Fpt University

TRƯỜNG ĐẠI HỌC FPT

GIFT ECOMMERCE WEBSITE

Software Design Document

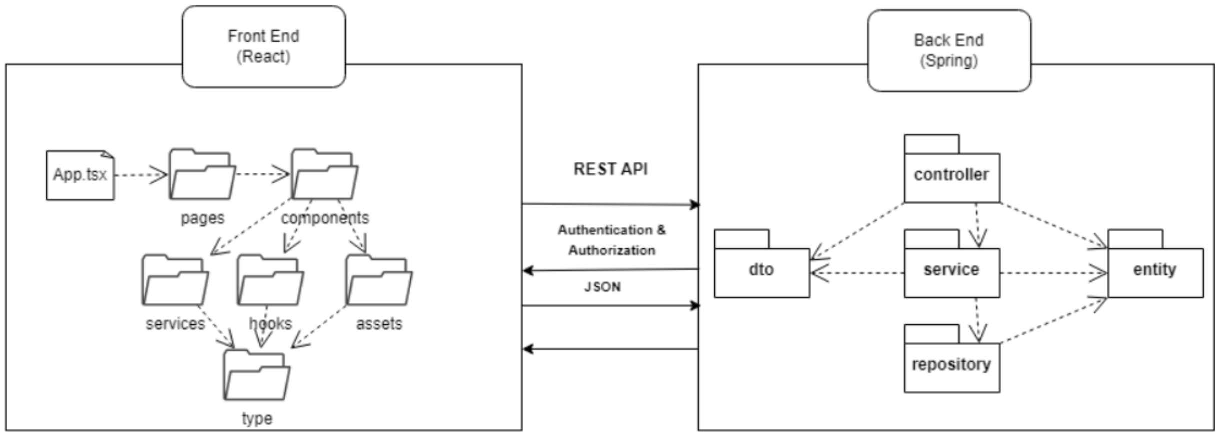
– Ho Chi Minh City, May 2023 –

Table of Contents

I. Overview

1. Code Packages/Namespaces

[Provide the package diagram for each sub-system. The content of this section including the overall package diagram, the explanation, package and class naming conventions in each package. Please see the sample and description table format below – following Java project naming convention]



Package descriptions & package class naming conventions

No	Package	Description
01	*controller*	- *Where to hold REST Controller classes.* - *Classname + "Controller" (ex: HomeController.java)*
02	*service*	- *Where to hold service classes for business logic.* - *Classname + "Service" (ex: OrderService.java)*
03	*repository*	- *Where to hold repository classes for interacting with database.* - *Classname + "Repository" (ex: OrderRepository.java)*
04	*entity*	- *Where to hold entity classes for mapping with database's table.* - *TableName (ex: Order.java)*
05	*dto*	- *Where to hold classes for transferring data for REST APIs.* - *ClassName + "DTO" (ex: LoginInfoDTO.java)*
06	*pages*	- *Where to hold comprehensive components for pages.* - *PageName.tsx (ex: Home.tsx)*
07	*components*	- *Where to hold basic React components.* - *ComponentName.tsx (ex: Header.tsx)*
08	*services*	- *Where to hold service file like fetch api,...* - *Name + "Service.ts" (ex: ProductService.ts)*
09	*hooks*	- *Where to hold React custom hook.* - *"use" + HookName.ts (ex: useFetchAPI.ts)*
10	*assets*	- *Where to hold web assets like images...*
11	*type*	- *Where to hold defined type & interfaces for dto.* - *TypeName.ts (ex: UserDTO.ts)*

2. Database Schema

[Provide the tables relationship like example below – following MySQL database naming convention]

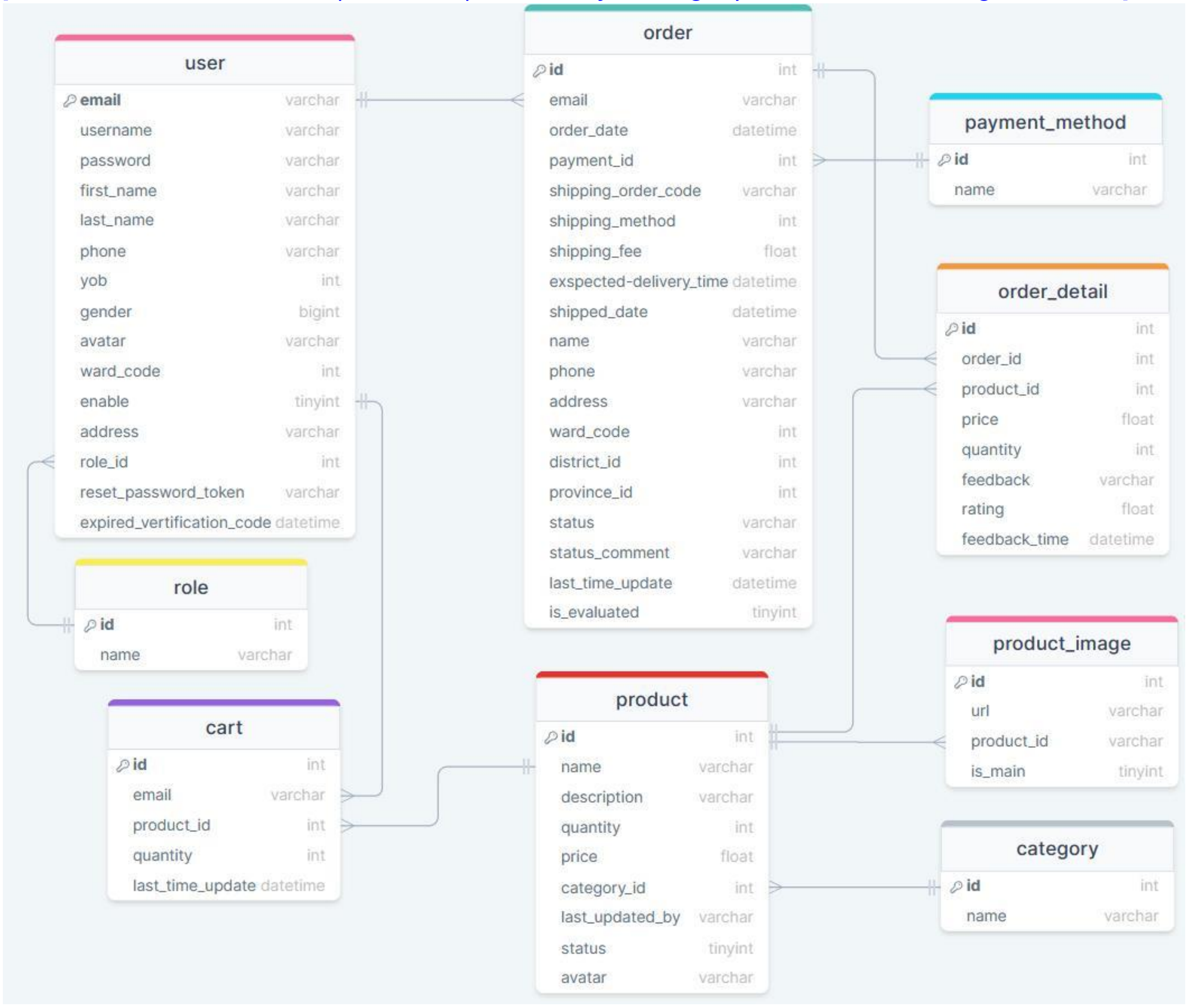


Table descriptions & package class naming conventions are as below

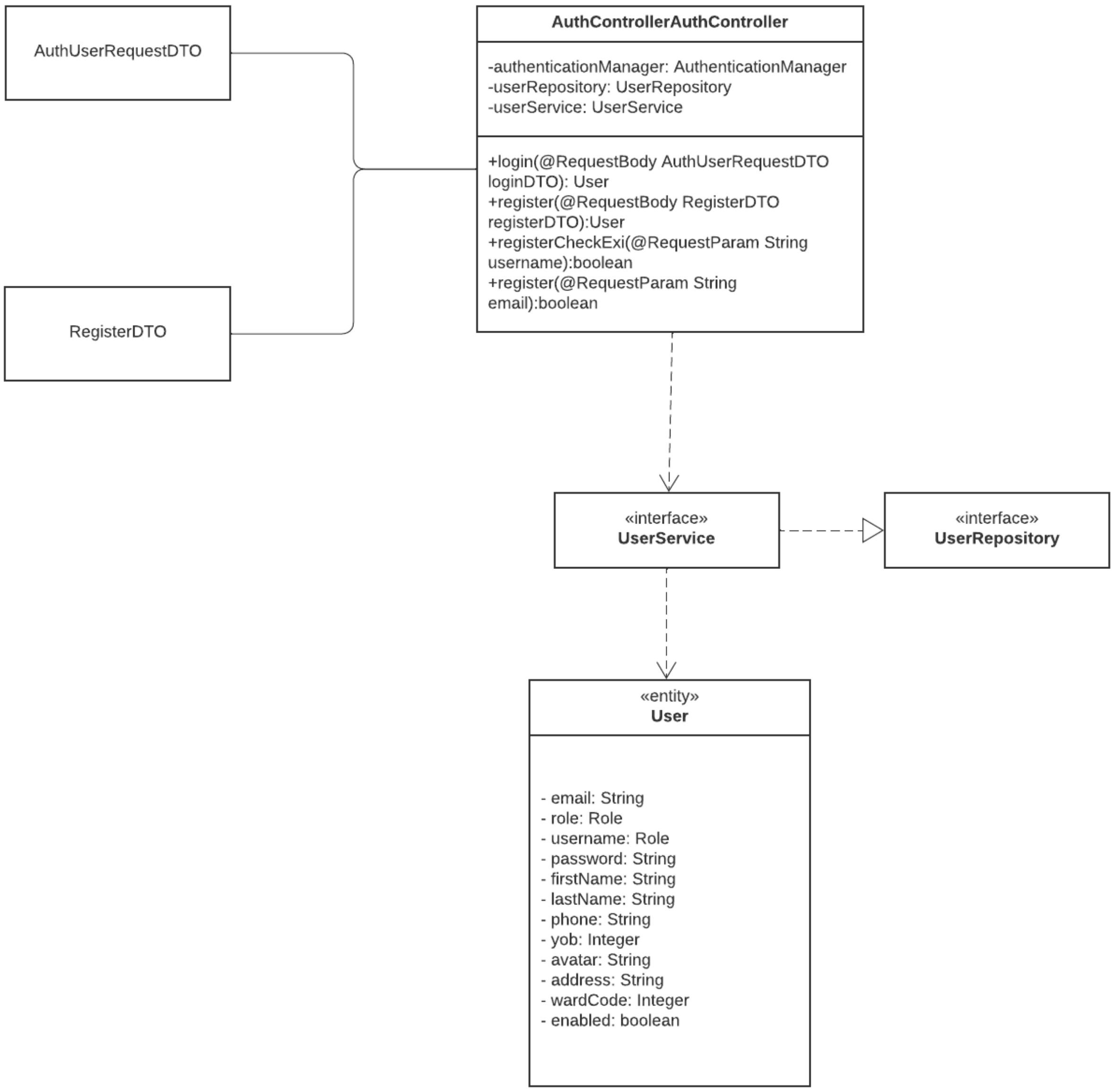
No	Package	Description
01	*user*	*contains users' information* *Primary keys: email* *Foreign keys: ward_id*
02	*role*	*contains users' roles.* *Primary keys: id*
03	*order*	*contains information about orders.* *Primary keys: id* *Foreign keys:* *-email* *-payment_id* *-ward_id*
04	*payment_method*	*contains information about orders.* *Primary keys: id*

		Foreign keys: *-email* *-payment_id* *-ward_id*
05	*order_detail*	*contains information about order details.* *Primary keys: id* *Foreign keys:* *-order_id* *-product_id*
06	*product*	*-contains information about the product.* *-Primary keys: id* *-Foreign keys:* *-category_id* *-last_updated_by*
07	*category*	*contains a product category list.* *Primary keys: id*
08	*product_image*	*contains product images' information.* *Primary keys: id* *Foreign keys:* *product_id*
09	*cart*	*contains customer's product in cart information.* *Primary keys:* *-email* *-product_id* *-Foreign keys:* *-email* *-product_idd*

II. Code Designs

1. <Authentication/Login/Register/Log out>

a. Class Diagram



AuthUserRequestDTO
RegisterDTO
AuthControllerAuthController
-authenticationManager: AuthenticationManager
-userRepository: UserRepository
-userService: UserService
+login(@RequestBody AuthUserRequestDTO loginDTO): User
+register(@RequestBody RegisterDTO registerDTO):User
+registerCheckExi(@RequestParam String username):boolean
+register(@RequestParam String email):boolean
«interface» UserService
«interface» UserRepository
«entity» User
- email: String
- role: Role
- username: Role
- password: String
- firstName: String
- lastName: String
- phone: String
- yob: Integer
- avatar: String
- address: String
- wardCode: Integer
- enabled: boolean

b. Class Specifications

[Provide the description for each class and the methods in each class, following the table format as below]

AuthController

No	Method	Description
01	*login*	*login(@RequestBody AuthUserRequestDTO loginDTO)* Require users to authenticate their identity before accessing certain features
02	*register*	register(@RequestBody RegisterDTO registerDTO)

		Allows new users to create accounts and gain access to the features and functionality available on the platform
03	*registerCheckExist*	*registerCheckExist(@RequestParam String username)* Prevent duplicate registrations and ensure that each user has a unique account within the platform
04	*register*	*register(@RequestParam String email)* Require users to authenticate their identity before accessing certain features. @PostMapping("/register/error/email") Prevent duplicate registrations

User

No	**Method**	**Description**
01	*get()*	*Returns the current value of the property*
02	*set()*	Set new value of the property
03	*editProfile*	*editProfile(UserProfileDTO userProfile, Role role)* Allows users to modify their personal profile information

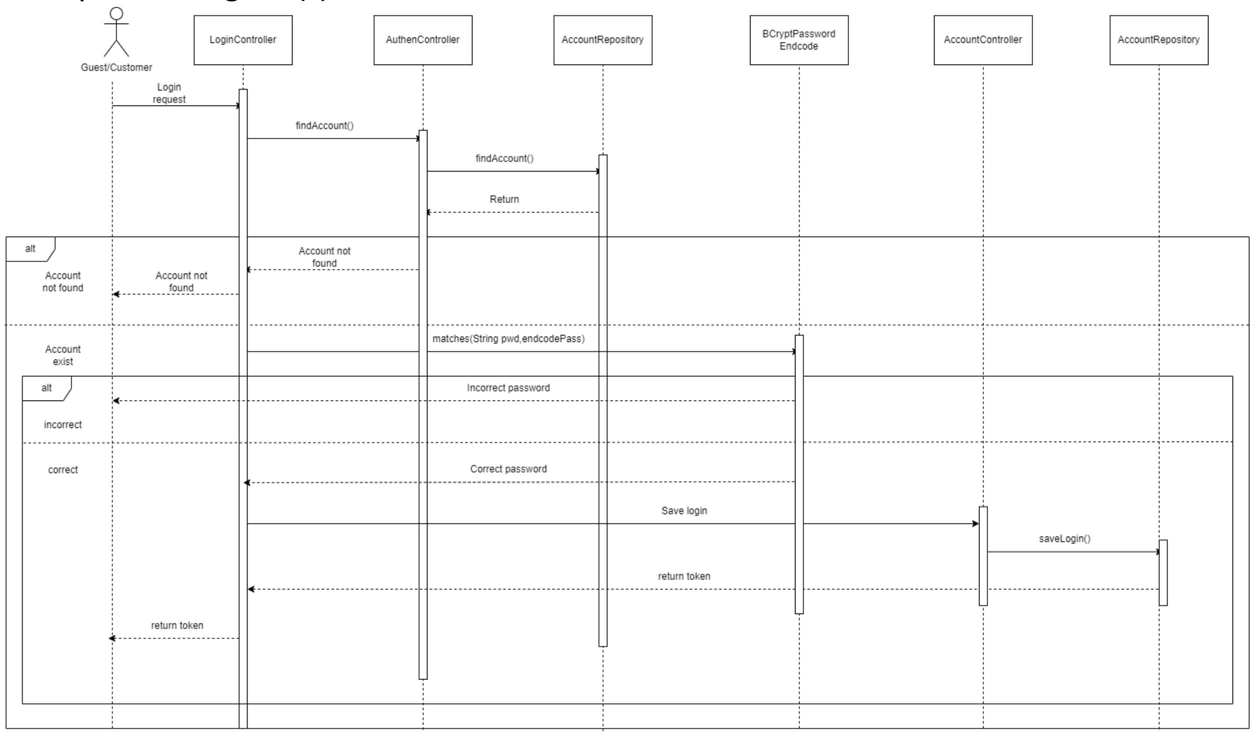
UserRepository

No	**Method**	**Description**
01	*getUserByUsername*	*Parameter: String username. This function will find a user by username*
02	*getUserByEmail*	*Parameter: String email. This function will find a user by email*
03	*getUsersByRoleId*	*Parameter: Pageable pageable and int roleId. Find the user list by role id*
04	*getUserByUsernameOrEmail*	*Parameter: String usernameOrEmail. Find the user by username or email*
05	*getUsersByRoleId*	*Parameter: Pageable pageable and int roleId and boolean enabled. Find the user list by role id with an enabled status: TRUE or FALSE*
06	*setEnabledByEmail*	*Parameter: String email and enabled. Return integer number of row affected (set enabled)*
07	*filterUsersByRoleId*	*Parameter: Pageble pageable, int roleId, boolean enabled, and String search. Find the user list by role id with an enabled status: TRUE or FALSE with search element.*

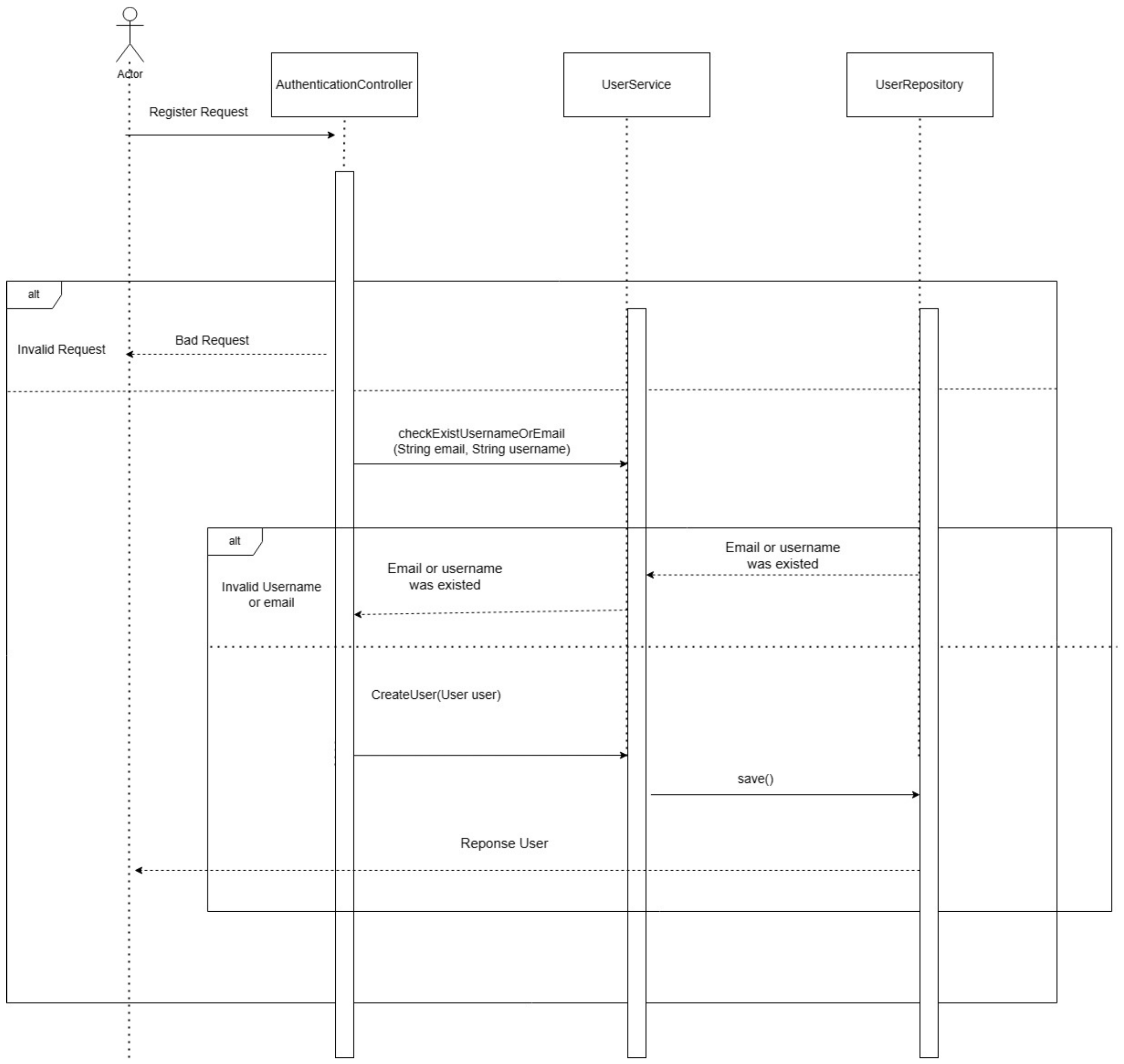
Interface UserService was implemented by UserServiceImpl

