FPT UNIVERSITY

Jewelry Management System

In Store

Software Requirement

Specification

1. Project Overview

1.1/ Project information

- **Project Name:** Jewelry Sale System at Store
- **Software Type:** Web Application

1.2/ Project Team

- **Team member**

Name	Role	Student ID
Hà Thái Hoàng	Full- Stack, Tester	SE181747
Lâm Tấn Phú	Back-end, Tester	SE182607
Tô Triều Vỹ	Front-end, Tester	SE183178
Huỳnh Nguyễn Thái Đăng	Back-end, Tester	SE182714

1.3/ Project Background

- The jewelry industry is highly valued, demanding a management system to streamline operations and enhance customer enjoyment. For that reason, we seek to create a robust web application to manage sales, promotions, inventory, staff workflow and customer interactions effectively
- This system support for the below type of user:
  - Staff: The sale members and cashiers at the register
  - Administrator: the organization leader, acts as the system administrator
  - Manager: The sale manager in the organization

1.4/ Project Scope and Limitations

Scope:

- **Order Management:** Create and manage sales orders at the counter, support barcode scanning, and manual product code entry.
- **Pricing Calculation:** Automate pricing based on product cost factors, adjustable by managers or administrators.
- **Promotions Management:** Handle discount codes and special offers, with managerial approval required for customer-specific discounts.
- **Buy-Back Transactions:** Manage the process of buying back jewelry from customers, including finding past sales receipts and applying buy-back pricing policies.
- **Inventory Management:** Track and update inventory, display real-time gold prices, and categorize products.
- **Customer Loyalty Program:** Accumulate points for customers and display their loyalty points.
- **Employee Management:** Assign staff to counters, track sales performance, and manage accounts.
- **Reporting and Analytics:** Provide dashboards for revenue tracking and staff performance.

Limitations:

- The system is still slow in some areas with complex functionalities such as scheduling.

1.5/ Software product vision

This web application aims to provide a comprehensive platform for jewelry store staff, managers, and administrators to effectively manage business operations. With user-friendly interfaces and intuitive navigation, the system ensures minimal training requirements for staff. Key features include:

- Real-time display of gold prices.
- Flexible pricing and promotions management.
- Efficient buy-back and refund policies.
- Detailed inventory and sales tracking.
- Comprehensive customer loyalty program.
- Performance analytics for staff and sales.

1.6/ System Functions:

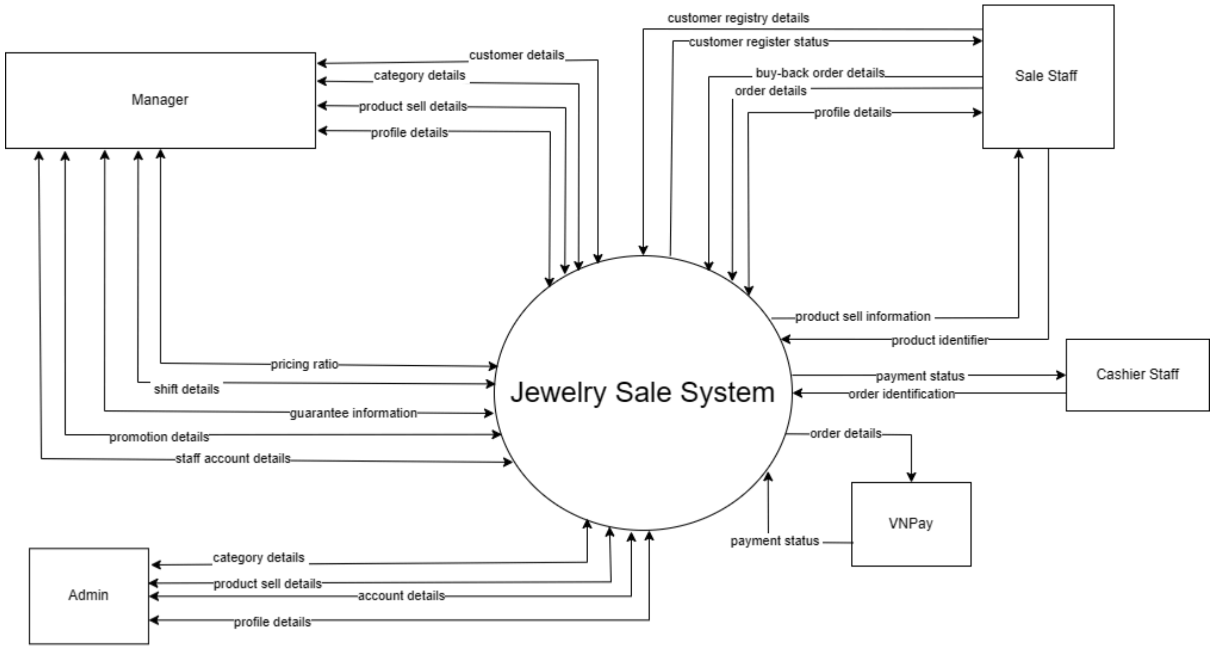
a) User Authorization:

Screen	**Admin**	**Manager**	**Staff**	**Public**
<<Home Screen>>	x	x	x	x
View display products	x	x	x	x
View detail of a product	x	x	x	x
Access login page	x	x	x	x
<<Login Screen>>				
Enter Login Details	x	x	x	
Access “Forgot password” page	x	x	x	
Login via Google	x	x	x	
<<After Login>>				
<<Profile Page>>	x	x	x	
Check profile	x	x	x	
Change Password	x	x	x	
Update Profile	x	x	x	
Logout	x	x	x	
<<Product Page>>	x	x	x	
View product	x	x	x	
Update product	x	x		
Delete product	x	x		
Create product	x	x		
<<Category Page>>	x	x	x	
View category	x	x	x	
Create category	x			
Update category		x		
Delete category	x	x		

<<Dashboard Page>>	x	x		
View Top Products sells	x	x		
View transaction	x	x		
View product sell by category	x			
View revenue	x			
View number of registered customer	x			
View type of registered customer	x			
View total number of manager and staff	x			
View revenue, number of product sell and registered customer between 2 years		x		
View staff list	x	x		
<<Admin Page>>	x			
View manager list	x			
Update manager	x			
Delete manager	x			
Create manager	x			
<<Manager Page>>		x		
Create staff		x		
Update staff		x		
Delete staff		x		
View Schedule of staff		x		
Assign Schedule and shift to staff		x		
View customer list		x		
Update customer		x		
Delete customer		x		
View Promotion		x		

Add new Promotions		x		
Adjust Pricing Ratio		x		
<<Staff Page>>			x	
Verify Order			x	
Buy Product from customer			x	
Create Order			x	
Processing payment			x	

1.7/ Context Diagram



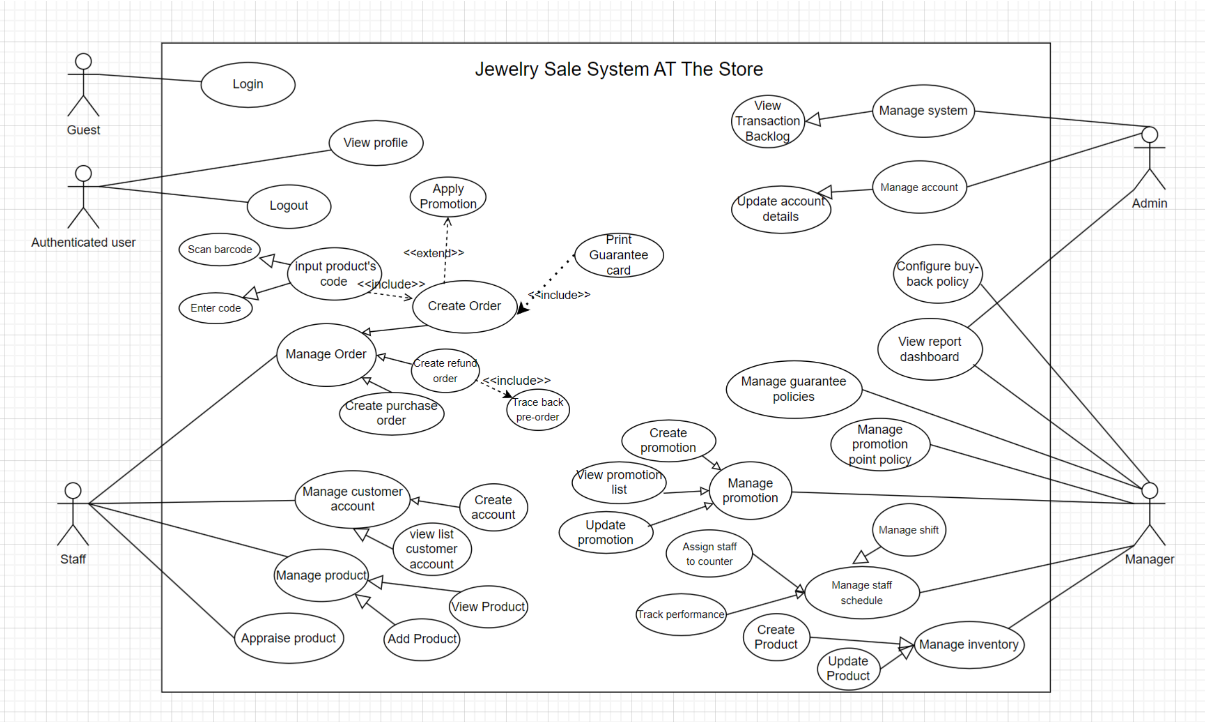
2. User Requirements

2.1/ Actors

#	Actor	Description

1	Administrator	The administrators who manage the system application. They are responsible for managing accounts, application settings, and permissions.
2	Staff	Staff members are responsible for managing sales orders, handling customer queries, creating, viewing, editing, and deleting products.
3	Manager	Managers oversee staff performance, approve inventory updates, manage promotions, and adjust pricing policies. They use their accounts to log in and access the system's managerial features.

Table - Actors list and description



Use Case

2.2/ Use Case List with description

ID	Use Case	Actors	Description
UC-1	As a staff member, I want to create an order for a customer	Staff	Allow staff to create an order for a customer with product information.
UC-2	As a staff member, I want to scan barcodes or input product codes	Staff	Allow staff to scan product barcodes or manually input product codes.
UC-3	As a staff member, I want to print guarantee cards	Staff	Allow staff to print out physical guarantee cards for products purchased by customers.
UC-4	As a manager, I want to adjust pricing ratios	Manager	Allow managers to adjust pricing ratios for products.
UC-5	As a staff member, I want to calculate prices based on current gold prices and other factors	Staff	Allow staff to calculate prices using current gold prices, weight, labor cost, and gemstone cost.
UC-6	As a manager, I want to create promotions and discount codes	Manager	Allow managers to create promotions and discount codes for products.
UC-7	As a staff member, I want to apply promotions to customer orders	Staff	Allow staff to apply promotions and discount codes to customer orders.
UC-8	As a staff member, I want to create buy-back orders with valid details	Staff	Allow staff to create buy-back orders for customers with valid details.

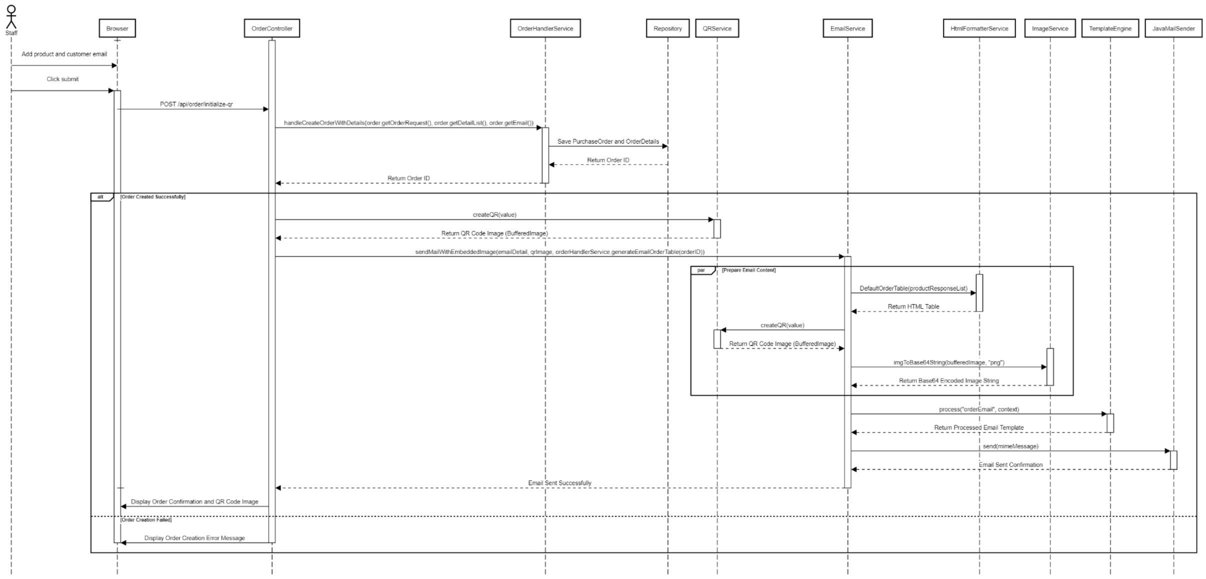
UC-9	As a manager, I want to update existing guarantees for the product to ensure that the correct guarantee details are applied.	Manager	Allow manager to update new information for existing guarantees for the product.
UC-10	As a staff member, I want to add new products to the inventory	Staff	Allow staff to add new products to the inventory.
UC-11	As a staff member, I want to update existing products in the inventory	Staff	Allow staff to update information for existing products in the inventory.
UC-12	As a staff member, I want to delete products from the inventory	Staff	Allow staff to delete products from the inventory.
UC-13	As an admin, I want to create staff accounts	Admin	Allow the admin to create accounts for staff members.
UC-14	As an admin, I want to update staff account details	Admin	Allow the admin to update details for staff accounts.
UC-15	As an admin, I want to delete staff accounts	Admin	Allow the admin to delete staff accounts.
UC-16	As a customer, I want to view my loyalty points	Customer	Allow customers to view their accumulated loyalty points.
UC-17	As a staff member, I want to register customers for the loyalty program	Staff	Allow staff to register customers into the loyalty program.

