Software Architecture Document

CoffeeBean Purchasing System

YorkWay Coffee Shop

Table of Content

System Overview

1. Context

The nominated system is used for coffee industry practicers to sell and manage their products online, which are home-branded coffee beans. It suits for businesses with different sizes, which is supposed to be an enterprise system with features of logical complexity, big data processing, scalability, concurrency.

2. Intended User

Role name in System	Description
Staff	Internal staff with different classifications, who manage product, category and placed orders
User	External customer, who browses product and makes purchases

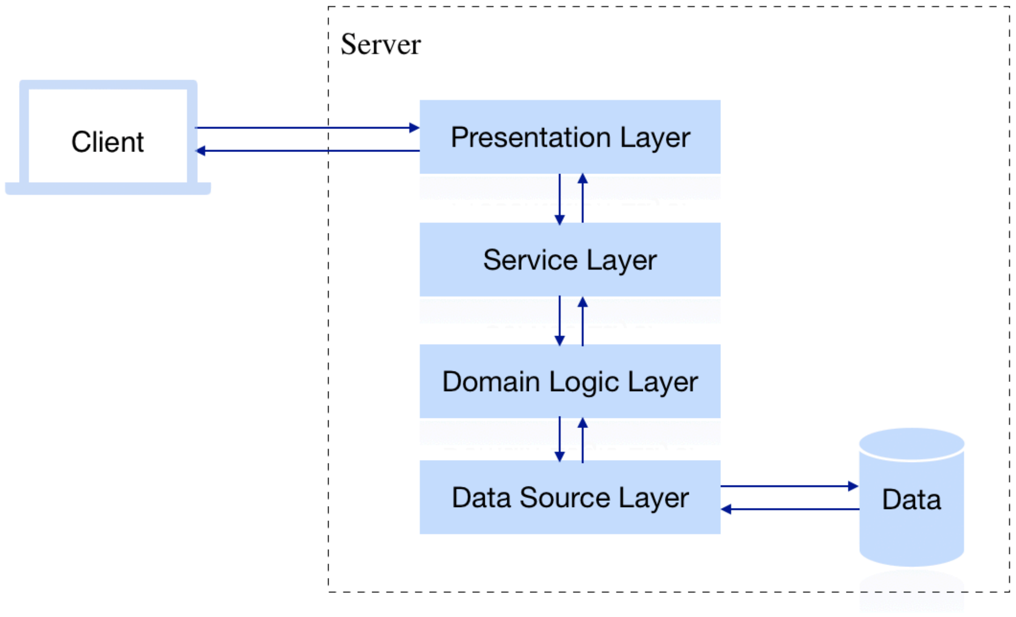
3. Feature

ID	Name	Description
Feature A	Category & Product Management	Staff is able to manage all category and product User is able to view and search product and category, manage associated shipping cart
Feature B	Order Management	Staff is able to view received orders, cancel order and modify order’s status

1. Hight-level Architecture

1.1. Layer Digram

Four layers have been defined in the intended system, and each layer will only interact with its directly related layers, by providing functional interfaces to upper layer and calling the lower layer's services.



1.2. Layers Responsibility

Layer	Description
Presentation	Facade of system, which parse and forward client's HTTP request also format and return server' response
Service	Presentation-oriented system service which is rearrangement and encapsulation of domain logic behaviour
Domain Logic	Business-related data and behaviour encapsulated in different domain model
Data Source	Mapping between domain model and database table in terms of create, retrieve, update, delete operation towards enterprise data

2. Detail Architecture

2.1. Presentation Layer (Full implementation in submission 3)

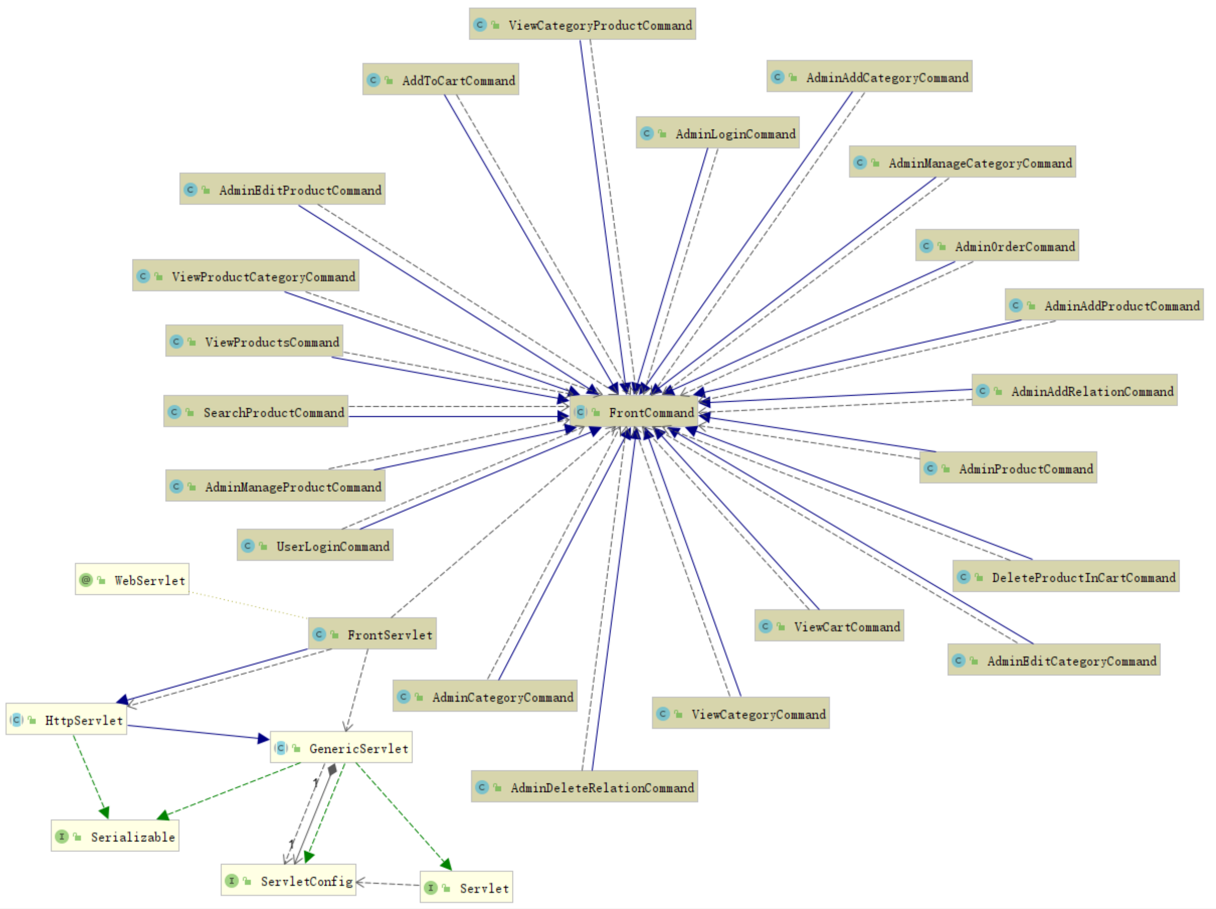
2.1.1. Responsibility

Presentation layer serves as a facade of system server to interact with end users. It parses and dispatches received HTTP requests to according services, also collects and displays the server-generated view to enquiring user.

2.1.2. Pattern Applied with Justification

Currently, Presentation Layer is constructed complying to Model-View-Controller structure, and particularly Front Controller pattern is applied in Controller module.

2.1.3. Layer Decomposition



2.1.4. Module Responsibility & Interface Specification

Module name	Responsibility
AddToCartCommand	Add a product to cart with its amount, category, and user_id of the customer
AdminAddCategoryComman	Parse parameters to create a new category
AdminAddProductCommand	Parse parameters to create a new product

AdminAddRelationComman	Parse parameters to add a product in a category
AdminCategoryCommand	Forward to category manage page
AdminDeleteRelationComma	Delete a product from a category
AdminEditCategoryComman	Parse parameters to modify the name of the category
AdminEditProductCommand	Parse parameters to modify the name, info, price, weight or inventory of a product.
AdminLoginCommand	Forward to admin home page.
AdminManageCategoryCom mand	Handle different category manage requests, parse parameters and redirect to other command.
AdminManageProductCom mand	Handle different product manage requests, parse parameters and redirect to other command.
AdminOrderCommand	Feature B
AdminProductCommand	Forward to product manage page
DeleteProductInCartComma	Parse parameters to delete a product from a shopping cart.
FrontCommand	The command extended by other commands.
FrontServlet	The only entry servlet of the system, handle all requests and parse to different command.
SearchProductCommand	Parse parameter to search a product.
UserLoginCommand	Forward to user home page.
ViewCartCommand	Forward to user cart page.
ViewCategoryProductComm and	View all products in a particular category.
ViewProductCategoryComm and	View all categories of a particular product.
ViewProductsCommand	Parse parameters and show the products list.