No	Method	Description
01	*getPageableUsers*	*Parameter: int pageNo, int pageSize, int roleId . This function will return a page of user list with role Id and page information is PageNo and PageSize*
02	*getUserByEmail*	*Parameter: String email. This function will return a user by email*
03	*getPageableUsers*	*Parameter: int pageNo, int pageSize, int roleId, and boolean enabled . This function will return a page of user list with role Id and enabled status and page information is PageNo and PageSize*
04	*checkExistUser*	*Parameter: String usernameOrEmail. Return boolean, check that user is exist or not: TRUE or FALSE*
05	*getUserByEmailOrUsername*	*Parameter: String usernameOrEmail and boolean enabled. Return user by username or email and their enabled*
06	*setEnabledUserByEmail*	*Parameter: String email and UserProfileDTO. Return user after update their profile*
07	*updateUserProfile*	*Parameter: Pageable pageable, int roleId, boolean enabled, and String search. Find the user list by role id with an enabled status: TRUE or FALSE with search element.*
08	*createUser*	*Parameter: User user. Return user after that created*
09	*searchUsers*	*Parameter: int pageNo, int pageSize, int roleId, boolean enabled and String search. This function will return a page of user list with role Id and enabled status and page information is PageNo and PageSize with a search element*
10	*register*	*Parameter: RegisterDTO form. Return an user after register successfully*
11	*updateResetPassword*	*Parameter: String token, String email. This function was used to reset the password of a user account*
12	*getResetPasswordToken*	*Parameter: String resetPasswordToken. This function will return the User that is reset password token*
13	*updateUserPassword*	*Parameter: User user, String password. This function was used to update new password of the user with that parameter*
14	*getExTime*	*Parameter: String token. This function will return the user that get ex time*

c. Sequence Diagram(s)



Login sequence diagram



Register User

d. Database queries

GetUserByUsername

SELECT u FROM User u WHERE u.username = :username

GetUserByEmail

SELECT u FROM User u WHERE u.email = :email

GetUsersByRoleId

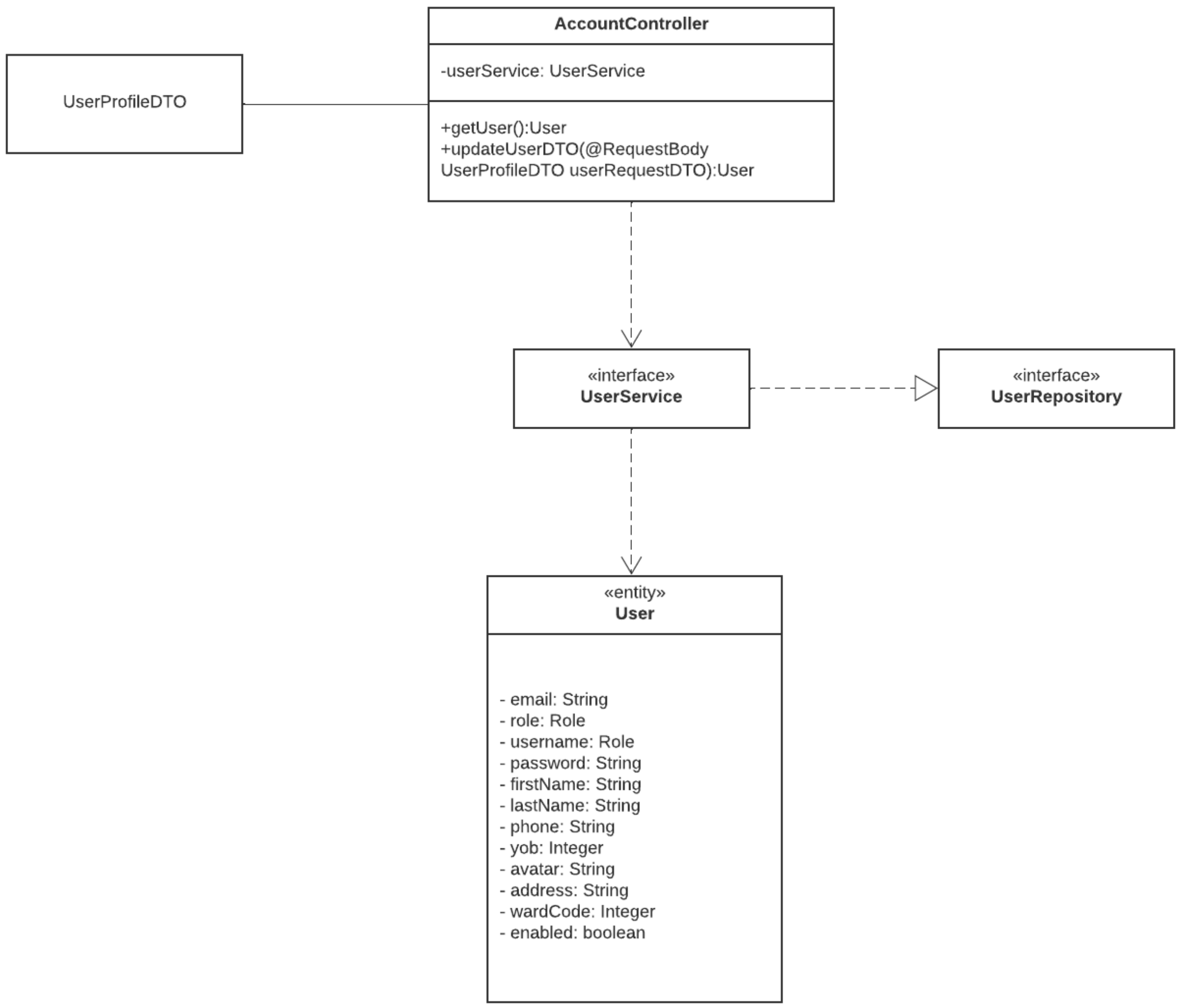
SELECT u FROM User u WHERE u.role.id = :roleId

FilterUsersByRoleId

SELECT u FROM User u WHERE u.role.id = :roleId AND u.enabled = :enabled AND (u.username LIKE %:search% "OR u.email LIKE %:search% OR (u.firstName || ' ' || u.lastName LIKE %:search%))

2. <Manage Account Profile/View profile/Edit profile>

a. Class Diagram



b. Class Specifications

UserRepository

No	Method	Description
01	*getUser*	*Parameter: No parameter. This function will return a user that login in the system*
02	*updateUserDTO*	*Parameter: UserProfileDTO u. This function was used to update the user in database with the parameter profile*

Interface UserService was implemented by UserServiceImpl

No	Method	Description
01	*getPageableUsers*	*Parameter: int pageNo, int pageSize, int roleId . This function will return a page of user list with role Id and page information is PageNo and PageSize*
02	*getUserByEmail*	*Parameter: String email. This function will return a user by email*
03	*getPageableUsers*	*Parameter: int pageNo, int pageSize, int roleId, and boolean enabled . This function will return a page of user list with role Id and enabled status and page information is PageNo and PageSize*
04	*checkExistUser*	*Parameter: String usernameOrEmail. Return boolean, check that user is exist or not: TRUE or FALSE*
05	*getUserByEmailOrUsername*	*Parameter: String usernameOrEmail and boolean enabled. Return user by username or email and their enabled*
06	*setEnabledUserByEmail*	*Parameter: String email and UserProfileDTO. Return user after update their profile*
07	*updateUserProfile*	*Parameter: Pageable pageable, int roleId, boolean enabled, and String search. Find the user list by role id with an enabled status: TRUE or FALSE with search element.*
08	*createUser*	*Parameter: User user. Return user after that created*
09	*searchUsers*	*Parameter: int pageNo, int pageSize, int roleId, boolean enabled and String search. This function will return a page of user list with role Id and enabled status and page information is PageNo and PageSize with a search element*
10	*register*	*Parameter: RegisterDTO form. Return an user after register successfully*
11	*updateResetPassword*	*Parameter: String token, String email. This function was used to reset the password of a user account*
12	*getResetPasswordToken*	*Parameter: String resetPasswordToken. This function will return the User that is reset password token*
13	*updateUserPassword*	*Parameter: User user, String password. This function was used to update new password of the user with that parameter*
14	*getExTime*	*Parameter: String token. This function will return the user that get ex time*

UserRepository

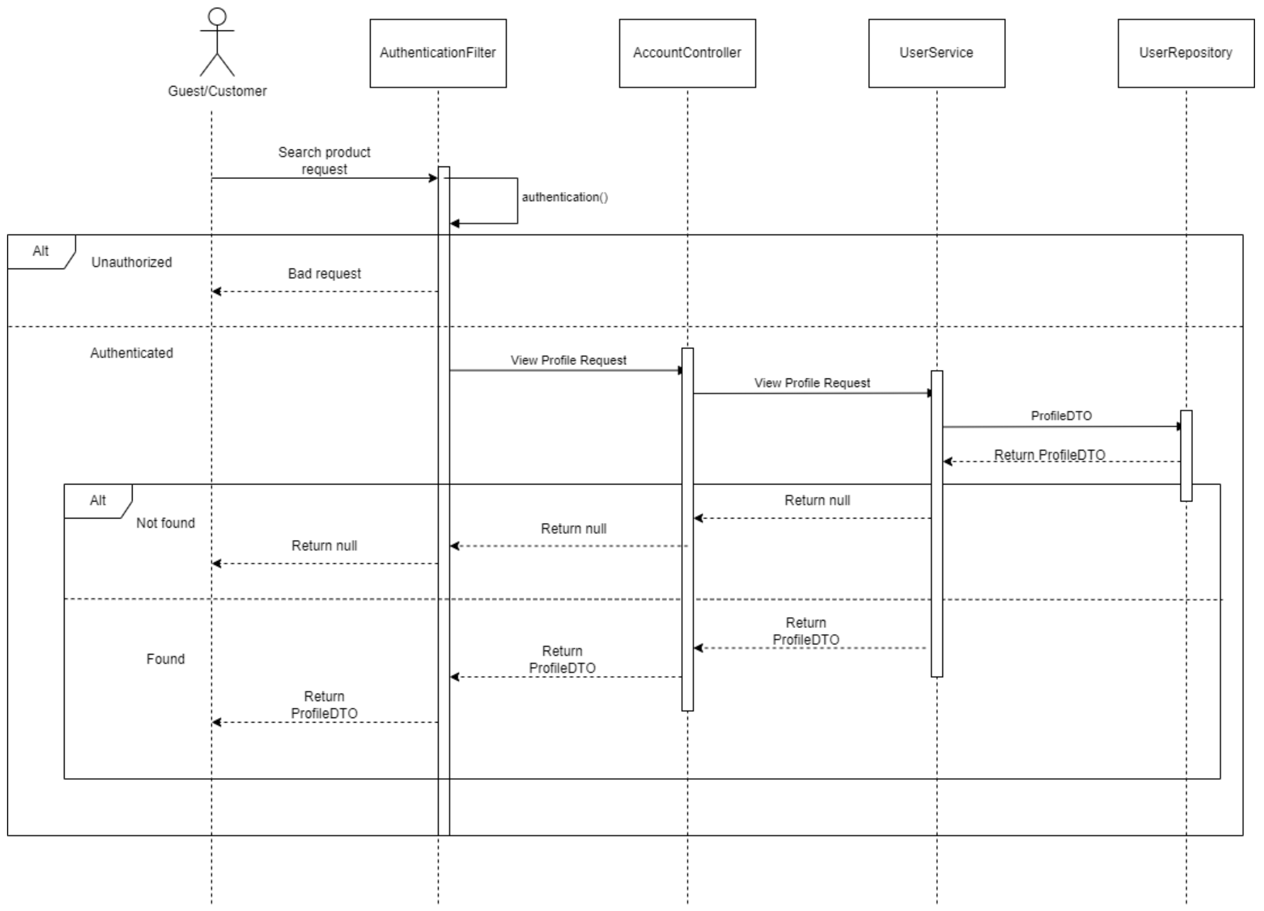
No	Method	Description

01	*getUserByUsername*	*Parameter: String username. This function will find a user by username*
02	*getUserByEmail*	*Parameter: String email. This function will find a user by email*
03	*getUsersByRoleId*	*Parameter: Pageable pageable and int roleId. Find the user list by role id*
04	*getUserByUsernameOrEmail*	*Parameter: String usernameOrEmail. Find the user by username or email*
05	*getUsersByRoleId*	*Parameter: Pageable pageable and int roleId and boolean enabled. Find the user list by role id with an enabled status: TRUE or FALSE*
06	*setEnabledByEmail*	*Parameter: String email and enabled. Return integer number of row affected (set enabled)*
07	*filterUsersByRoleId*	*Parameter: Pageable pageable, int roleId, boolean enabled, and String search. Find the user list by role id with an enabled status: TRUE or FALSE with search element.*
08	*getResetPasswordToken*	*Parameter: String token. This function will return the reset password token*
10	*getExTime*	*Parameter: String token. This function will return the user that get ex time*

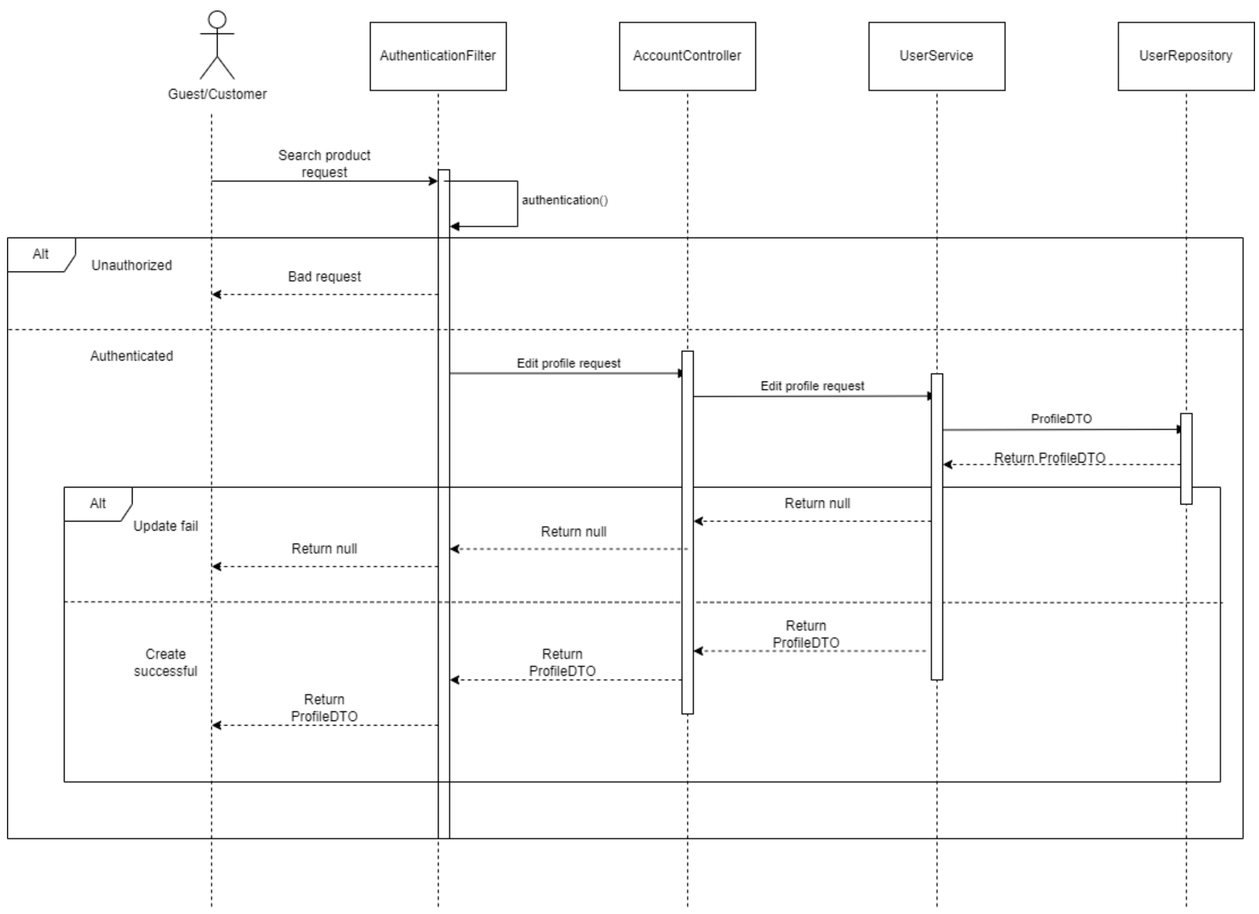
User

01	*get()*	*Returns the current value of the property*
02	*set()*	Set new value of the property
03	*editProfile*	*editProfile(UserProfileDTO userProfile, Role role)* Allows users to modify their personal profile information

c. Sequence Diagram(s)



View Account profile



Update Account profile

d. Database queries

Get user by username

SELECT u FROM User u WHERE u.username = :username

Get user by email

SELECT u FROM User u WHERE u.email = :email

Get user by roleId

SELECT u FROM User u WHERE u.role.id = :roleId

Get user with enabled status by username or email

SELECT u FROM User u WHERE u.enabled = :enabled and (u.email = :check OR u.username = :check)

Get users by role id and enabled

SELECT u FROM User u WHERE u.role.id = :roleId AND u.enabled = :enabled

Update user enabled by username or email

```
UPDATE User u SET u.enabled = :enabled WHERE
                              u.email = :emailOrUsername OR u.username = :emailOrUsername)
```

Filter user by firstname and lastname by role id and enabled

```
SELECT u FROM User u
        WHERE u.role.id = :roleId
        AND u.enabled = :enabled AND (u.username LIKE %:search%
        OR u.email LIKE %:search% OR (u.firstName || ' ' || u.lastName LIKE %:search%))"
```

Get reset token password

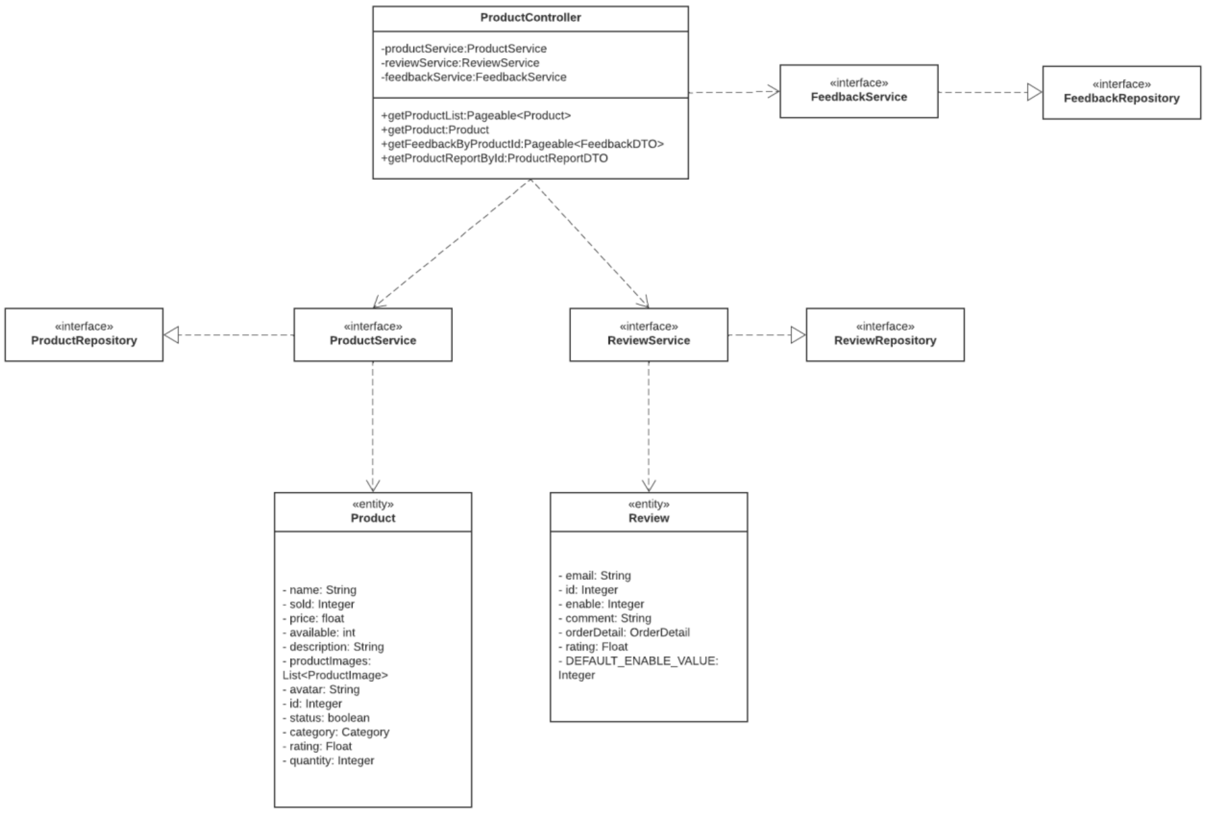
```
SELECT u FROM User u WHERE u.reset_password_token = :token
```

