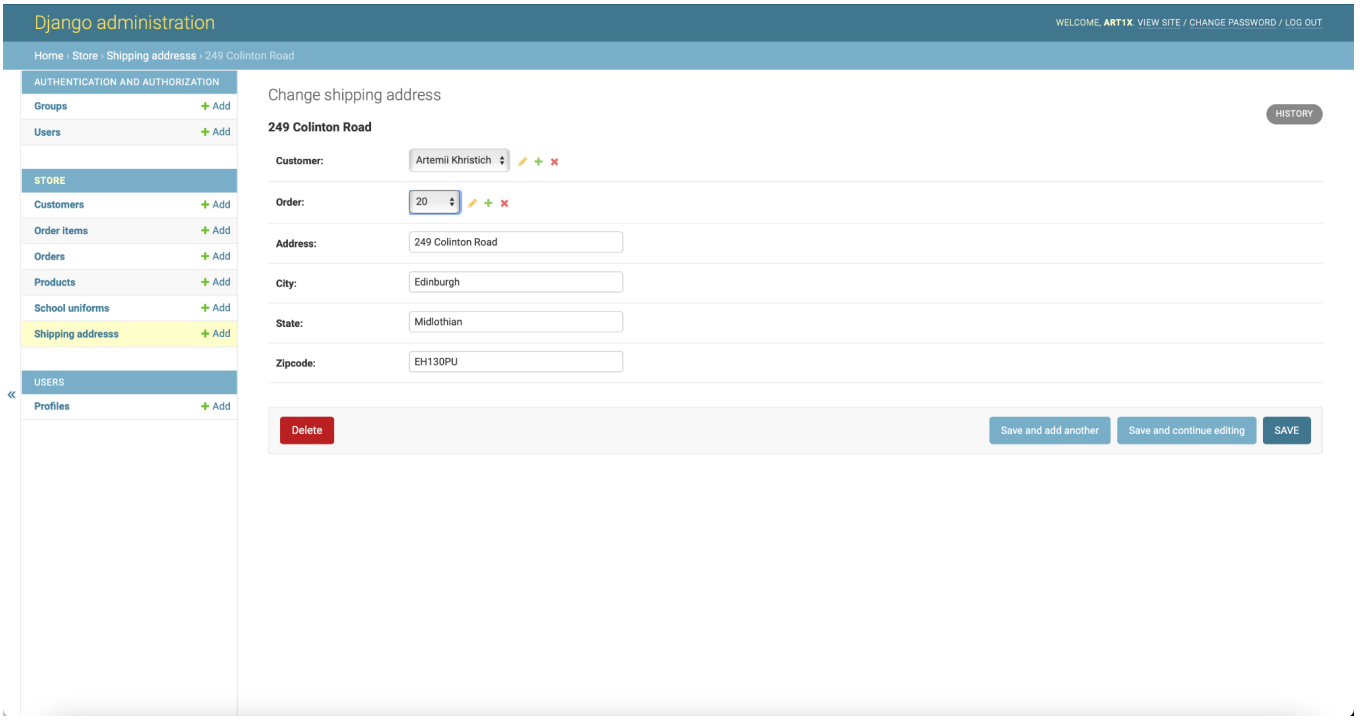


2 c

I have changed the structure of the website and implemented a different solution. All orders are listed in the same table. This increases the efficiency of the website and makes it more suitable for mobile devices.







2 d i

I have implemented a different solution for the returned items due to school policy. Students should email the school shop manager to return items

Testing

Here I will be testing objectives that I have written in the analysis section and testing them to see if they have been met.

Here are the objectives that I am going to test:

Objectives - Customers

7. The website will include a registration page
    a. Details required for registration:
        i. username, password, password verification, email address

8. My system must include a store page
    a. Panel with item groups and a return option:
        i. Item groups
            1. Names of the item categories (t-shirts, trousers, PE kit, etc.)

        ii. Return option
            1. Specification of items that need to be returned

    b. Displayed items of a chosen item group
        i. Each item will have an image and price
            1. Each item will have an item page that will include a little description and a table with sizes

9. The website will have a basket
    a. Product that is desired to be purchased will be displayed
    b. Shipping Address
        i. Address, town, postcode, confirmation of postcode, state

Objectives - Admin

3. My system will have an admin panel

    a. Admin user will be able to add items option:
        i. Here the admin user will be able to add items to certain categories. This will include price, images, sizes grid, and name of item

    b. My system will include deleting items option:
        i. Here the admin user will be able to delete items from the school shop

    c. My project provides an orders option:
        i. Page with orders
            1. Shipping address
                a. Adress, city, postcode, country
            2. Product information
                a. Quantity
                b. Product name

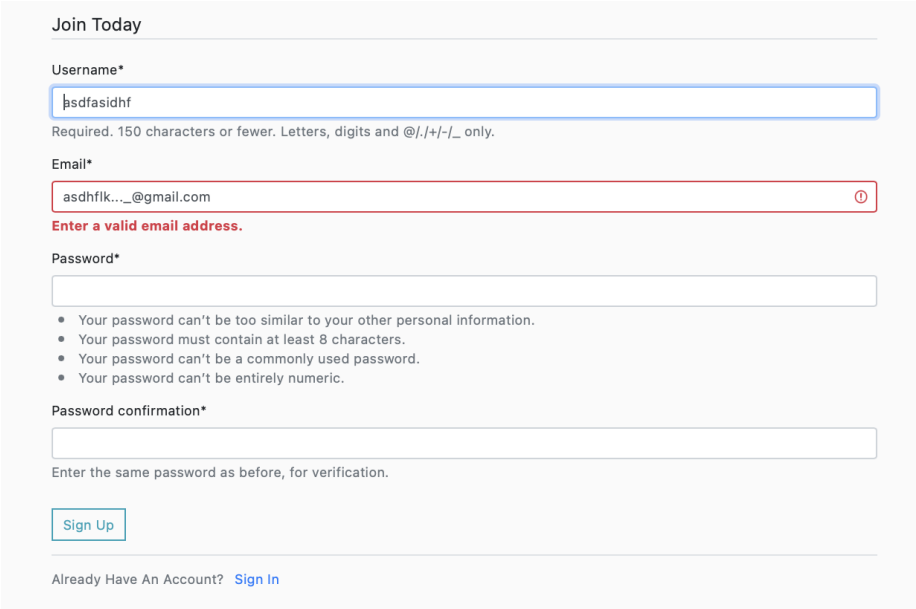
    d. Returned items option will be included:
        i. Database with returned items

Testing Objectives for Customers

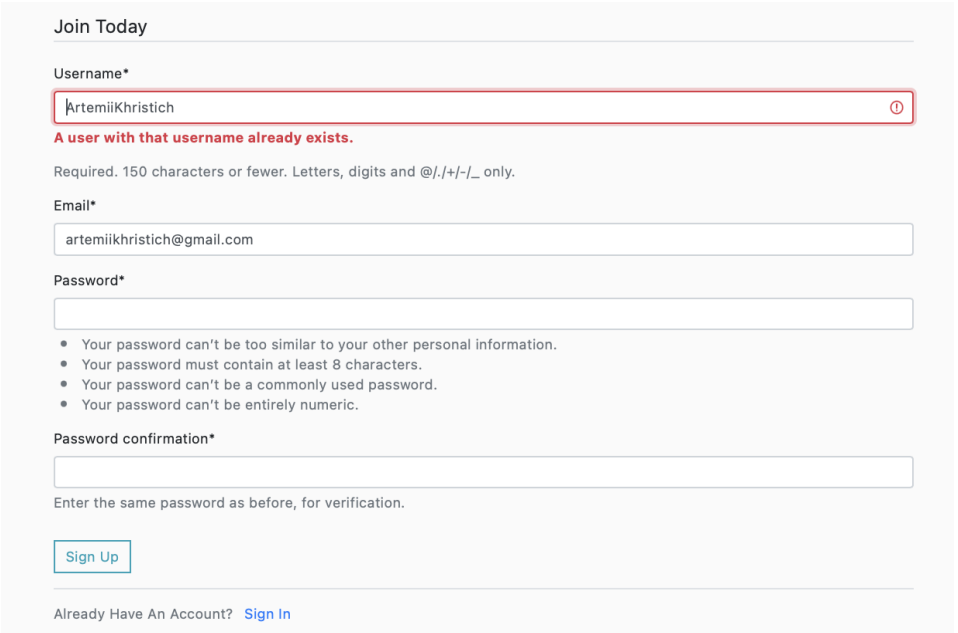
Testing for Objective 7

Objective 7 a. i.