3. Functional Requirements

Use Case: UC-1: Create an order for a customer

Objective: Allow staff member to create an order for a customer.

Actor	**Staff member**
Trigger	Staff member decides to create a new order for a customer.
Pre-condition	- Staff member is logged into the system. - Staff member has access to customer and product information.
Post-condition	An order is successfully created and saved in the system for the customer.
Assumption	N/A
Dependency	Availability of up-to-date product inventory and customer details in the system.

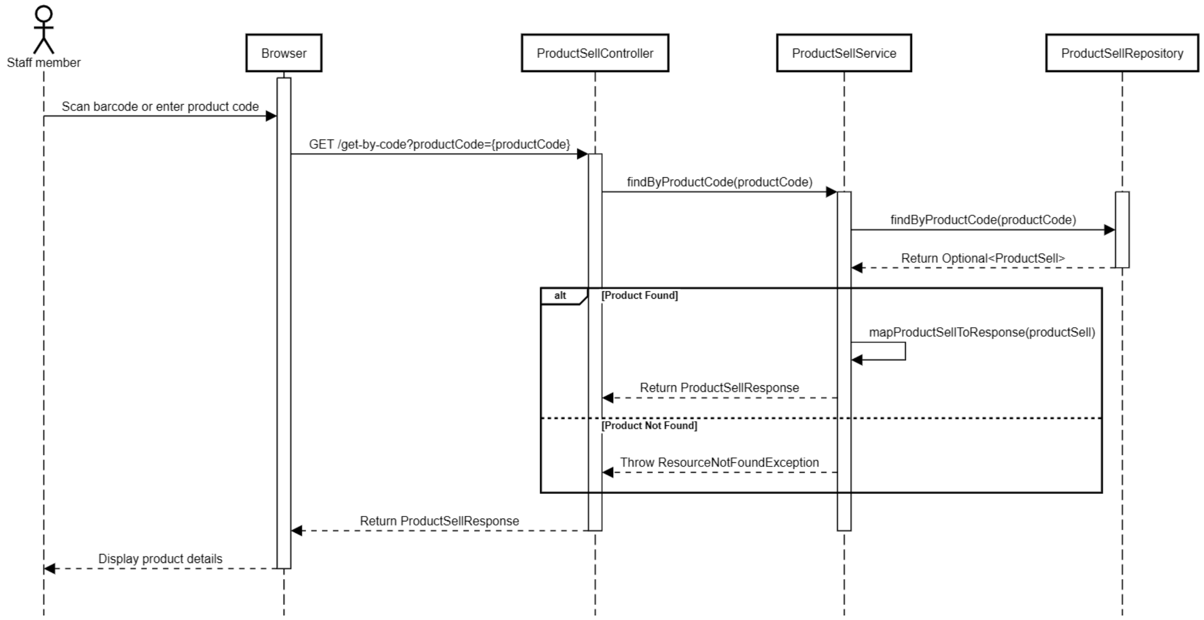

Staff
Browser
OrderController
OrderHandlerService
Repository
QRService
EmailService
HtmlFormatterService
ImageService
TemplateEngine
JavaMailSender
Add product and customer email
Click submit
POST /api/order/initialize-qr
handleCreateOrderWithDetails(order.getOrderRequest(), order.getDetailList(), order.getEmail())
Save PurchaseOrder and OrderDetails
Return Order ID
Return Order ID
alt
[Order Created Successfully]
createQR(value)
Return QR Code Image (BufferedImage)
sendMailWithEmbeddedImage(emailDetail, qrImage, orderHandlerService.generateEmailOrderTable(orderID))
par
[Prepare Email Content]
DefaultOrderTable(productResponseList)
Return HTML Table
createQR(value)
Return QR Code Image (BufferedImage)
imgToBase64String(bufferedImage, "png")
Return Base64 Encoded Image String
process("orderEmail", context)
Return Processed Email Template
send(mimeMessage)
Email Sent Confirmation
Email Sent Successfully
Display Order Confirmation and QR Code Image
[Order Creation Failed]
Display Order Creation Error Message

Use Case: UC-2: Scan barcodes or input product codes

Objective: Allow staff member to scan barcodes or input product codes for product identification.

Actor	Staff member
Trigger	Staff member needs to identify products either by scanning their barcodes or entering product codes manually.
Pre-condition	- Staff member is logged into the system. - Staff member has access to the scanning/input interface.
Post-condition	Product details are retrieved and displayed for further actions such as ordering or updating inventory.
Assumption	N/A
Dependency	Availability of a functional barcode scanner or product code input field in the system.

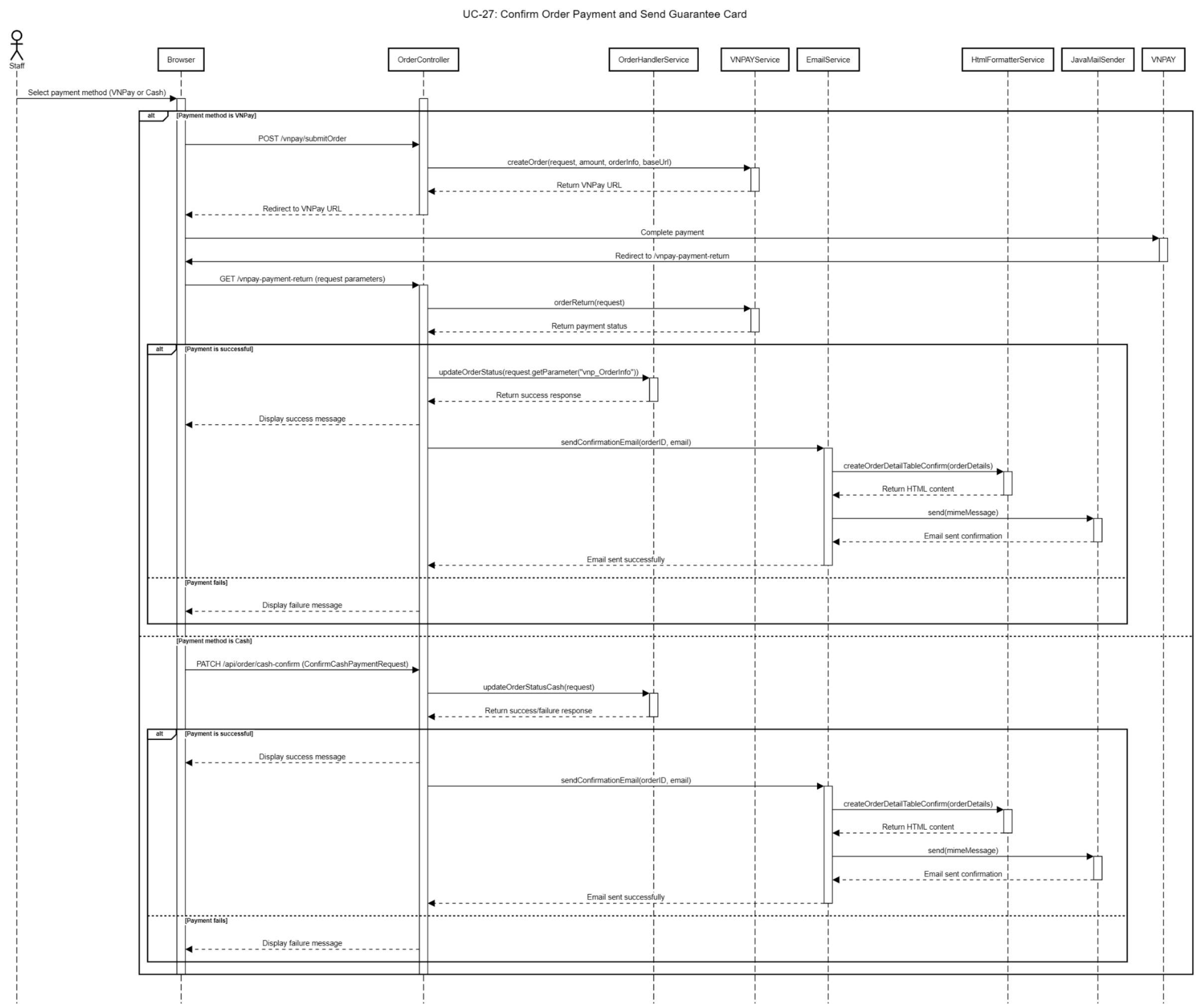
UC-2: Scan barcodes or input product codes


Staff member
Browser
ProductSellController
ProductSellService
ProductSellRepository
Scan barcode or enter product code
GET /get-by-code?productCode={productCode}
findByProductCode(productCode)
findByProductCode(productCode)
Return Optional<ProductSell>
alt
[Product Found]
mapProductSellToResponse(productSell)
Return ProductSellResponse
[Product Not Found]
Throw ResourceNotFoundException
Return ProductSellResponse
Display product details

Use Case: UC-3: Confirm Order Payment and Send Guarantee Card

Objective: Allow staff to confirm order payment through VNPay or cash and send a guarantee card via email upon successful payment.

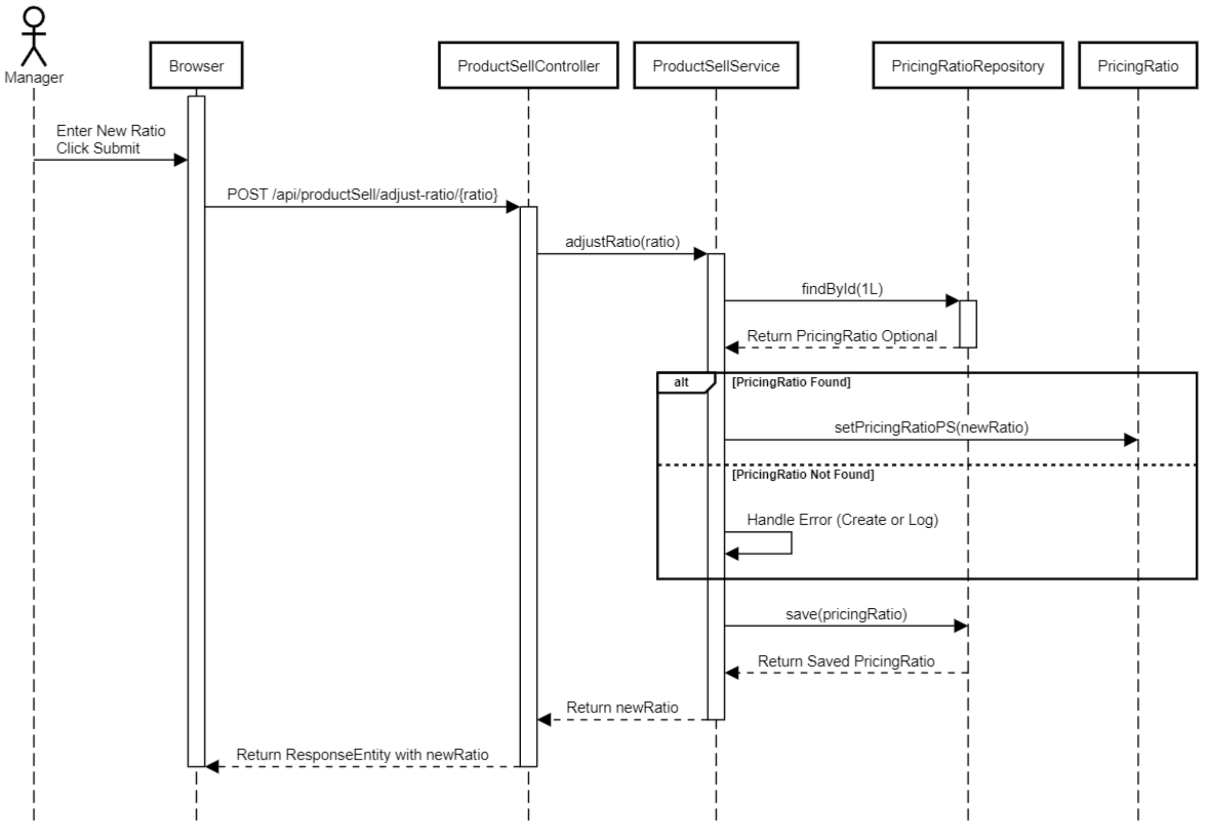
Actor	Staff member
Trigger	Staff member decides to confirm the order payment
Pre-condition	- Staff member is logged into the system. - Payment methods (VNPay and cash) are available. - Guarantee card template is available in the system.
Post-condition	Guarantee card is sent via email upon successful payment.
Assumption	N/A
Dependency	Availability of VNPay integration, cash payment processing tools, and guarantee card template in the system.



Use Case: UC-4: Adjust pricing ratios

Objective: Allow manager to adjust pricing ratios.