Get user ex time

```
SELECT u FROM User u WHERE u.reset_password_token = :token and expired_vertification_code >
CURRENT_TIMESTAMP
```

3. <View product in shop/View product/Search product>

a. Class Diagram


ProductController
-productService:ProductService
-reviewService:ReviewService
-feedbackService:FeedbackService
+getProductList:Pageable<Product>
+getProduct:Product
+getFeedbackByProductId:Pageable<FeedbackDTO>
+getProductReportById:ProductReportDTO
«interface» FeedbackService
«interface» FeedbackRepository
«interface» ProductRepository
«interface» ProductService
«interface» ReviewService
«interface» ReviewRepository
«entity» Product
- name: String
- sold: Integer
- price: float
- available: int
- description: String
- productImages: List<ProductImage>
- avatar: String
- id: Integer
- status: boolean
- category: Category
- rating: Float
- quantity: Integer
«entity» Review
- email: String
- id: Integer
- enable: Integer
- comment: String
- orderDetail: OrderDetail
- rating: Float
- DEFAULT_ENABLE_VALUE: Integer

b. Class Specifications

ProductController

No	Method	Description
01	*getProductList*	*Parameter:* *+ int pageNo, int pageSize. This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ String search, Integer category, String sortField, Boolean sortOrder, Integer related act as data filters according to the condition*
02	*getProduct*	*Product getProduct(@PathVariable int productId) The implementation of the getProduct() method would typically use the productId parameter to query a database or other data source for details about a specific product*
03	*getReviewOfProduct*	*Parameter:* *+ int pageNo, int pageSize. This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ getReviewOfProduct is a method name that suggests it retrieves reviews for a product.*

ProductService

No	Method	Description
01	*getPageableProducts*	*getPageableProducts(int pageNo, int pageSize)* *Parameter:* *+ int pageNo, int pageSize . This function will return a page of user list with role Id and page information is PageNo and PageSize*
02	*getProductByRelated*	*Product getProduct(@PathVariable int productId) The implementation of the getProduct() method would typically use the productId parameter to query a database or other data source for details about a specific product.* *+ function that returns a collection or list of products that are related to a particular product or item*
03	*searchProductsByName*	*APIPageableResponseDTO<Product> searchProductsByName(int pageNo, int pageSize, String search, String sortField);* *+ function that searches for products based on a given name or search query*
04	*searchProductsByNameInCategory*	*APIPageableResponseDTO<Product> searchProductsByNameInCategory(Integer pageNo, Integer pageSize, String search, Integer category, String sortField);* *Parameter:* *+ int pageNo, int pageSize . This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ function that returns a collection or list of products that are related to a particular product or item*

ReviewService

No	Method	Description
01	*findReviewsByProductId*	*Parameter:* *+ int pageNo, int pageSize . This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ int productId, int enable. The ID of the product being reviewed* *The function would then query a database or other data source to find all reviews associated with the specified product ID*
02	*save*	*Parameter:* *+ RatingRequestDTO ratingRequestDTO contains several properties*

		+ String email The email parameter may be used to identify the user who submitted the rating, allowing the system to associate the rating with a specific user account.

FeedbackService

No	Method	Description
01	*getFeedbackByProductId*	*Parameter:* *+ int pageNo, int pageSize . This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ int productId. The ID of the product being feedback* *The function would then query a database or other data source to find all reviews associated with the specified product ID*

ProductRepository

No	Method	Description
01	*finfAllByRealated*	*finfAllByRealated(@Param("status") boolean status,@Param("realated")* + *function that returns a collection or list of products that are related to a particular product or item* + *finds all products that are related in some way to a particular product, based on the value of the related parameter*
02	*findProductById*	*Product findProductById(@Param("productId") int productId,* *@Param("status") boolean status);* *+ int productId* *The function would then query a database or other data source with the specified product ID*
03	*finfAllByName*	*Page<Product> finfAllByName(@Param("status") boolean status,@Param("search") String search,Pageable pageable);* *+ Pageable pageable . This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ The function would then query a database or other data source the specified product ID*

04	*findAllByStatusByNameByCategory*	*Page<Product> findAllByStatusByNameByCategory(@Param("status") boolean status,@Param("search") String search,@Param("category") Integer category, Pageable pageable);* *+ function that searches for products based on a given name or search query*

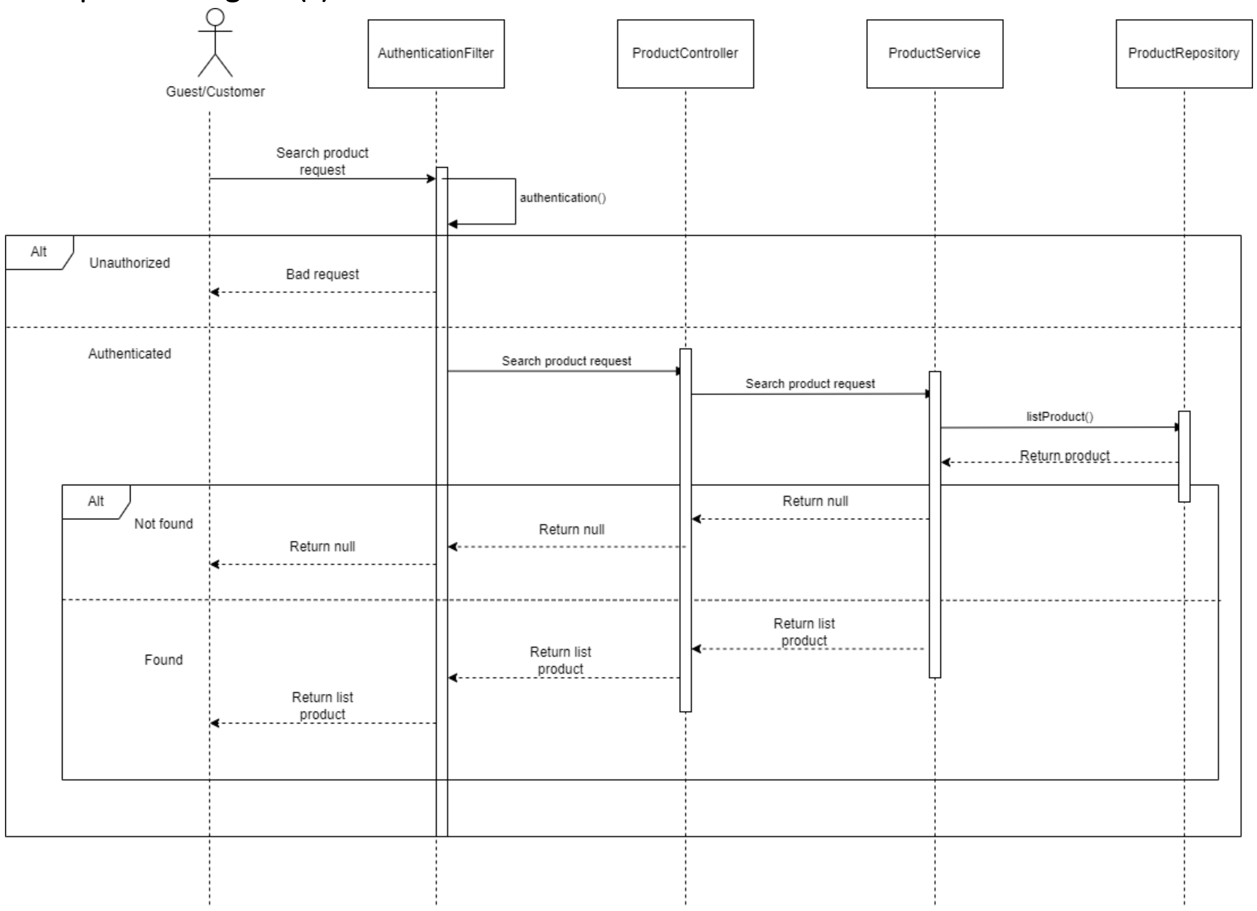
ReviewRepository

No	Method	Description
01	*findReviewsByProductId*	*Page<Review> findReviewsByProductId(Pageable pageable, @Param("productId") int productId, int enable);+ int productId. The ID of the product being reviewed* *Pageable pageable . This function will return a page of user list with role Id and page information is PageNo and PageSize*

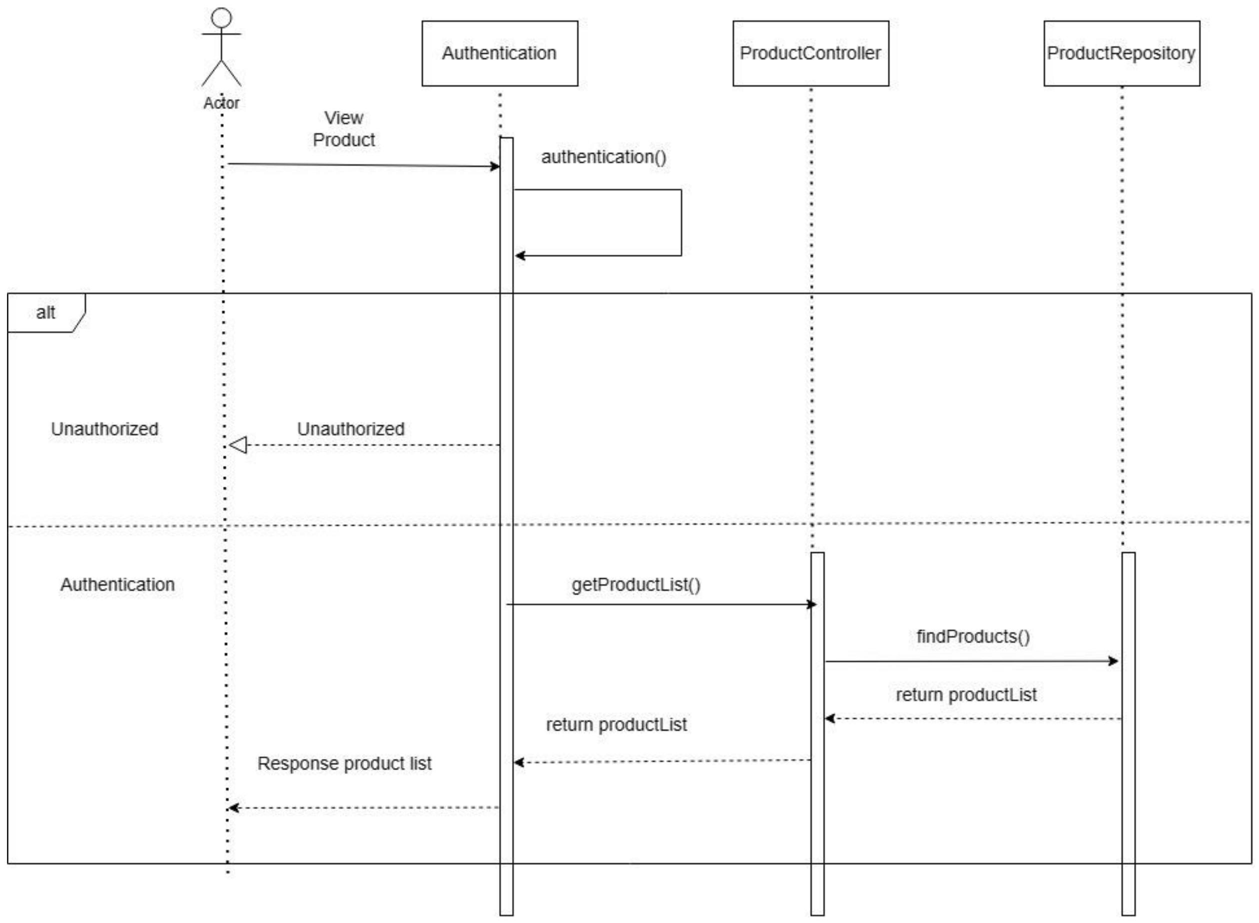
Product

01	*get()*	*Returns the current value of the property*
02	*set()*	Set new value of the property
03	*getAvatar()*	Allows users to modify product avatar information

c. Sequence Diagram(s)



Search product sequence diagram



View Products Sequence diagram

d. Database queries

findAllByStatusByNameByCategory :

@Query("SELECT p FROM Product p WHERE p.status = :status AND p.category.id = :category AND p.name LIKE %:search%")

finfAllByName:

@Query("SELECT p FROM Product p WHERE p.name LIKE %:search% and p.status = :status")

finfAllByRealated:

@Query("SELECT p FROM Product p WHERE p.category.id = (SELECT p2.category.id FROM Product p2 WHERE p2.id = :realated) AND p.status = :status ORDER BY RAND()")

findProductById:

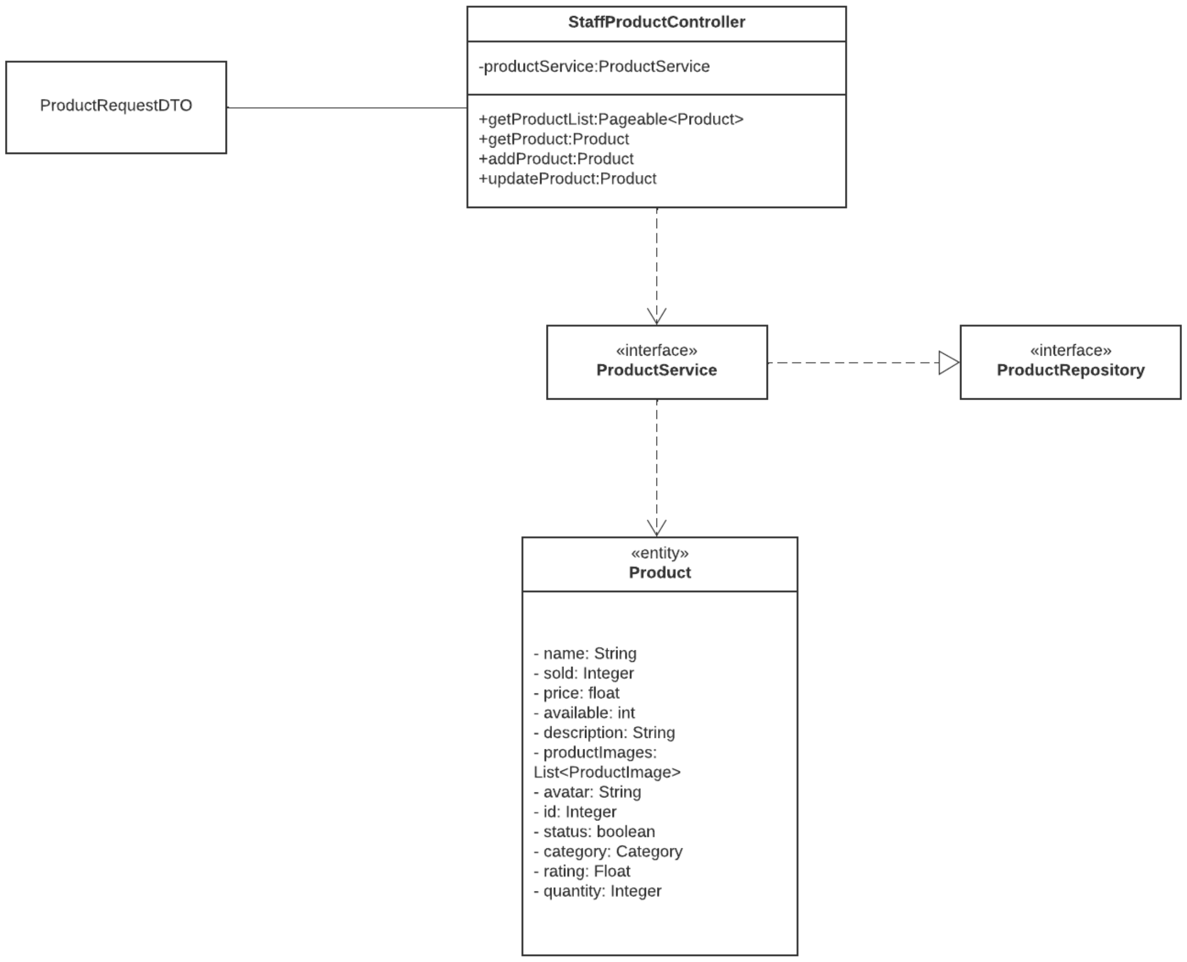
```
@Query("SELECT p FROM Product p WHERE p.id = :productId AND p.status = :status")
```

findReviewsByProductId:

```
 @Query("SELECT r FROM Review r WHERE r.orderDetail.productId = :productId AND r.enable =
:enable")
```

4. <Manage Product/View product/Create product/edit product/delete product/search product>

a. Class Diagram



b. Class Specifications

StaffProductController

No	Method	Description
01	*getProductList*	*Parameter:* *+ int pageNo, int pageSize . This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ String search, Integer category, String sortField, Boolean sortOrder, Integer related act as data filters according to the condition*
02	*getProduct*	*Parameter: @PathVariable int productId. This function will return a product with the parameter productId is the id of the product*
03	*addProduct*	*Parameter: @RequestBody ProductRequestDTO productRequestDTO . This function will return a product that you created and save it in database.*

04	*updateProduct*	*Parameter: @PathVariable int productId, @RequestBody ProductRequestDTO productRequestDTO. This function will return a product that you updated with information productRequestDTO and save it in database*
05	*deleteCustomerById*	*Parameter:@PathVariable int productId. This function will delete a product in database by parameter productId*

ProductService

No	Method	Description
01	*getPageableProducts*	*getPageableProducts(int pageNo, int pageSize)* *Parameter:* *+ int pageNo, int pageSize. This function will return a page of user list with role Id and page information is PageNo and PageSize*
02	*getProductByRelated*	*Product getProduct(@PathVariable int productId) The implementation of the getProduct() method would typically use the productId parameter to query a database or other data source for details about a specific product.* *+ function that returns a collection or list of products that are related to a particular product or item*
03	*searchProductsByName*	*APIPageableResponseDTO<Product> searchProductsByName(int pageNo, int pageSize, String search, String sortField);* *+ function that searches for products based on a given name or search query*
04	*searchProductsByNameInCategory*	*APIPageableResponseDTO<Product> searchProductsByNameInCategory(Integer pageNo, Integer pageSize, String search, Integer category, String sortField);* *Parameter:* *+ int pageNo, int pageSize . This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ function that returns a collection or list of products that are related to a particular product or item*
05	*save*	*Parameter:Product product. This function will save a product in parameter to the database*
06	*checkExist*	*Parameter:Product product. This function return true/false check that the product is exist or not*
07	*delete*	*Parameter:Product product. This function will delete the product in the parameter in the database. return true/false*
08	*update*	*Parameter:int productId, ProductRequestDTO productRequestDTOt. This function will return user that updated by the information product request by product id*

09	*createNewProductFrom*	*Parameter: ProductRequestDTO productRequestDTOt. This function will return user that created an save in the database*

