Project7

Buff shop

Group Members: Sijia Ge, Xiaosong Wang, Zhiyong Wang

Final Statement

Basically, we have implemented almost all the features we have stated in Project5 and Project6.

In summary, We have designed a shopping website that mimics the CU book store. It can perform the most common functions of a real-world e-commerce website like adding to the cart and checking out the order, etc. In specific, the website contains the following functions:

1. Sign up a new user for the website, input the username,full name,password, email.phone and security question, answer, all fields need to be authenticated
2. Login to the website by username and password, and authentication,
3. Reset the password by entering the username and answering the security question.
4. Log out of the website
5. A decorator(wrap up a function rather than a class), required login in before all operations.
6. Modify the personal information in the user center. including email, phone number, and reset of the password.
7. View the history orders.
8. Log in to the backstage management system via the superuser account and manage the website information, managing all tables through the Django built-in API.
9. Search the products by the keyword, and look through the products based on the department.
10. Add items to the cart, edit the cart. Prompt up the number of items in the cart for each click ‘adding the item’

11. Load the shipping address of the last order automatically and support editing the shipping address.

What we didn't implement:

1. The database we did not convert to MySQL, since our group members don't install the MySQL, and the less convenient to view the tables than the Django built-in SQL database SQLite.
2. The total price and more information shown in the prompt window are not implemented due to the limited time.
3. We cannot implement more design patterns since we found that Django is not a OO framework, since there is no necessary place to instantiate an object from defined class except the ORM objects. The Django framework is a functional like framework, and main class is just ORM, we just need the function to get the request from the front end, and pull out the json-like information from the request and process the request or extract information from the database, then send out the response to the template and render to the user. So it's hard to find the scenario and easy approach to implement the pattern.

Final Class Diagram

Third-Party Code

The code for resetting the password, the personal center information and the layout for the main page and some models definition were done by ourselves. Other functional code and main business logic in this project is adapted by and inspired by a daily fresh project. Besides this, we use a lot Django built-in methods, API to operate the database, managing the website and some code for UI elements.Moreover, we using the JQuery and Javascript code to implement some user interaction.

A navigation bar for the department.https://www.w3schools.com/bootstrap/bootstrap_navbar.asp

The javascript tutorial from w3school.https://www.w3schools.com/js/default.asp

The JQuery tutorial from W3school https://www.w3schools.com/jquery/default.asp

Django official documentation:https://docs.djangoproject.com/zh-hans/4.0/

Django tutorial:https://pythondjango.cn/django/basic-tutorials

Django-tinymce, it's a rich text editor that can be defined in the models and offer the editor for the administers to edit the description of the products.https://django-tinymce.readthedocs.io/en/latest/

Statement on the OOAD process

1. The **MTV** (a variant of MVC, means Model, Template and Views) pattern built-in Django really brings the whole system loose coupling and high cohesion. It split different functions into different modules. The models in the MTV pattern, which are defined in the models.py files, handle the object-relational mapping between the system and the database.It offer the API for us to execute query and extract or store the data. The templates in the MTV pattern are made up of multiple HTML for different pages, which plays the role of view in MVC to display the UI to the users. They contain the surface static contents, including text and pictures of the web pages, and they also contain functions written in javascript, which can fulfill the basic interactions between the user and the system. The views in the MTV pattern mediate between the models and the templates, like the controller in MVC, passing front-end requests to the model and passing back-end information to the templates (or to the user).
2. The Django framework is not a really OO framework, which means that in the Django projects, we don't need to a lot of objects to interact with each other except the objects generated automatically by the ORM. The whole views in the MTV pattern can be just the functions to receive the request, process the quest via ORM or not, then render the HTTP request back to the clients. Even with the class-based views, the different method just in charge of processing the request sent to the backend via different methods like POST,GET,etc. The request is a json-like objects that are instantiated by the framework automatically, the view just pull out the significant information like session, cookies from the request objects. And all url assignments are done by the urls file which is a route. So it's more close to a function oriented framework. The Django

supports some OO concepts like inheritance while the inheritance for ORMs is just add more fields to the base tables, and inheritance for the class-based views is just for reusing some duplicated logic from the base class.

3. Django built-in ORM is a really easy to use. We defined the fields in the models, and the Django offer various different kinds of fields objects for the developers, and it support some frequently used function like log in authentication. And we can set up some property of the table by the metadata in the table class like whether the table is a proxy table or order by what fields and so on. Through the model we defined, we can easily register the model to the backstage system and decide what to display to the administers and what fields can be editable to the administers. Besides, the paginator is another powerful tool, we can instantia the paginator objects and pass the data into that, then we get a paginator which we manage the pagination through the method conveniently, it encapsulates a lot of logic for splitting the pages and abstract for the developers.
4. The method decorator in the views of Django makes us know another way of using decorator in programming, which is functional-based. Namely, in our system, we force the user to log in if they want to add items to or edit the shopping cart. To achieve this, a login function wrap the functions written in the view.py of the cart folder, which handles the add item request, edit request, visit cart request, and delete cart request. Before running those functions, the system will call the decorator function login first, judging if the user is in login status, and then it will call the function wrapped in the decorator. This novel usage of the decorator is eye-opening to us.