2.2. Service Layer

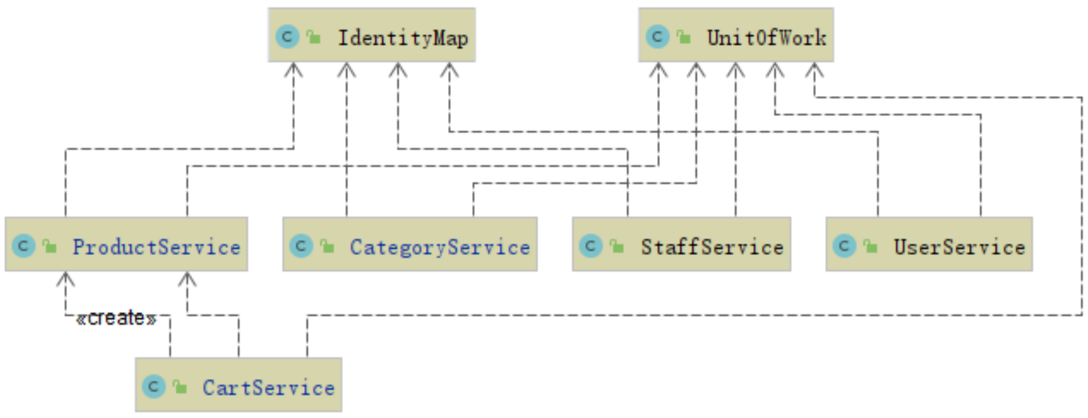
2.2.1. Responsibility

Service layer defines the business boundary of the system by providing a series of presentation-oriented services which are the rearrangement and encapsulation of domain logic.

2.2.2. Pattern Applied with Justification

Service layer is optional and rearrangement of domain logic, hence there is no design pattern applied here.

2.2.3. Service Layer Decomposition



2.2.4. Module Responsibility

Module name	Responsibility
Concrete Services	Provide rearranged domain logic methods forming individual service, which can be directly called by presentation layer

2.2.5. Interface Spécification

- **Service Provided**

Module name	Services	Interface
ProductService	Get a instance of ProductService	ProductService()
	Retrieve all existing product instances in the system	getAll() getAllAvailableProducts()
	Retrieve specific product instance by attributes id, name, category	findProductById(), findProductByName(), findProductByCategory()
	Insert a new product or delete or update an existing product	insertProduct(), deleteProduct(), updateProduct(),
	Add, delete product-category relationship, or destroy all existing relationships in the system	addRelation(), deleteRelation(), deleteAllRelations(), findRelation(),
CategoryService	Get an instance of CategoryService	CategoryService()
	Retrieve all existing category instances in the system	getAllCategories()
	Retrieve specific category instance by attributes id, name, product	findCategoryById(), findCategoryByName(), findCategoryByProduct()
	Insert a new category or delete or update an existing category	newCategory(), deleteCategory(), updateCategory()
CartService	Get an instance of CartService	CartService()

	Retrieve specific cart object by certain attributes	findCartByUserId()
	Add new product to cart, build a new cart or delete an existing cart	addToCart(), newCart(),deleteCart()
	Manipulate each record in user associated cart	insertCartDetail(), updateCartDetail(), deleteCartDetail(), findCartDetailByUserId()
StaffService	Feature B	
UserService	Feature B	
OrderService	Feature B	

- **Service Required**

Data Source level operations:

- Create a new record into database
- Retrieve an existing record in the database
- Update an existing record in the database
- Delete a exiting record in the database

2.3. Domain Logic Layer

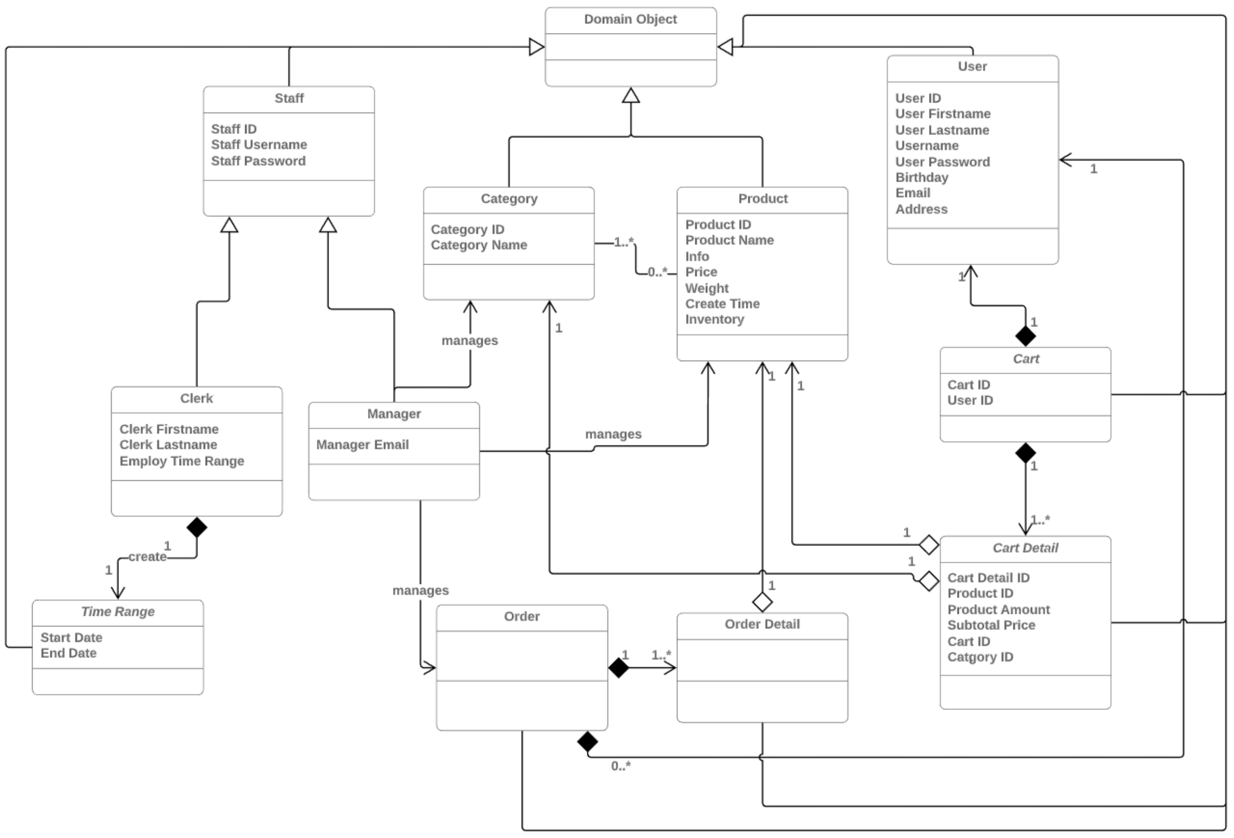
2.3.1. Responsibility

Domain logic layer defines the domain models by providing template classes within which encapsulate both object behaviour and data. Concrete domain objects are instantiated using domain model and used for data transmission

2.3.2. Pattern Applied with Justification

Pattern name	Domain Model
Description	An object model of the domain that incorporates both behaviour and data.
Choice	Only create objects related to a particular contract ID
Justification	1. **Extendability** It reduces the effort need for progress extension of this enterprise system. When we need to add new business logic, we only need add new field, method or class into current object-oriented structure 2. **Programming Intuition** It follows the object-oriented programming intuition and implementer's experience

2.3.3. Domain Logic Layer Decomposition



2.3.4. Module Responsibility

Module name	Responsibility
DomainObeject	Represent general domain object and contains only id field
Prodcut	Represent product in the system and contains id, information, price, weight, name, inventory fields
Category	Represent product in the system and contains id and name fields
Cart	Represent cart in the system and contains id and associated user fields
CartDetail	Represent individual item in cart in the system and contains id, associated product, product category, amount, total price and associated cart fields
Staff	Feature B
Manager	Feature B
Clerk	Feature B
TimeRange	Feature B
User	Feature B
Order	Feature B
OrderDetail	Feature B

2.3.5. Interface Specification

- **Service Provided**

Module name	Service	Interface
DomainObject	Instantiate a general DomainObject	DomainObejct()
	Get DomainObject instance ID	getId()
Product	Instantiate a Product object	Poduct()
	Get and set Product fields includes id, info, price, weight, name, inventory	get/setProductId(), get/setInfo(), get/setPrice(), get/setWeight(), get/setProductName(), get/setInventory()
	Lazy load the object when any of its field is accessed	load()
Category	Instantiate a Category object	Category()
	Get and set Category fields includes id and name	get/setCategoryId(), get/setCategoryName()
	Lazy load the object when any of its field is accessed	load()
Cart	Instantiate a Cart object	Cart()
	Get and set Product fields includes id and associated user	get/setCartId(), get/setUser()
	Lazy load the object when any of its field is accessed	load()
CartDetail	Instantiate a CartDetail object	CartDetail()
	Get and set CartDetail fields includes id, associated product, amount, Total price, associated cart and product category	get/setCartDetailId(), get/setProduct(), get/setAmount(), get/setTotalPrice(), get/setCart(), get/setCategory()
	Lazy load the object when any of its field is accessed	load()
Staff	Feature B	
Manager	Feature B	
Clerk	Feature B	
TimeRange	Feature B	
User	Feature B	
Order	Feature B	
OrderDetail	Feature B	

- **Service Required**

Instantiation by other layers for data transmission.