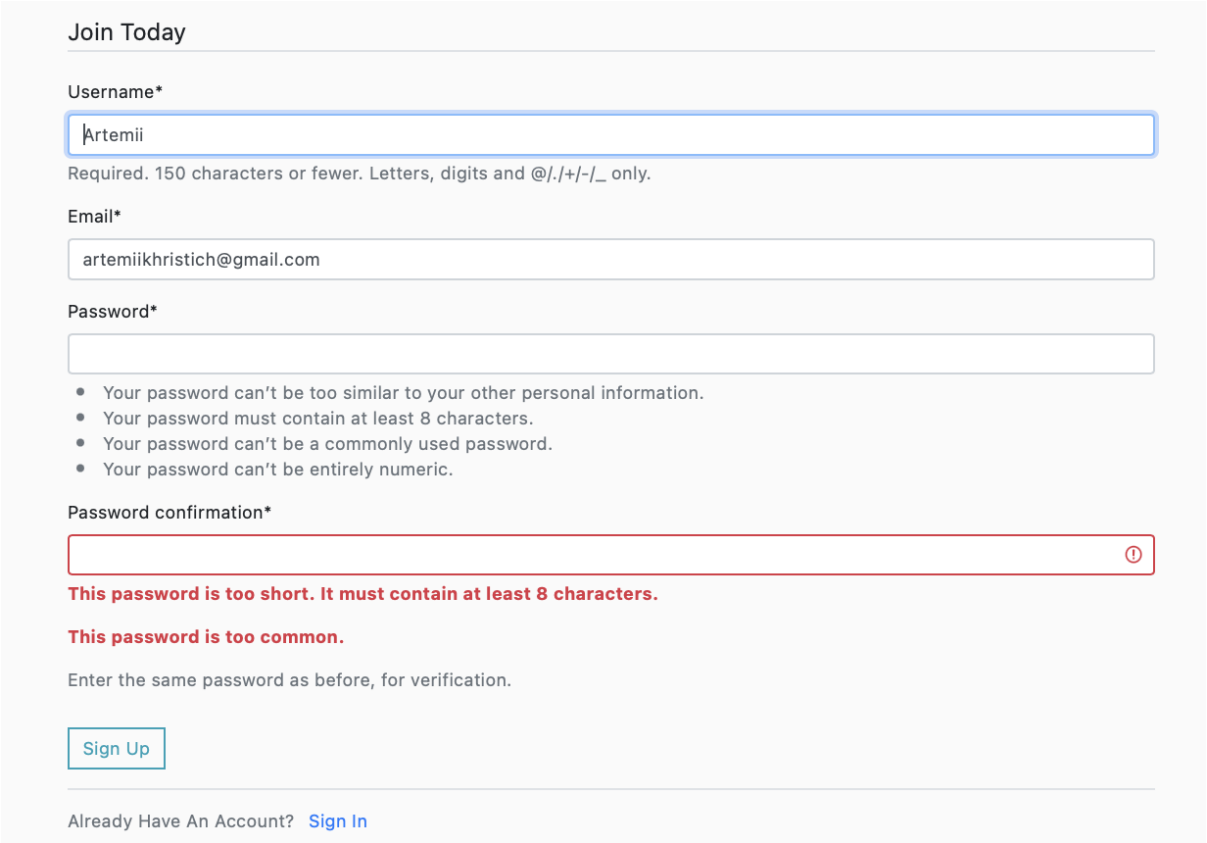
Here is an example of email address validation, if the user enters an email address that is not valid, the program doesn't allow the user to be registered.



If the client enters a username that already exists, the program will not allow the registration.



There is a certain password standard that will not allow registration to have proceeded if the requirements are not met.

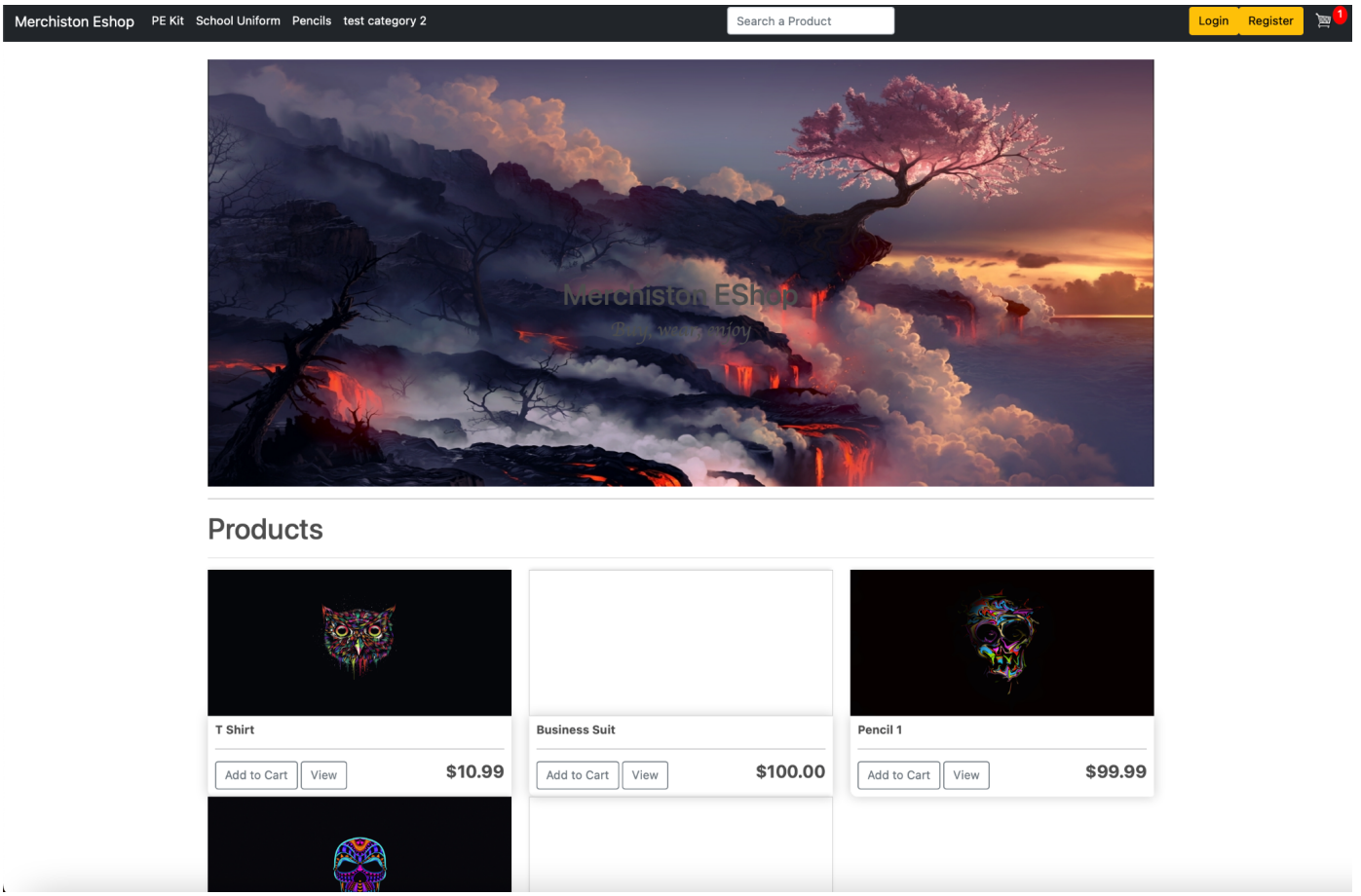


Testing Objective 8

Objective 8:

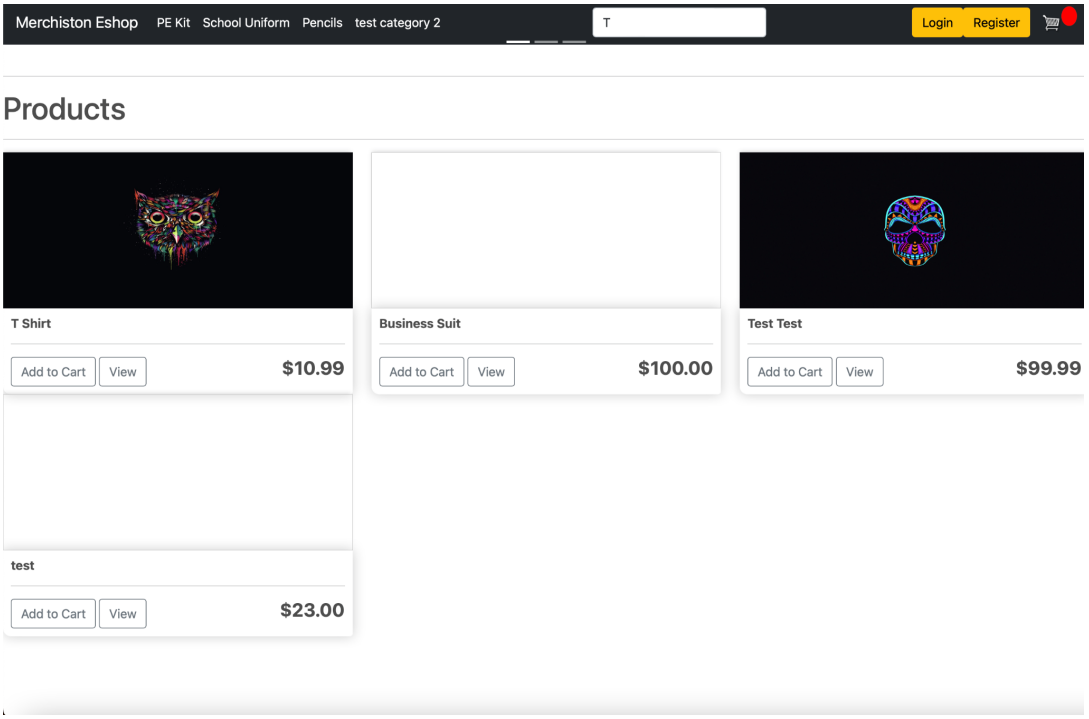
Here is a store page with a carousel and items displayed. A carousel is an additional feature that is not a part of my objectives, but it will give a more user-friendly interface. Also, the image and text on the carousel can be edited from the admin panel. It will be shown later, in the admin objectives testing.

I have implemented a search function on my website so that it would be easier for clients to find desired items. This is an additional feature to my website that will make use of my web application easier and more efficient.

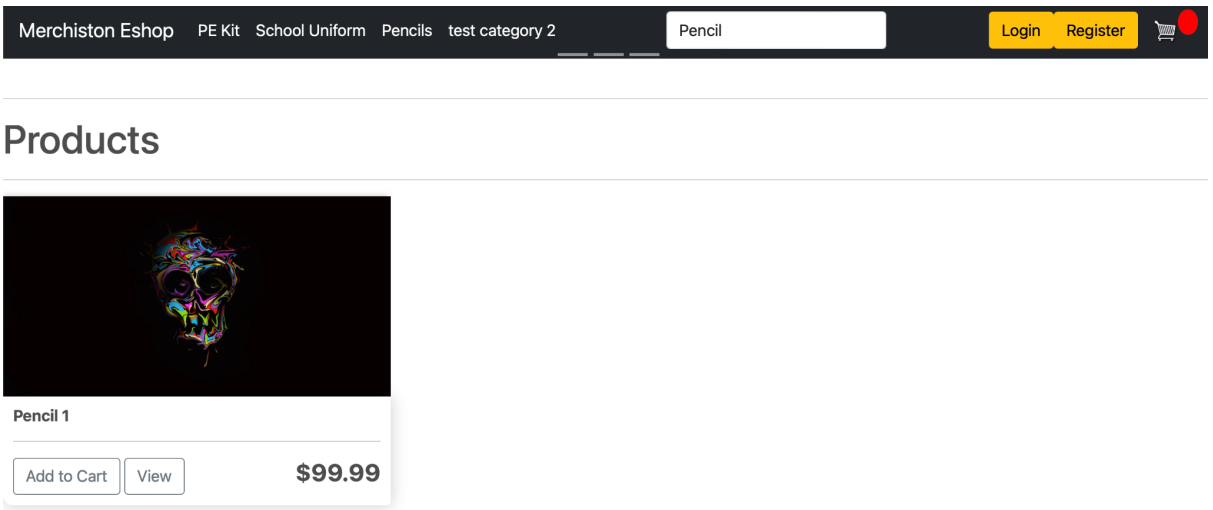


Testing for the relevance of the Search function

This is not part of my objectives, but it has to be tested for the relevance of the searched items.