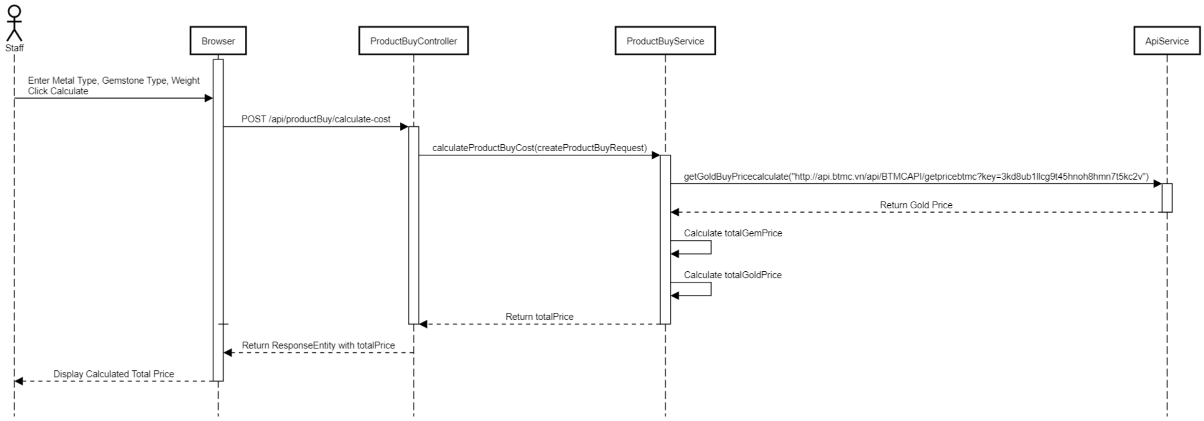
Actor	Manager
Trigger	Manager decides to adjust pricing ratios for products or services.
Pre-condition	- Manager is logged into the system. - Manager has permission to adjust pricing settings.
Post-condition	Pricing ratios are updated and applied to products or services as defined by the manager.
Assumption	N/A
Dependency	Availability of pricing adjustment tools and system access permissions for the manager.



Use Case: UC-5: Calculate prices based on current gold prices and other factors

Objective: Allow staff member to calculate prices based on current gold prices and other relevant factors.

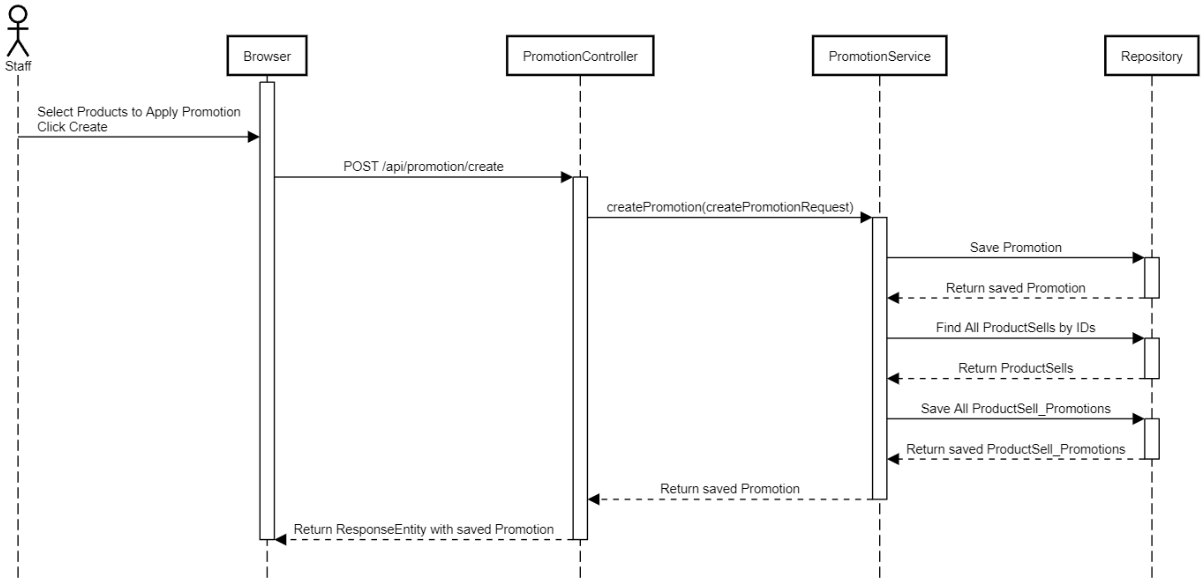
Actor	Staff member
Trigger	Staff member needs to calculate prices for products based on dynamic factors such as gold prices.
Pre-condition	- Staff member is logged into the system. - System has up-to-date information on gold prices and relevant factors.
Post-condition	Prices for products are calculated and displayed accurately.
Assumption	N/A
Dependency	Availability of real-time data on gold prices and relevant factors in the system.


Staff
Browser
ProductBuyController
ProductBuyService
ApiService
Enter Metal Type, Gemstone Type, Weight
Click Calculate
POST /api/productBuy/calculate-cost
calculateProductBuyCost(createProductBuyRequest)
getGoldBuyPricecalculate("http://api.btmc.vn/api/BTMCAPI/getpricebtmc?key=3kd8ub1llcg9t45hnoh8hmn7t5kc2v")
Return Gold Price
Calculate totalGemPrice
Calculate totalGoldPrice
Return totalPrice
Return ResponseEntity with totalPrice
Display Calculated Total Price

Use Case: UC-6: Create promotions and discount codes

Objective: Allow manager to create promotions and discount codes.

Actor	Manager
Trigger	Manager decides to create promotional offers or discount codes.
Pre-condition	- Manager is logged into the system. - Manager has authorization to create promotions.
Post-condition	Promotional offers or discount codes are successfully created and stored in the system.
Assumption	N/A
Dependency	Availability of promotion creation tools and system access permissions for the manager.

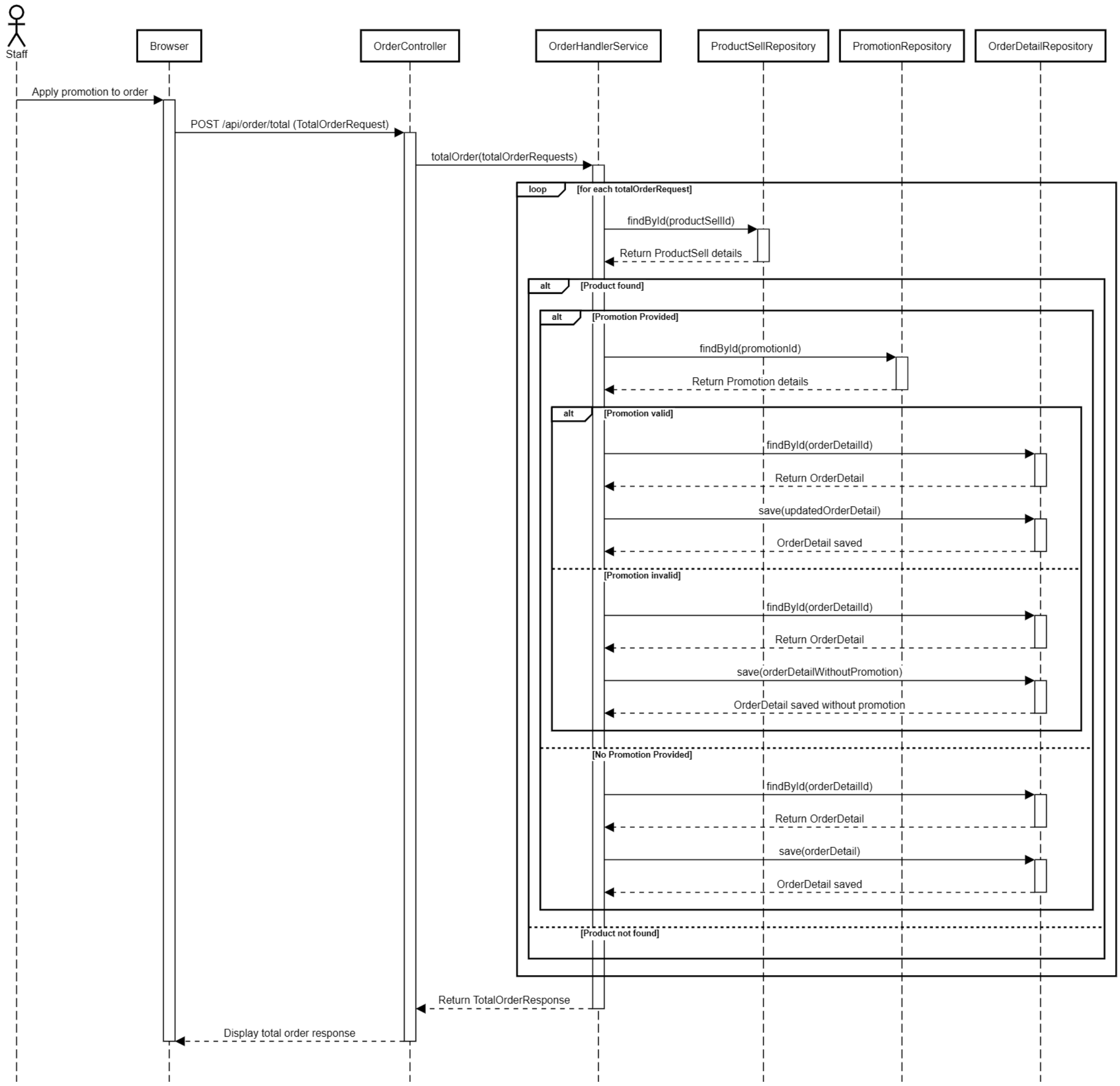

Staff
Browser
PromotionController
PromotionService
Repository
Select Products to Apply Promotion
Click Create
POST /api/promotion/create
createPromotion(createPromotionRequest)
Save Promotion
Return saved Promotion
Find All ProductSells by IDs
Return ProductSells
Save All ProductSell_Promotions
Return saved ProductSell_Promotions
Return saved Promotion
Return ResponseEntity with saved Promotion

Use Case: UC-7: Apply promotions to customer orders

Objective: Allow staff member to apply promotions to customer orders.

Actor	**Staff member**
Trigger	Staff member needs to apply a promotion or discount to a customer's order.
Pre-condition	- Staff member is logged into the system. - Customer order is in the process of being finalized.
Post-condition	Promotion or discount is successfully applied to the customer's order.
Assumption	N/A
Dependency	Availability of promotion application tools and access to customer order details in the system.

UC-7: Apply promotions to customer orders

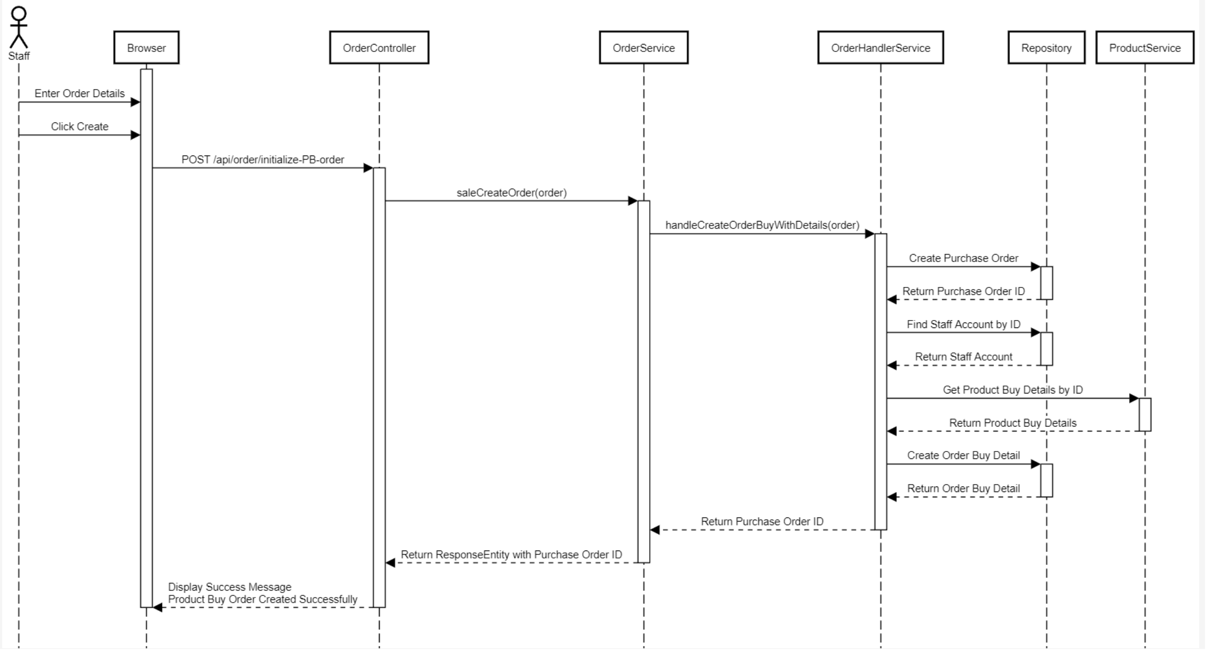


Use Case: UC-8: Create buy-back orders with valid details

Objective: Allow staff member to create buy-back orders with valid details.

Actor	Staff member
Trigger	Staff member decides to create a buy-back order for a product.
Pre-condition	- Staff member is logged into the system. - Product details and buy-back policies are available.
Post-condition	Buy-back order with valid details is successfully created and recorded in the system.