2.4. Data Source Layer

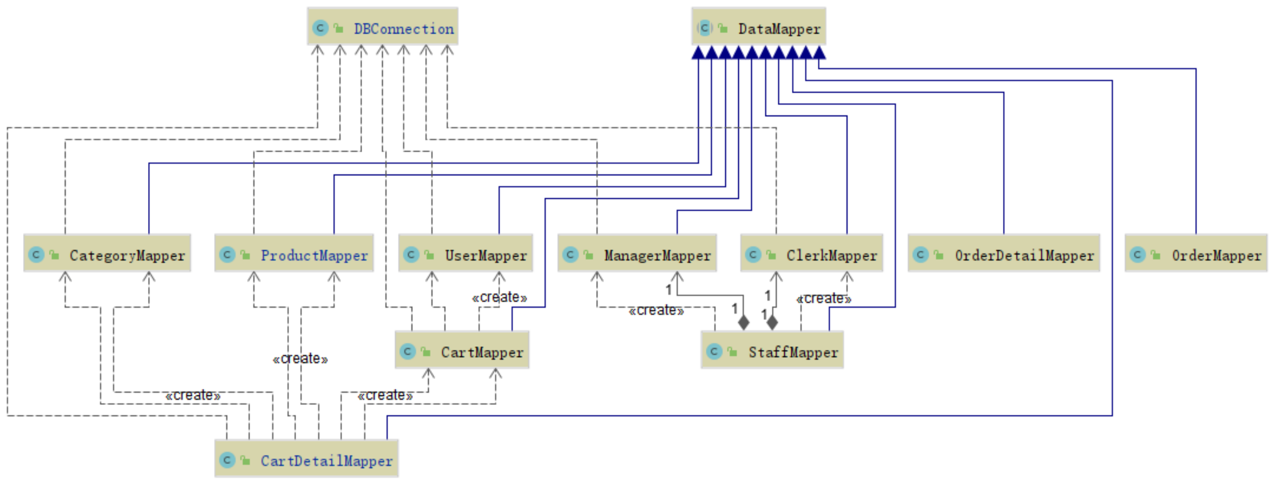
2.4.1. Responsibility

Data source layer deals with the real interactions with database, including receive request from service layer and order database CRUD transaction by constructing domain object and encapsulating necessary SQL statement.

2.4.2. Pattern Applied with Justification

Pattern name	Data Mapper
Design briefing	Encapsulate data manipulating methods in domain object associated Mappers
Choice	One Mapper per domain class
Justification	1. **Decoupling** It effectively decoupled the domain logic with low-level data accessing, means that services in Service Layer will only need to know the interfaces provided by Mappers in Data Source Layer, and changes in one of these layers will not necessarily effect another's implementation. 2. **Compatibility** It's highly compatible with chosen Domain Model pattern in Domain Logic Layer compared to other Data Source Layer patterns

2.4.3. Data Source Layer Decomposition



2.4.4. Module Responsibility

Module name	Responsibility
DataMapper	Abstract class defines the common structure of concrete Mappers and provides unified interfaces to service layer
Concrete Mappers	Inherit DataMapper and implement, or extend methods tailored to individual domain model

2.4.5. Interface Specification

- **Service Provided**

Module name	Service	Interface
DataMapper	Reflect to get concrete Mappers instance, by parsing argument class	getMapper()
	General interface required implementation in concrete Mappers	insert(), delete(), update()
ProductMapper	Retrieve product instance by certain attributes or return all product meet certain criteria from database product table	findProductById(), findProductByName(), findProductByCategory(), getAllProducts(), getAllAvailableProducts(),
	Write certain product-category relationship into database product_category table	addRelation() deleteRelation() deleteAllRelationsByProduct()
	Insert, delete or update certain records in database product table	insert(), delete(), update()
CategoryMapper	Retrieve category instance by certain attributes or return all category from database category table	findCategoryById(), findCategoryByName(), findCategoryByProduct(), getAllCategory(),
	Insert, delete or update certain records in database category table	insert(), delete(), update()
CartMapper	Retrieve cart instance by certain attributes from database cart table	findCartById(), indCartByUserId()
	Insert, delete or update certain records in database cart table	insert(), delete(), update()
CartDetailMapper	Retrieve cart detail instance by certain attributes from database cart_detail table	findCartDetailByCartId(), findProductInCart(), findCategoryByProduct(),
	Retrieve cart detail instance by certain attributes from database cart_datail table	insert(), delete(), update()
StaffMapper	Feature B	
ManagerMapper	Feature B	
ClerkMapper	Feature B	
UserMapper	Feature B	
OrderMapper,	Feature B	
OrderDetailMapper	Feature B	

- **Service Required**

Database level operation.

2.5. Other Applied Pattern Rationale

The realisation of this enterprise system involves application of a bunch of design pattern, each of them serves for a certain optimisations purpose and can be referred to code pieces.

2.5.1. Object-to-Relation Mapping Behavioural Optimisation Patterns

Pattern 1	Unit of Work
Justification	Unit of Word pattern decreases the time that mapper write new/delete/modified data into database by tracking data status and writing an "Unit of Data" in one commit, which significantly save machine's turnover on frequent writing behaviour
Reference code	CoffeeWeb/src/util/UnitOfWork

Pattern 2	Lazy Load (Ghost)
Justification	Lazy Load patter decreases the time that mapper load data from database by only loading a model instance when it's actually accessed, which save machine's turnover on memory space and database accessing. Particularly, Ghost implementation is light-weight and efficient, since it will load complete object when any of its field is accessed, compared other implementation, it shows better performance in meeting system goal and reducing data-accessing time
Reference code	CoffeeWeb/src/domain

Pattern 3	Identity Map
Justification	Identity Map pattern protects the data integrity from preventing reading in same records multiple times by maintaining maps for each domain class. The maps keep tracking which records are already loaded and are examined first when system issue a loading behaviour
Reference code	CoffeeWeb/src/IdentityMap

2.5.2. Object-to-Relation Mapping Structural Patterns

Pattern 1	Identity Field
Justification	It's a necessary strategy to realise Object-to-Relation Mapping by assigning one unique identifier for each domain object which is then used as primary key in according database table In this system design, each object is instantiated with a globally unique identifier which is used as indexing primary key in relational database tables
Reference code	CoffeeWeb/src/domain All domain classes

Pattern 2	Foreign Key Mapping

Justification	It solves the ORM one-to-many relationship mapping problems by persisting this domain logic relationship into foreign key schema in database. In this system, one Cart object is associated with many CartDetail objects, therefore in database, CartDetail will store Cart table’s primary key as a foreign key for referencing query
Reference code	CoffeeWeb/src/domain Cart-CartDetail

Pattern 3	Association Table Mapping
Justification	It solves the ORM many-to-many relationship mapping problems by persisting this domain logic relationship into an extra relational table in database. In this system, Product and Category share this many-to-many relationship, therefore in database, an extra table named product_category is made and stores both Product primary key and Category primary key for referencing query
Reference code	CoffeeWeb/src/domain Product-Category

Pattern 4	Embedded Value
Justification	The work time range is always requested at the same time with other information of a clerk. So we use embedded value here to reduce requests.
Reference code	CoffeeWeb/src/domain Clerk-TimeRange

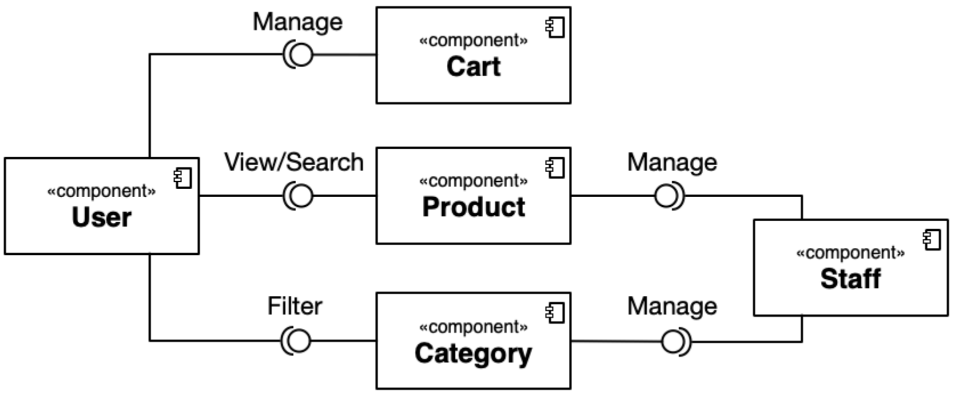
Pattern 5	Concrete Table Inheritance
Justification	It solves the ORM extend relationship mapping problem. We use concrete table here to achieve an easy-to-understand table design and reduce joins between tables. It also prevent to store all data in a single table.
Reference code	CoffeeWeb/src/domain Staff-Clerk & Manager