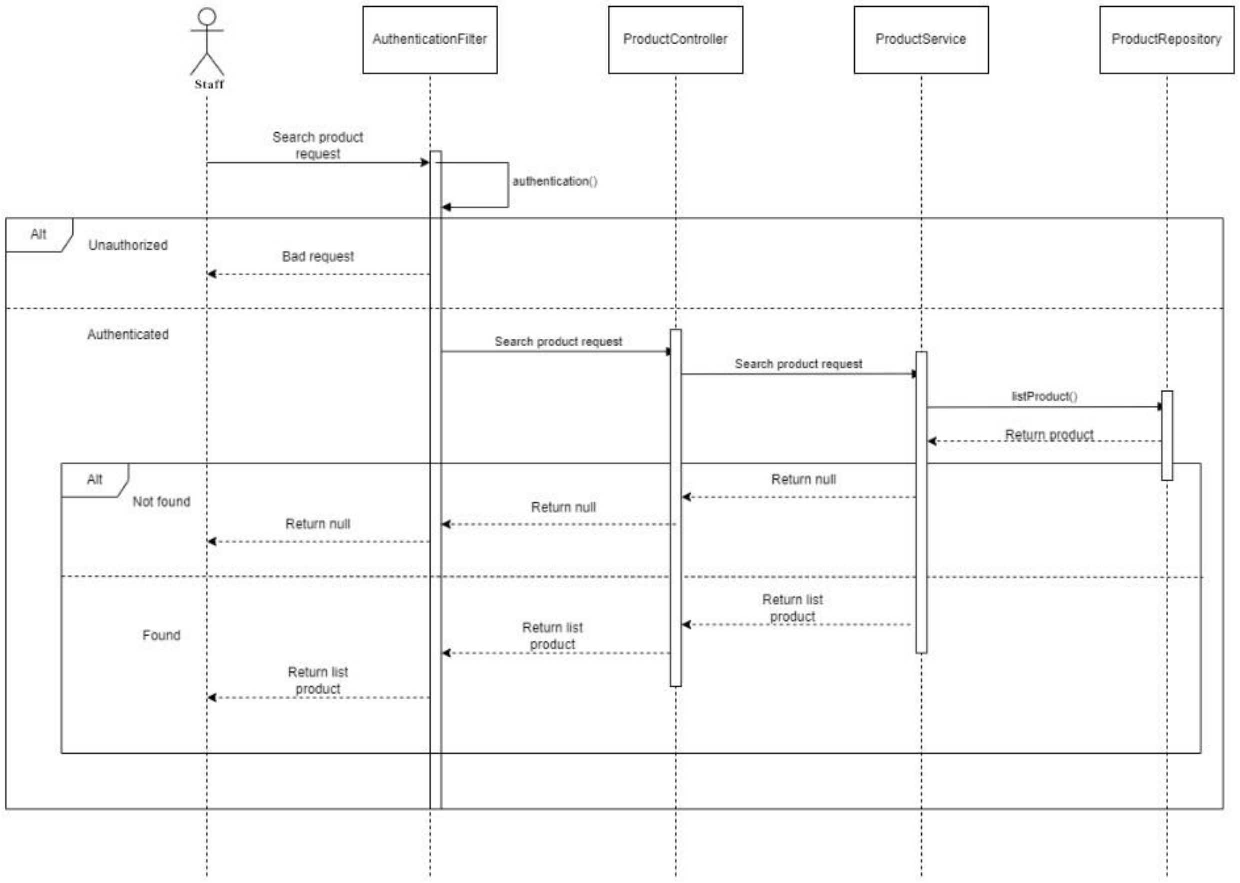
ProductRepository

No	Method	Description
01	*findAllByRealated*	*finfAllByRealated(@Param("status") boolean status,@Param("realated")* + *function that returns a collection or list of products that are related to a particular product or item* + *finds all products that are related in some way to a particular product, based on the value of the related parameter*
02	*findProductById*	*Product findProductById(@Param("productId") int productId,* *@Param("status") boolean status);* *+ int productId* *The function would then query a database or other data source with the specified product ID*
03	*finfAllByName*	*Page<Product> finfAllByName(@Param("status") boolean status,@Param("search") String search,Pageable pageable);* *+ Pageable pageable . This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ The function would then query a database or other data source the specified product ID*
04	*findAllByStatusByNameByCategory*	*Page<Product> findAllByStatusByNameByCategory(@Param("status") boolean status,@Param("search") String search,@Param("category") Integer category, Pageable pageable);* *+ function that searches for products based on a given name or search query*

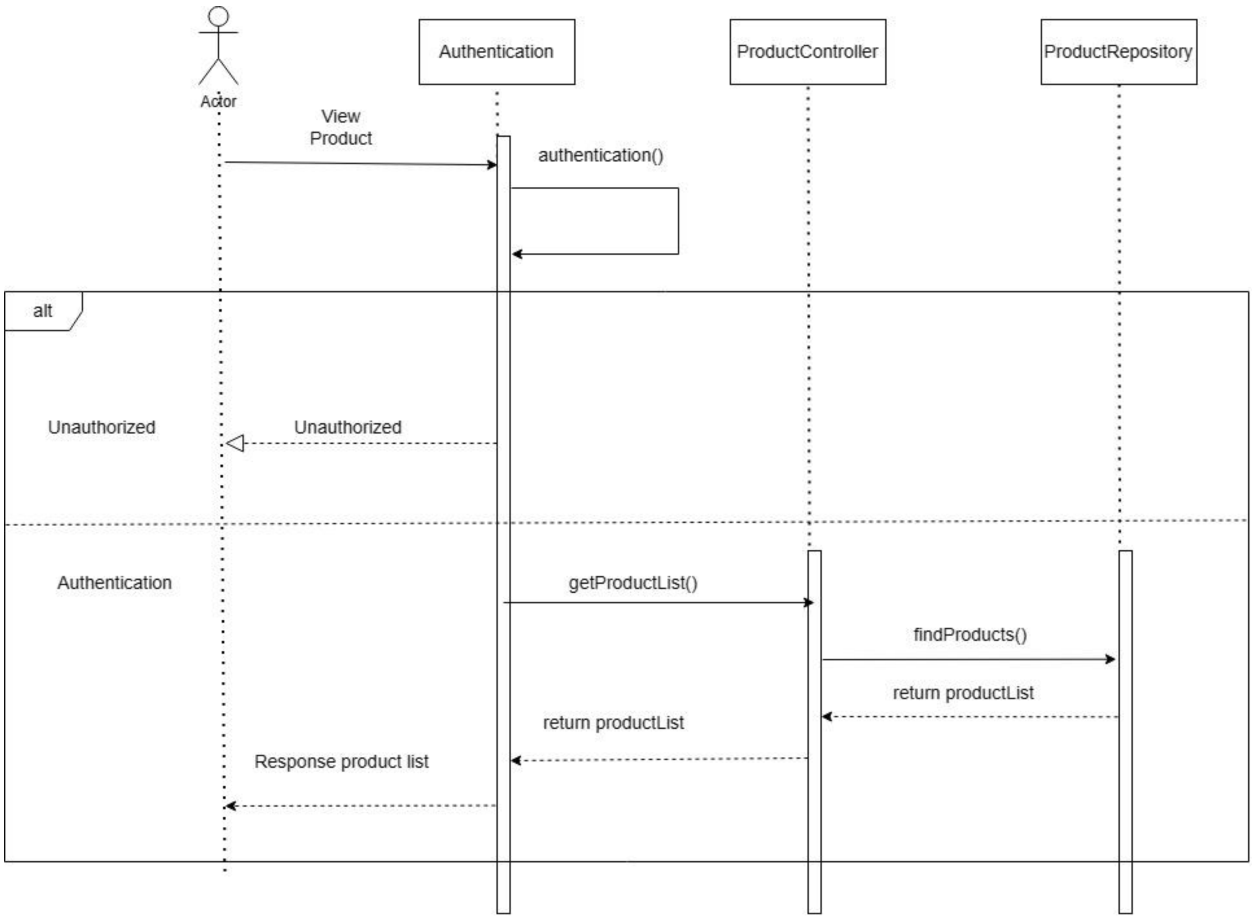
Product

No	Method	Description
01	*get()*	*Returns the current value of the property*
02	*set()*	Set new value of the property
03	*getAvatar()*	Allows users to modify product avatar information

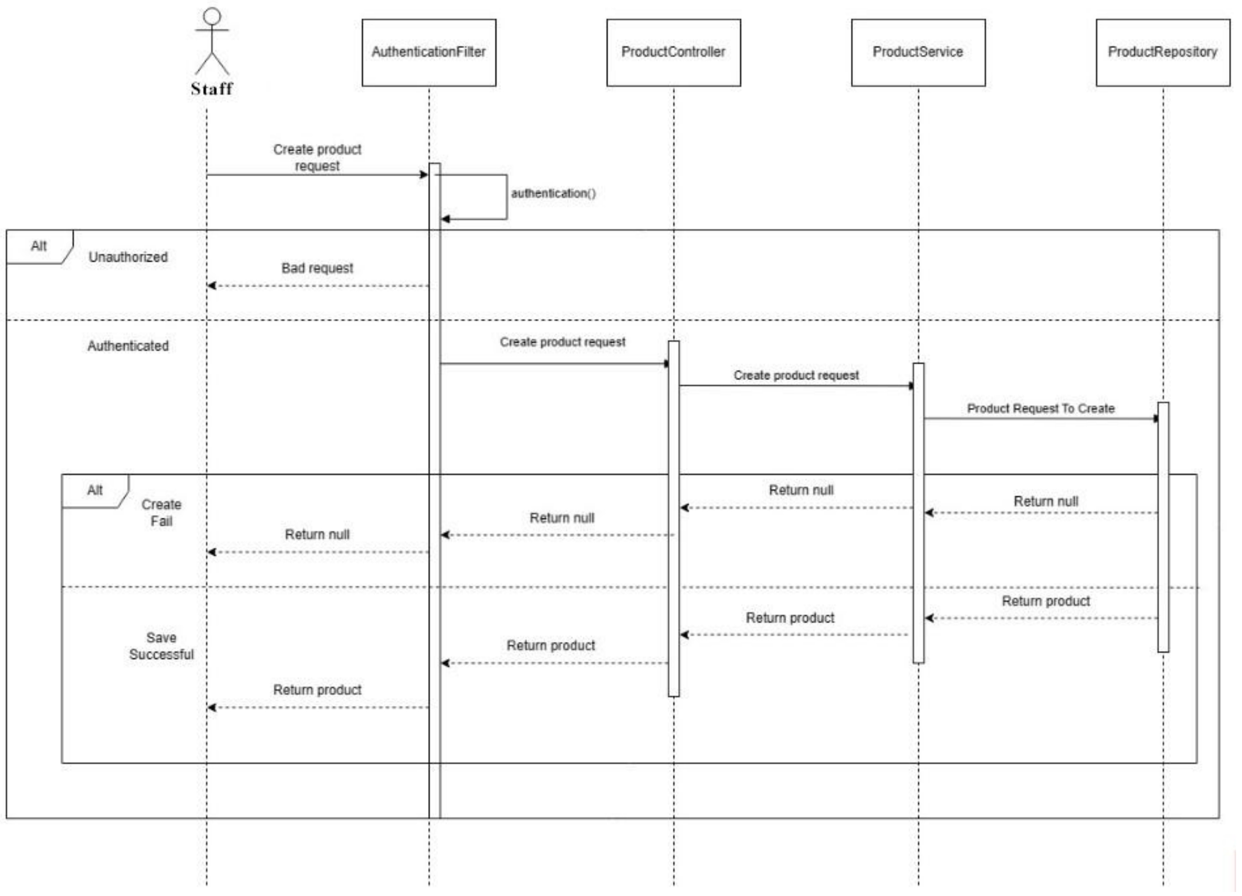
c. Sequence Diagram(s)



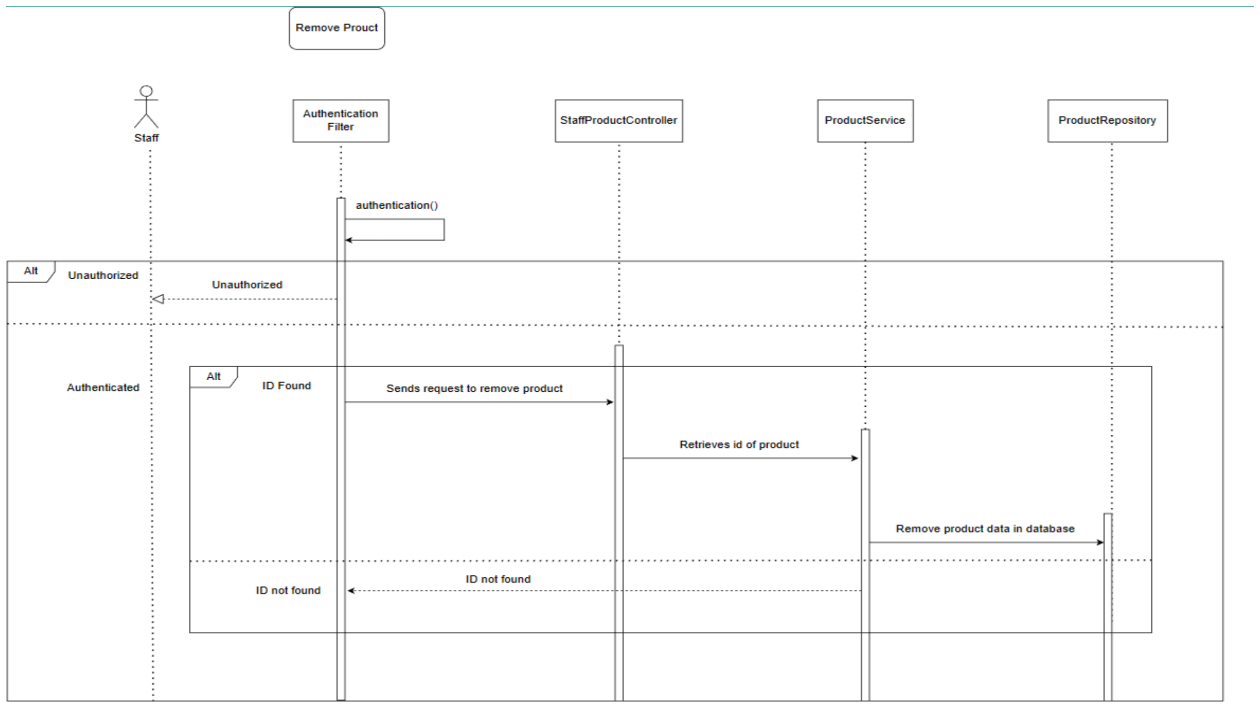
Search product sequence diagram



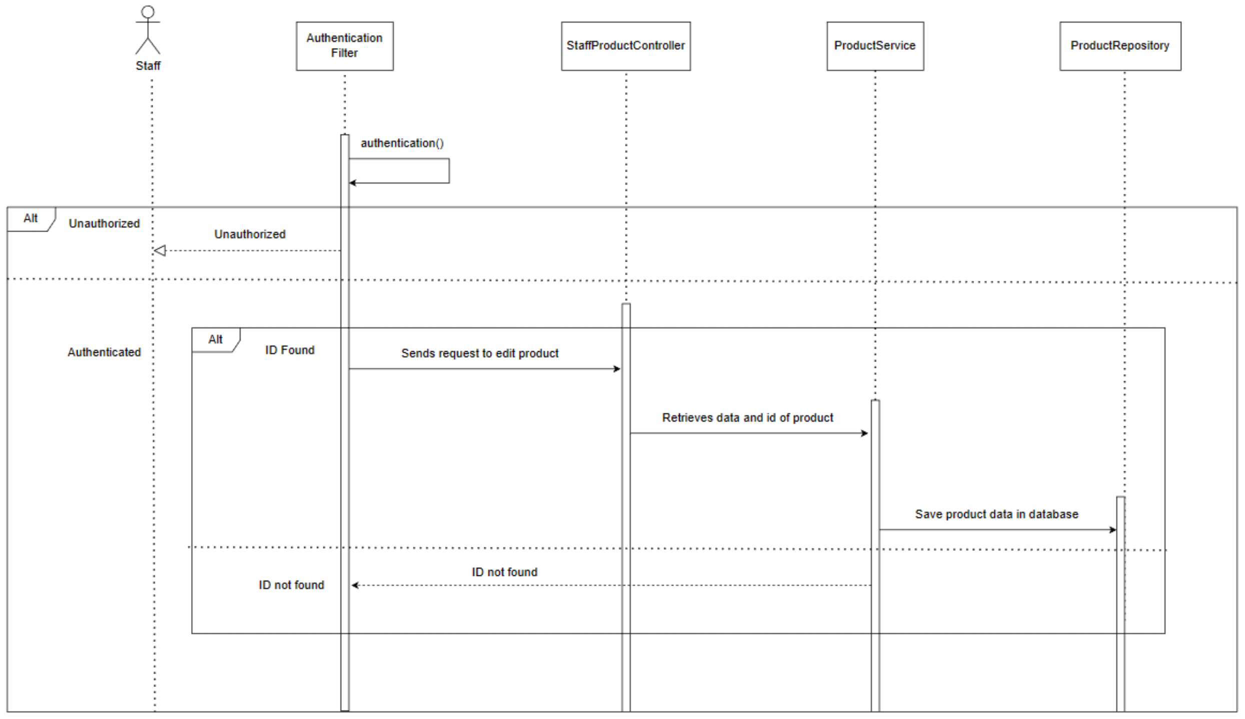
View Products Sequence diagram



Create Product



Remove Product



Edit Product

d. Database queries

Filter products by category and name

SELECT p FROM Product p WHERE p.status = :status AND p.category.id = :category AND p.name LIKE %:search

Filter the products with their name and their status

SELECT p FROM Product p WHERE p.name LIKE %:search% and p.status = :status

Get product by productId and status

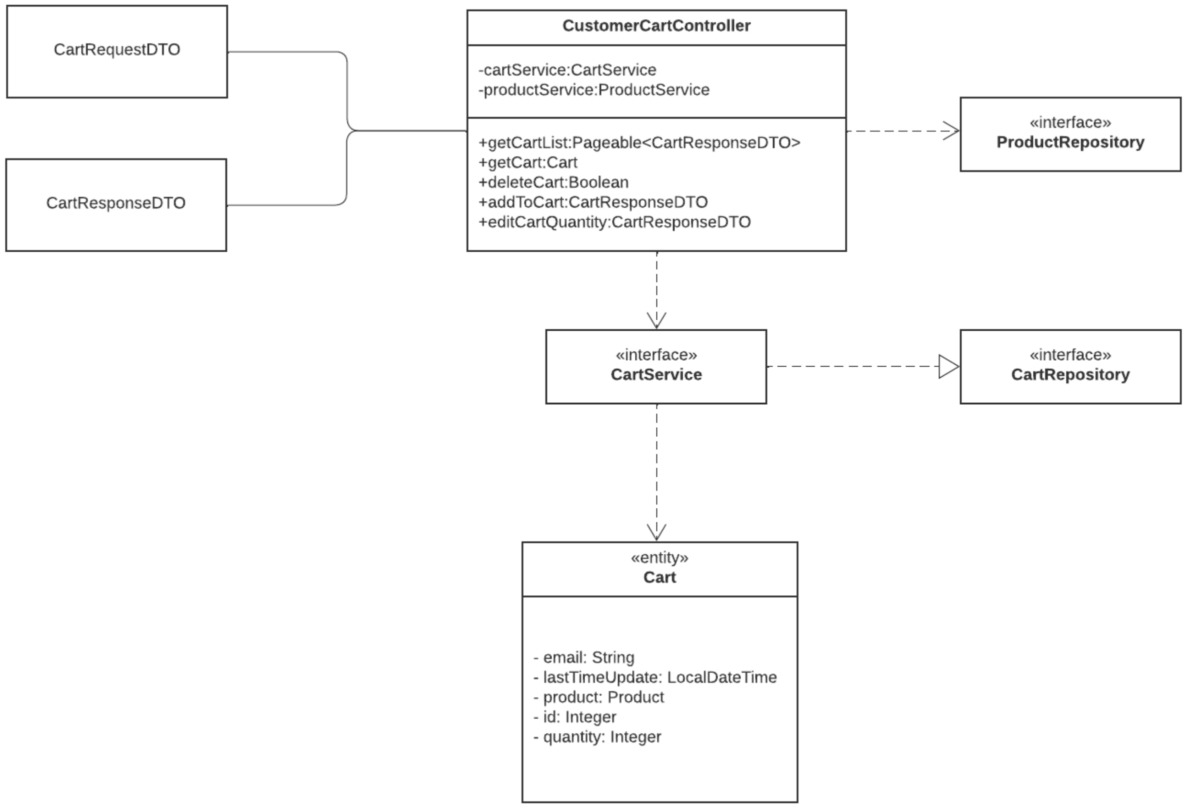
SELECT p FROM Product p WHERE p.id = :productId AND p.status = :status

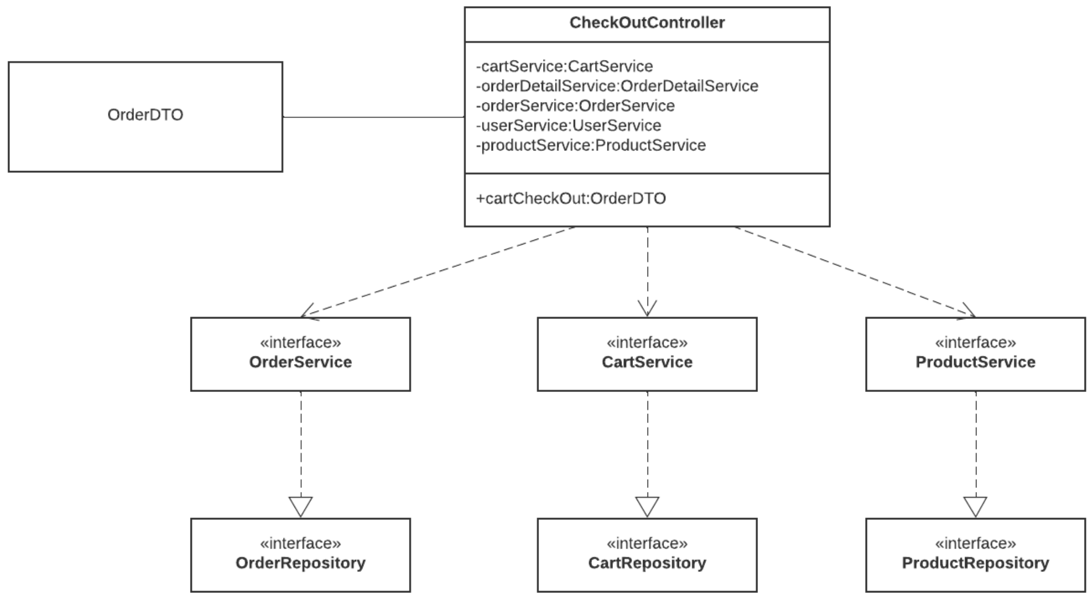
Get Products by email and order by p.id ASC

```
SELECT DISTINCT p

                FROM Product p

                JOIN OrderDetail od ON p.id = od.productId

                JOIN Orders o ON od.orderId = o.id

                JOIN User u ON o.email = u.email

                WHERE u.email = :email

                Order by p.id ASC
```

