Herb Shopping

Object Design

Version 1.0

29.12.2017

Orhan Aykut Kardeş

Berke Kağan Yıldız

Batuhan Kurt

Mert Günay

Prepared for

SE301 Software Engineering



Table of Contents

[1. Introduction 1](#_Toc436772639)

[1.1. Object Design Trade-offs 1](#_Toc436772640)

[1.2. Interface Documentation Guidelines 2](#_Toc436772641)

[1.3. Definitions, Acronyms, and Abbreviations 2](#_Toc436772642)

[1.4. References 2](#_Toc436772643)

[2. Packages 2](#_Toc436772644)

[3. Class Interfaces 3](#_Toc436772645)

OBJECT DESIGN DOCUMENT

# Introduction

The Herb Shopping Project consists of modules such as product management, the distributor portal, the manager portal, the administration portal, and so on. Therefore, our content management module has to interact with the other modules. Inter-company relationships are an important topic to consider and we have increased our workload to facilitate other modules for the storage and retrieval of data from the database. By providing these interfaces we take control of the database and obey the rules of object-oriented programming. So, when you make a modification to a table in the DB, other modules will not get into trouble.

As the user enters data, the module will force the user to follow the predefined pattern designed for this purpose. This will surely prevent a faulty entry of the data into the database. Therefore, the consistency and integration of the data are provided with these patterns. Of course, this procedure increases the complexity of our source code.

## Object Design Trade-offs

**Reliability**

We tested all the steps of the software, we developed it with different inputs and tried to improve the experience of the users by fixing all the mistakes. We tested the system after every new feature we added. When we encountered any problem, we figured out where we made a mistake.

**Expandability**

While designing our system, we paid attention to extensibility. Thanks to its modular structure, our system is ready for new technologies that can be added later.

**Programmability**

In our project, we tried to create a simple, functional product using the Python Django Framework. The reason we use the framework is that simple operations such as logging in and signing up are readily available and accordingly we can spend more time on other important parts of the software.

**Maintainability**

After we finished our project, we gave importance to give the necessary technical support. When we sold our restaurant reservation system to a business in our daily life we needed to explain the operation of the system over a period of time, so we tried to provide all the features with an understandable interface.

**Compatibility**

Another reason why we use Python in our project is that we know that it will not hurt us about compatibility. The Django Framework is currently the most used framework on Python and has a lot of resources. Of course, new features can be added to the system later, so we chose to use a software language that is known to survive the problem and make the software long-lived.

**Adaptability**

The software we have developed is designed to adapt to changing conditions. Of course, there are no restrictions from the beginning, since later searches for restaurants and

new features can be added to the database section to make some additions and deletions, as

needed.

**Availability**

Availability is the ratio of time a system or component is functional to the total time it is required or expected to function. The system we developed must be accessible 24/7 and we designed it this way.

## Interface Documentation Guidelines

**Top Navigation**

At the top of the screen we have name label of our system, search box, category list and the buttons that allow users to login and register.

**Icons**

We focused primarily on functionality on our site, but we did not forget to skip some small details. We added the necessary icons like shopping cart icon, add product icon

**Buttons**

We used bootstrap to build the buttons on the site. We chose to use the ready button designs and measures found here because Django automatically supports it. In order to better understand the code, we have written, we've put some rules in naming it, you can find these rules in the following table.

## Definitions, Acronyms, and Abbreviations

Model: A schematic description of a system that accounts for its known or inferred properties

Template: A visual representation of a model which might

URL: Name and address of website.

HTTP: is a protocol for secure communication over a computer network which is widely used on the Internet.