1. Class Diagram

Feature A's complete domain class diagram is linked to https://www.dropbox.com/s/p7g18gnu98zh9yo/ClassDiagram.png?dl=0

2. Component Diagram

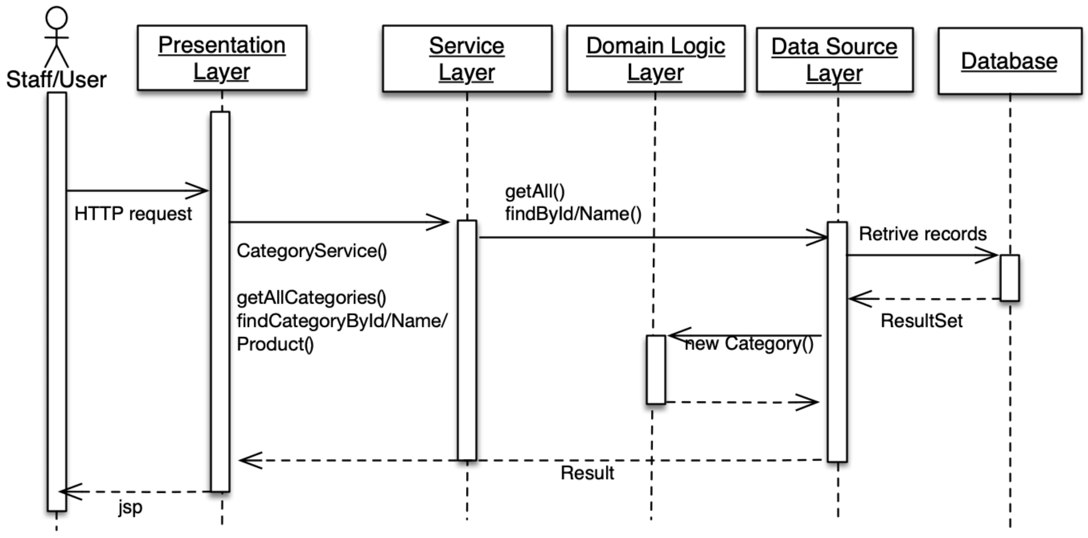
Feature A mainly refers to interactions between five components, they are system product, category, system staff, shopping user and its associated shopping cart