5. <Manage Cart/View cart/Add to cart/Delete product in cart/Edit product quantity in cart>

a. Class Diagram


CartRequestDTO
CartResponseDTO
CustomerCartController
-cartService:CartService
-productService:ProductService
+getCartList:Pageable<CartResponseDTO>
+getCart:Cart
+deleteCart:Boolean
+addToCart:CartResponseDTO
+editCartQuantity:CartResponseDTO
«interface»
ProductRepository
«interface»
CartService
«interface»
CartRepository
«entity»
Cart
- email: String
- lastTimeUpdate: LocalDateTime
- product: Product
- id: Integer
- quantity: Integer



CheckOutController
-cartService:CartService
-orderDetailService:OrderDetailService
-orderService:OrderService
-userService:UserService
-productService:ProductService
+cartCheckOut:OrderDTO
OrderDTO
«interface» OrderService
«interface» CartService
«interface» ProductService
«interface» OrderRepository
«interface» CartRepository
«interface» ProductRepository

b. Class Specifications

CustomerCartController

No	Method	Description
01	*getCartList*	*APIPageableResponseDTO<CartResponseDTO> getCartList* *Parameter:* *+ int pageNo, int pageSize . This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ String search.* *Function would retrieve a list of items currently present in a user's shopping cart on an e-commerce website or web application*
02	*getCart*	*Cart getCart(@PathVariable int cartId)* *Parameter: @PathVariable int cartId. This function will return a Cart with the parameter cartIdis the id of the Cart*
03	*deleteCart*	*boolean deleteCart(@PathVariable int cartId)* *Parameter: @PathVariable int cartId. This function will delete a Cart with the parameter cartIdis the id of the Cart*

04	*addToCart*	*CartResponseDTO addToCart(@RequestBody CartRequestDTO cartDTO)* *The CartRequestDTO object contains information about the items to be added to the cart* *The purpose of this method is to add the items specified in the cartDTO object to the user's shopping cart.*
05	*editCartQuantity*	*CartResponseDTO editCartQuantity(@RequestBody CartRequestDTO cartDTO)* *The CartRequestDTO object contains information about the item whose quantity needs to be edited* *The purpose of this method is to edit the quantity of an item in the user's shopping cart.*

CartService

No	Method	Description
01	*getCartByEmailAndProductId*	*Cart getCartByEmailAndProductId(String email, int productId);* *Find Cart base on String email, int productId*
02	*getPagableCart*	*APIPageableResponseDTO<CartResponseDTO> getPagableCart(Integer pageNo, Integer pageSize,String email);* *Get List Cart IPageable*
03	*getCartByEmailAndCartId*	*Cart getCartByEmailAndCartId(String emai, int cartId);* *Find Cart base on String email, int CartId*
04	*refreshCart*	*Cart refreshCart(Cart cart)* *Reload the cart every time there is an action with the cart*
05	*getShopAvailableQuantity*	*int getShopAvailableQuantity(int productId);* *Show AvailableQuantity of product User can add to cart*
06	*getCustomerAvailableQuantity*	*int getCustomerAvailableQuantity(String email, int productId)* *Limit quantity of product User can add to cart*
07	*save*	*Cart save(Cart cart)* *Save cart info*
08	*deleteCart*	*deleteCart(String email, int productId);* *Delete product in cart base on email, productId*

CartRepository

No	Method	Description

01	*findCartByEmailAndProductId*	*Cart findCartByEmailAndProductId(@Param("email") String email, @Param("productId") int productId);* *Find Cart base on email, producId*
02	*findCartByEmailAndCartId*	*Cart findCartByEmailAndCartId(@Param("email") String email, @Param("cartId") int cartId);* *Find Cart base on email, cartId*
03	*findAllByEmail*	*Page<Cart> findAllByEmail(@Param("email") String email,PageRequest pageRequest);* *Return a Page Cart of Email parameter (Sort DESC)*
04	*findAllByEmail*	*Page<Cart> findAllByEmail(@Param("email") String email,PageRequest pageRequest);* *Return a Page Cart of Email parameter*

ProductService

No	Method	Description
01	*getPageableProducts*	*getPageableProducts(int pageNo, int pageSize)* *Parameter:* *+ int pageNo, int pageSize. This function will return a page of user list with role Id and page information is PageNo and PageSize*
02	*getProductByRelated*	*Product getProduct(@PathVariable int productId) The implementation of the getProduct() method would typically use the productId parameter to query a database or other data source for details about a specific product.* *+ function that returns a collection or list of products that are related to a particular product or item*

ProductRepository

No	Method	Description
01	*findProductById*	*Product findProductById(@Param("productId") int productId,* *@Param("status") boolean status);* *+ int productId* *The function would then query a database or other data source with the specified product ID*
02	*finfAllByName*	*Page<Product> finfAllByName(@Param("status") boolean status,@Param("search") String search,Pageable pageable);* *+ Pageable pageable . This function will return a page of user list with role Id and page information is PageNo and PageSize*

		+ The function would then query a database or other data source the specified product ID

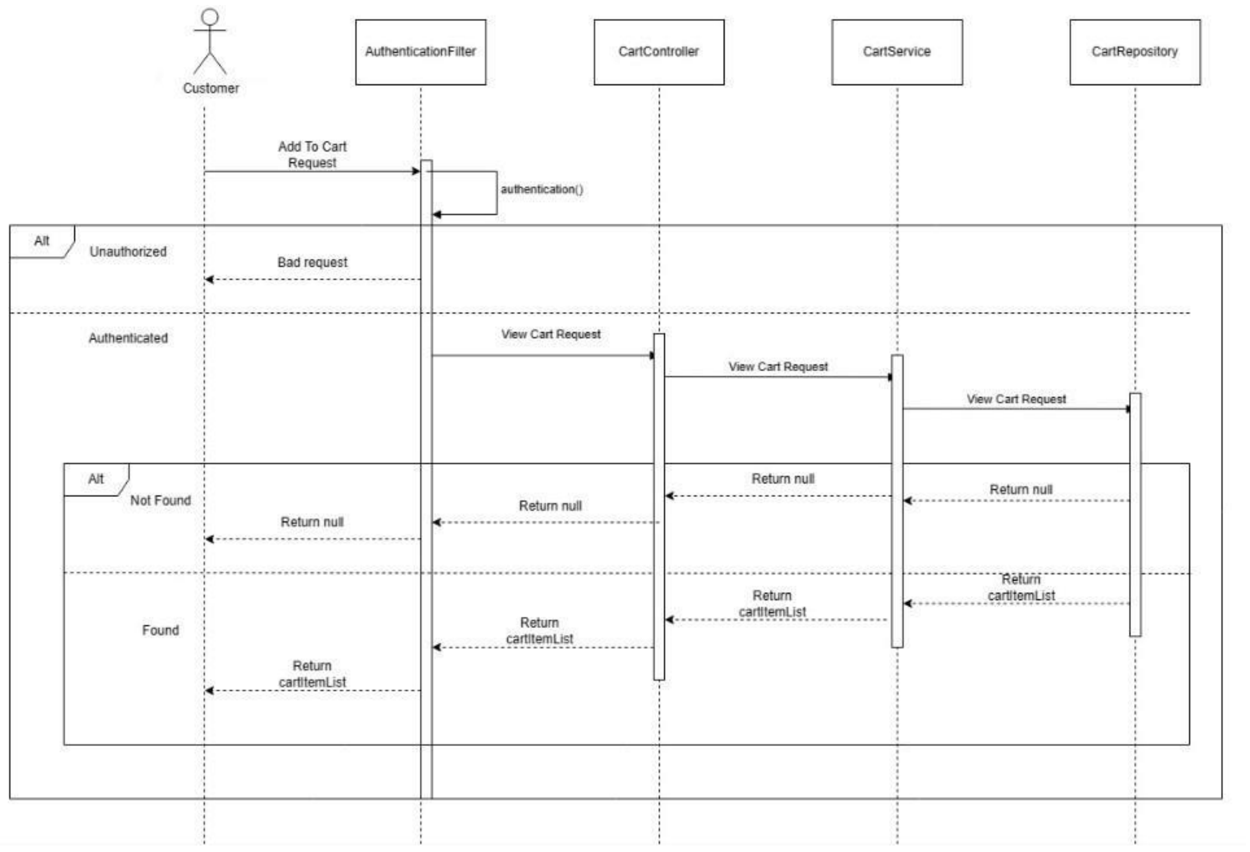
Carts

No	Method	Description
01	*get()*	*Returns the current value of the property*
02	*set()*	Set new value of the property

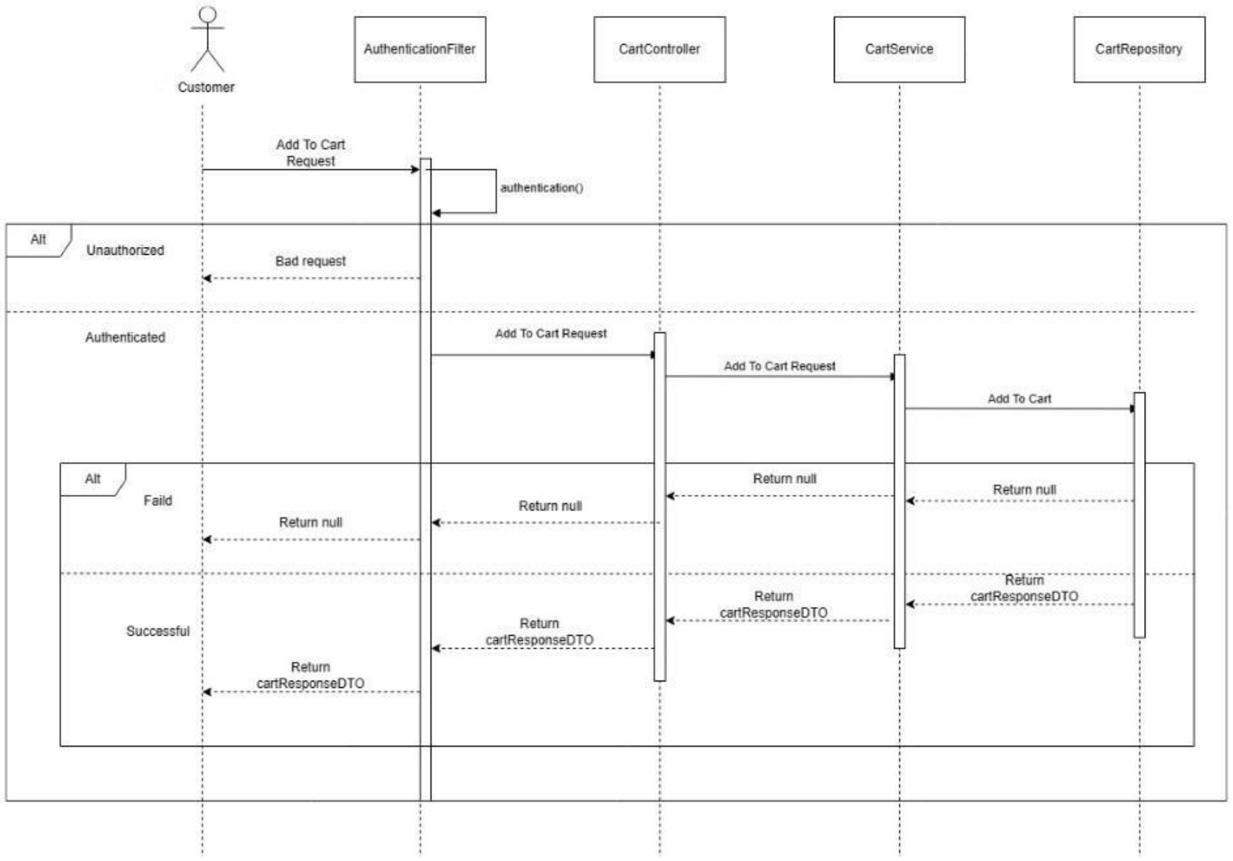
Product

No	Method	Description
01	*get()*	*Returns the current value of the property*
02	*set()*	Set new value of the property

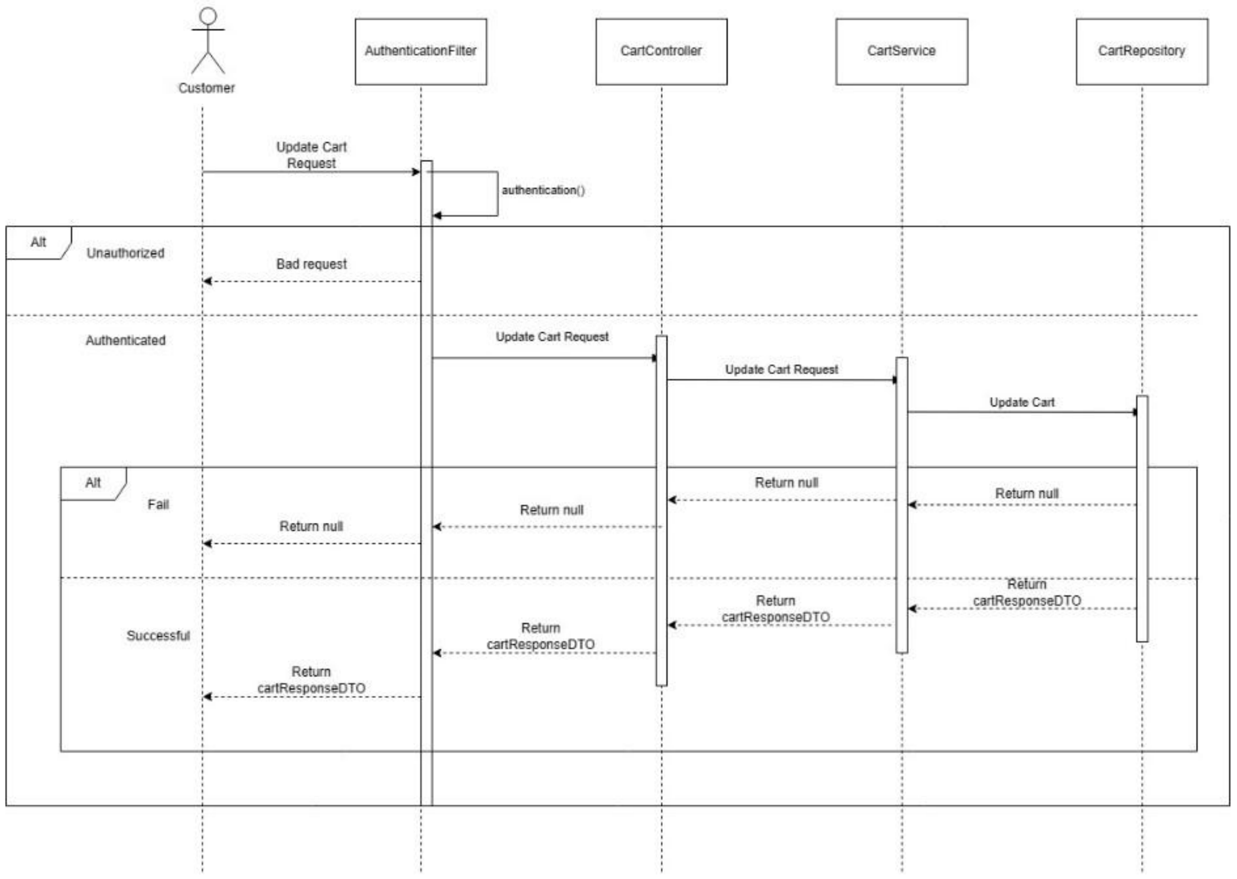
c. Sequence Diagram(s)



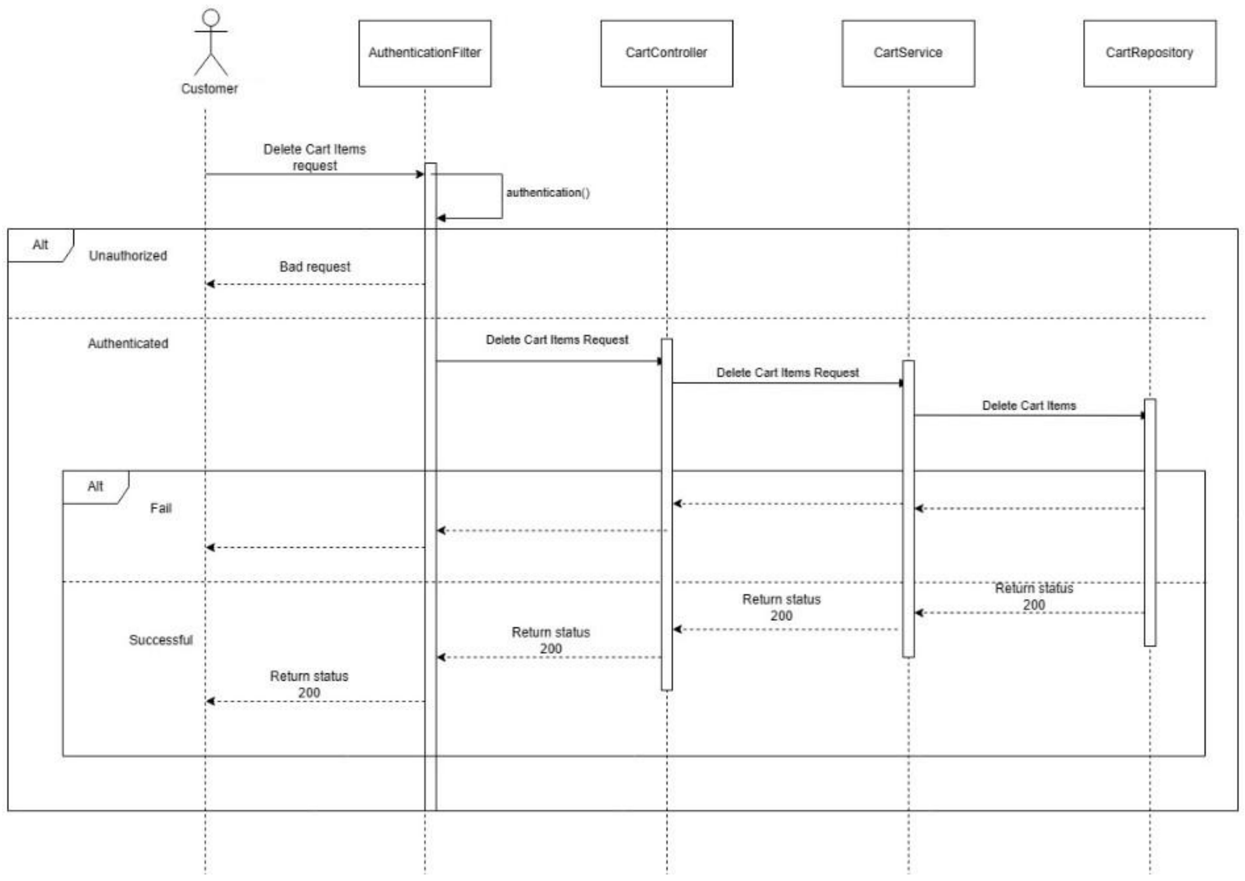
View Cart


Customer
AuthenticationFilter
CartController
CartService
CartRepository
Add To Cart Request
authentication()
Alt
Unauthorized
Bad request
Authenticated
Add To Cart Request
Add To Cart Request
Add To Cart
Alt
Faild
Return null
Return null
Return null
Return null
Successful
Return cartResponseDTO
Return cartResponseDTO
Return cartResponseDTO
Return cartResponseDTO

Add To Cart


Customer
AuthenticationFilter
CartController
CartService
CartRepository
Update Cart Request
authentication()
Alt
Unauthorized
Bad request
Authenticated
Update Cart Request
Update Cart Request
Update Cart
Alt
Fail
Return null
Return null
Return null
Return null
Successful
Return cartResponseDTO
Return cartResponseDTO
Return cartResponseDTO
Return cartResponseDTO

Edit Cart



Delete Cart Item

d. Database queries

FindCartByEmailAndProductId:

SELECT cart FROM Cart cart WHERE cart.email = :email AND cart.product.id = :productId")

FindCartByEmailAndCartId:

SELECT c FROM Cart c WHERE c.email = :email AND c.id = :cartId

FindAllByEmail:

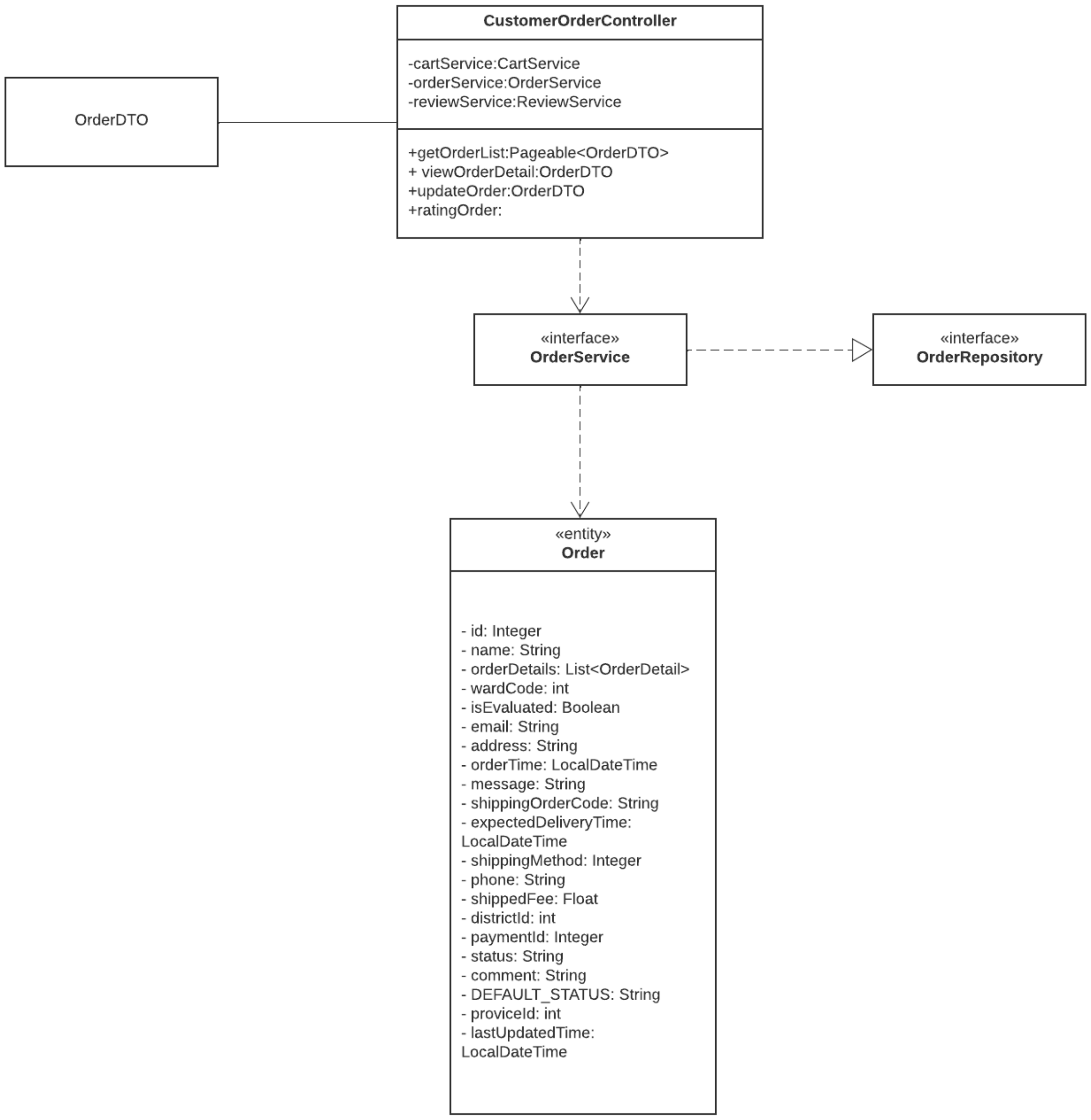
SELECT c FROM Cart c WHERE c.email = :email ORDER BY c.lastTimeUpdate DESC

FindAllByEmail:

SELECT c FROM Cart c WHERE c.email = :email

6. <Customer Manage Order/Checkout/View orders placed/View detail of each order>

a. Class Diagram



b. Class Specifications

CustomerOrderController

No	Method	Description
01	*getOrderList*	*Parameter:*

		+ @RequestParam Integer pageNo, @RequestParam Integer pageSize, @RequestParam String status . This function will return a page of order list with pageable
02	*viewOrderDetail*	*Parameter: @PathVariable int orderId. This function will return OrderDTO to view the order detail of a order with orderId*

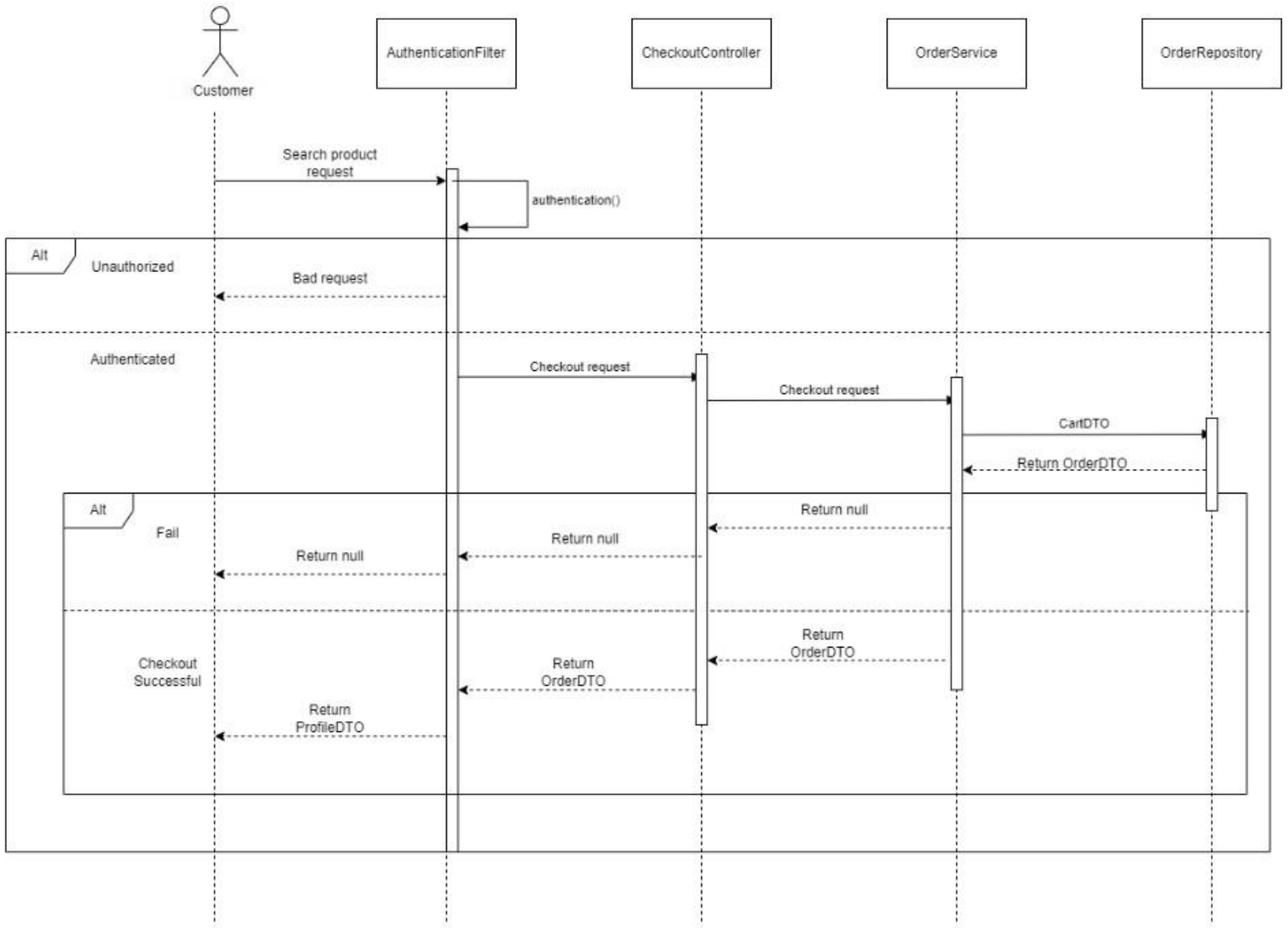
OrderService

No	Method	Description
01	*getOrderByOrderId*	*Parameter:Integer orderId. This function will return a order with orderId*
02	*getOrderList*	*Parameter: Integer pageNo, Integer pageSize, String email. This function will return the page of orders with pageable*

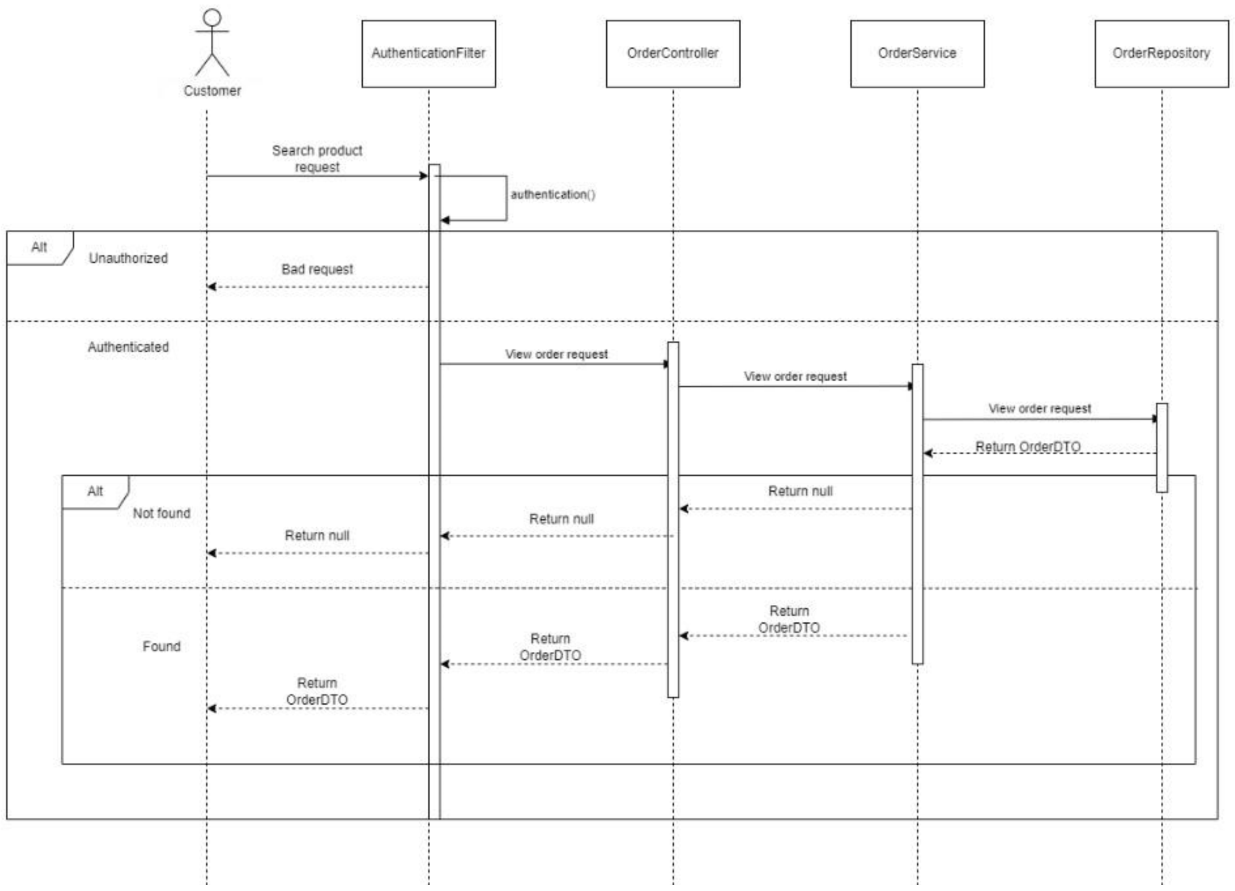
OrderRepository

No	Method	Description
01	*findOrderByOrderId*	*Parameter:@Param Integer orderId. This function will return a order with orderId*
02	*getOrderedWithStatu s*	*Parameter: String status, PageRequest pageRequest. This function will return the page of orders with pageable*

c. Sequence Diagram(s)



Check out diagram



View orders

d. Database queries

Get order by order id

SELECT o FROM Orders o WHERE o.id = :orderId

Get order list by with status

SELECT p FROM Orders p WHERE p.status like %:status%

FindAllByEmail

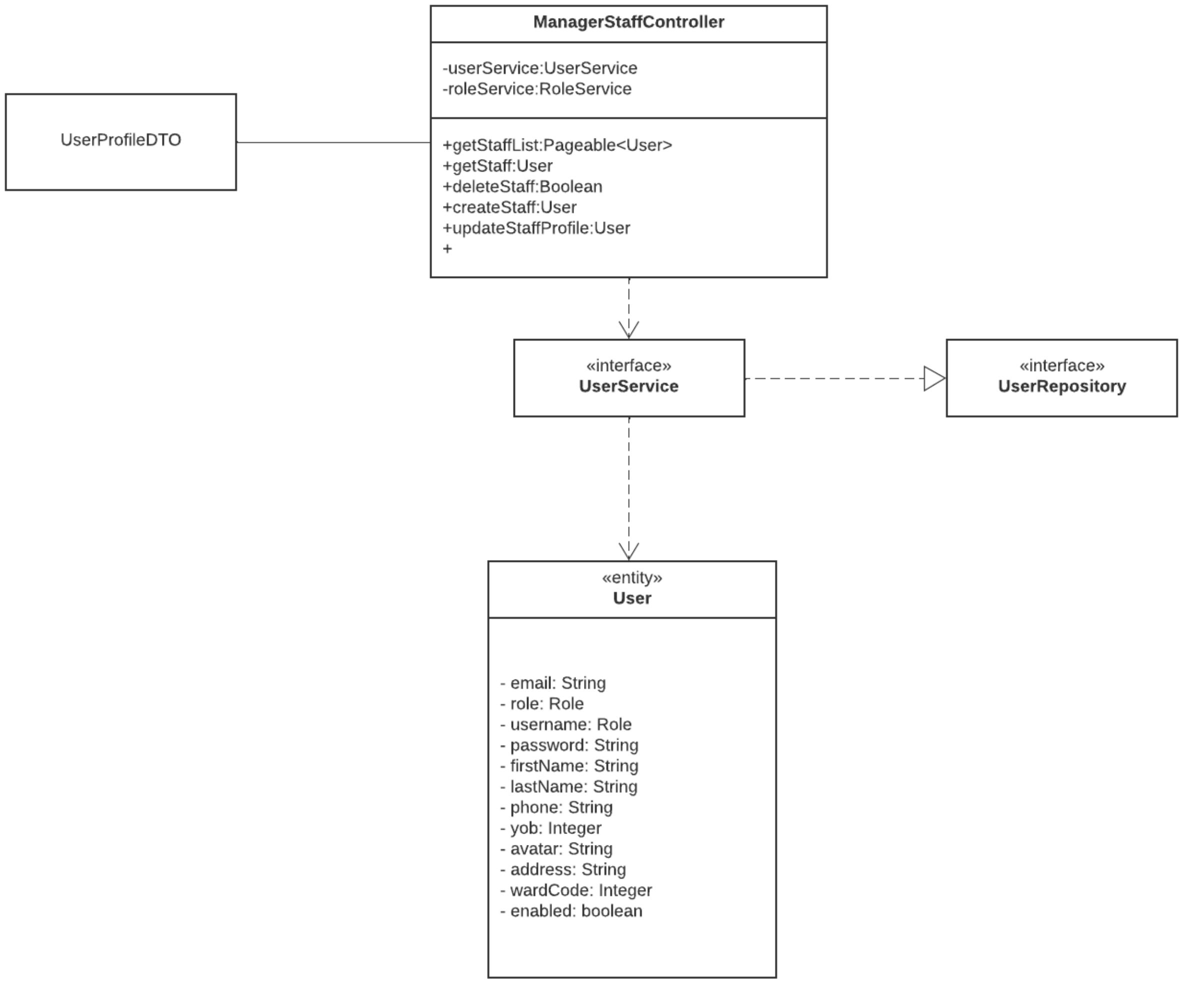
SELECT o FROM Orders o WHERE o.email = :email ORDER BY o.orderTime DESC

FindAllWithStatus

SELECT o FROM Orders o WHERE o.status in :statuses ORDER BY o.orderTime DESC

7. <Manage staff/View staff list/View detail staff profile/Edit staff profile/Add new staff/ Delete staff>

a. Class Diagram



b. Class Specifications

MangerStaffController

No	Method	Description
01	*getStaffList*	*APIPageableResponseDTO<User> getStaffList()* *Parameter:* *+ int pageNo, int pageSize . This function will return a page of user list with role Id and page information is PageNo and PageSize* *+ String search.* *Function would retrieve a list of Staff*

02	*getStaff*	*User getStaff(@PathVariable String check)* *Parameter: @PathVariable String check. This function will return a User staff with the parameter String check*
03	*deleteCart*	*boolean deleteStaff(@PathVariable String check)* *Parameter: @PathVariable String check. This function will delete a User staff with the parameter String check*
04	*createStaff*	*User createStaff(@RequestBody User user)* *The user object contains information about the items to be added to the cart* *The purpose of this method is to add the user specified in the role staff.*
05	*updateStaffProfile*	*User updateStaffProfile(@PathVariable String check, @RequestBody UserProfileDTO userProfile)* *The userProfile object contains information about the information needs to be edited* *The purpose of this method is to edit the information of the user's.*

UserService

No	**Method**	**Description**
01	*getPageableUsers*	*Parameter: int pageNo, int pageSize, int roleId . This function will return a page of user list with role Id and page information is PageNo and PageSize*
02	*getUserByEmail*	*Parameter: String email. This function will return a user by email*
03	*getPageableUsers*	*Parameter: int pageNo, int pageSize, int roleId, and boolean enabled . This function will return a page of user list with role Id and enabled status and page information is PageNo and PageSize*
04	*checkExistUser*	*Parameter: String usernameOrEmail. Return boolean, check that user is exist or not: TRUE or FALSE*
05	*getUserByEmailOrUsername*	*Parameter: String usernameOrEmail and boolean enabled. Return user by username or email and their enabled*
06	*setEnabledUserByEmail*	*Parameter: String email and UserProfileDTO. Return user after update their profile*
07	*updateUserProfile*	*Parameter: Pageable pageable, int roleId, boolean enabled, and String search. Find the user list by role id with an enabled status: TRUE or FALSE with search element.*
08	*createUser*	*Parameter: User user. Return user after that created*
09	*searchUsers*	*Parameter: int pageNo, int pageSize, int roleId, boolean enabled and String search. This function will return a page of user list with role Id and enabled status and page information is PageNo and PageSize with a search element*

UserRepository

No	Method	Description
01	*getUserByUsername*	*User getUserByUsername(@Param("username") String username);* *Return user base on username*
02	*updateUserDTO*	*Parameter: UserProfileDTO u. This function was used to update the user in database with the parameter profile*
03	*getUsersByRoleId*	*Page<User> getUsersByRoleId(Pageable pageable, @Param("roleId") Integer role_id);* *Return a list of user have role_id*
04	*filterUsersByRoleId*	*Page<User> filterUsersByRoleId(Pageable pageable, @Param("roleId") int roleId,* *@Param("enabled") boolean enabled,* *@Param("search") String search);* *Return a list of user as Page and have search condition*