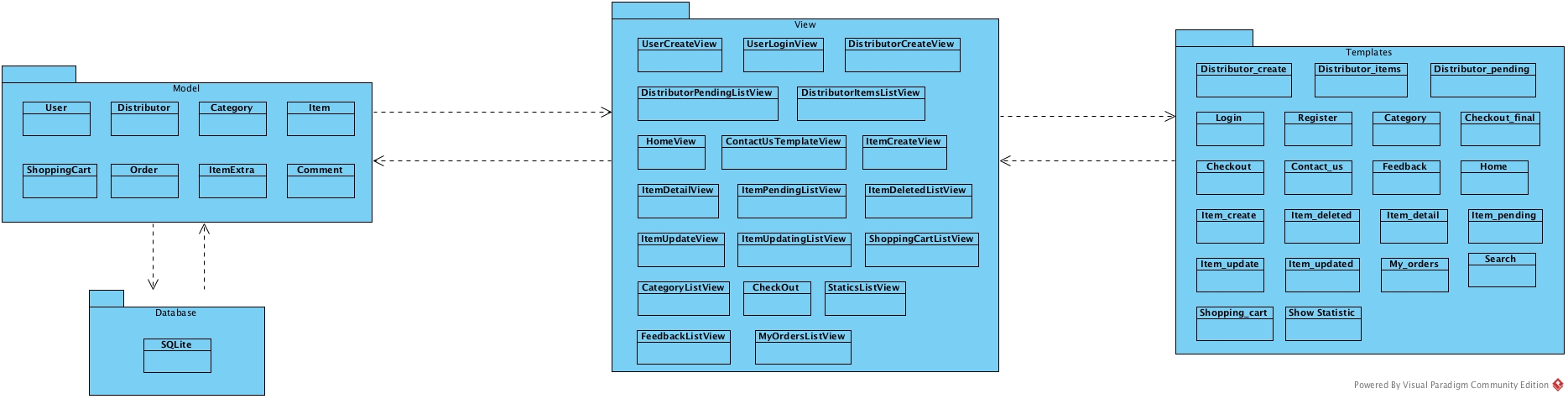
View: The controller translates interactions with the view into actions to be performed by the model.

DB: Database service

## References

1. https://www.herbaffair.com
2. https://www.iherb.com.tr
3. https://www.herbandspiceshop.com

# Packages



Note: The original image is on GitHub with .jpg extension in ODD materials.

Web framework is the framework for creating webpages. Each framework follows an MVC architecture pattern for building website to separate the data with business rules from the user interface. In Django web development, MVC is renamed as MTV framework. In this project, we used Django Framework to build the system through the MTV design pattern.

**M stands for “Mode**l,” the data access layer. This layer contains anything and everything about the data; how to access it, how to validate it and the relationships between the data.

**T stands for “Template,”** the presentation layer. This layer contains presentation-related decisions: how something should be displayed on Web page.

**V stands for “View,”** the business logic layer. This layer contains the logic that accesses the model and defers to the appropriate templates. It is bridge between models and templates.

# Class Interfaces

**Model Name:** User (Abstract User)

**Model Attributes/Explanation/Dependencies**

* ‘user\_type’: Indicates the role of the user in the system
* ‘adress’: Contains the address of the user
* ‘postcode’: Contains the postcode of the user
* ‘country’: Contains the country of the user

**Model Name:** User (Costumer)

**Model Attributes/Explanation/Dependencies**

We used the user fields that Django Framework provided itself by

from django.contrib.auth.models import AbstractUser

**Model Operations:**

* class UserCreateView(FormView): The customer registers on the system
* class UserLoginView(LoginView): The customer logins on the system
* class HomeView(ListView): The customer displays home and products
* def search(request): The customer can search for desired product
* def add\_to\_cart(request, id=None): The customer adds product to own shopping cart
* def delete\_from\_cart(request, id=None): The customer removes product to own shopping cart
* class ShoppingCartListView(LoginRequiredMixin, ListView): The customer shows the products in the customer's basket.
* def update\_quantity(request): The customer increase or decrease the amount of products in the own basket
* class CategoryListView(ListView): The customer displays products based on category
* class CheckOut(LoginRequiredMixin, ListView): The customer orders the products he adds to the basket
* class MyOrdersListView(LoginRequiredMixin, ListView): The customer displays orders that has already given
* def comment(request): The customer reports views and status about orders