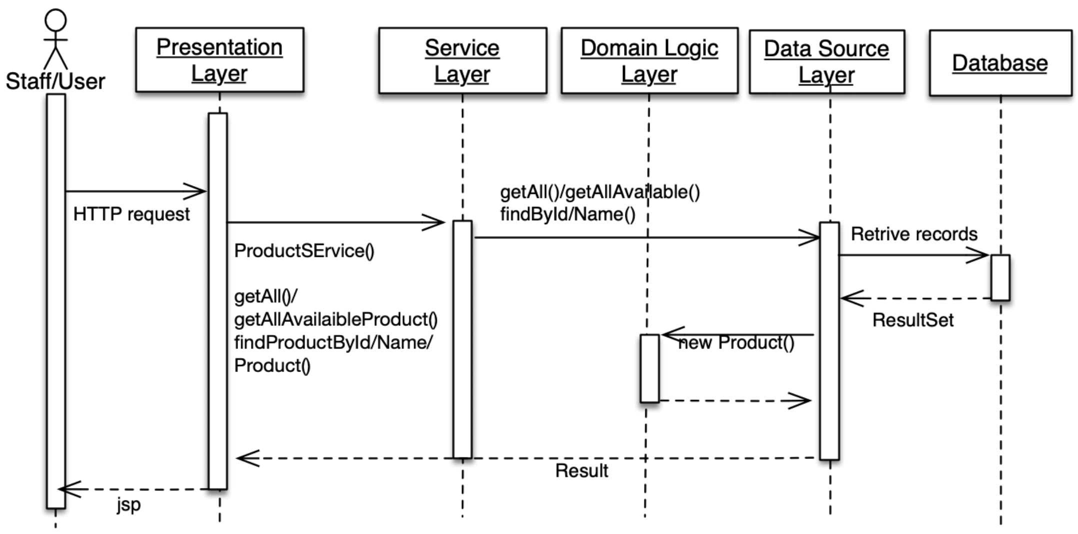
3. Interaction Diagram

Interactions in feature A mainly includes 10 scenarios depicted as below

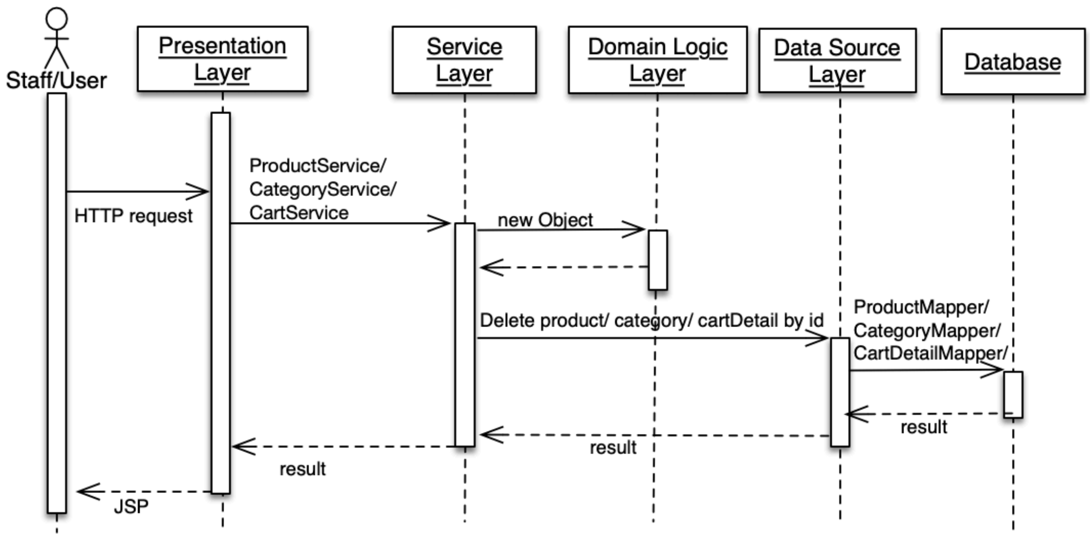
3.1. Staff/User view categories



3.2.Staff/User view products

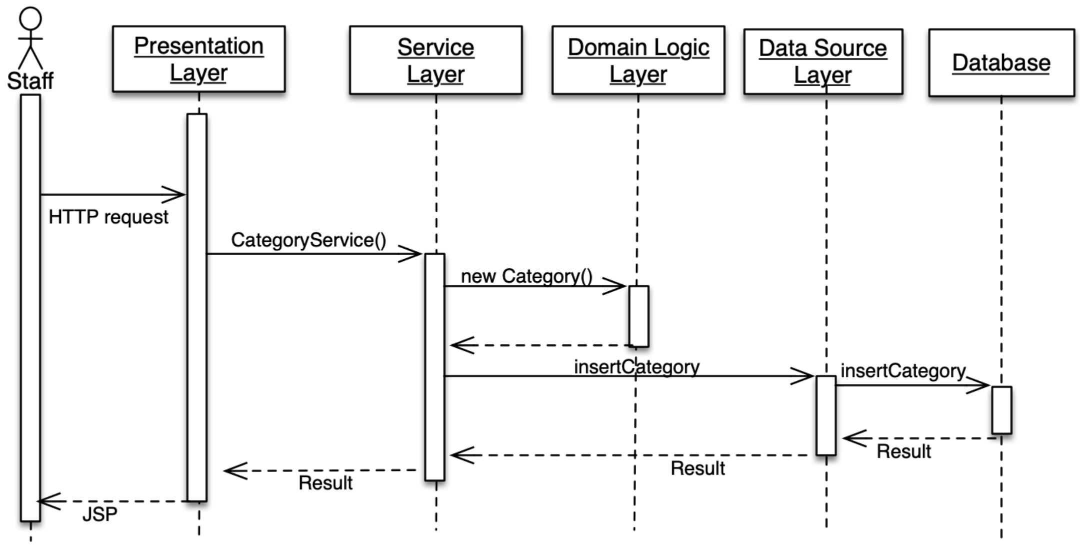


3.3.Staff Delete Product/Category; User delete cartDetail

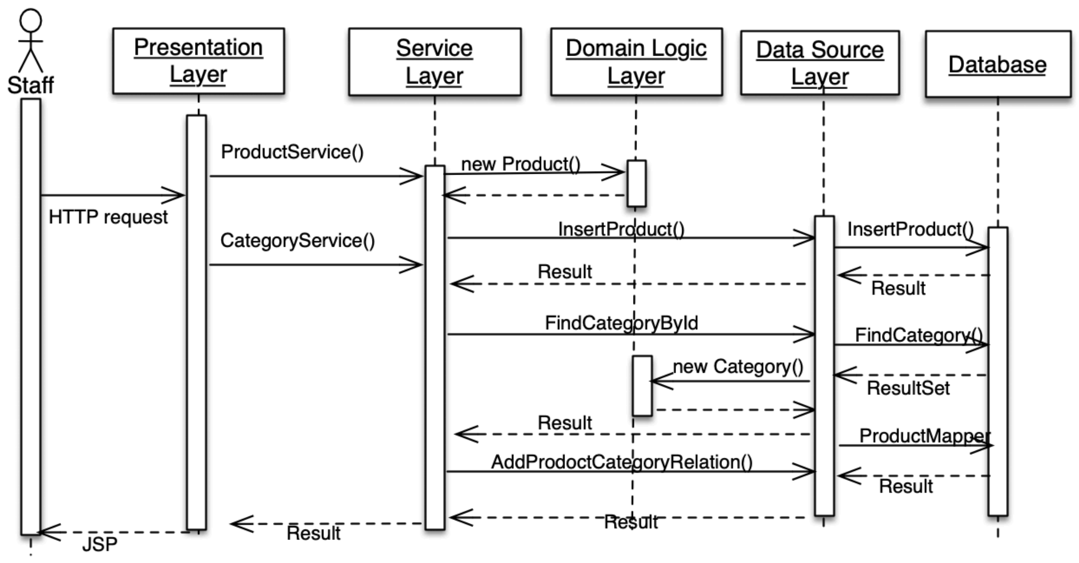


Staff/User
Presentation Layer
Service Layer
Domain Logic Layer
Data Source Layer
Database
HTTP request
ProductService/ CategoryService/ CartService
new Object
Delete product/ category/ cartDetail by id
ProductMapper/ CategoryMapper/ CartDetailMapper/
result
result
result
JSP