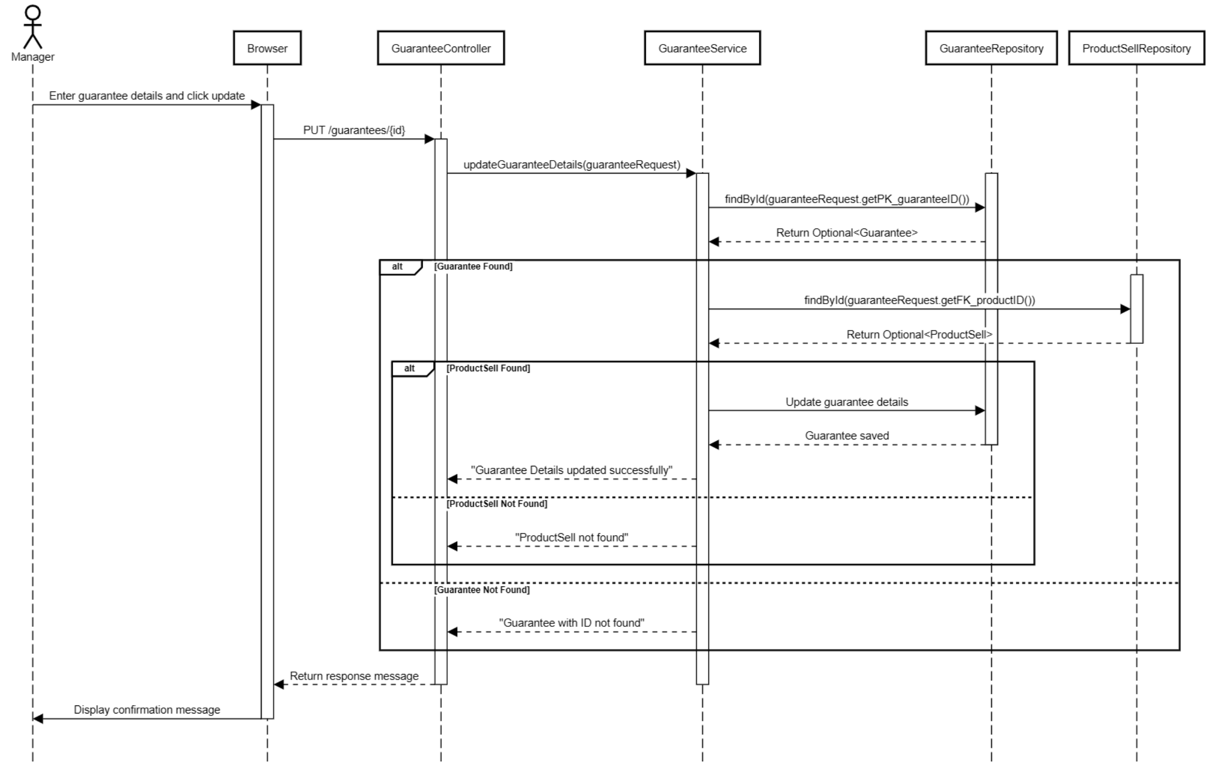
Assumption	N/A
Dependency	Availability of buy-back order creation tools and accurate product information in the system.



Use Case: UC-9: Update Guarantee Details

Objective: Allow staff members to update existing guarantee detail for a product in the Guarantee table

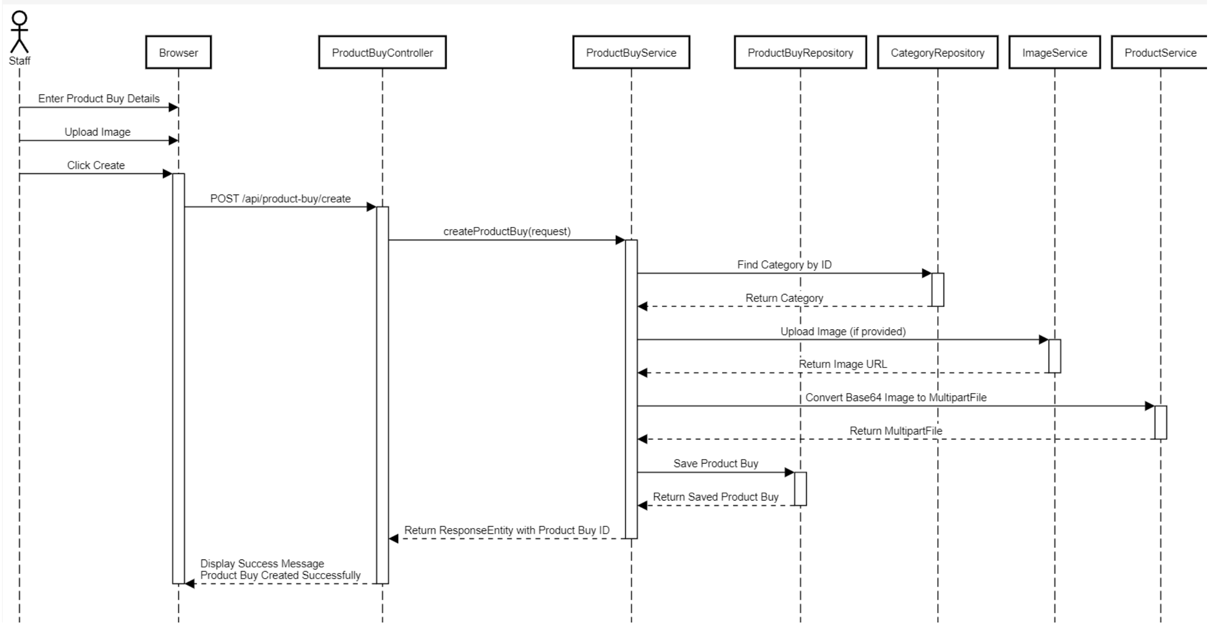
Actor	**Manager**
Trigger	Manager needs to update the guarantee details for a specific product.
Pre-condition	- Manager is logged into the system. - System has accurate information about the products and their existing guarantees.
Post-condition	The guarantee detail for the specified product are successfully updated in the Guarantee table.
Assumption	N/A
Dependency	Availability of the update tools and accurate product information in the system.


Manager
Browser
GuaranteeController
GuaranteeService
GuaranteeRepository
ProductSellRepository
Enter guarantee details and click update
PUT /guarantees/{id}
updateGuaranteeDetails(guaranteeRequest)
findById(guaranteeRequest.getPK_guaranteeID())
Return Optional<Guarantee>
alt [Guarantee Found]
findById(guaranteeRequest.getFK_productID())
Return Optional<ProductSell>
alt [ProductSell Found]
Update guarantee details
Guarantee saved
"Guarantee Details updated successfully"
[ProductSell Not Found]
"ProductSell not found"
[Guarantee Not Found]
"Guarantee with ID not found"
Return response message
Display confirmation message

Use Case: UC-10: Add new products to the inventory

Objective: Allow staff member to add new products to the inventory.

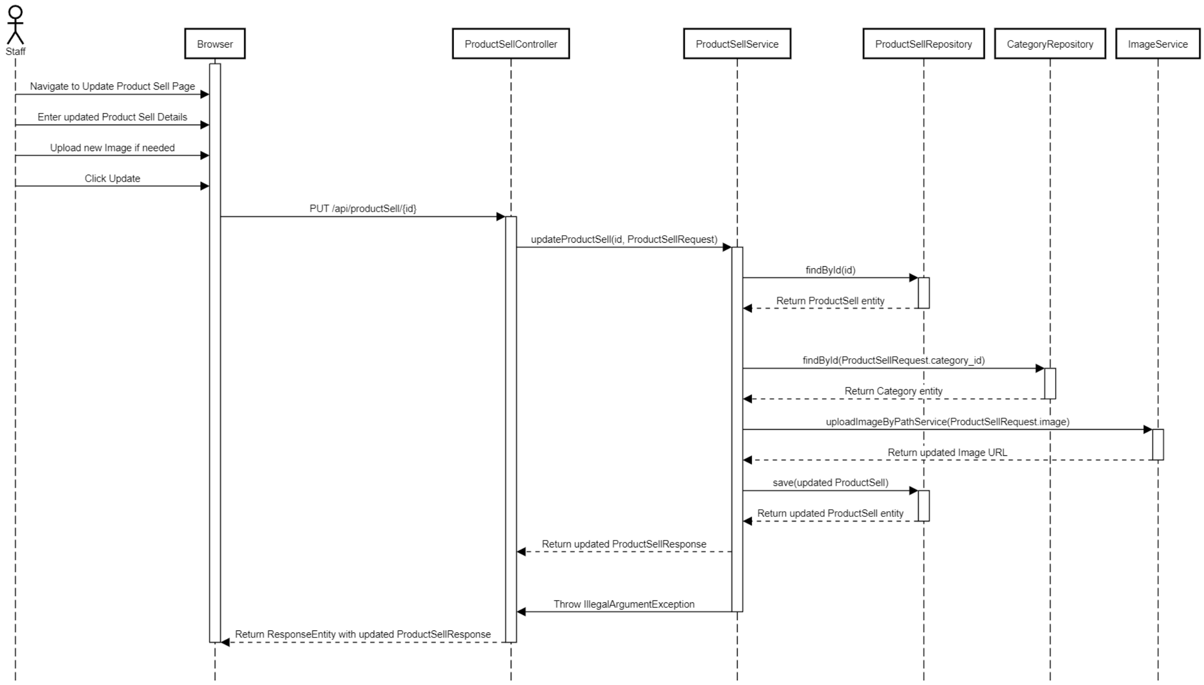
Actor	Staff member
Trigger	Staff member decides to add a new product to the inventory.
Pre-condition	- Staff members are logged into the system. - Inventory management interface is accessible.
Post-condition	New product is successfully added to the inventory and available for sale.
Assumption	N/A
Dependency	Availability of product addition tools and access to inventory management functions in the system.



Use Case: UC-11: Update existing products in the inventory

Objective: Allow staff member to update existing products in the inventory.

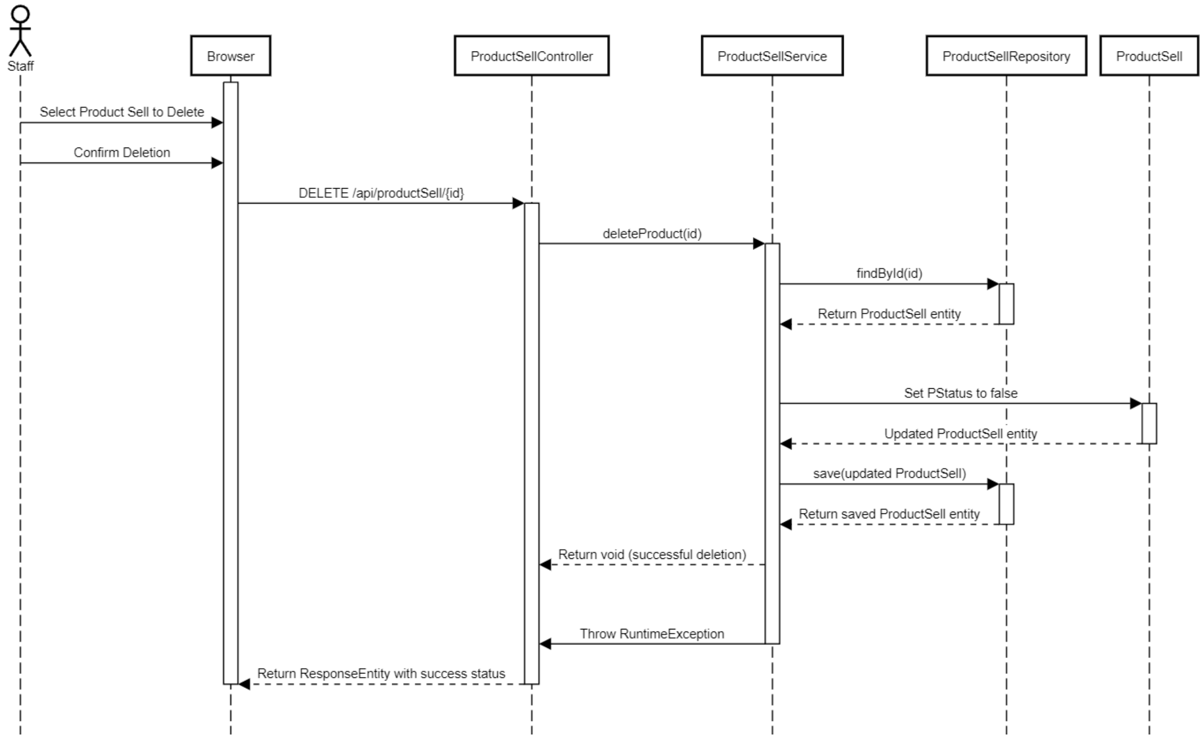
Actor	Staff member
Trigger	Staff member decides to update details of an existing product in the inventory.
Pre-condition	- Staff member is logged into the system. - Product to be updated is already in the inventory.
Post-condition	Changes to product details (e.g., price, description) are successfully updated in the inventory.
Assumption	N/A
Dependency	Availability of product update tools and access to inventory management functions in the system.


Staff
Browser
ProductSellController
ProductSellService
ProductSellRepository
CategoryRepository
ImageService
Navigate to Update Product Sell Page
Enter updated Product Sell Details
Upload new Image if needed
Click Update
PUT /api/productSell/{id}
updateProductSell(id, ProductSellRequest)
findById(id)
Return ProductSell entity
findById(ProductSellRequest.category_id)
Return Category entity
uploadImageByPathService(ProductSellRequest.image)
Return updated Image URL
save(updated ProductSell)
Return updated ProductSell entity
Return updated ProductSellResponse
Throw IllegalArgumentException
Return ResponseEntity with updated ProductSellResponse

Use Case: UC-12: Delete products from the inventory

Objective: Allow staff member to delete products from the inventory.