**Model Name:** User(Distributor)

**Model Attributes/Explanation/Dependencies**

* ‘user’: The ID from the database for the user's role
* ‘brand’: Brand represented by the distributor
* ‘ceo\_name’: The name of the person who manages the brand in the system
* ‘average\_earnings’: Annual average revenue of the brand
* ‘company\_created’: Date of establishment of the company
* ‘is\_approveled’: The mark is not approved or approved

**Model Operations:**

* class UserLoginView(LoginView): The distributor logins on the system
* class UserCreateView(FormView): The distributor registers on the system as customer
* class DistributorCreateView(LoginRequiredMixin, CreateView): The distributor registers on the system with distributor register form
* class DistributorItemsListView(LoginRequiredMixin, ListView): The distributor displays products it sells
* class ItemCreateView(LoginRequiredMixin, CreateView): The distributor adds products to the system
* class ItemUpdateView(LoginRequiredMixin, UpdateView): The distributor updates products details it sells to the system
* def item\_delete(request, pk=None): The distributor deletes products that abandoned from selling

**Model Name:** User(Admin)

**Model Attributes/Explanation/Dependencies**

Admin has the username and password that was originally created by the system

**Model Operations:**

* class UserLoginView(LoginView): The admin logins on the system
* def confirm\_item(request): The admin accepts request to added product from distributor.
* def confirm\_u\_item(request): The admin accepts request to updated product from distributor.
* def confirm\_deleted\_item(request): The admin accepts request to deleted product from distributor.
* class FeedBacksListView(LoginRequiredMixin, ListView): The admin manages comments from the user

**Model Name:** User(Manager)

**Model Attributes/Explanation/Dependencies**

Manager has the username and password that was originally created by the system

**Model Operations:**

* class UserLoginView(LoginView): The manager logins on the system
* class DistributorPendingListView(LoginRequiredMixin, ListView): The manager displays distributor’s sends requests for sales product
* def confirm\_distributor(request): The manager accepts incoming distributor requests
* class StaticsListView(LoginRequiredMixin, ListView): The manager displays how much of the distributor’s merchandise has been sold

**Model Name:** Category

**Model Attributes/Explanation/Dependencies**

* ‘name’: The place where products are classified
* ‘slug’: The link with the products in the system.

**Model Operations:**

This model does not contain any distinctive operations at any controller.

**Model Name:** Item

**Model Attributes/Explanation/Dependencies**

* ‘owner’: It shows which distributor added the product
* ‘name’: Indicates the product's name
* ‘slug’: Indicates the url on the link
* ‘description’: Show product description
* ‘price’: Show product price
* ‘sold’: Indicates the sales status of the product
* ‘weight’: Show product weight
* ‘stock’: Show product stock
* ‘is\_approvaled’: Indicates the product's approved status
* ‘is\_updated’: Indicates the product's updated status
* ‘is\_deleted’: Indicates the product's deleted status
* ‘timestamp’: Indicates when the product was added
* ‘updated\_date’: Indicates when the product was updated
* ‘category’: Indicates when the product was added
* ‘related\_product’: Shows product category
* ‘image’: Shows the product's picture

**Model Operations:**

This model does not contain any distinctive operations at any controller.

**Model Name:** ItemExtra

**Model Attributes/Explanation/Dependencies**

* ‘itemEx’: the product is related to the other product
* ‘shopcart’: it shows which is the cart
* ‘quantity’: shows the amount of product in the basket

**Model Operations:**

This model does not contain any distinctive operations at any controller.

**Model Name:** Order

**Model Attributes/Explanation/Dependencies**

* ‘owner’: Specify the person who placed the order
* ‘shop\_cart’: Which cart indicates this is happening
* ‘orders\_item’: Show ordered products

**Model Operations:**

This model does not contain any distinctive operations at any controller.

**Model Name:** Comment

**Model Attributes/Explanation/Dependencies**

* ‘owner’: Indicates the person making the comment
* ‘comment’: Show the comment made on the system

**Model Operations:**

This model does not contain any distinctive operations at any controller.