Here I tested search functionality for relevance and it can be seen that if the user searches for items that begin with T for instance, the result is quite relevant.



If the user searches for a specific item, it also gives a relevant result.

Testing Objective 8 a. i. 1.

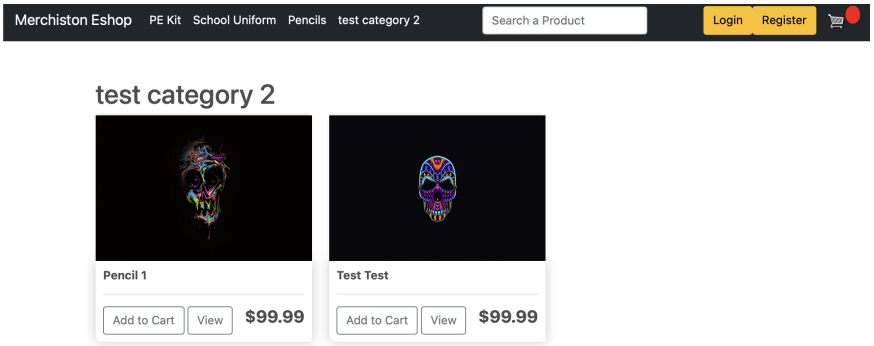
Here is the panel with category groups which can be edited through the admin panel and will be tested in the admin objective section later.

Testing for Objective 8 a. ii. 1.

This objective cannot be done since the school policy says that if the student wants to return an item, they have to go to the school shop for the school shop manager to look at the state of the item and decide on a return. Therefore, this objective is not done because it would add additional useless functionality in this case.

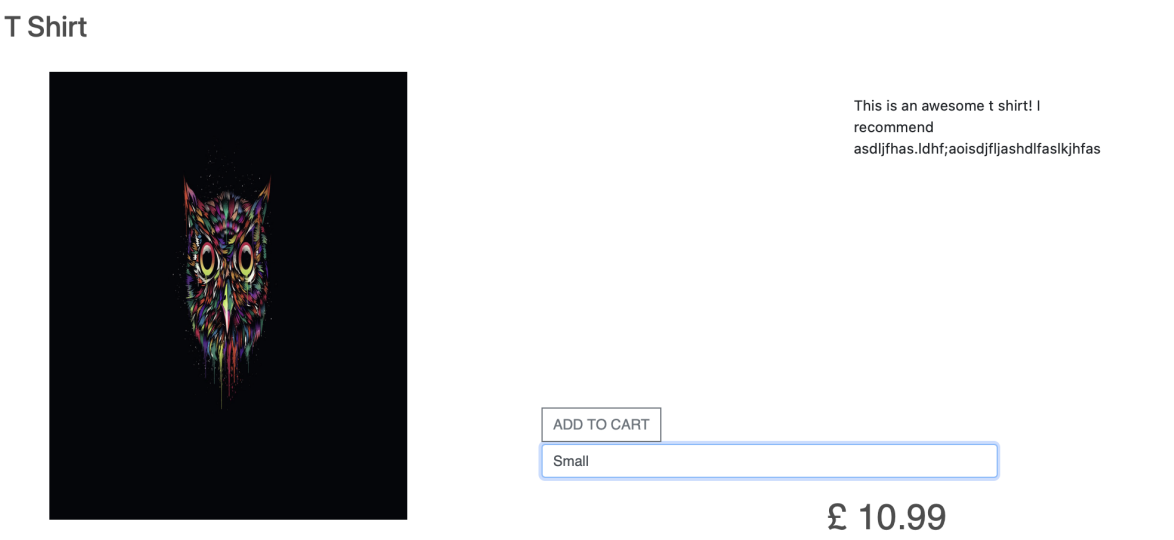
Testing Objective 8 b.

Here are the items for a certain category, they can be added through the admin panel and it will be shown in more detail later in the admin testing section.

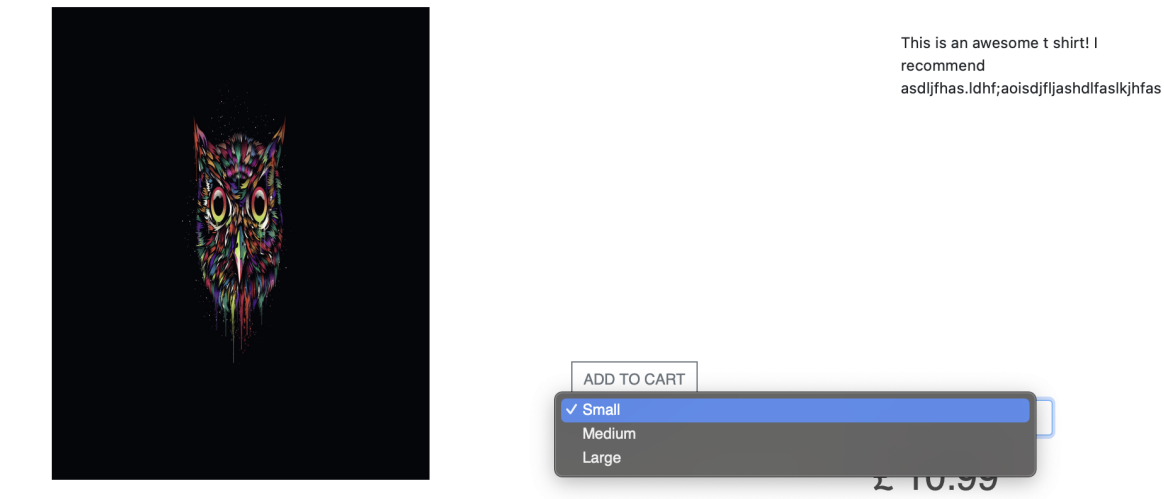


Testing Objective 8 b. i. 1.

This is the detailed page for an item that is automatically generated with every item. This page contains names, prices, descriptions and sizes for items that can be edited through the admin panel.

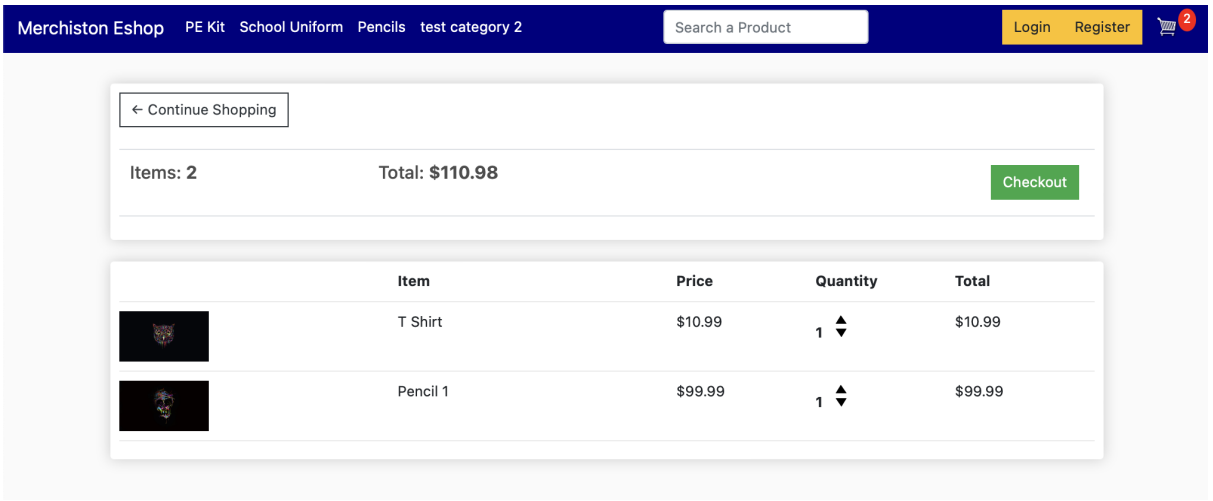


Here the user can choose a specific variation of sizes.

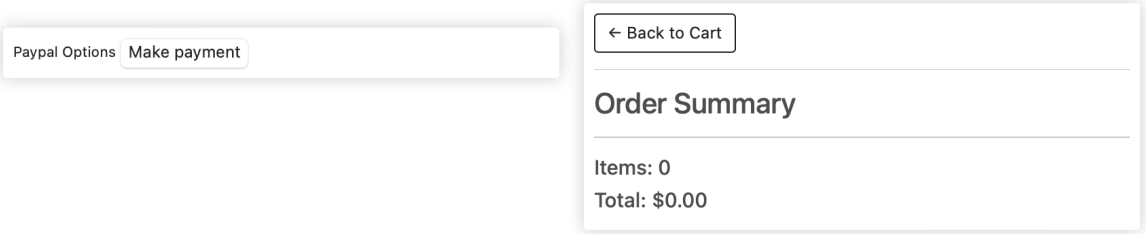


Testing Objective 9 a.

Here is the basket that includes the name of the item, quantity that can be changed directly from the basket, the price for individual items and the total price of the items in the basket.