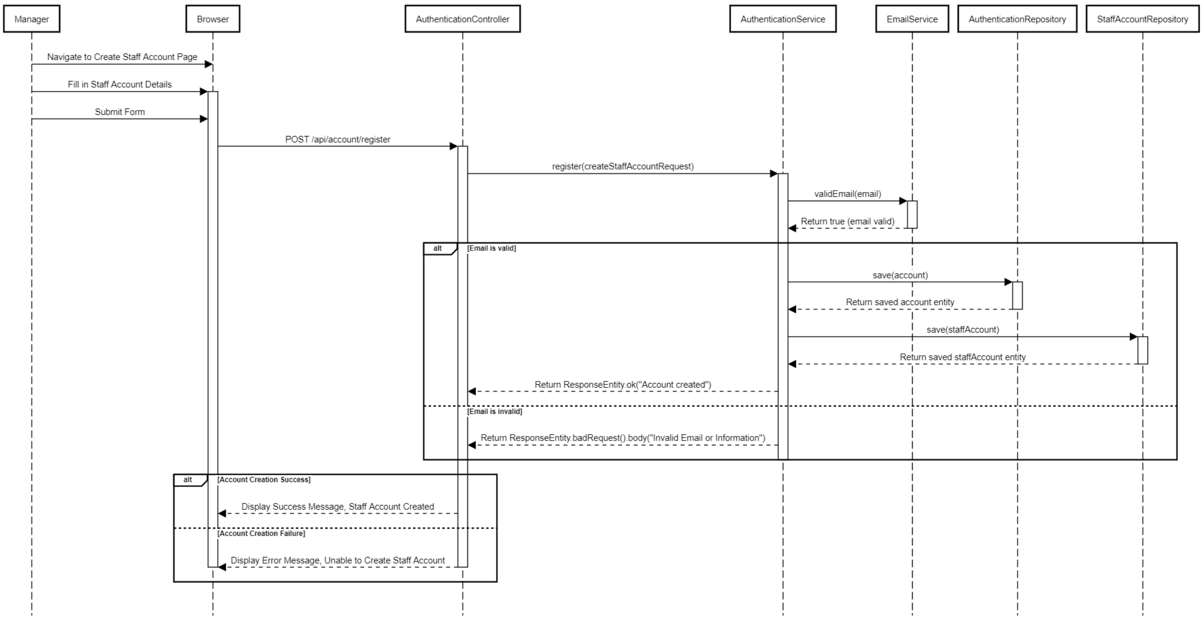
Actor	Staff member
Trigger	Staff member decides to remove a product from the inventory.
Pre-condition	- Staff member is logged into the system. - Product to be deleted exists in the inventory.
Post-condition	Product is successfully removed from the inventory and no longer available for sale.
Assumption	N/A
Dependency	Availability of product deletion tools and access to inventory management functions in the system.


Staff
Browser
ProductSellController
ProductSellService
ProductSellRepository
ProductSell
Select Product Sell to Delete
Confirm Deletion
DELETE /api/productSell/{id}
deleteProduct(id)
findById(id)
Return ProductSell entity
Set PStatus to false
Updated ProductSell entity
save(updated ProductSell)
Return saved ProductSell entity
Return void (successful deletion)
Throw RuntimeException
Return ResponseEntity with success status

Use Case: UC-13: Create staff accounts

Objective: Allow admin to create new staff accounts.

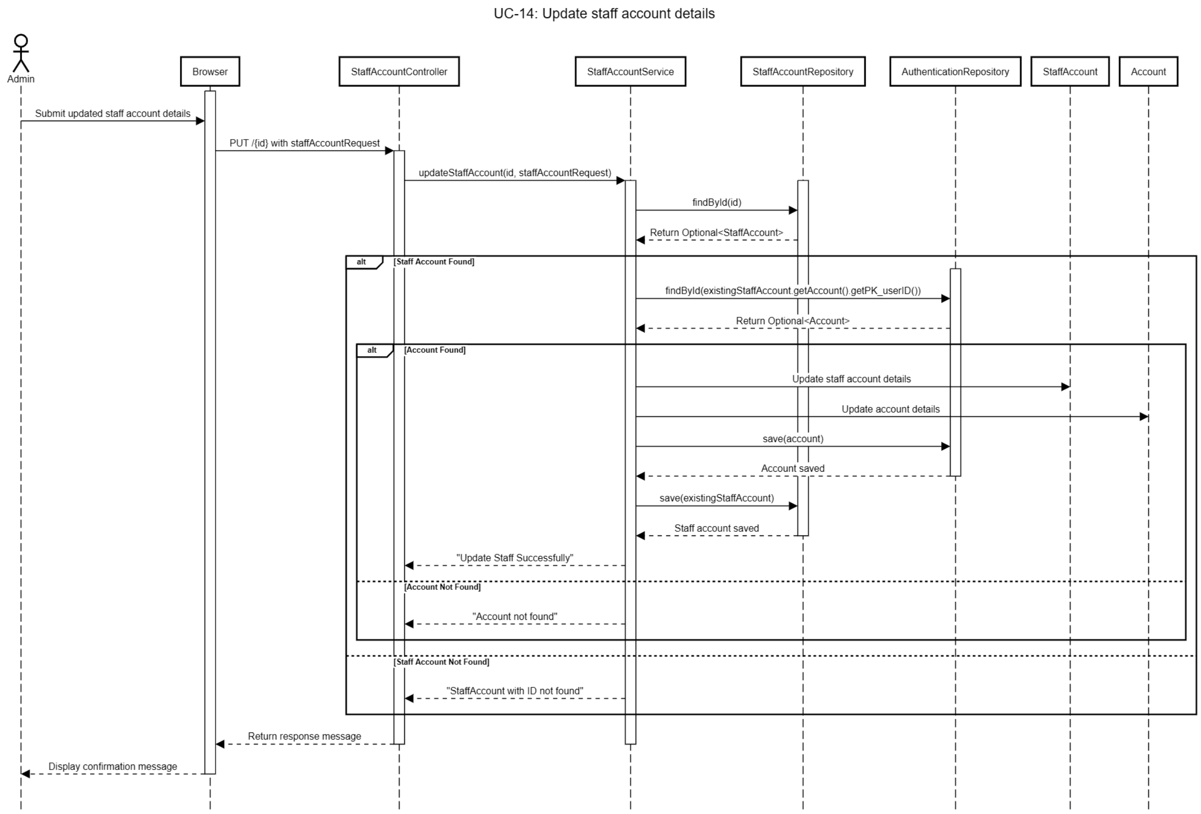
Actor	Admin
Trigger	Admin decides to create a new staff member account.
Pre-condition	- Admin is logged into the system. - Authorization to create staff accounts is granted.
Post-condition	New staff account is successfully created and added to the system.
Assumption	N/A
Dependency	Availability of staff account creation tools and admin access rights in the system.


Manager
Browser
AuthenticationController
AuthenticationService
EmailService
AuthenticationRepository
StaffAccountRepository
Navigate to Create Staff Account Page
Fill in Staff Account Details
Submit Form
POST /api/account/register
register(createStaffAccountRequest)
validEmail(email)
Return true (email valid)
alt
[Email is valid]
save(account)
Return saved account entity
save(staffAccount)
Return saved staffAccount entity
Return ResponseEntity.ok("Account created")
[Email is invalid]
Return ResponseEntity.badRequest().body("Invalid Email or Information")
alt
[Account Creation Success]
Display Success Message, Staff Account Created
[Account Creation Failure]
Display Error Message, Unable to Create Staff Account

Use Case: UC-14: Update staff account details

Objective: Allow admin to update details of staff accounts.

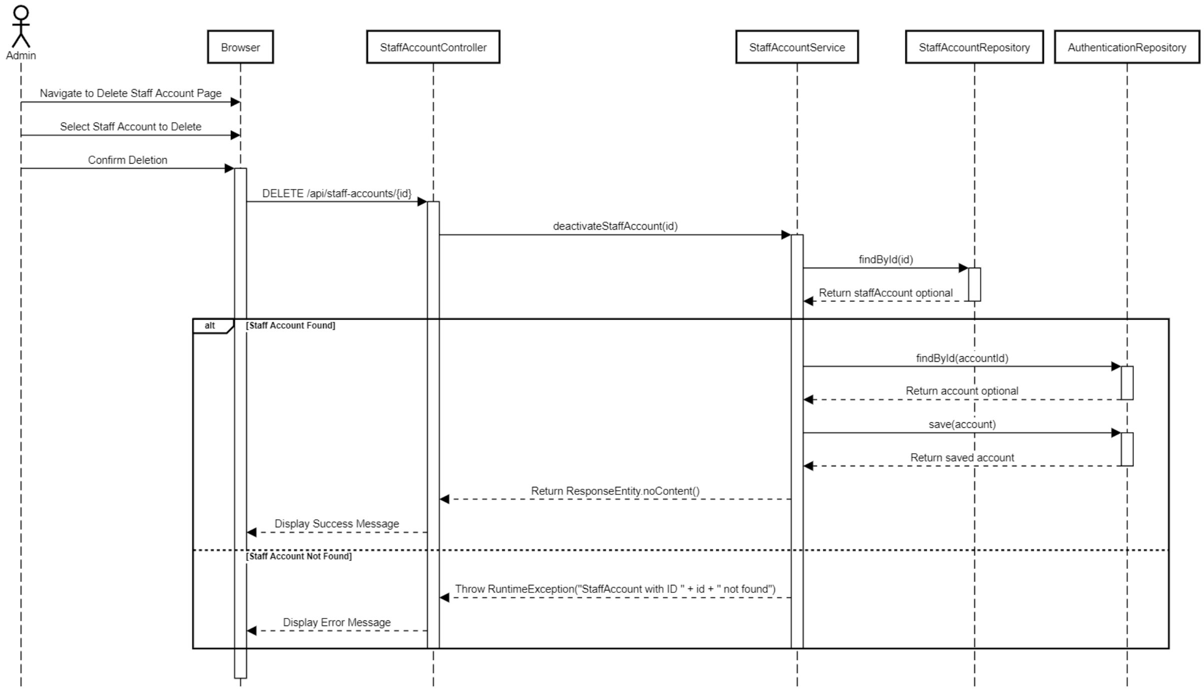
Actor	Admin
Trigger	Admin decides to modify details of an existing staff account.
Pre-condition	- Admin is logged into the system. - Staff account to be updated exists in the system.
Post-condition	Changes to staff account details (e.g., role, contact information) are successfully updated.
Assumption	N/A
Dependency	Availability of staff account update tools and admin access rights in the system.


UC-14: Update staff account details
Admin
Browser
StaffAccountController
StaffAccountService
StaffAccountRepository
AuthenticationRepository
StaffAccount
Account
Submit updated staff account details
PUT /{id} with staffAccountRequest
updateStaffAccount(id, staffAccountRequest)
findById(id)
Return Optional<StaffAccount>
alt
[Staff Account Found]
findById(existingStaffAccount.getAccount().getPK_userID())
Return Optional<Account>
alt
[Account Found]
Update staff account details
Update account details
save(account)
Account saved
save(existingStaffAccount)
Staff account saved
"Update Staff Successfully"
[Account Not Found]
"Account not found"
[Staff Account Not Found]
"StaffAccount with ID not found"
Return response message
Display confirmation message

Use Case: UC-15: Delete staff accounts

Objective: Allow admin to delete staff accounts.

Actor	Admin
Trigger	Admin decides to remove an existing staff account from the system.
Pre-condition	- Admin is logged into the system. - Staff account to be deleted exists in the system.
Post-condition	Staff account is successfully deleted and no longer accessible in the system.
Assumption	N/A
Dependency	Availability of staff account deletion tools and admin access rights in the system.

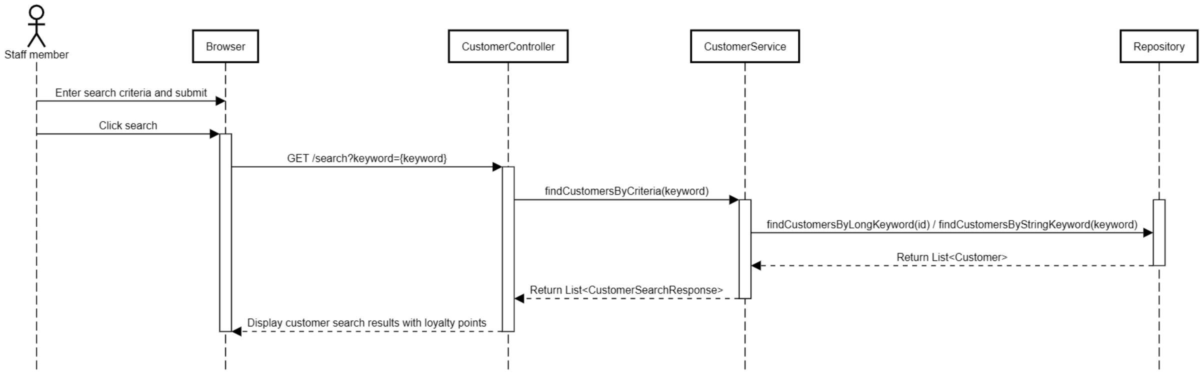

Admin
Browser
StaffAccountController
StaffAccountService
StaffAccountRepository
AuthenticationRepository
Navigate to Delete Staff Account Page
Select Staff Account to Delete
Confirm Deletion
DELETE /api/staff-accounts/{id}
deactivateStaffAccount(id)
findById(id)
Return staffAccount optional
alt
[Staff Account Found]
findById(accountId)
Return account optional
save(account)
Return saved account
Return ResponseEntity.noContent()
Display Success Message
[Staff Account Not Found]
Throw RuntimeException("StaffAccount with ID " + id + " not found")
Display Error Message

Use Case: UC-16: View loyalty points

Objective: Allow staff to view customers loyalty points.