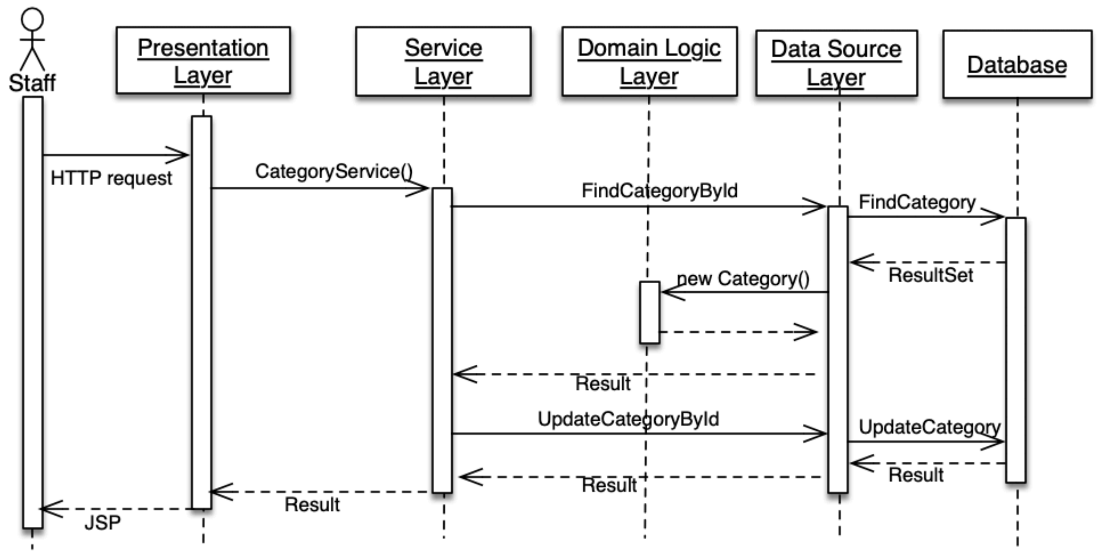
3.4.Staff add new Category



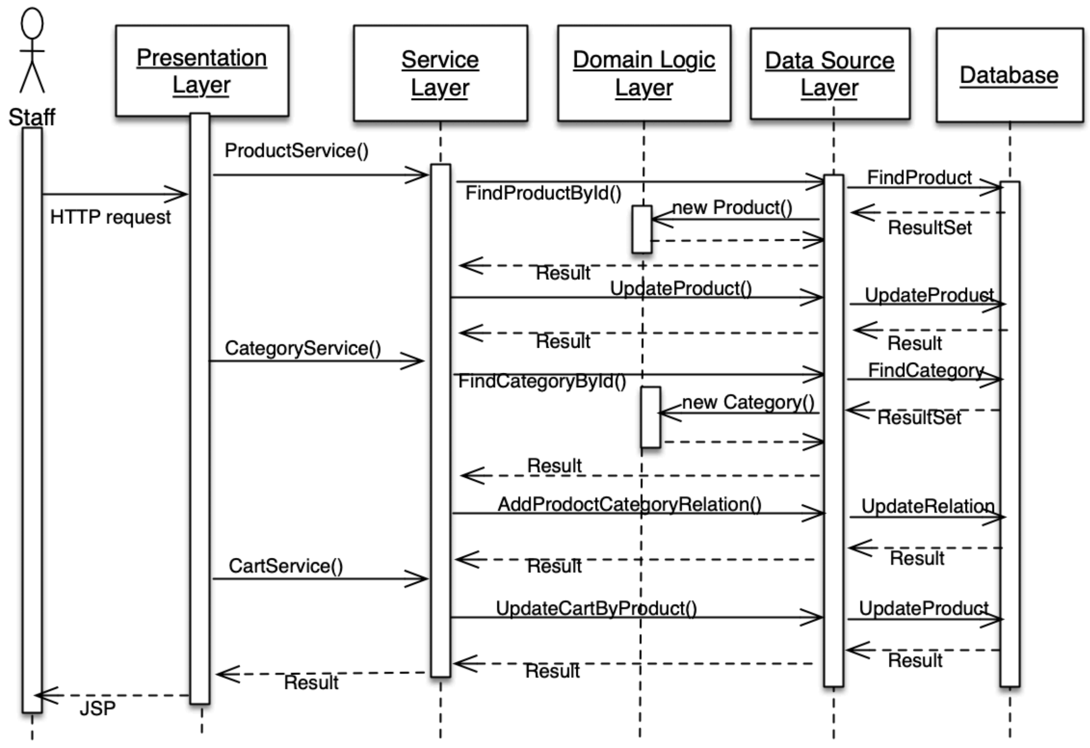
3.5.Staff add new Product



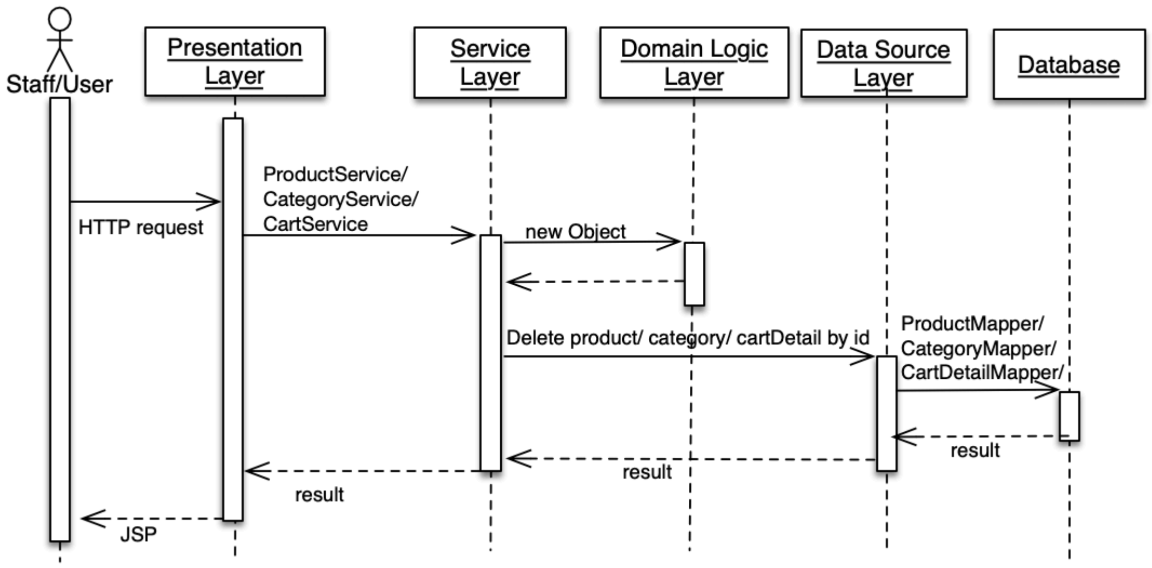
3.6.Staff edit Category



3.7.Staff edit Product

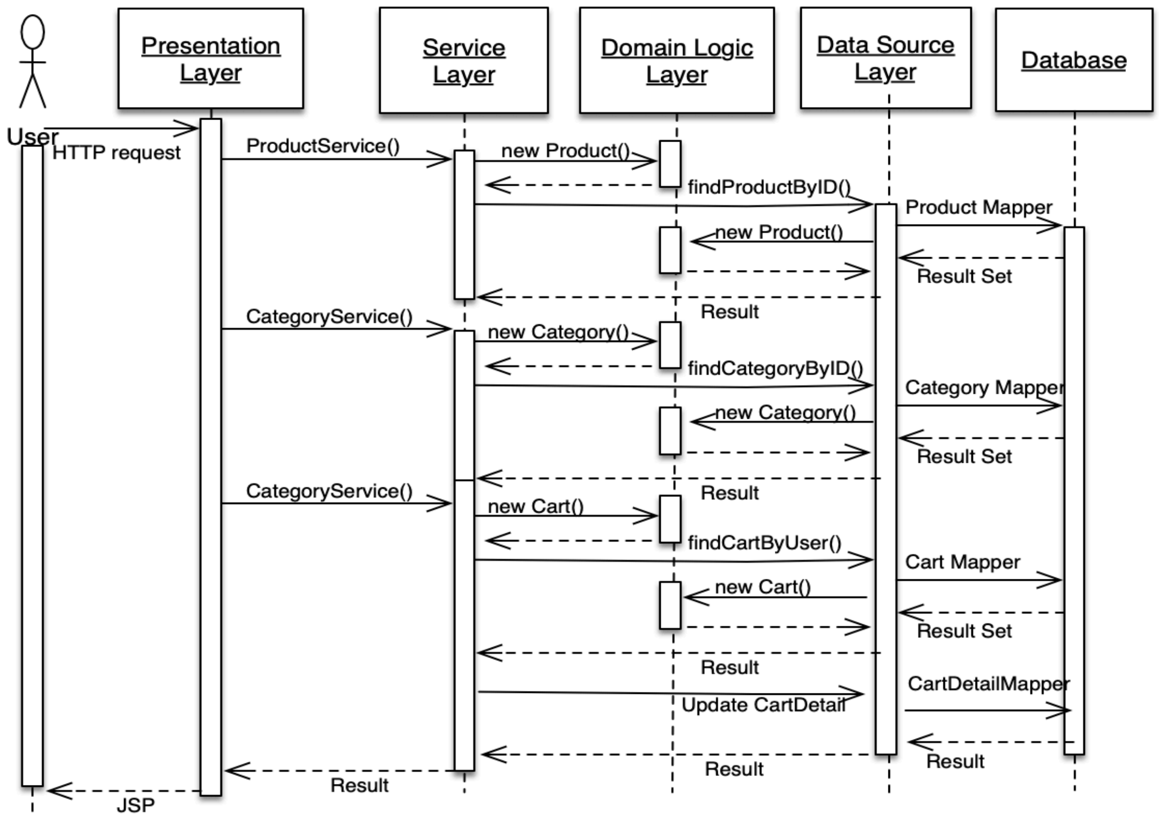


3.8.Staff delete Product/Category; User delete cartDetail

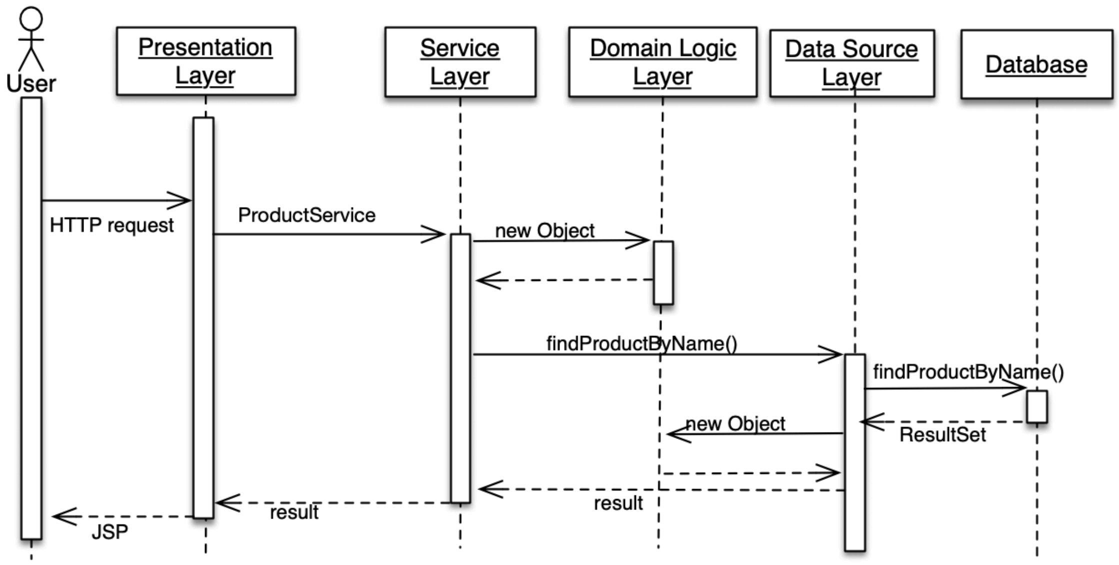


Staff/User
Presentation Layer
Service Layer
Domain Logic Layer
Data Source Layer
Database
HTTP request
ProductService/ CategoryService/ CartService
new Object
Delete product/ category/ cartDetail by id
ProductMapper/ CategoryMapper/ CartDetailMapper/
result
result
result
JSP

3.9.User add product to cart



3.10.User search product



Appendix

Code repository: https://github.com/DennyLee515/SWEN90007-SDA-2019-Project

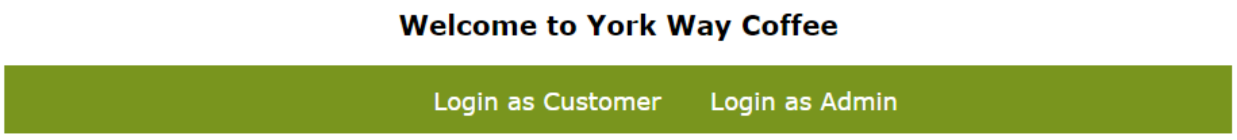
Heroku Link: https://yorkwaycoffee.herokuapp.com/index.jsp

When we are testing the system on Heroku, remote database down time is faced. It will recover in a few minutes or hours.

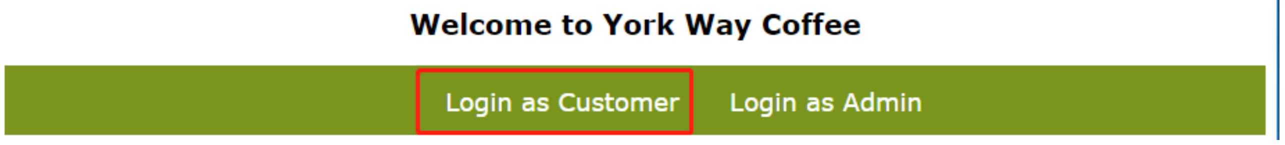
Besides, as we are using Heroku for our deployment, the database credentials are not permanent. Thus it is possible when you are testing the system, the remote database cannot be connected. If this happens, please contact dongming@student.unimelb.edu.au or zhuxiny1@student.unimelb.edu.au and we will check the attributes and redeploy the system.