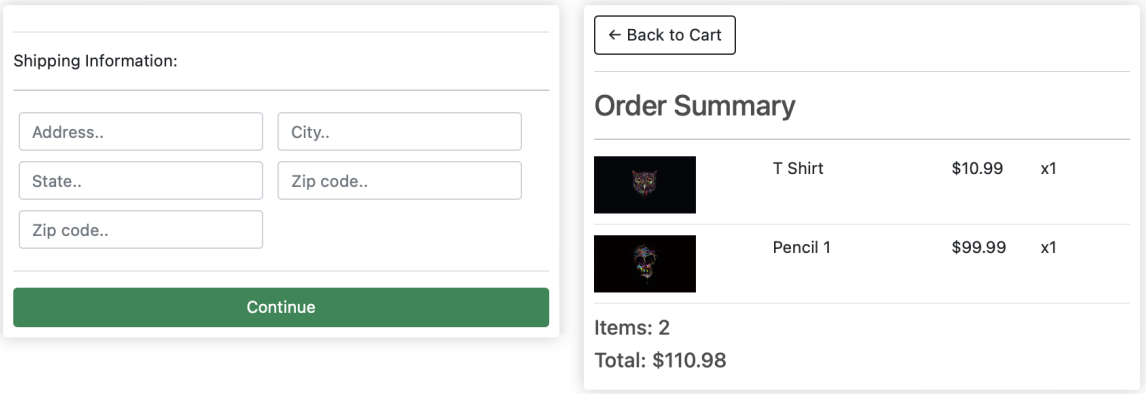


Here is the example of when the user removes all items from the basket. It can be seen that the price is 0 and therefore, it will not allow the user to enter the destination to avoid pointless data in the admin panel.

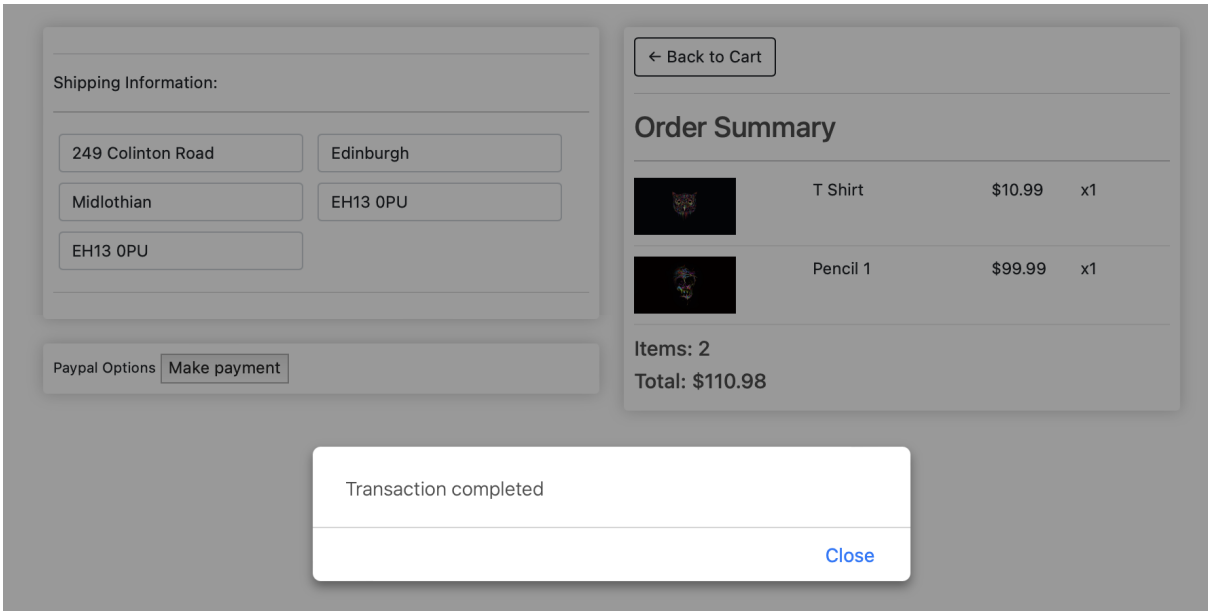


Testing Objective 9 b. i.

Here are the shipping address fields that are required to be filled in to make a purchase.



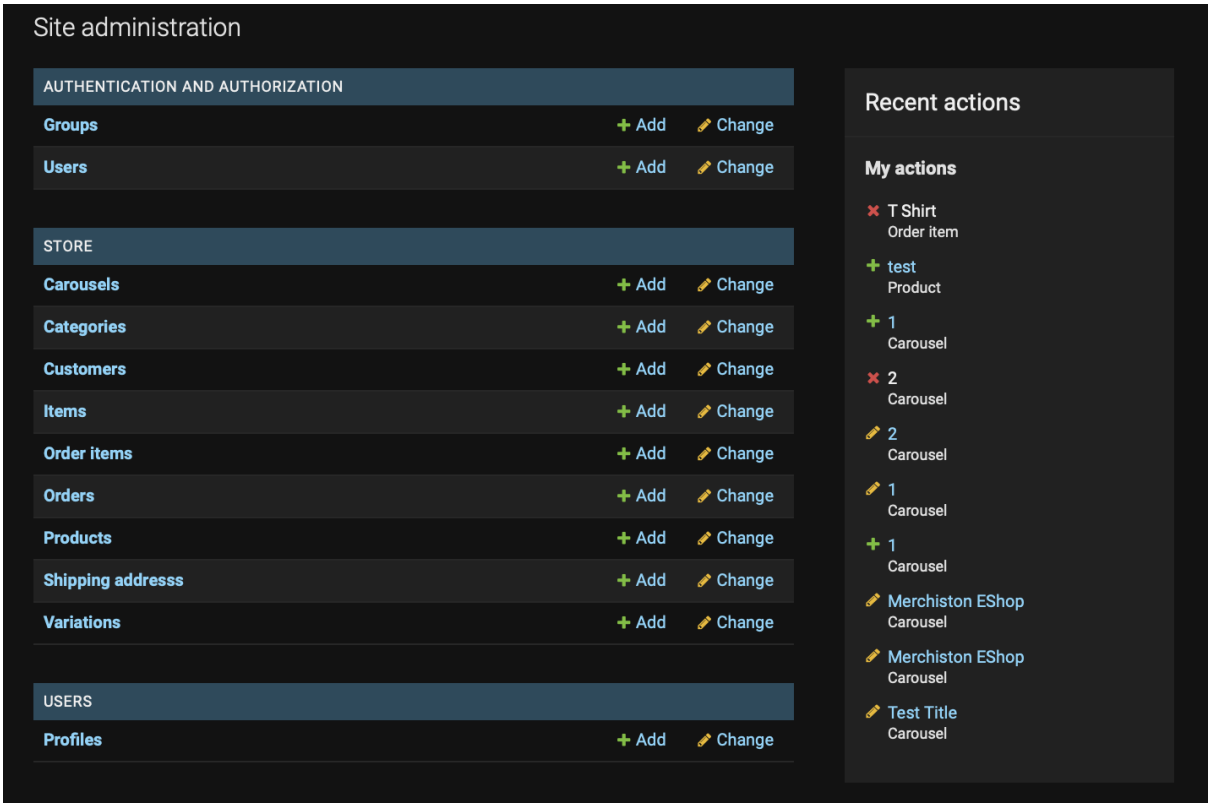
Since payment integration is out of A-Level bounds, I have not implemented it. This is the test message shown after the user has filled in shipping information and pressed ‘Make a payment’. After this, the data is sent to thedatabasee which can be viewed in the admin panel.



Testing Objectives for Admin

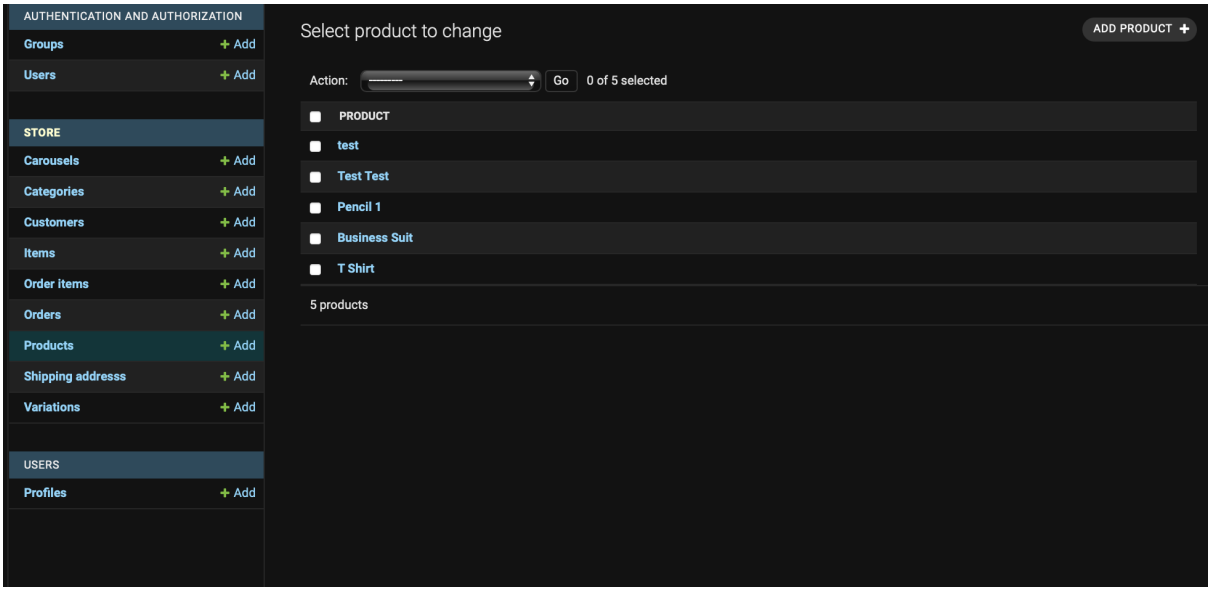
Testing Objective 3

Here is the admin page with all the required information to process orders and maintain the website.