Actor	Staff member
Trigger	Staff decides to check customers loyalty points balance in customer accounts.
Pre-condition	- Staff is logged into their account. - Loyalty program is active and points are recorded in customer accounts.
Post-condition	Staff can view customers loyalty points balance in customer accounts accurately.
Assumption	N/A
Dependency	Availability of loyalty points display feature in the staff account interface.

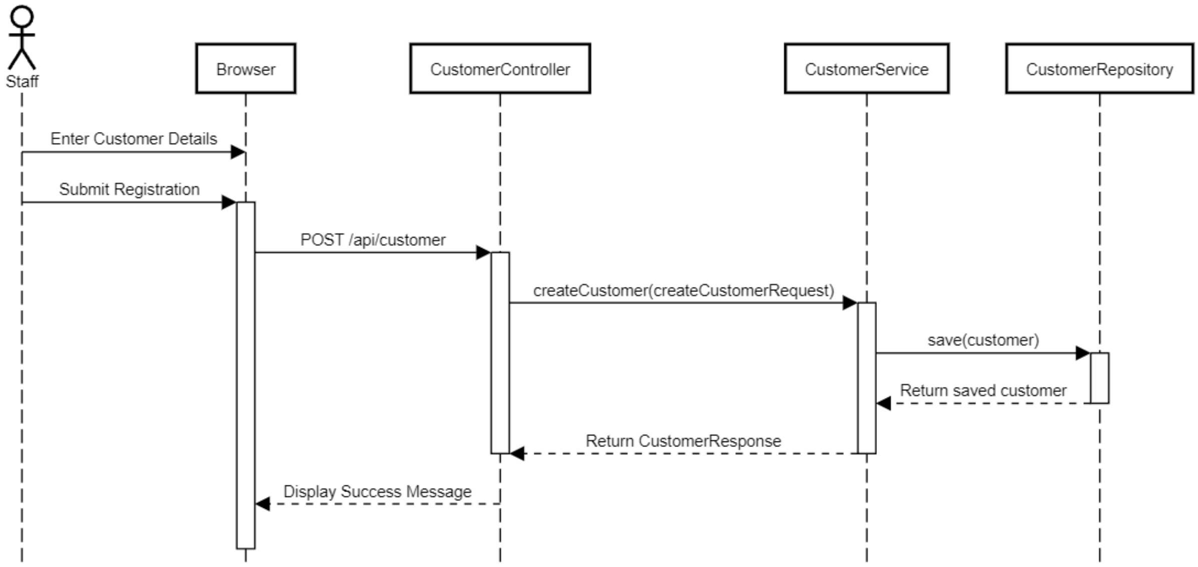
UC-16: View loyalty points


Staff member
Browser
CustomerController
CustomerService
Repository
Enter search criteria and submit
Click search
GET /search?keyword={keyword}
findCustomersByCriteria(keyword)
findCustomersByLongKeyword(id) / findCustomersByStringKeyword(keyword)
Return List<Customer>
Return List<CustomerSearchResponse>
Display customer search results with loyalty points

Use Case: UC-17: Register customers for the loyalty program

Objective: Allow staff member to register customers for the loyalty program.

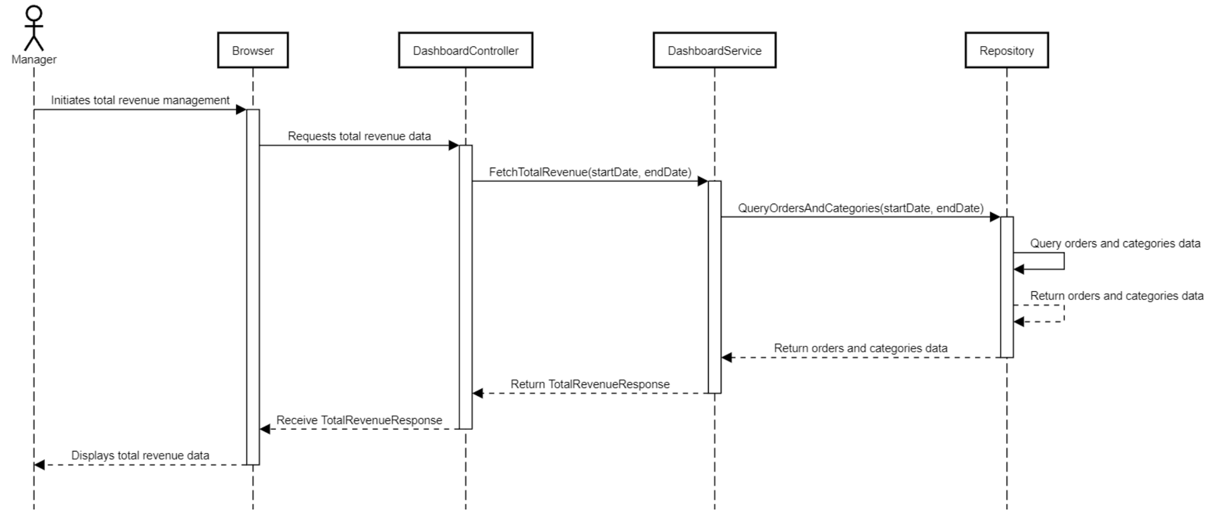
Actor	Staff member
Trigger	Staff member decides to enroll a customer in the loyalty program.
Pre-condition	- Staff member is logged into the system. - Customer wishes to join the loyalty program.
Post-condition	Customer is successfully registered and enrolled in the loyalty program.
Assumption	N/A
Dependency	Availability of customer registration tools for the loyalty program in the system.



Use Case: UC-18: Track total revenue

Objective: Allow manager to track total revenue generated by sales.

Actor	Manager
Trigger	Manager decides to view total revenue statistics.
Pre-condition	- Manager is logged into the system. - Sales transactions have been recorded in the system.
Post-condition	Total revenue figures are displayed accurately for the specified period.
Assumption	N/A
Dependency	Availability of revenue tracking tools and access to sales data in the system.

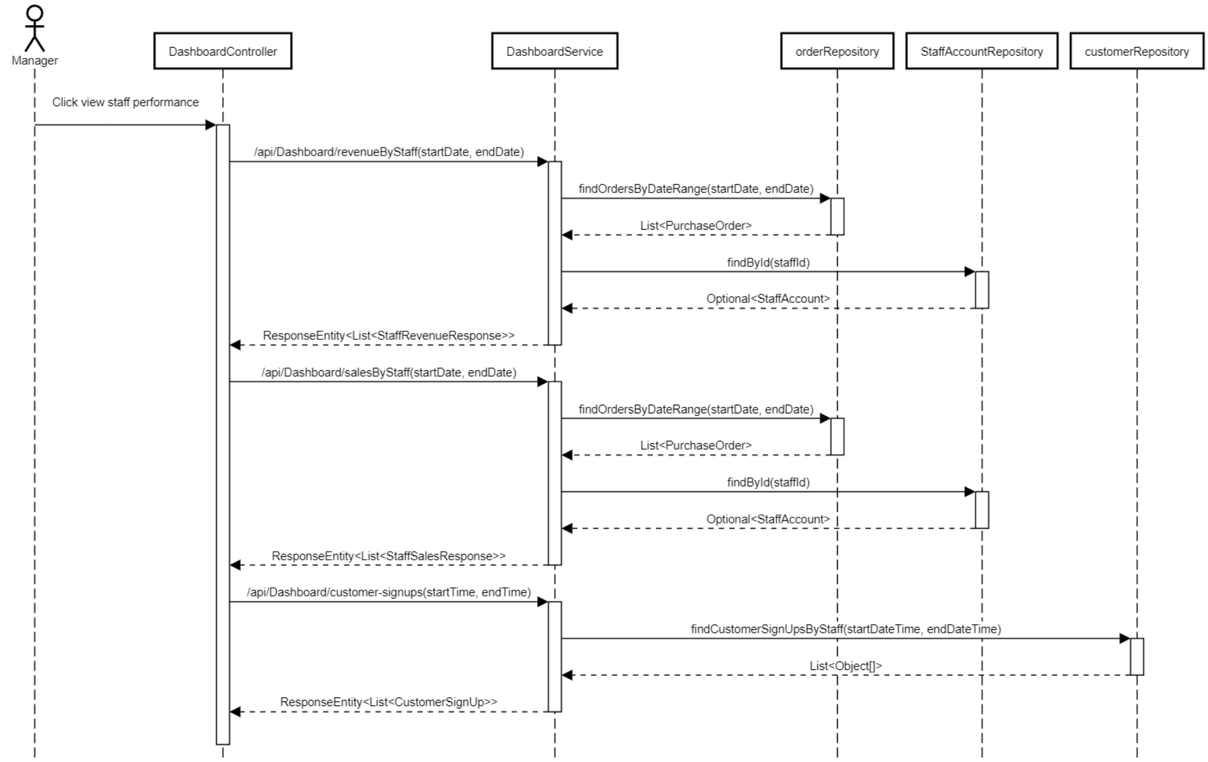

Manager
Browser
DashboardController
DashboardService
Repository
Initiates total revenue management
Requests total revenue data
FetchTotalRevenue(startDate, endDate)
QueryOrdersAndCategories(startDate, endDate)
Query orders and categories data
Return orders and categories data
Return orders and categories data
Return TotalRevenueResponse
Receive TotalRevenueResponse
Displays total revenue data

Use Case: UC-19: Track staff performance metrics

Objective: Allow manager to track performance metrics of staff members.

Actor	Manager
Trigger	Manager decides to review staff performance metrics.
Pre-condition	- Manager is logged into the system. - Staff performance data is recorded in the system.
Post-condition	Performance metrics (e.g., sales targets, customer satisfaction) are displayed accurately.

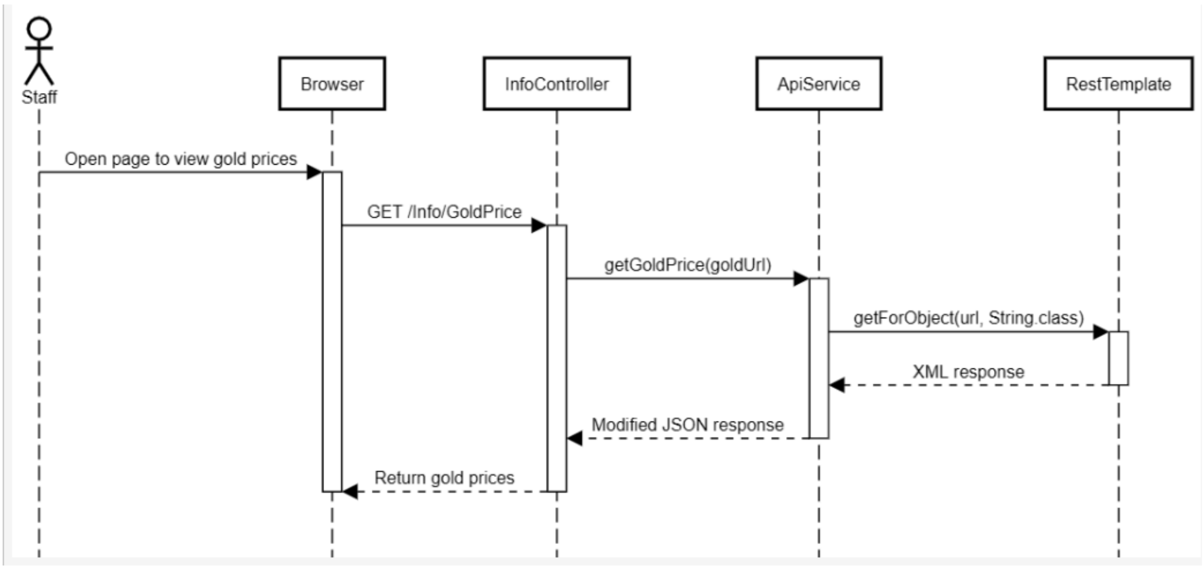
Assumption	N/A
Dependency	Availability of staff performance tracking tools and access to performance data in the system.


Manager
DashboardController
DashboardService
orderRepository
StaffAccountRepository
customerRepository
Click view staff performance
/api/Dashboard/revenueByStaff(startDate, endDate)
findOrdersByDateRange(startDate, endDate)
List<PurchaseOrder>
findById(staffId)
Optional<StaffAccount>
ResponseEntity<List<StaffRevenueResponse>>
/api/Dashboard/salesByStaff(startDate, endDate)
findOrdersByDateRange(startDate, endDate)
List<PurchaseOrder>
findById(staffId)
Optional<StaffAccount>
ResponseEntity<List<StaffSalesResponse>>
/api/Dashboard/customer-signups(startTime, endTime)
findCustomerSignUpsByStaff(startDateTime, endDateTime)
List<Object[]>
ResponseEntity<List<CustomerSignUp>>

Use Case: UC-20: View real-time gold prices

Objective: Allow user to view real-time gold prices.