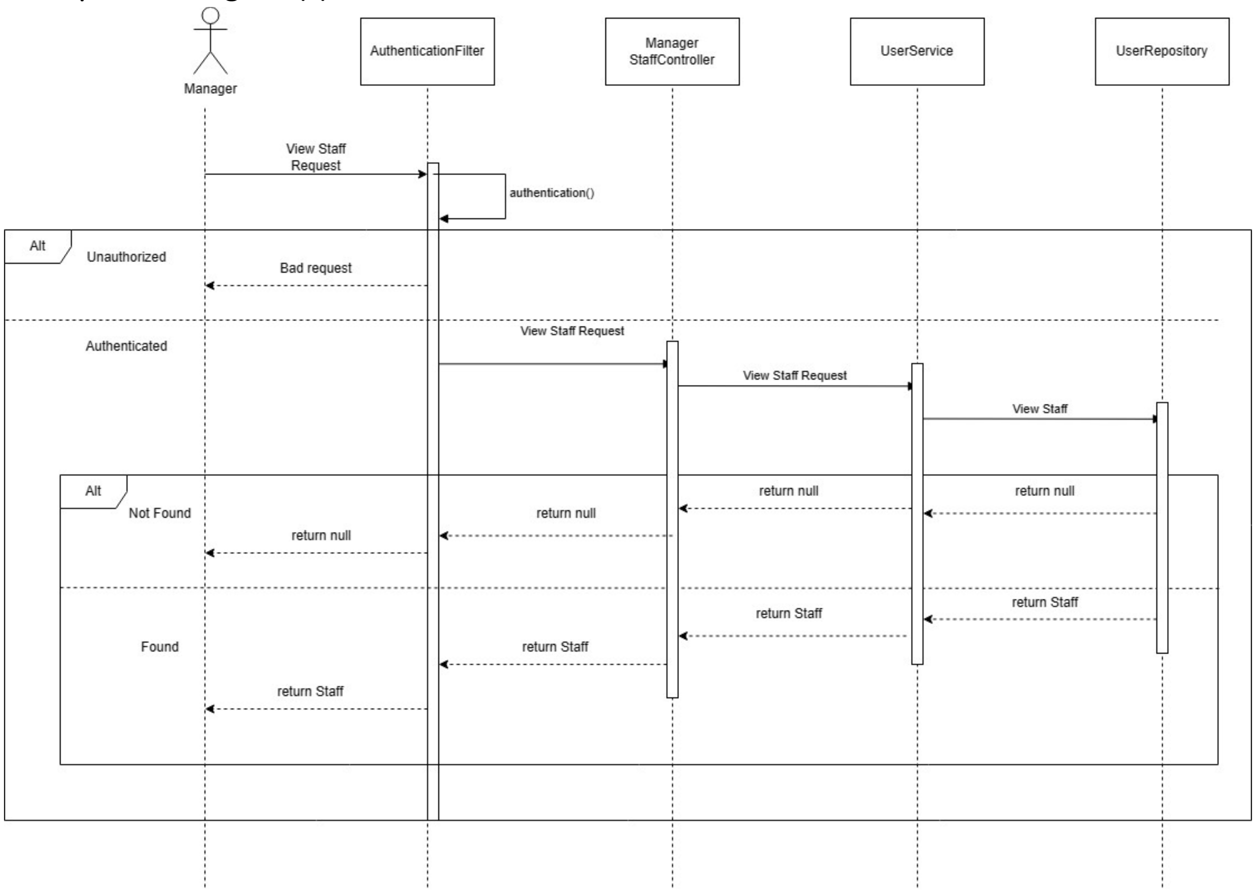
RoleService

No	Method	Description
01	*getUserByUsername*	*User getUserByUsername(@Param("username") String username);* *Return user base on username*
02	*updateUserDTO*	*Parameter: UserProfileDTO u. This function was used to update the user in database with the parameter profile*
03	*getUsersByRoleId*	*Page<User> getUsersByRoleId(Pageable pageable, @Param("roleId") Integer role_id);* *Return a list of user have role_id*
04	*filterUsersByRoleId*	*Page<User> filterUsersByRoleId(Pageable pageable, @Param("roleId") int roleId,* *@Param("enabled") boolean enabled,* *@Param("search") String search);* *Return a list of user as Page and have search condition*

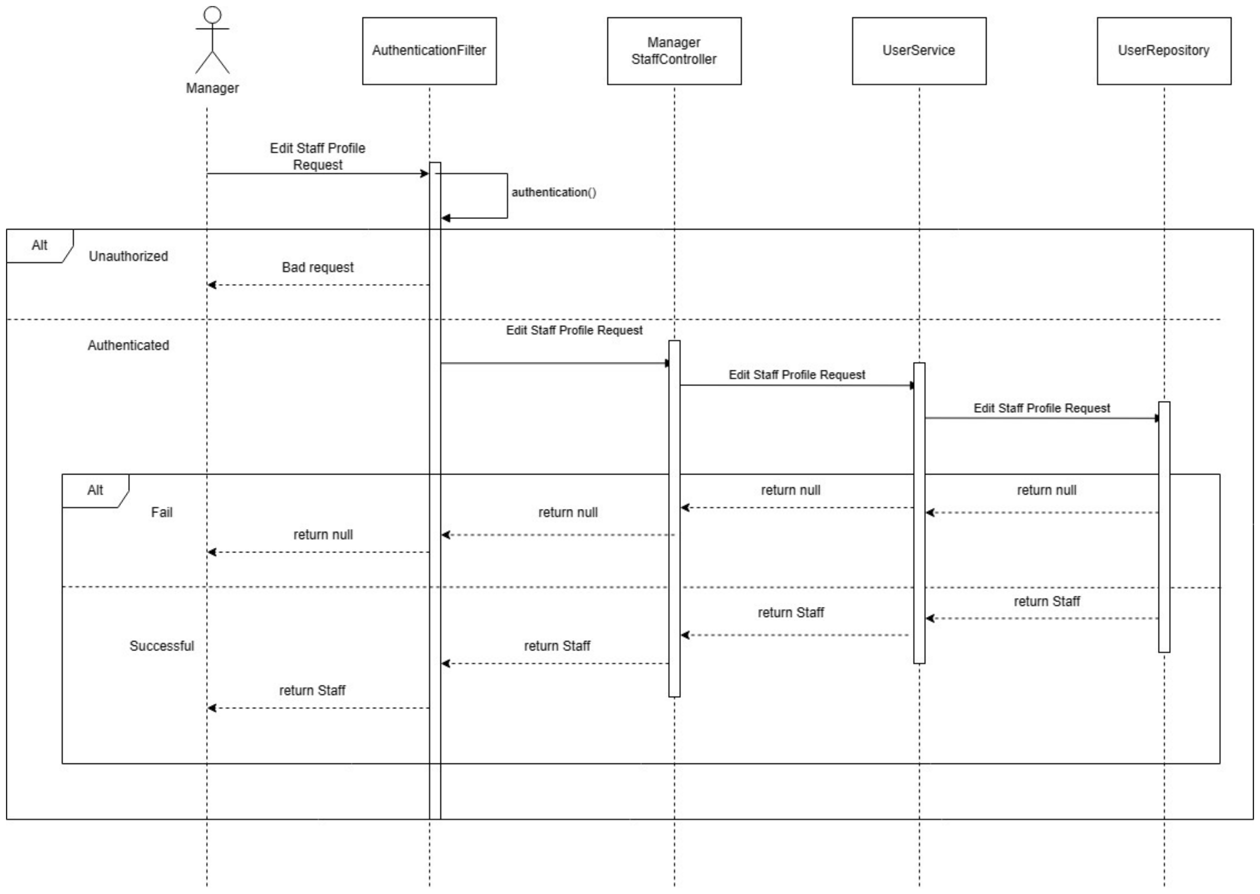
RoleRepository

No	Method	Description
01	*getRoleByRoleName*	*Role getRoleByRoleName(@Param("roleName") String roleName);* *Return user base on roleName*
02	*getRoleById*	*Role getRoleById(@Param("id") Integer id);* *Return Role with id*

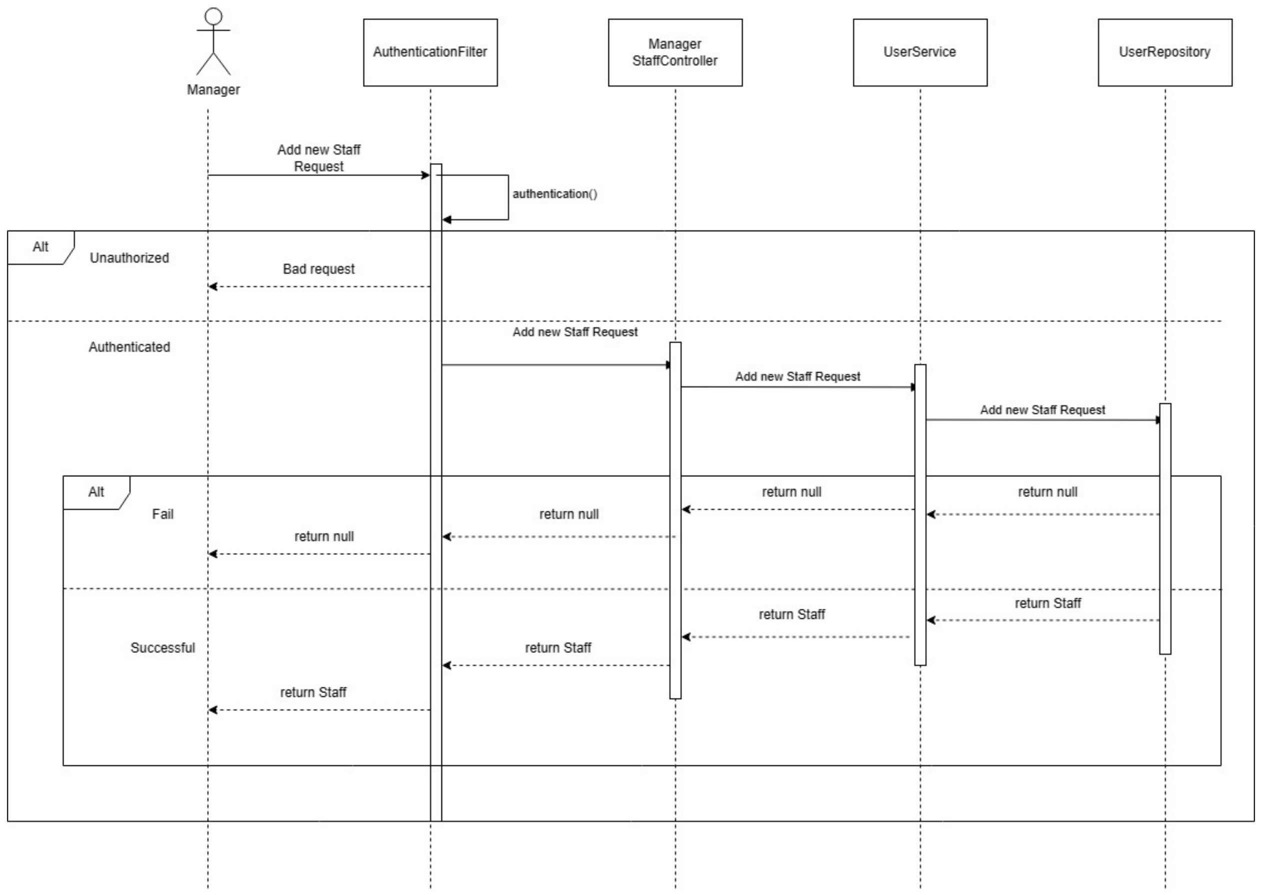
c. Sequence Diagram(s)



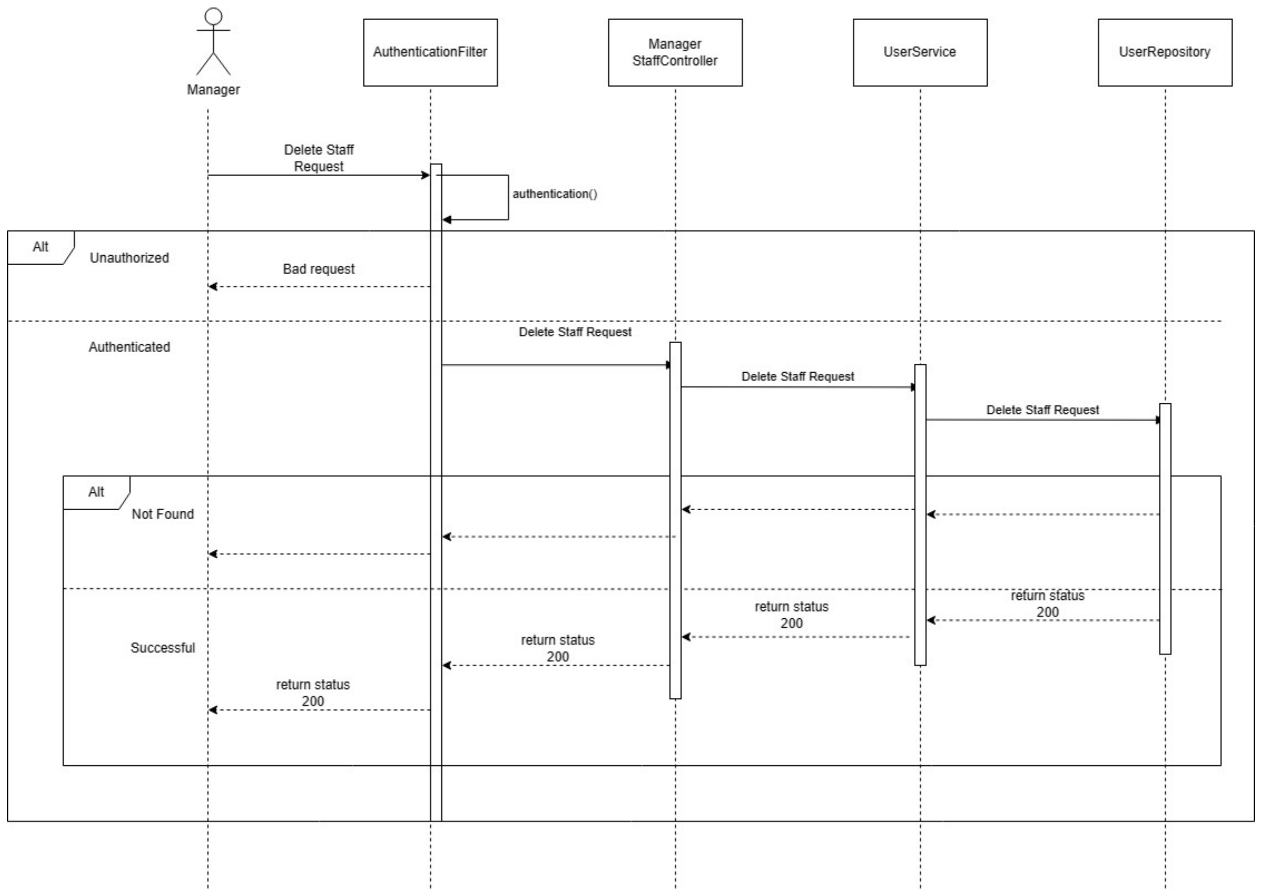
View Staff List / View Staff



Edit Staff Profile



Add New Staff



Delete Staff

d. Database queries

Get user by username

SELECT u FROM User u WHERE u.username = :username

Get user by email

SELECT u FROM User u WHERE u.email = :email

Get user by roleId

SELECT u FROM User u WHERE u.role.id = :roleId

Get user with enabled status by username or email

SELECT u FROM User u WHERE u.enabled = :enabled and (u.email = :check OR u.username = :check)

Get users by role id and enabled

SELECT u FROM User u WHERE u.role.id = :roleId AND u.enabled = :enabled

Update user enabled by username or email

UPDATE User u SET u.enabled = :enabled WHERE

u.email = :emailOrUsername OR u.username = :emailOrUsername)

Filter user by firstname and lastname by role id and enabled

SELECT u FROM User u

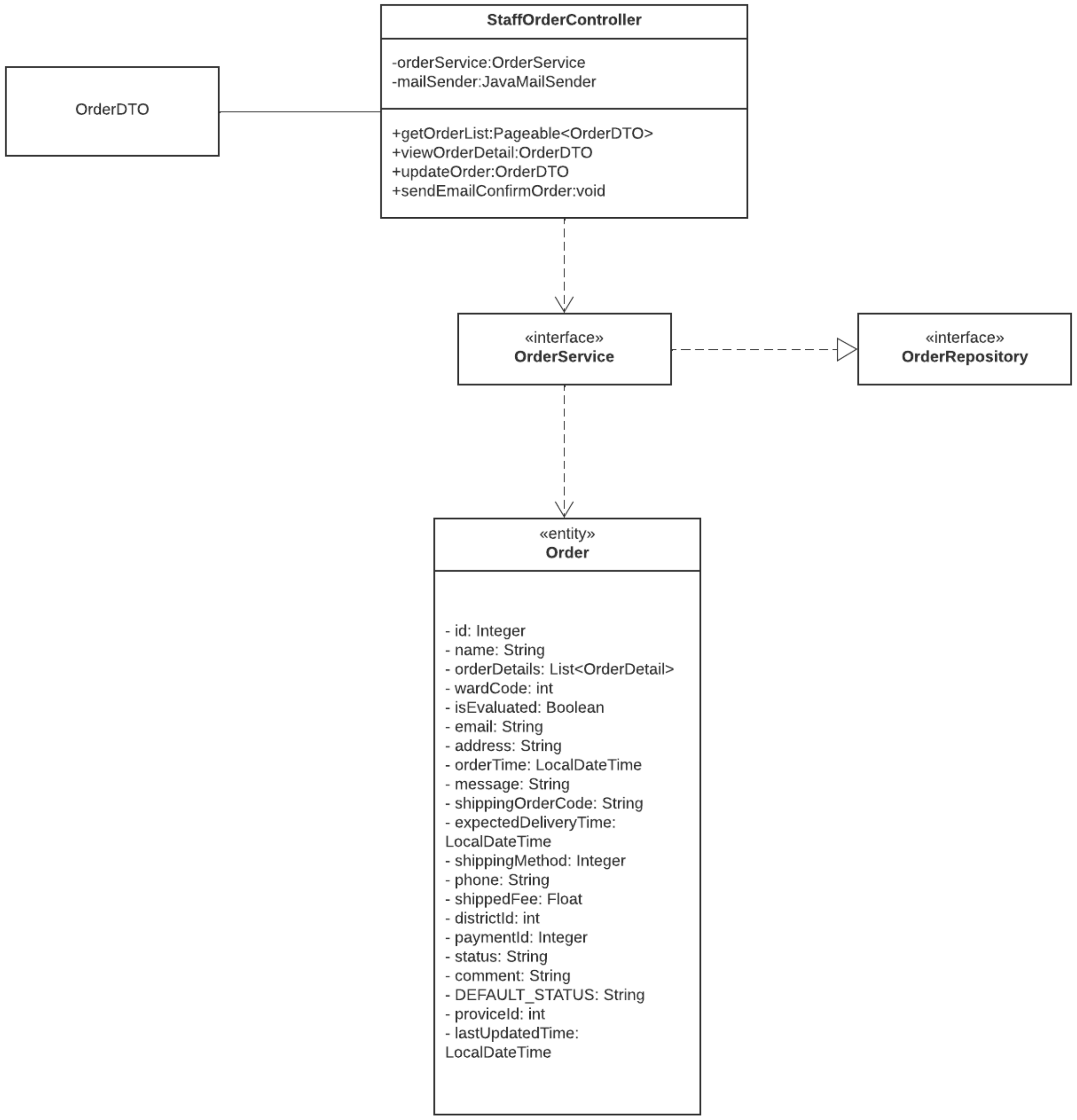
WHERE u.role.id = :roleId

AND u.enabled = :enabled AND (u.username LIKE %:search%

OR u.email LIKE %:search% OR (u.firstName || ' ' || u.lastName LIKE %:search%))".