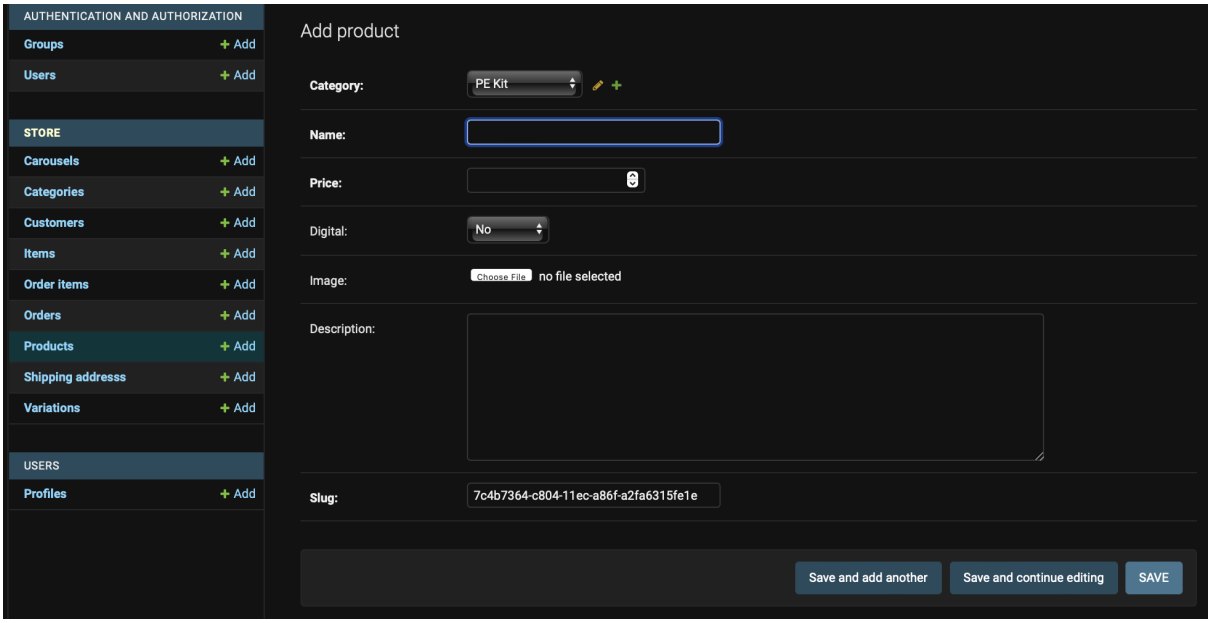
Testing Objective 3 a.

Here the admin can add/delete products to/from the page.

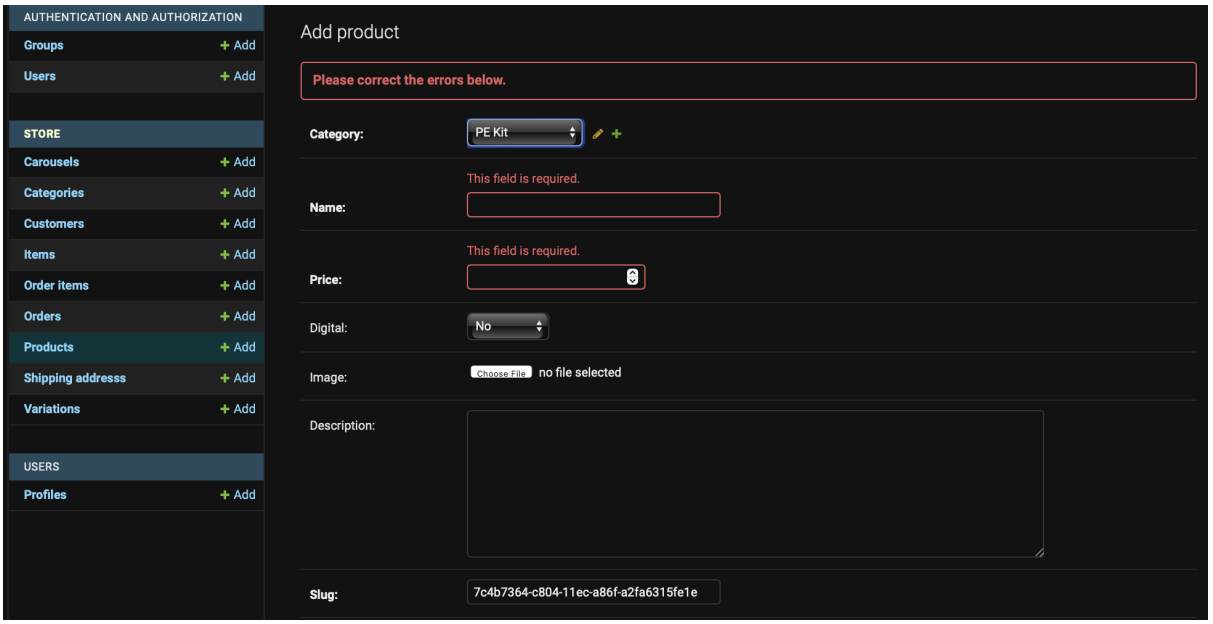


Testing Objective 3 a. i. and 3 b. i.

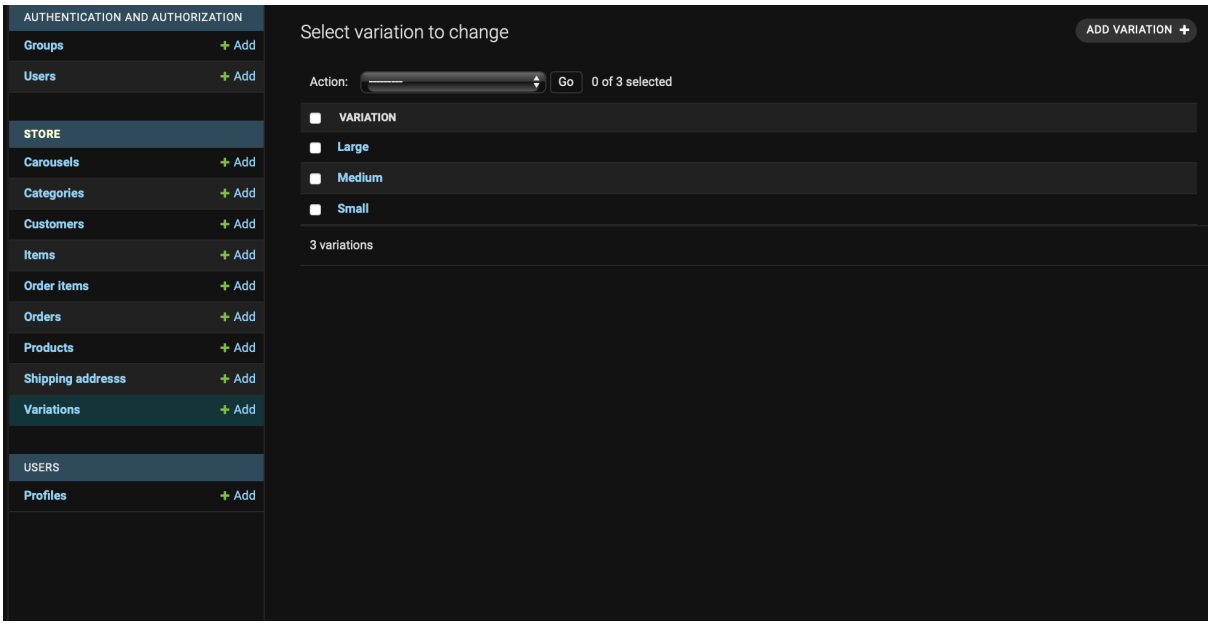
Here the admin can create products for the category pages. The creation of the product includes name, price, whether the product is digital or not (if digital, shipping address is not required), image and description. The slug field is the unique product identifier, it is generated automatically or it can be edited by the admin.



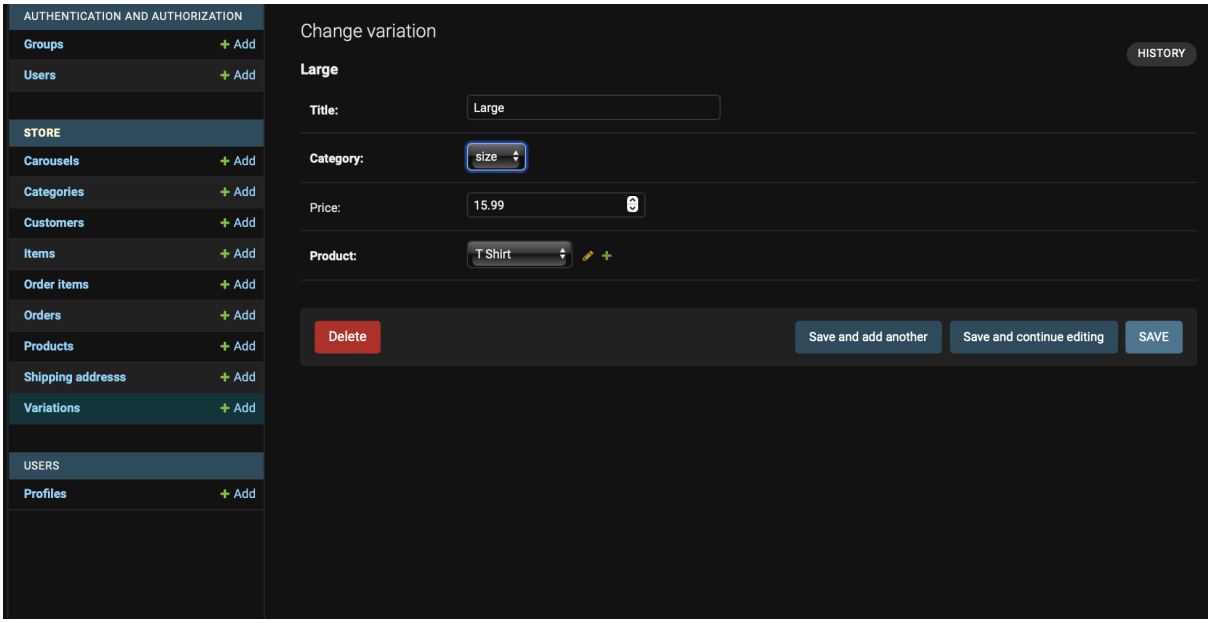
Here is the example when the admin tries to add a product with empty fields. The required fields are name and price.