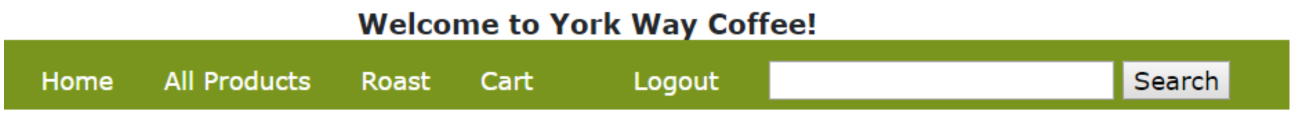
1. Feature A Test Scenarios

- **Website Home Page**

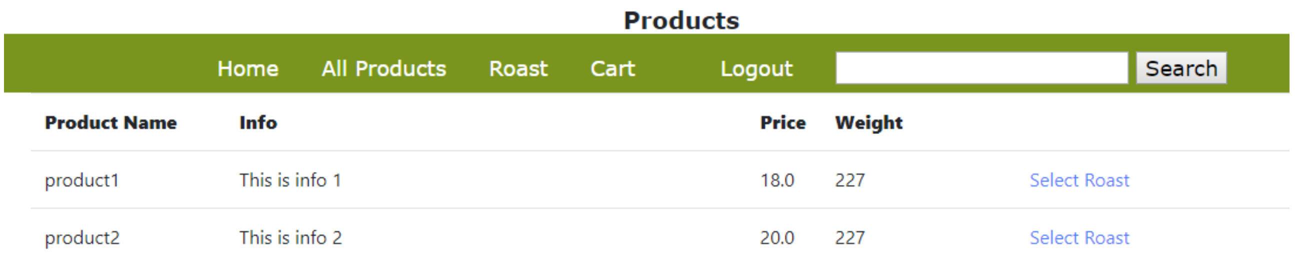


- **Customer Scenarios**
  1. Login / Logout



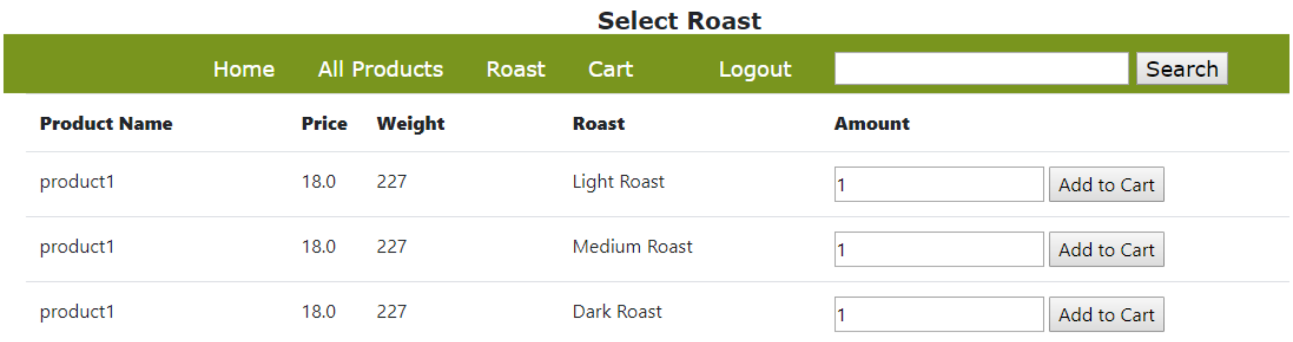


  2. View Product
     2.1. View all coffee by clicking the "All products"
     2.2. View all roast level of a particular coffee by clicking the "Select Roast"



Product Name	Info	Price	Weight	
product1	This is info 1	18.0	227	Select Roast
product2	This is info 2	20.0	227	Select Roast

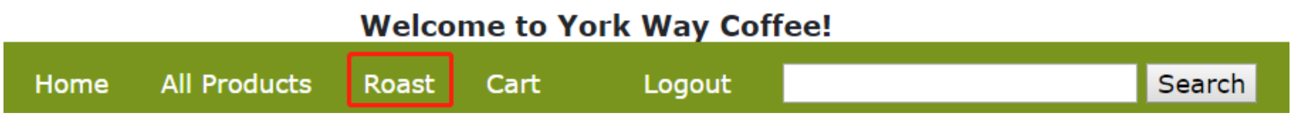
     2.3. Add a product to cart by inputting an amount and clicking the "Add to Cart", if success, the website will redirect to "My Cart" page



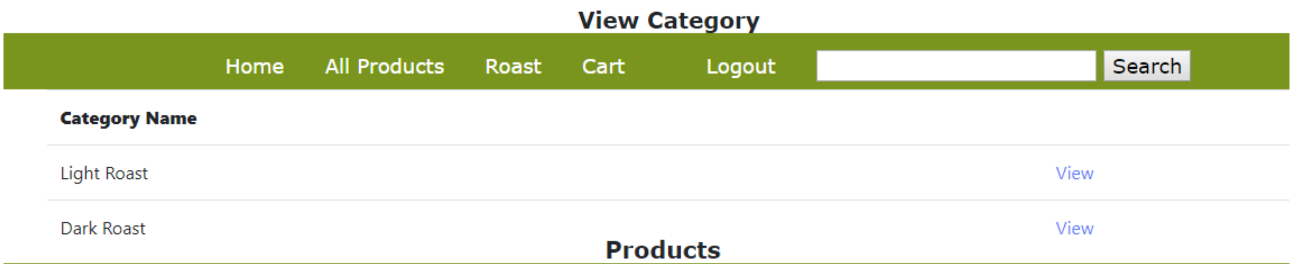
Select Roast

Product Name	Price	Weight	Roast	Amount
product1	18.0	227	Light Roast	1
product1	18.0	227	Medium Roast	1
product1	18.0	227	Dark Roast	1

3. View all Roast categories
   3.1. View all categories by clicking the "Roast"
   3.2. View all products in a roast category by click the "View"
   3.3. Add a product to cart by inputting an amount and clicking the "Add to Cart", if success, the website will redirect to "My Cart" page.

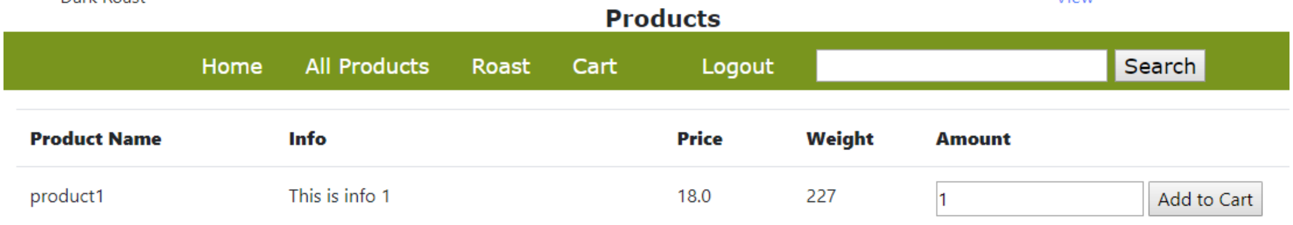


Welcome to York Way Coffee!



View Category

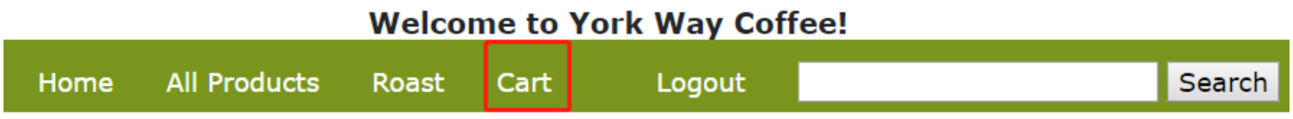
Category Name	
Light Roast	View
Dark Roast	View



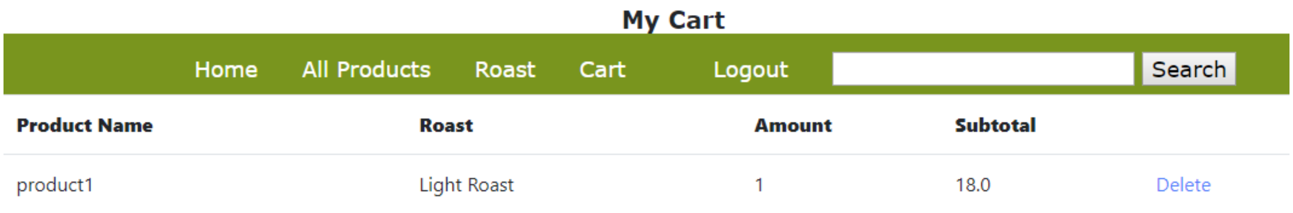
Products

Product Name	Info	Price	Weight	Amount
product1	This is info 1	18.0	227	1

4. View cart
   4.1. View all the products in the shopping cart in "My Cart" page



Welcome to York Way Coffee!



My Cart

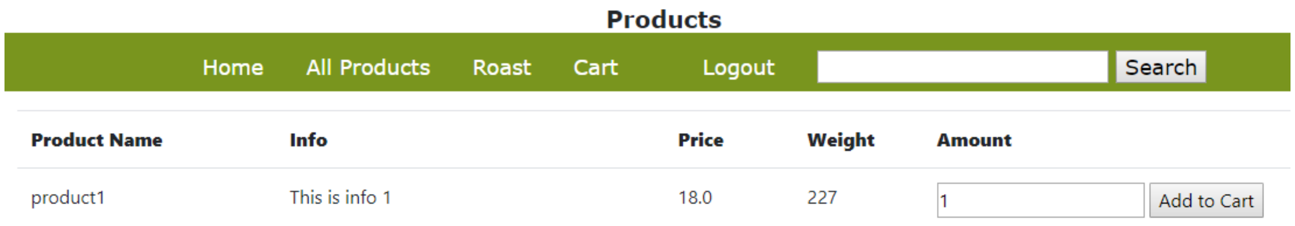
Product Name	Roast	Amount	Subtotal	
product1	Light Roast	1	18.0	Delete

5. Search product
   5.1. Search a product by inputting a name in the search textbox such as "product1" and clicking the "search

5.2. View all roast level of the particular coffee by clicking the "Select Roast"

5.3. Add a product to cart by inputting an amount and clicking the "Add to Cart", if success, the website will redirect to "My Cart" page.