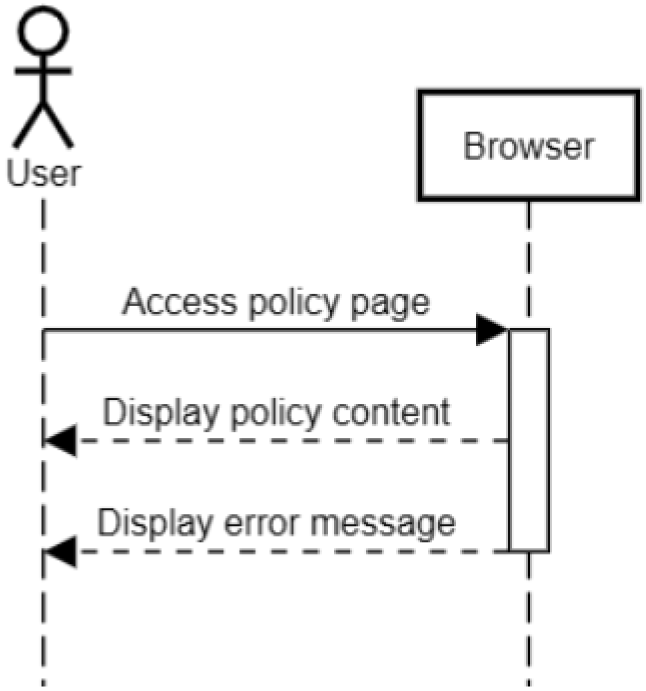
Actor	User
Trigger	User decides to check real-time gold prices.
Pre-condition	User has access to the web application. Real-time gold prices are available and updated in the system
Post-condition	User can view current gold prices accurately.
Assumption	N/A
Dependency	Availability of view real-time gold prices functionality in the web application.



Use Case: UC-21: View the store's return policies

Objective: Allow user to view the store's return policies.

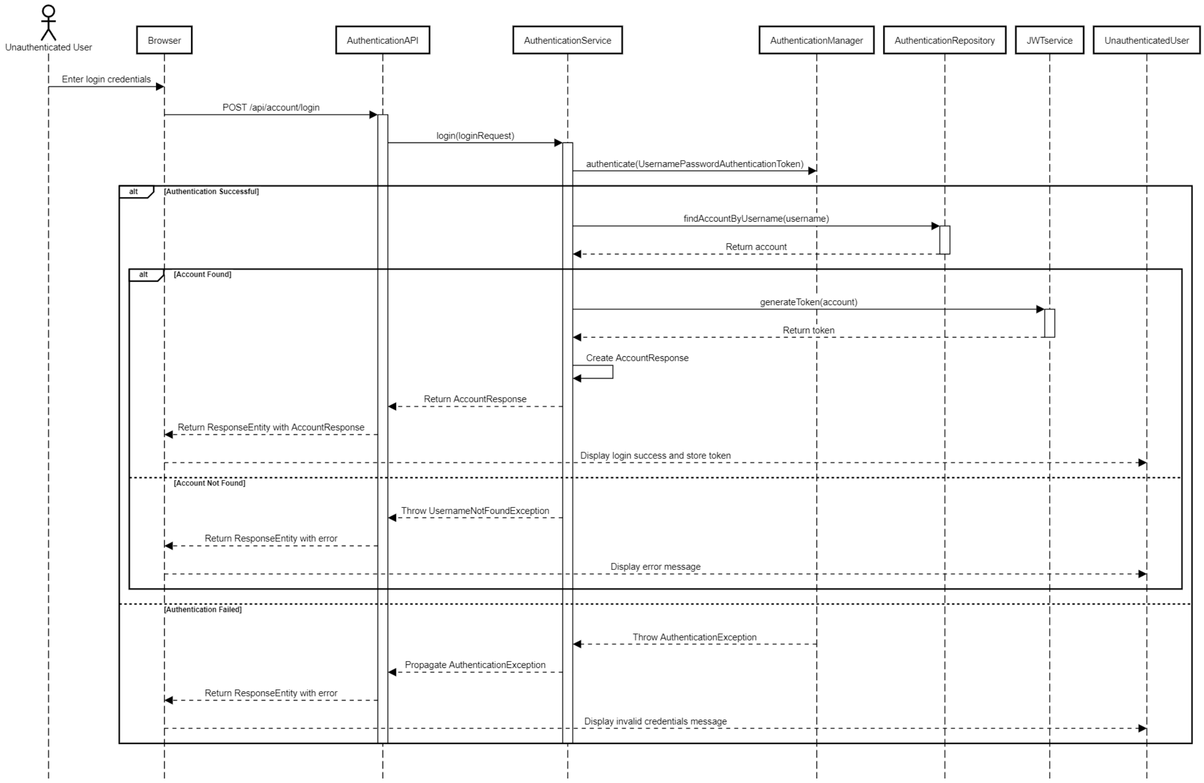
Actor	**User**
Trigger	User decides to review the store's return policies.
Pre-condition	User has access to the web application. Return policies are updated and available in the system.
Post-conditio n	User can access and understand the store's return policies.
Assumption	N/A
Dependency	Availability of view policy functionality in the web application.



Use Case: UC-22: Logging into a Web Application

Objective: Allow user to log into a web application.

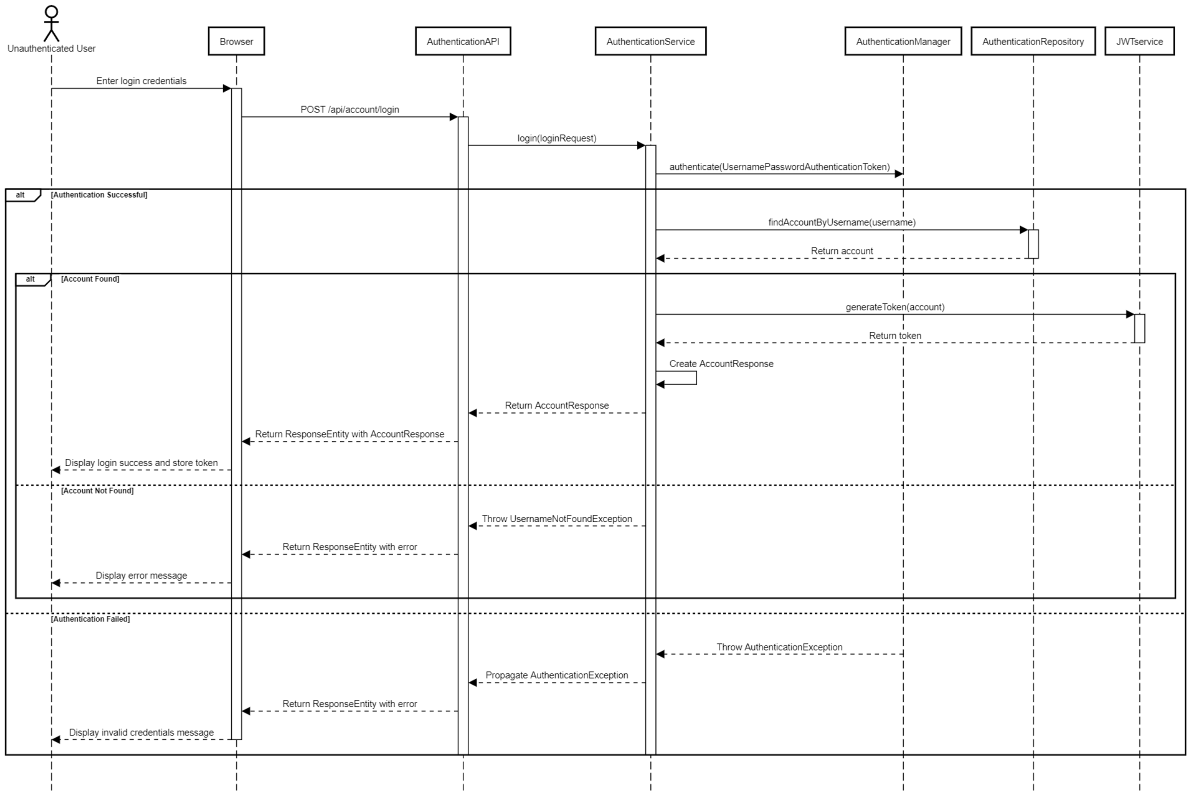
Actor	User
Trigger	User decides to log into the web application.
Pre-condition	- User has a registered account on the web application. - Login credentials are correct.
Post-condition	User successfully logs into the web application and gains access to their account.
Assumption	N/A
Dependency	Availability of login functionality and correct credentials verification in the web application.



Use Case: UC-23: Change password

Objective: Allow user to change their password.

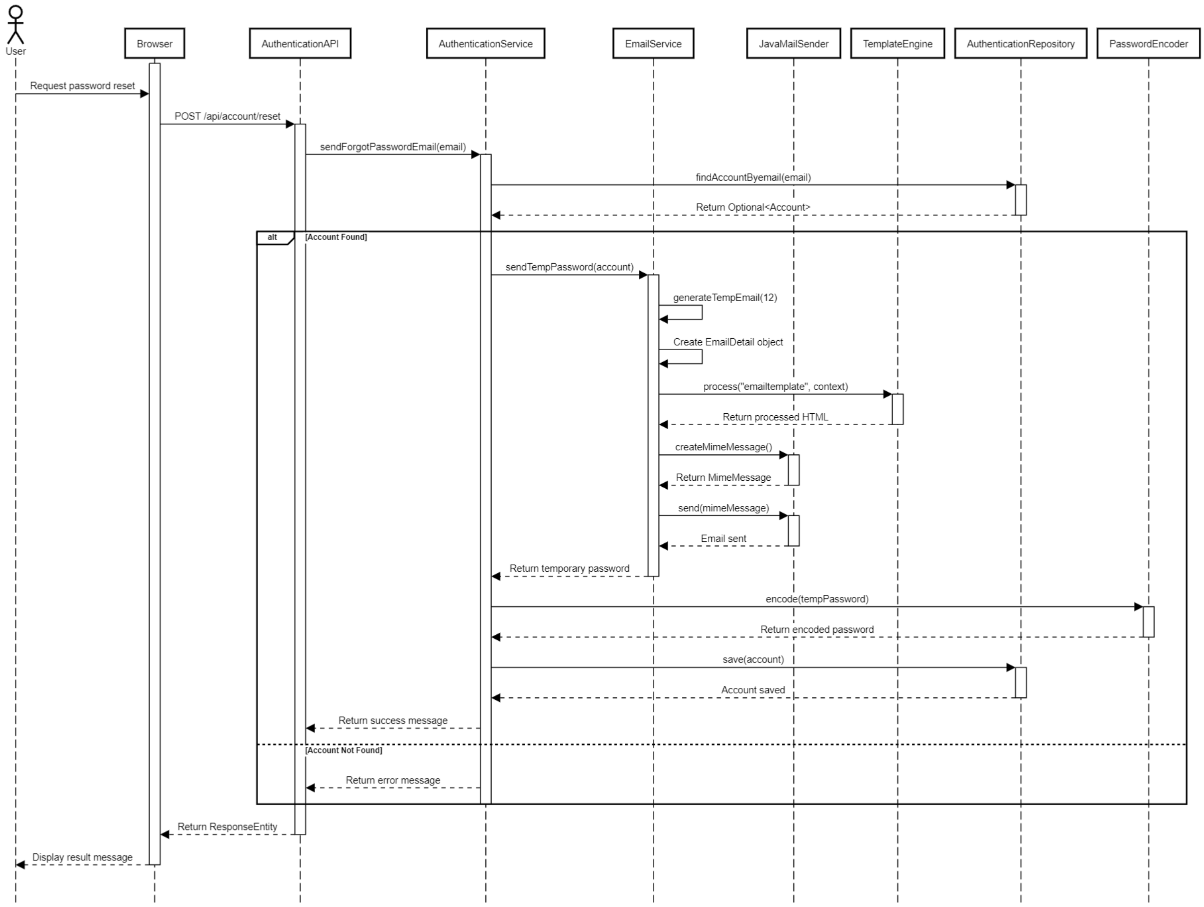
Actor	**User**
Trigger	User decides to change their password.
Pre-condition	- User is logged into their account. - User remembers their current password.
Post-condition	User successfully changes their password to a new one.
Assumption	N/A
Dependency	Availability of password change functionality and user authentication in the system.


Unauthenticated User
Browser
AuthenticationAPI
AuthenticationService
AuthenticationManager
AuthenticationRepository
JWTservice
Enter login credentials
POST /api/account/login
login(loginRequest)
authenticate(UsernamePasswordAuthenticationToken)
alt
[Authentication Successful]
findAccountByUsername(username)
Return account
alt
[Account Found]
generateToken(account)
Return token
Create AccountResponse
Return AccountResponse
Return ResponseEntity with AccountResponse
Display login success and store token
[Account Not Found]
Throw UsernameNotFoundException
Return ResponseEntity with error
Display error message
[Authentication Failed]
Throw AuthenticationException
Propagate AuthenticationException
Return ResponseEntity with error
Display invalid credentials message

Use Case: UC-24: Password Reset via Email

Objective: Allow user to reset their password via email.

Actor	User
Trigger	User decides to reset their password using the email reset option.
Pre-condition	- User has access to the email account associated with their account. - User requests a password reset.
Post-condition	User successfully resets their password using the link sent to their email.
Assumption	N/A
Dependency	Availability of password reset functionality and email communication integration in the system.