Here is how I implemented the size option for products. They are variations that can be set for any of the existing products on the site.

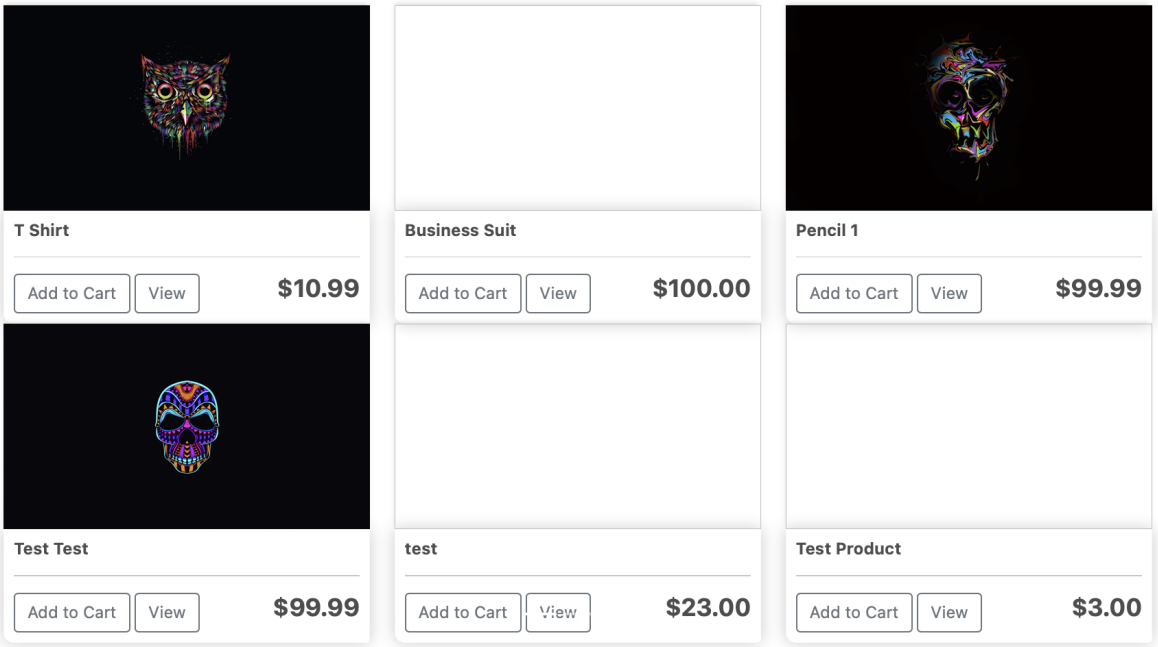


Size has its title, and category for the HTML to recognize whether it is a size or a colour and a product that this variation is set to. In this example, the variation ‘Large’ is set to the product ‘T-Shirt’.

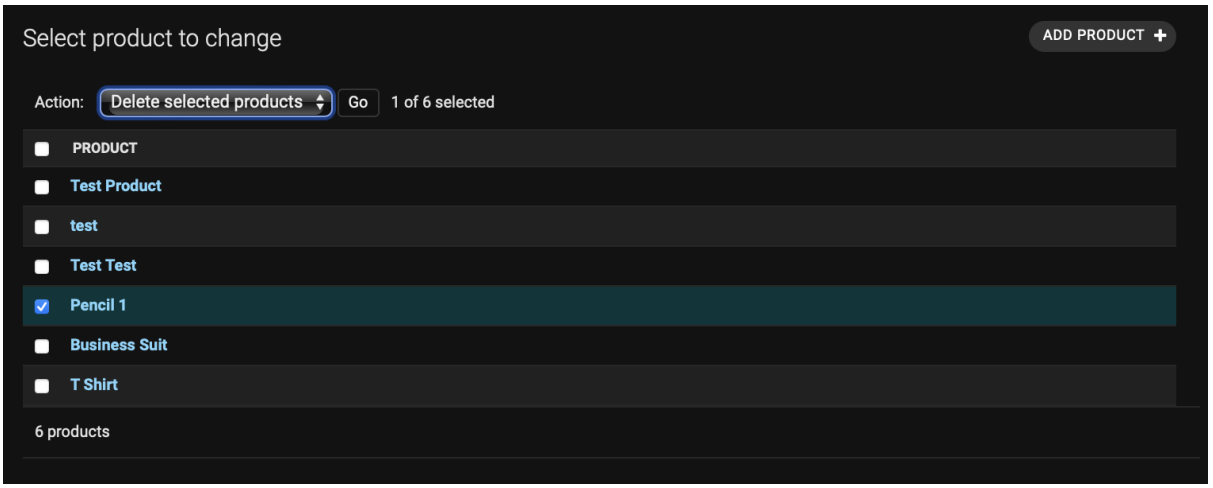


Testing Objective 3 b. i.

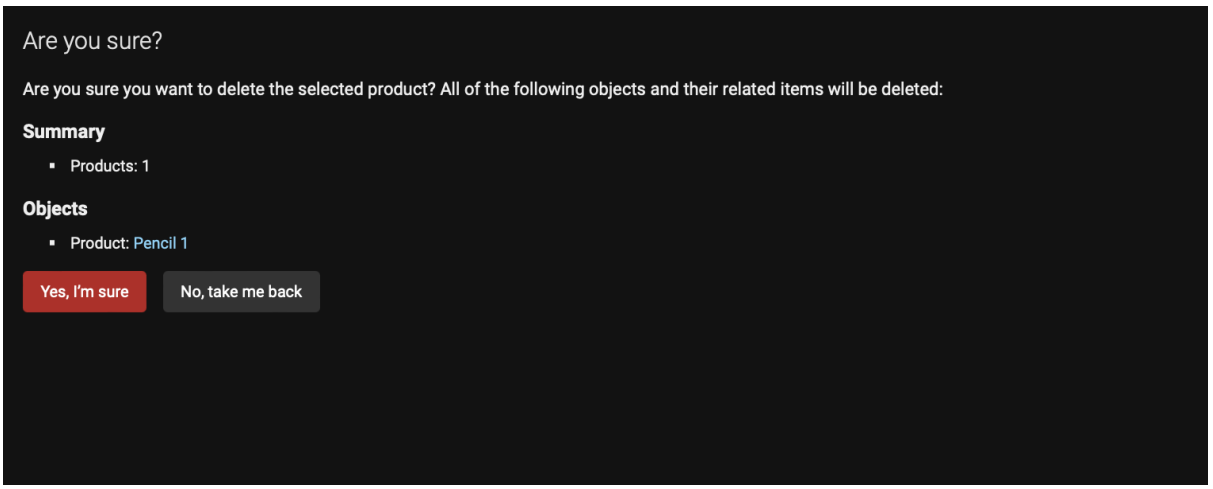
Here are existing products on the main page, lets's say that we don't need product 'Pencil 1' and instead we would like to add product 'Pen 2'.



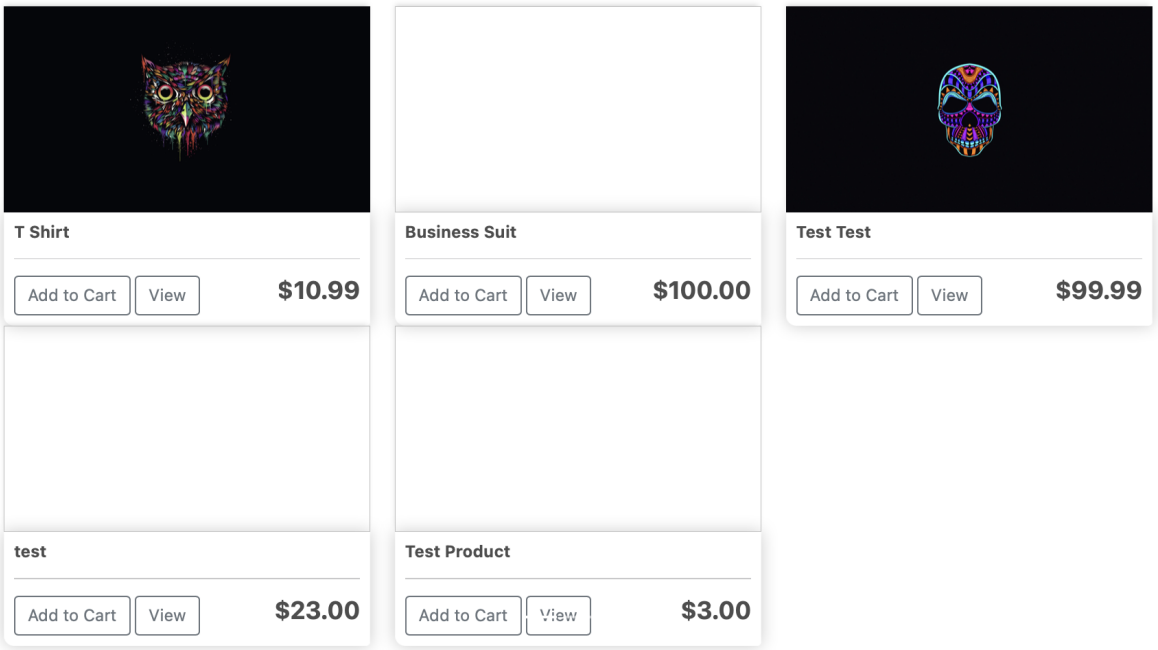
Here we can delete the item from the page after the admin pressed 'Go' button.