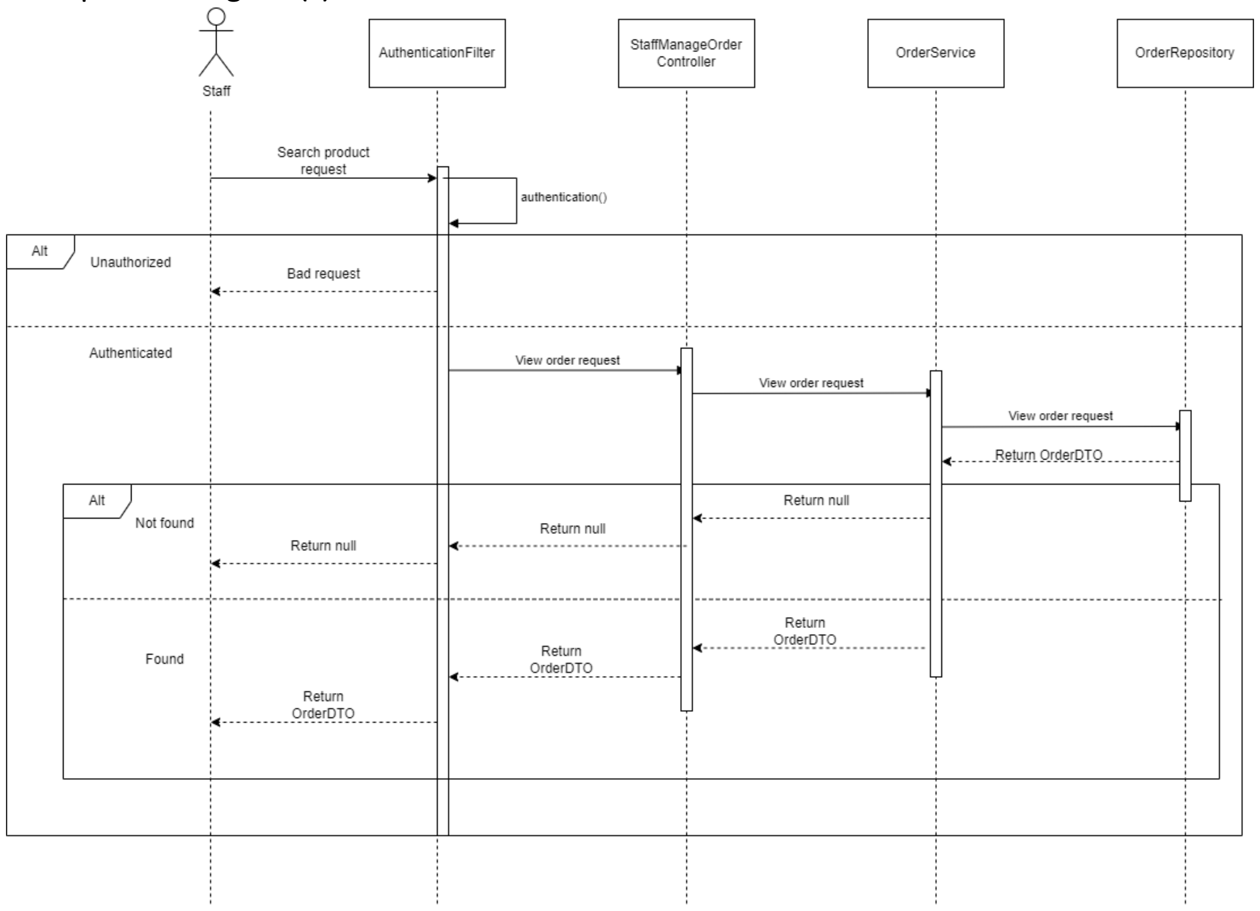
8. <Staff Manage order/View orders/View detail of each order/Refuse/Confirm orders>

a. Class Diagram

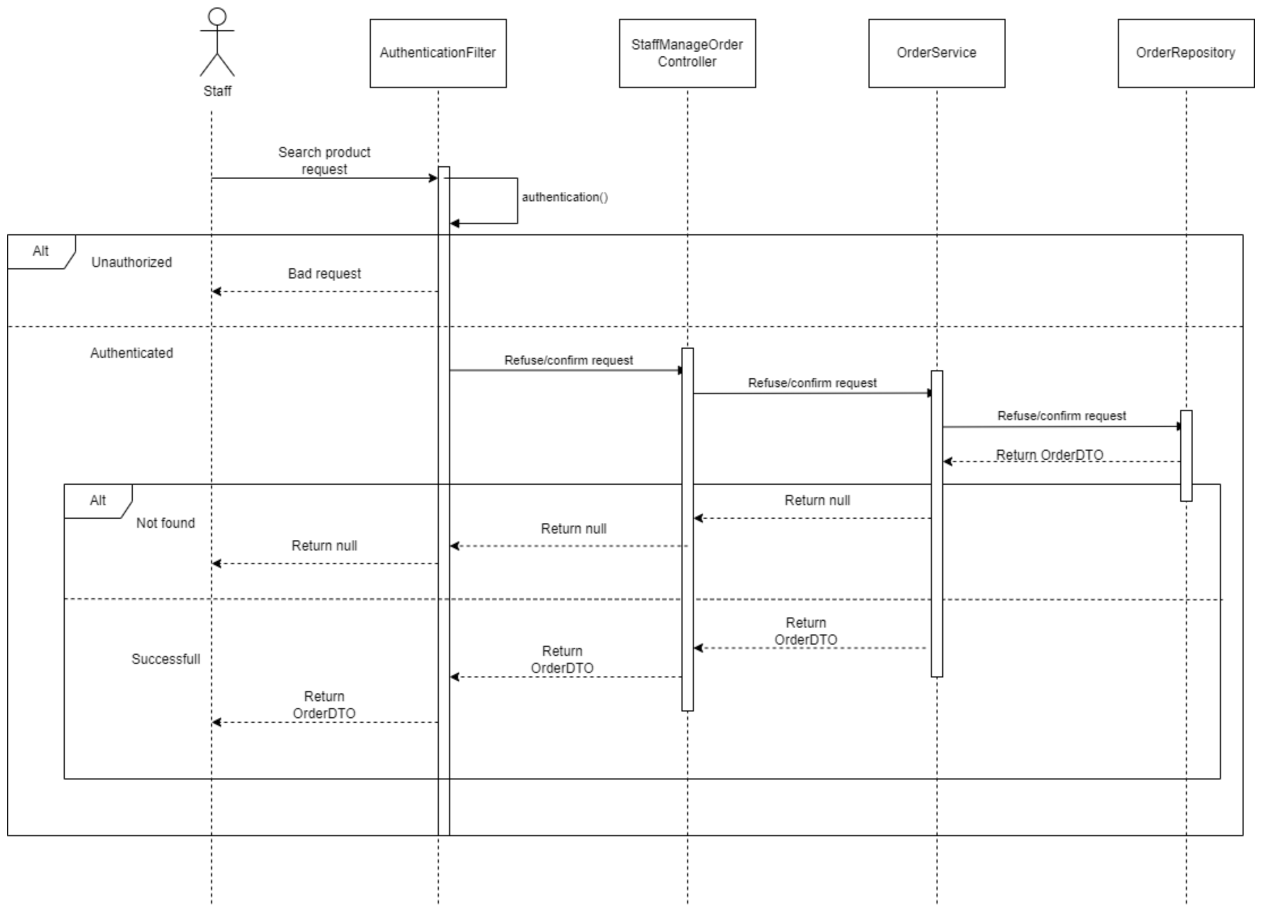


b. Class Specifications

c. Sequence Diagram(s)



View order diagram



Refuse/Confirm order

d. Database queries

FindOrderByOrderId

SELECT o FROM Orders o WHERE o.id = :orderId

FindOrdersWithoutStatus

SELECT o FROM Orders o WHERE (o.status NOT IN :status)

FindOrdersByProductIdWithoutStatus

SELECT o FROM Orders o WHERE o.status NOT IN :status AND :productId IN (SELECT od.productId FROM o.orderDetails od)

FindAllByEmail

SELECT o FROM Orders o WHERE o.email = :email ORDER BY o.orderTime DESC

FindAllByEmailWithStatus

SELECT o FROM Orders o WHERE o.email = :email AND o.status in :statuses ORDER BY o.orderTime DESC

SetStatusOfOrderByOrderId

UPDATE Orders o SET o.status = :status WHERE o.id = :orderId

GetOrderedWithStatus

SELECT p FROM Orders p WHERE p.status like %:status%

FindSuccessfulOrdersFromTo

SELECT o FROM Orders o WHERE o.status = 'SUCCESSFUL' AND :startDate <= o.orderTime AND o.orderTime <= :endDate

FindAllByEmail

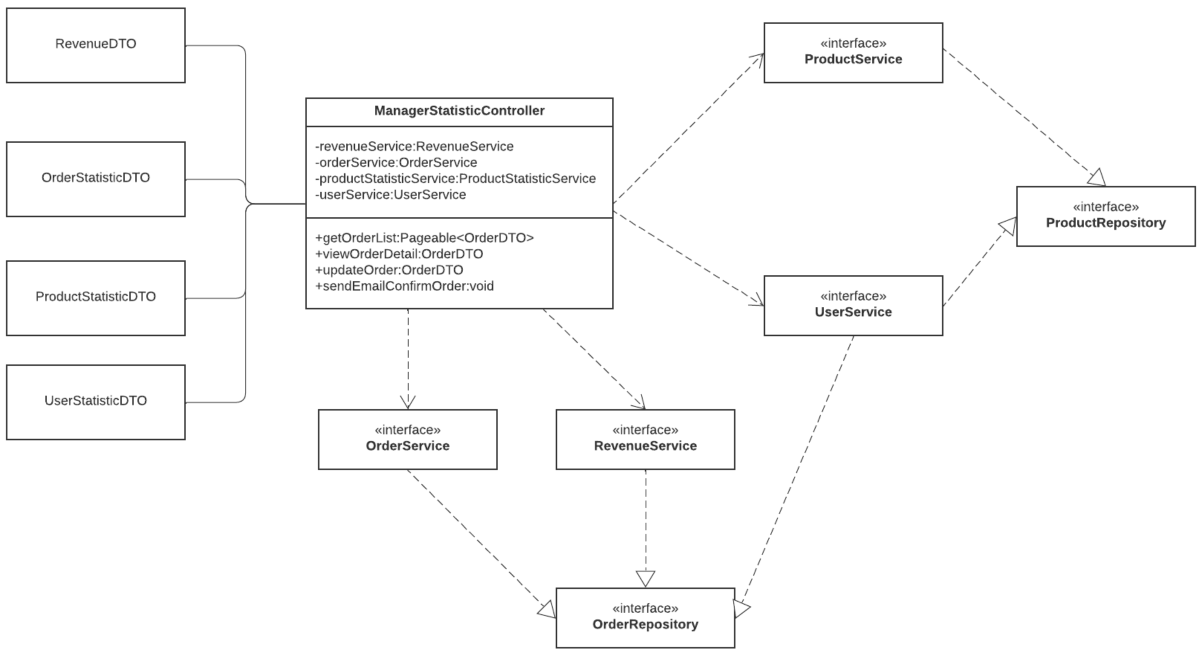
SELECT o FROM Orders o WHERE o.email = :email

FindOrderByDay

SELECT o FROM Orders o WHERE :startDate <= o.orderTime AND o.orderTime <= :endDate

9. <Statistic>

a. Class Diagram


RevenueDTO
OrderStatisticDTO
ProductStatisticDTO
UserStatisticDTO
ManagerStatisticController
-revenueService:RevenueService
-orderService:OrderService
-productStatisticService:ProductStatisticService
-userService:UserService
+getOrderList:Pageable<OrderDTO>
+viewOrderDetail:OrderDTO
+updateOrder:OrderDTO
+sendEmailConfirmOrder:void
«interface» ProductService
«interface» ProductRepository
«interface» UserService
«interface» OrderService
«interface» RevenueService
«interface» OrderRepository

b. Class Specifications

ManagerStatisticController

No	Method	Description
01	*getRevenue*	*Parameter:@RequestParam String date. This function will return the RevenueDTO Frontend in date*
02	*getOrderStatisticsDTO*	*Parameter:@RequestParam String date. This function will return the OrderStatisticDTO Frontend in date*
03	*getProductStatisticsDTO*	*Parameter:@RequestParam String date. This function will return the ProductStatisticDTO Frontend in date*
04	*getUserStatisticDTO*	*Parameter:@RequestParam String date. This function will return the UserStatisticDTO Frontend in date*

OrderService

No	Method	Description
01	*getOrderStatistic*	*Parameter:String date. This function will return a OrderStatisticDTO is that OrderStatistic of this date*

RevengeService

No	Method	Description
01	*getRevenue*	*Parameter:String date. This function will return a RevenueDTO of that statistic in date*

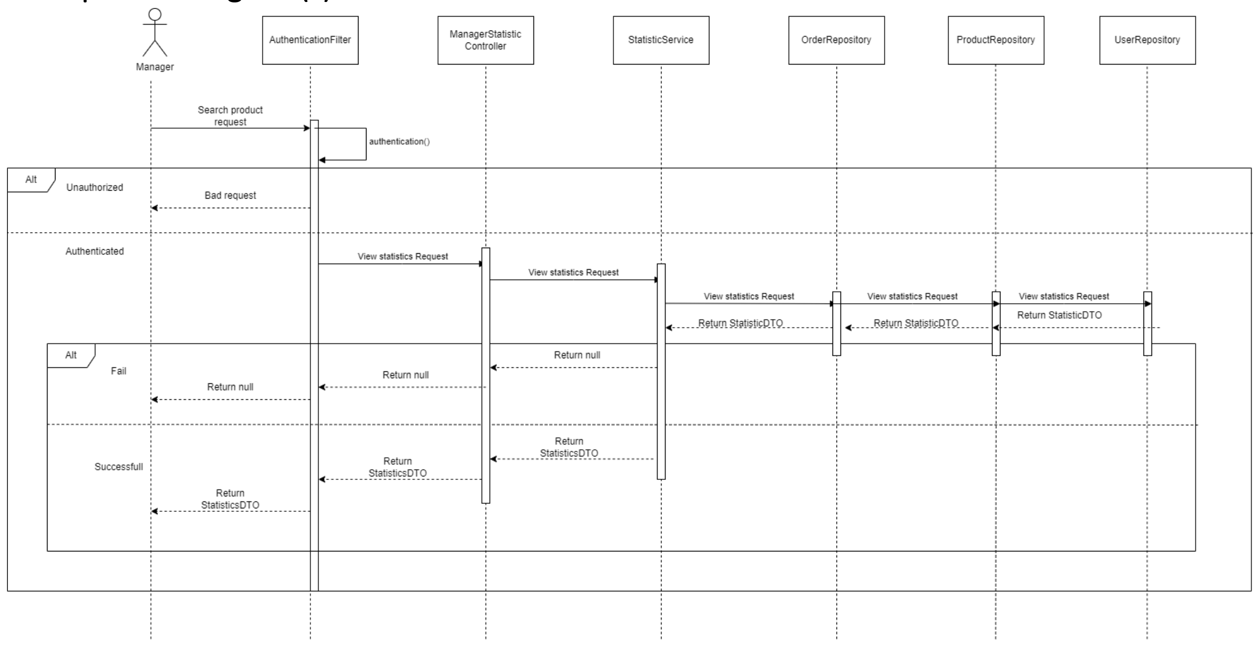
UserService

No	Method	Description
01	*getUserStatistic*	*Parameter:String date. This function will return a UserStatistic of that statistic in date*

ProductStatisticService

No	Method	Description
01	*getProductStatistic*	*Parameter:String date. This function will return a ProductStatisticDTO of that statistic in date*

c. Sequence Diagram(s)



d. Database queries

getRevenueFromTo

SELECT SUM(od.price * od.quantity) FROM Orders o JOIN o.orderDetails od

WHERE o.status = 'SUCCESSFUL'

AND :startDate <= o.orderTime

AND o.orderTime <= :endDate

findTopSoldProductFromTo

SELECT od.productId FROM Orders to JOIN

o.orderDetails od WHERE

o.status = 'SUCCESSFUL'

AND :startDate <= o.orderTime

AND o.orderTime <= :endDate

GROUP BY od.productId

ORDER BY SUM(od.quantity) DESC

findTopRatingProductFromTo

SELECT od.productId FROM Orders o JOIN

o.orderDetails od WHERE

```
                              o.status = 'SUCCESSFUL'
                              AND :startDate <= o.orderTime
                              AND o.orderTime <= :endDate
                              GROUP BY od.productId
                              ORDER BY AVG(od.rating) DESC
```

findAllByOrderCountDesc

```
SELECT u.* FROM user u
               LEFT JOIN (SELECT o.email, COUNT(*) AS order_count FROM orders o GROUP BY o.email)
o
               ON u.email = o.email
               WHERE o.order_count > 0
               ORDER BY o.order_count DESC
```

findAllByProductCountDesc

```
SELECT u.* FROM user u
                              LEFT JOIN ( SELECT o.email, COUNT(od.id) AS total_products FROM orders o
                              LEFT JOIN order_detail od ON o.id = od.order_id where o.status='SUCCESSFUL'
GROUP BY o.email) t
                               ON u.email = t.email
                               WHERE total_products > 0
                              ORDER BY t.total_products DESC
```

findAllOrderByTotalSpentDesc

```
SELECT u.* "
                              FROM user u
                              JOIN orders o ON u.email = o.email
                              JOIN order_detail od ON o.id = od.order_id
                              WHERE o.status = 'SUCCESSFUL'
                              GROUP BY  u.email, u.username, u.address
                              ORDER BY SUM(od.price * od.quantity)  DESC
```

III. Database Tables

1. <Order>

Order tables are commonly used in retail or e-commerce systems to store information about customer orders. The table will contain fields like the id , email of the customer and info date and order address of the order ()

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	id	INT	4		X	PK	Unique identifier for each order
2	email	VARCHAR	320		X	FK	Reference user(email) Email address of the customer who placed the order.
3	payment_id	INT	4			FK	Unique identifier for the payment Reference user(payment_method)
4	shipping_order_code	VARCHAR	45				This column stores a unique identifier for each shipping order.
5	shipping_method	INT	4				This column stores the method used to ship the order, such as air mail, ground shipping or express delivery.
6	shipping_fee	FLOAT	64				This column stores the amount charged for shipping the order.
7	expected_delivery_time	DATETIME	20				This column stores the estimated date and time when the order will be delivered.
8	shipped_date	DATETIME	400				This column stores the actual date and time when the order was shipped.
9	name	VARCHAR	64				This column stores the name of the recipient who placed the order
10	phone	VARCHAR	20				This column stores the phone number of the recipient
11	address	VARCHAR	4				This column stores the street address where the order should be delivered.
12	ward_code	VARCHAR	4				This column stores a code that identifies the ward or sub-district of the delivery address.
13	district_id	DATETIME					This column stores an identifier for the district or city of the delivery address.
14	province_id	INT	4				This column stores an identifier for the province or state of the delivery address.
15	status	INT	4				This column stores the current status of the order.

16	status_comment	VARCHAR	20				This column stores any comments or notes related to the status of the order.
17	last_updated_time	DATETIME					This column stores the date and time when the status of the order was last updated.
18	is_evaluated	TINYNT	4				This column stores a flag indicating whether the customer has provided feedback or evaluation of the order.

2. <Order Detail>

The "Order Detail" table is a relational database table that stores information about each product ordered in an order. This table usually has a foreign key constraint that links it to the "Order" table and the "Product" table.

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	id	INT	4		X	PK	The unique identifier for each item in the OrderDetail
2	order_id	INT	4			FK	Reference Order(id) The unique identifier for the order
3	product_id	INT	4			FK	Reference product(id) The unique identifier for the product
4	price	FLOAT	4				The price of the item
5	quantity	INT	4				The number of units of the product
6	feedback	VARCHAER	400				Teedback provided by customers or users.
7	rating	FLOAT	4				The rating given by customers or users
8	feedback_time	DATETIME					This column stores the date and time when the feedback was provided.

3. <payment_method>

The payment_method table is a database table that stores information about different payment methods that can be used to make payments in a system

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	id	INT	4		X	PK	Unique identifier for each payment method
2	name	INT	120				The name of the payment method

4. <cart>

The "cart" table is a crucial component of a database designed to handle the shopping cart functionality in an e-commerce or web-based application. It is responsible for storing information about the items selected by users for purchase before they proceed to checkout.

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	id	INT	4		X	PK	identity column
2	email	VARCHAR	320		X	PK	reference user(email) Cart of each user.
3	product_id	INT	4		X	FK	reference product(id) Product of each cart.
4	quantity	INT	4		X		Quantity of each product in cart.
5	last_time_update	DATETIME					Last time update by user.

5. <category>

The “category” table serves as a foundational structure for organizing and categorizing products, facilitating efficient search, navigation, and management of product data.

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	id	INT	4		X	PK	-reference product(category_id) identity column Product category for filter,...
2	name	VARCHAR	45	X	X		Name of each category.

6. <user>

The “user” table stores information about the users of the system, allowing for authentication, authorization, and personalized functionality

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	email	VARCHAR	320		X	PK	identity column -reference cart(email) Each cart of user

							-reference order(email) Order of user -reference review(email) All review of user
2	role_id	INT	4			FK	reference role(id) identity each role of user
3	username	VARCHAR	45	X	X		The username chosen by the user
4	password	VARCHAR	80		X		The encrypted password for user authentication
5	first_name	VARCHAR	50		X		The first name of the user.
6	last_name	VARCHAR	50				The last name of the user.
7	phone	VARCHAR	20				Phone number of user.
8	yob	INT	4				Year of birth of user.
9	avatar	VARCHAR	1000				The avatar was chosen by the user.
10	address	VARCHAR	400				Address of user.
11	ward_id	INT	4				Ward of the user.
12	enabled	TINYINT	4				Status of user in website(ban/unbanned)
13	reset_password_token	VARCHAR	45				Token that is generated when a user requests a password reset
14	expired_vertification_code	DATETIME					An expiration timestamp for a verification code that was sent to a user for some purpose

7. <role>

The "role" table is a common component of a database designed to manage user roles and permissions in a web application. It allows for the definition and assignment of different roles to users, controlling their access to certain features and resources.

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	id	INT	4		X	PK	reference user(email) identity role of each user
2	name	VARCHAR	45	X	X		The name or title of the role

8. <product>

The table 'product' is a database table that typically stores information related to various products within a system or organization. The 'product' table typically contains several columns to store specific details about each product. The 'product' table typically contains several columns to store specific details about each product.

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	id	INT	4		X	PK	What identify of each product

2	name	VARCHAR	200		X		The name of the product
3	description	VARCHAR	400				The description of the product
4	quantity	INT	4		X		The quantity of the product
5	price	FLOAT			X		The price of the product
6	category_id	INT	4			FK	Reference category(id)
7	last_updated_by	VARCHAR	32		X		The last time that the product was edited
8	status	TINYINT	1		X		The status of the product
9	avatar	VARCHAR	1000				The image of the product

9. <product_image>

The table "product_image" typically stores information related to images associated with products.

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	id	INT	4		X	PK	What identify of each product image
2	url	VARCHAR	1000		X		URL of image of product
3	product_id	INT	4			FK	Reference product(id)
4	is_main	TINYINT	1				This image is avatar or not