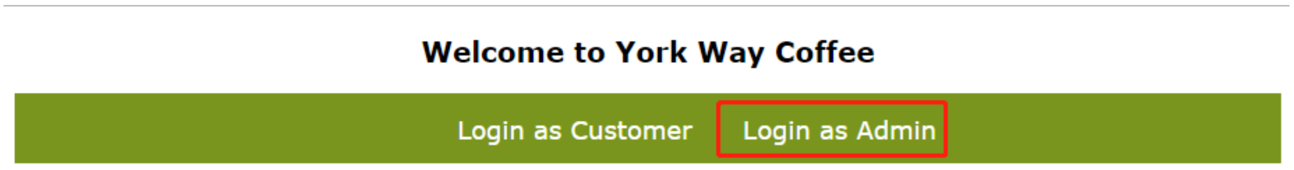
5.4. View all the products in the shopping cart in "My Cart" page

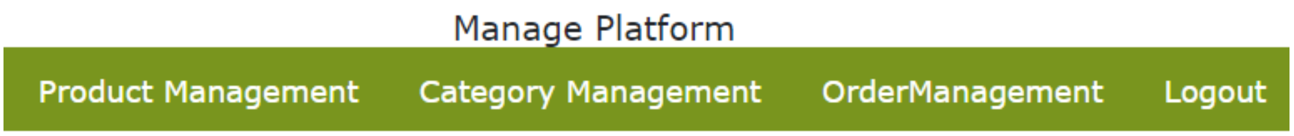


Product Name	Info	Price	Weight	Amount
product1	This is info 1	18.0	227	1

- **Admin Scenarios**

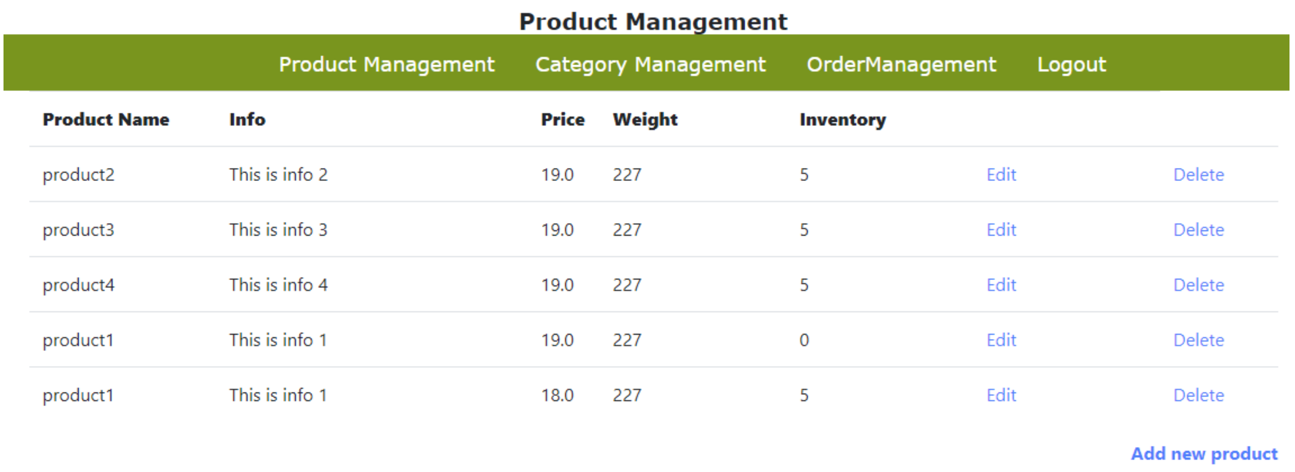
1. Login/Logout





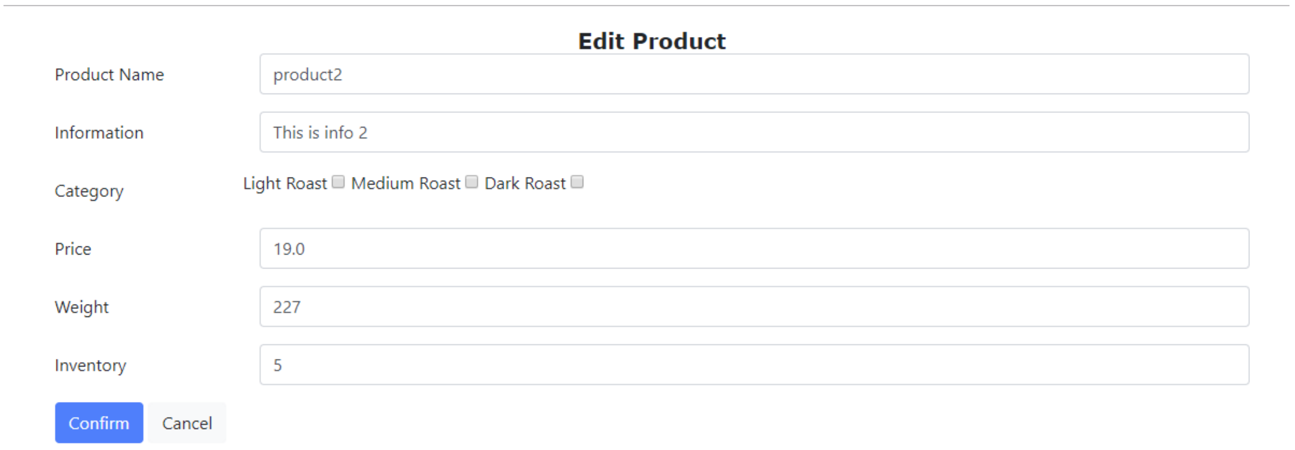
2. Manage Product

2.1. Manage all products by clicking "Product Management".



Product Name	Info	Price	Weight	Inventory		
product2	This is info 2	19.0	227	5	Edit	Delete
product3	This is info 3	19.0	227	5	Edit	Delete
product4	This is info 4	19.0	227	5	Edit	Delete
product1	This is info 1	19.0	227	0	Edit	Delete
product1	This is info 1	18.0	227	5	Edit	Delete

2.2. Add, delete or edit a product. Qualified input format is shown as below



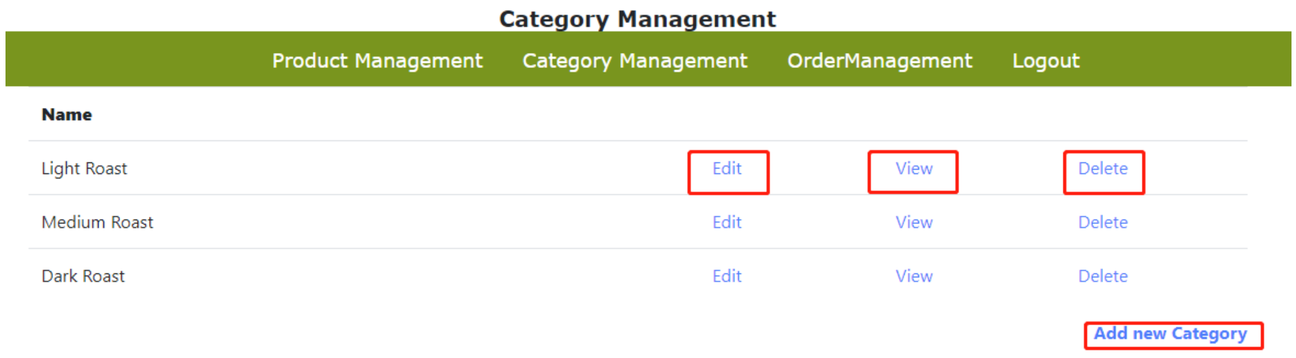
Product Name	product2
Information	This is info 2
Category	Light Roast ☐ Medium Roast ☐ Dark Roast ☐
Price	19.0
Weight	227
Inventory	5

3. Mange Category

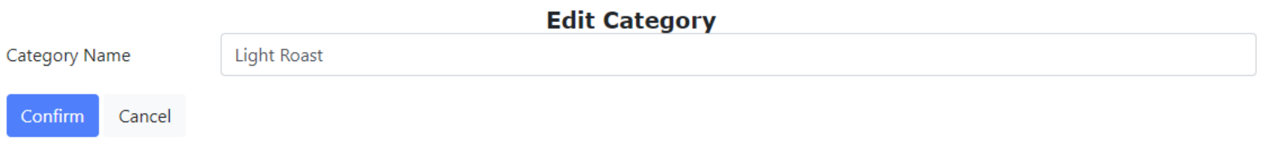
3.1. manage all categories by clicking the “Category Management”

3.2. Add, delete or edit a category. The input format is shown as below

3.3. View all products that belongs to a particular category and delete products from the category by clicking delete



Name			
Light Roast	Edit	View	Delete
Medium Roast	Edit	View	Delete
Dark Roast	Edit	View	Delete



Category Name: Light Roast