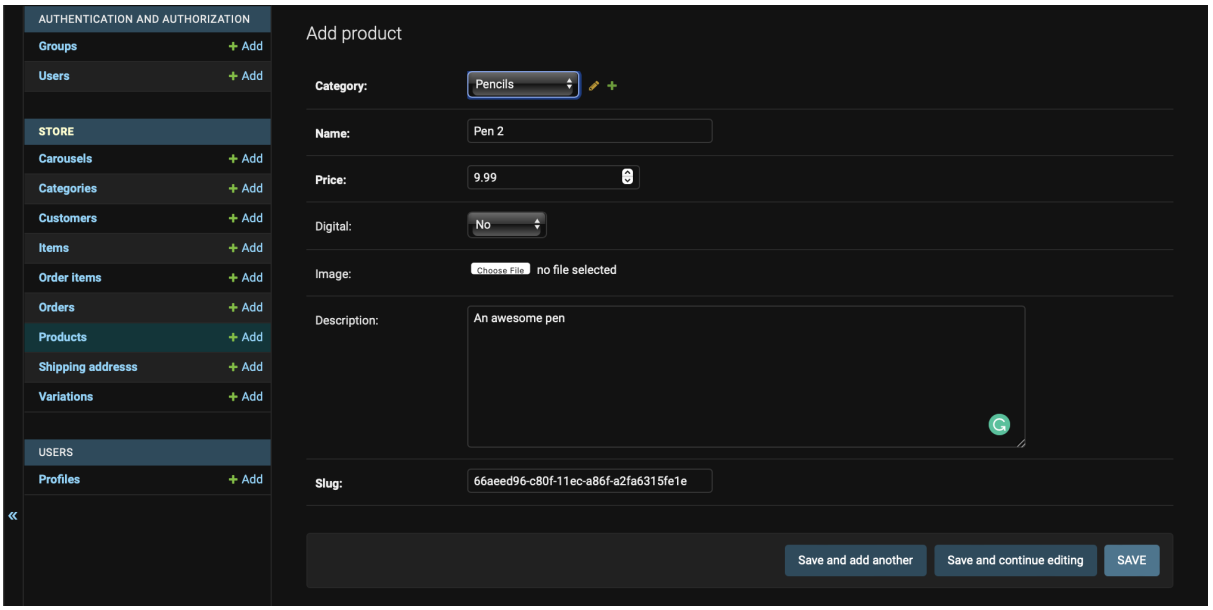
Verification if the admin wants to delete a list of following items. In this case, we want to delete ‘Pencil 1’.



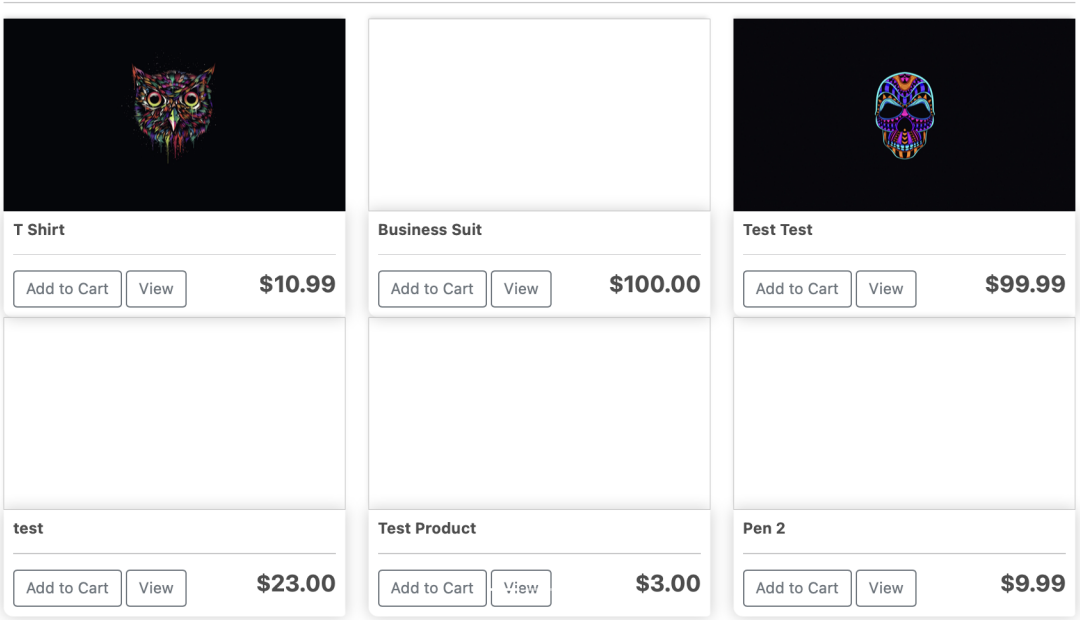
Here is the main page again and it can be seen that product ‘Pencil 1’ has been deleted from the site.



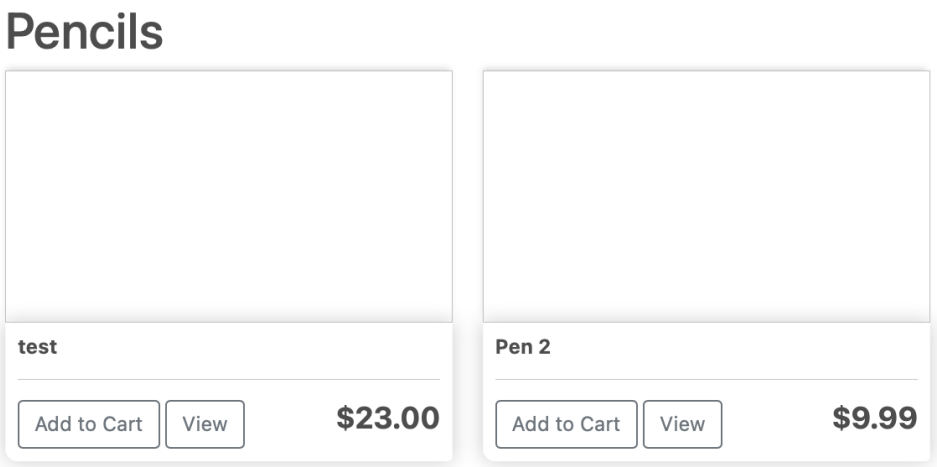
Here is an example of how the admin can add a product to the page.



Here the product has been added to the main page. Also, please note that the category of ‘Pen 2’ is ‘Pencils’, which means that if the users will go to that category, they will be able to see it.

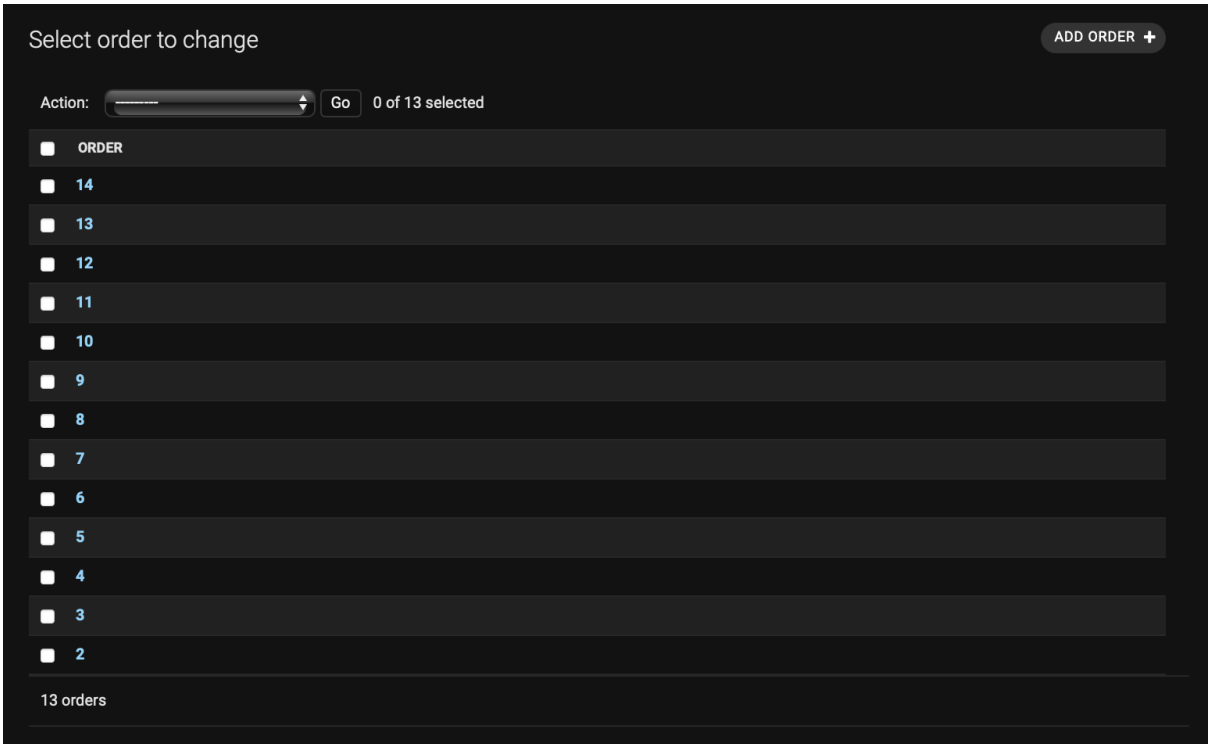


Here is the category ‘Pencils’ and it can be seen that the product has been added to this category.

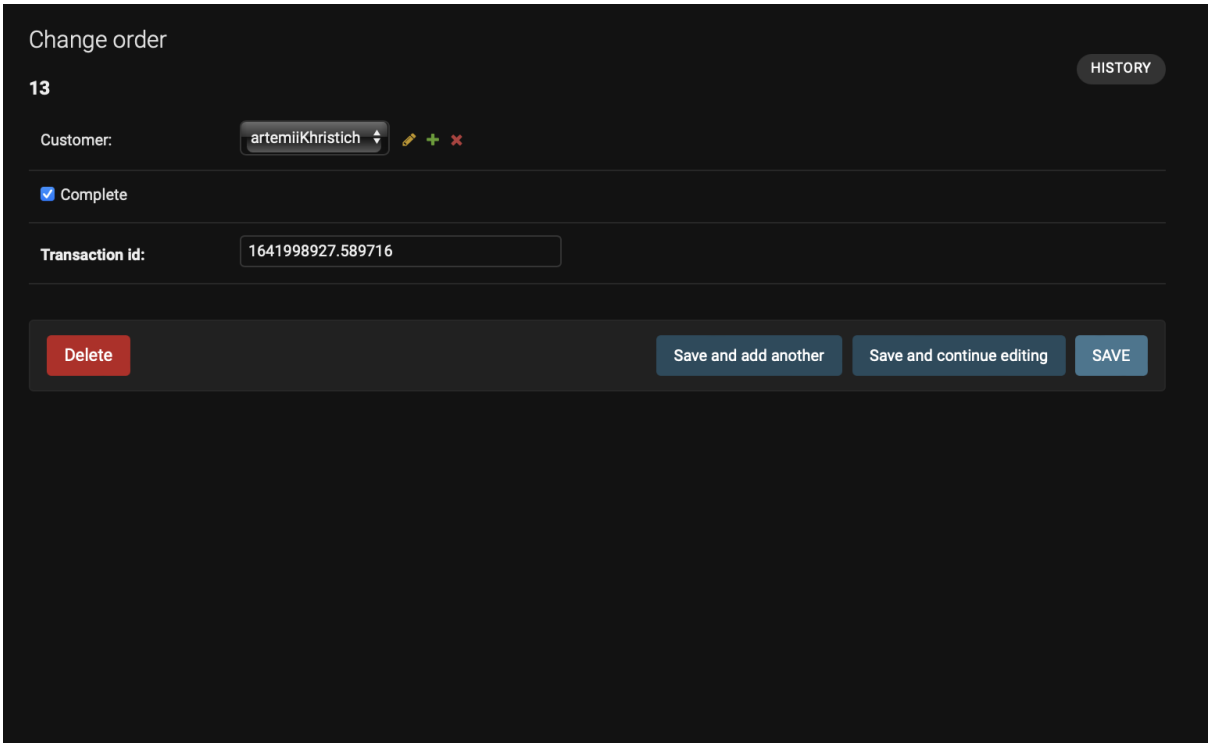


Testing Objective 3 c. i.

Here is the page for orders which are sorted in ascending order. It shows a history of all products that have been ordered.

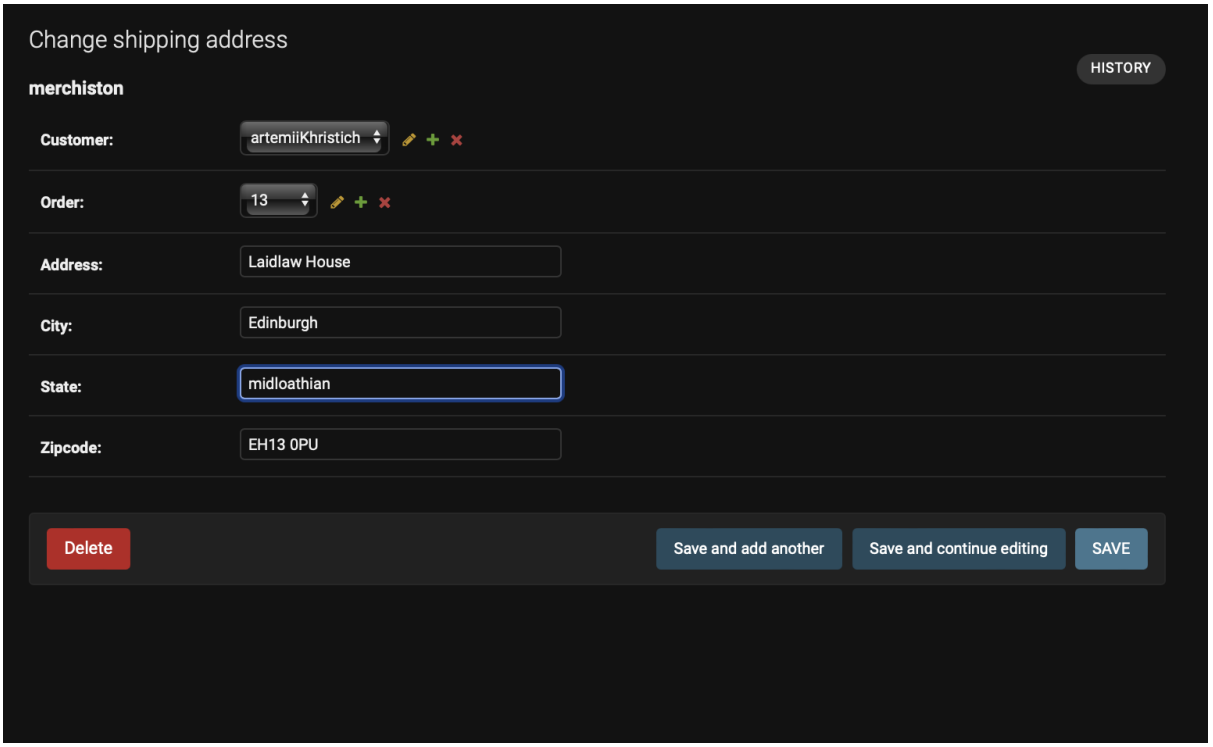


Here the admin can see the name of the customer who has been ordered an item, transaction it and a checkbox ‘complete’ to make it easier to determine which order has been complete.



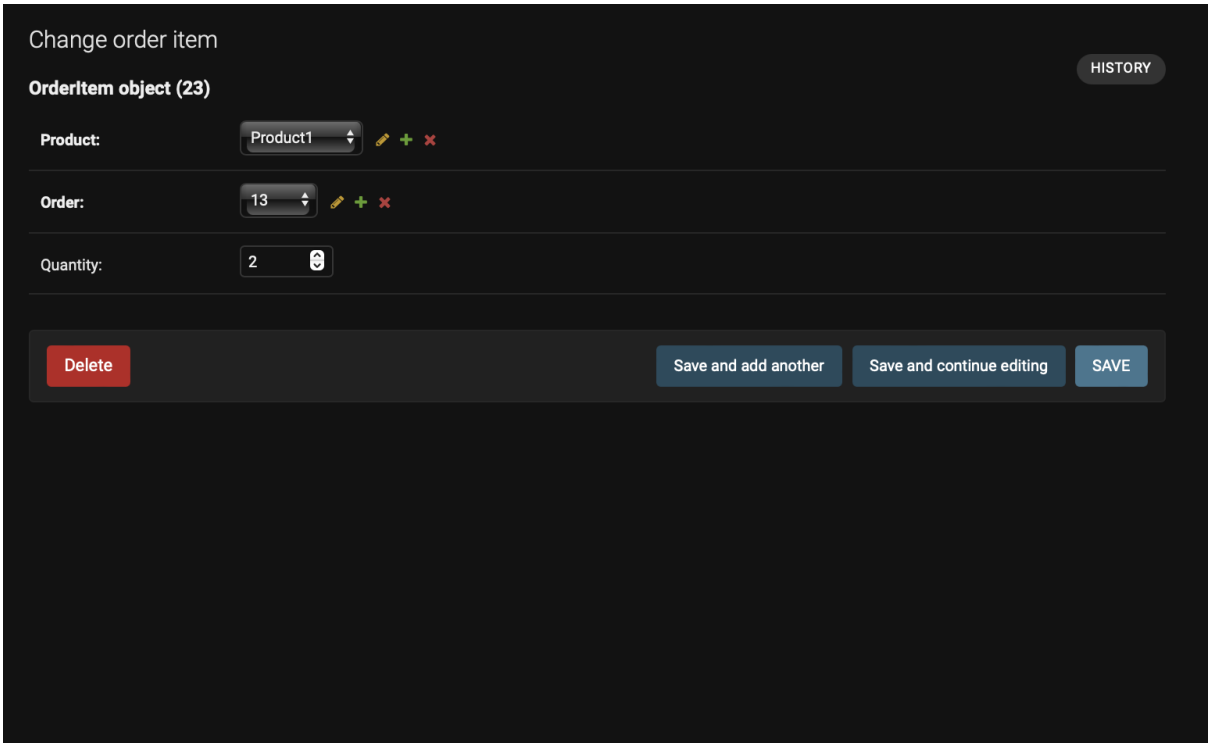
Testing Objective 3 c. i. 1.

Here is the page with shipping details for a corresponding order.



Testing Objective 3 c. i. 2

On this page, the admin can see information of a product for a corresponding order.



Testing Objective 3 d. i.

Due to the school policy, a database with returned items is irrelevant since it would add extra functionality that cannot be implemented under school regulations.

Evaluation

Overview

In my opinion I believe that my web application will significantly lower the amount of work that school shop managers need to put in. This is because previously, the school shop manager had to collect all the orders on paper and now this system will speed up the process of taking orders and collecting them. My program has fully met all my objectives that were set in the analysis. In addition, I have implemented some extra features beyond my objectives to make the web application more secure and more user friendly.

The feedback

I presented my project to a teacher (Mr Rowlands) for him to provide feedback on it. Mr Rowlands liked how ordered items are organised and that it is easy to see who ordered what. In addition, he said that this system is a good solution for the school shop. Mr Rowlands suggested that the current system is already good because it works well and has all the necessary features. But, for the school to take advantage of this system, at first I need to make sure that the system is secure and passed certain certifications. I have already implemented a CSRF token to protect against cookie attacks. Therefore, the security can be improved by making 2-factor authentication on a register page.

Also, I have shown my project to the school shop manager (Mrs Cordingley) to provide feedback. The feedback that I got from Mrs Cordingley was that she preferred my system as it simplifies a lot of work in comparison to the old system. In addition, she liked how orders are organised and it makes it quite easy to follow. One thing Mrs Cordingley suggested is to provide an ‘About’ page that would tell students what they should do in case they want to return an item for example. Also, another suggestion was that it would be nice if there would be a guide telling students how to measure chests for instance to determine their sizes. Therefore, the web application could be improved by implementing these things to make it easier in use and more efficient.

Reflection

In reflection, to make my program better, I would work more on a UI to make it perfect for the users. The UI for admin is already good and shows only necessary details for managing the school shop. Another feature that could be implemented is dynamically generated web pages from the admin panel. It means that the admin can add, delete, and edit pages with products. It means that the pages would be automatically generated from the admin panel and they will not be hardcoded.