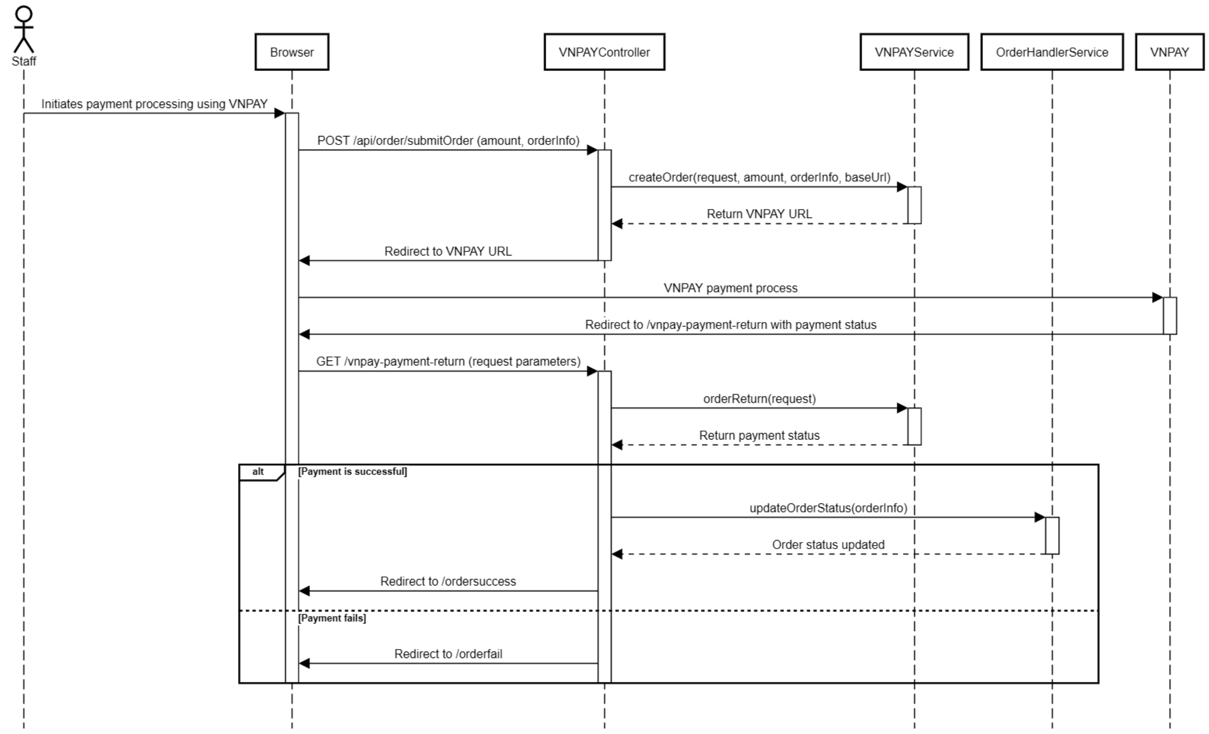


Use Case: UC-25: Use VNPAY to process payment for customer

Objective: Allow staff member to use VNPAY for processing customer payments.

Actor	**Staff member**
Trigger	Staff member initiates payment processing using VNPAY.
Pre-condition	- Staff member is logged into the system. - Customer's payment details are provided and verified.
Post-condition	Customer's payment is successfully processed using VNPAY.
Assumption	N/A
Dependency	Availability of VNPAY integration for payment processing and staff access rights in the system.

UC-25: Use VNPAY to process payment for customer

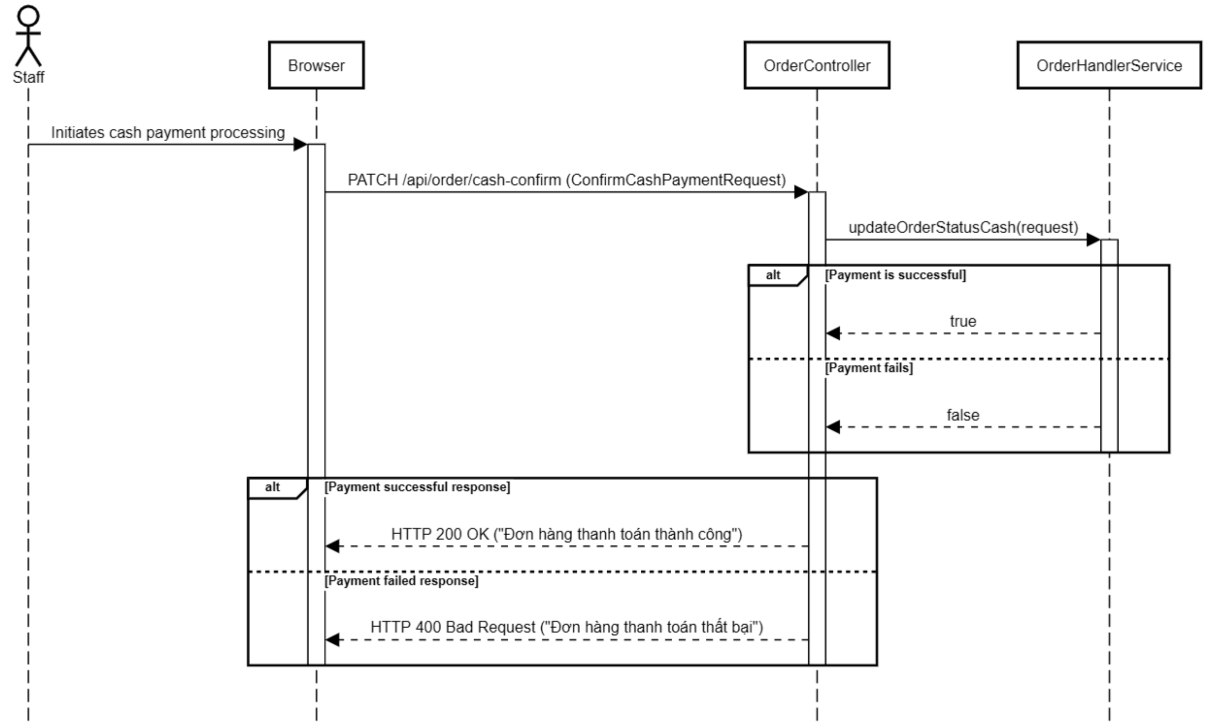

Staff
Browser
VNPAYController
VNPAYService
OrderHandlerService
VNPAY
Initiates payment processing using VNPAY
POST /api/order/submitOrder (amount, orderInfo)
createOrder(request, amount, orderInfo, baseUrl)
Return VNPAY URL
Redirect to VNPAY URL
VNPAY payment process
Redirect to /vnpay-payment-return with payment status
GET /vnpay-payment-return (request parameters)
orderReturn(request)
Return payment status
alt
[Payment is successful]
updateOrderStatus(orderInfo)
Order status updated
Redirect to /ordersuccess
[Payment fails]
Redirect to /orderfail

Use Case: UC-26: Process cash payment for customer

Objective: Allow staff member to process cash payment for customer.

Actor	Staff member
Trigger	Staff member initiates cash payment processing.
Pre-condition	- Staff member is logged into the system. - Customer chooses cash payment option.
Post-condition	Customer's cash payment is successfully processed.
Assumption	N/A
Dependency	Availability of cash payment processing tools and staff access rights in the system.

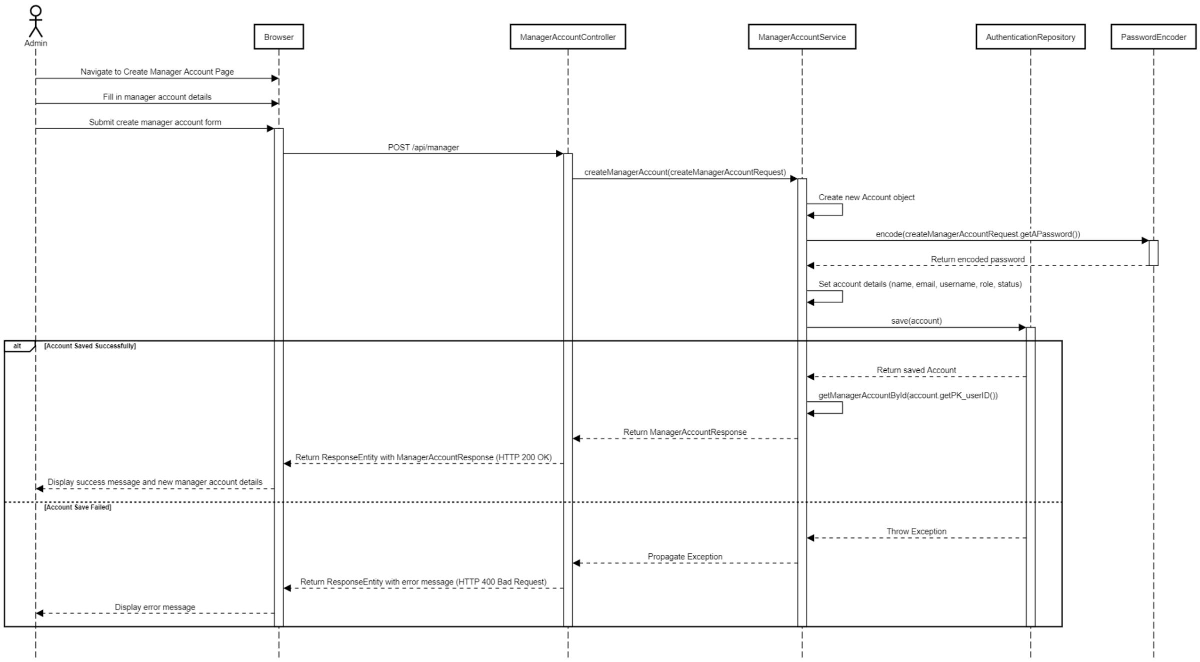
UC-26: Process cash payment for customer


Staff
Browser
OrderController
OrderHandlerService
Initiates cash payment processing
PATCH /api/order/cash-confirm (ConfirmCashPaymentRequest)
updateOrderStatusCash(request)
alt
[Payment is successful]
true
[Payment fails]
false
alt
[Payment successful response]
HTTP 200 OK ("Đơn hàng thanh toán thành công")
[Payment failed response]
HTTP 400 Bad Request ("Đơn hàng thanh toán thất bại")

Use Case: UC-27: Create manager account

Objective: Allow admin to create a manager account.

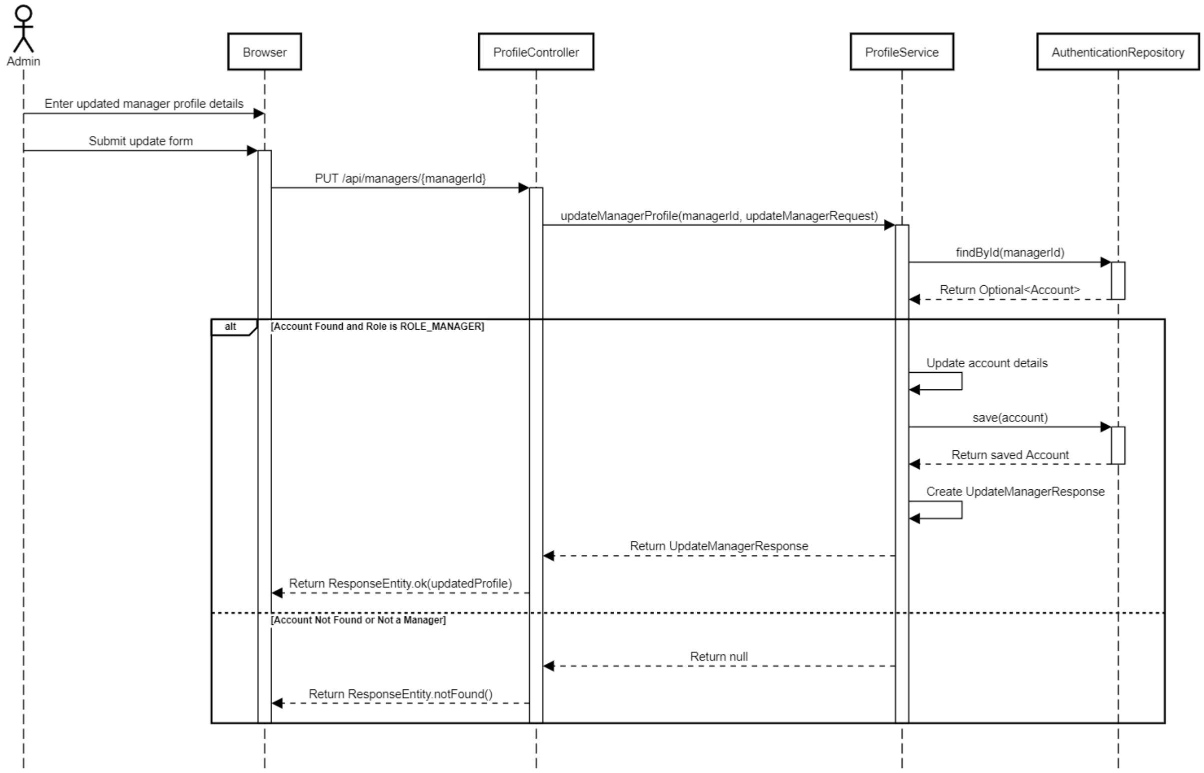
Actor	Admin
Trigger	Admin decides to create a new manager account.
Pre-condition	- Admin is logged into the system. - Authorization to create manager accounts is granted.
Post-condition	New manager account is successfully created and added to the system.
Assumption	N/A
Dependency	Availability of manager account creation tools and admin access rights in the system.


Admin
Browser
ManagerAccountController
ManagerAccountService
AuthenticationRepository
PasswordEncoder
Navigate to Create Manager Account Page
Fill in manager account details
Submit create manager account form
POST /api/manager
createManagerAccount(createManagerAccountRequest)
Create new Account object
encode(createManagerAccountRequest.getAPassword())
Return encoded password
Set account details (name, email, username, role, status)
save(account)
alt
[Account Saved Successfully]
Return saved Account
getManagerAccountById(account.getPK_userID())
Return ManagerAccountResponse
Return ResponseEntity with ManagerAccountResponse (HTTP 200 OK)
Display success message and new manager account details
[Account Save Failed]
Throw Exception
Propagate Exception
Return ResponseEntity with error message (HTTP 400 Bad Request)
Display error message

Use Case: UC-28: Update manager account details

Objective: Allow admin to update details of manager accounts.

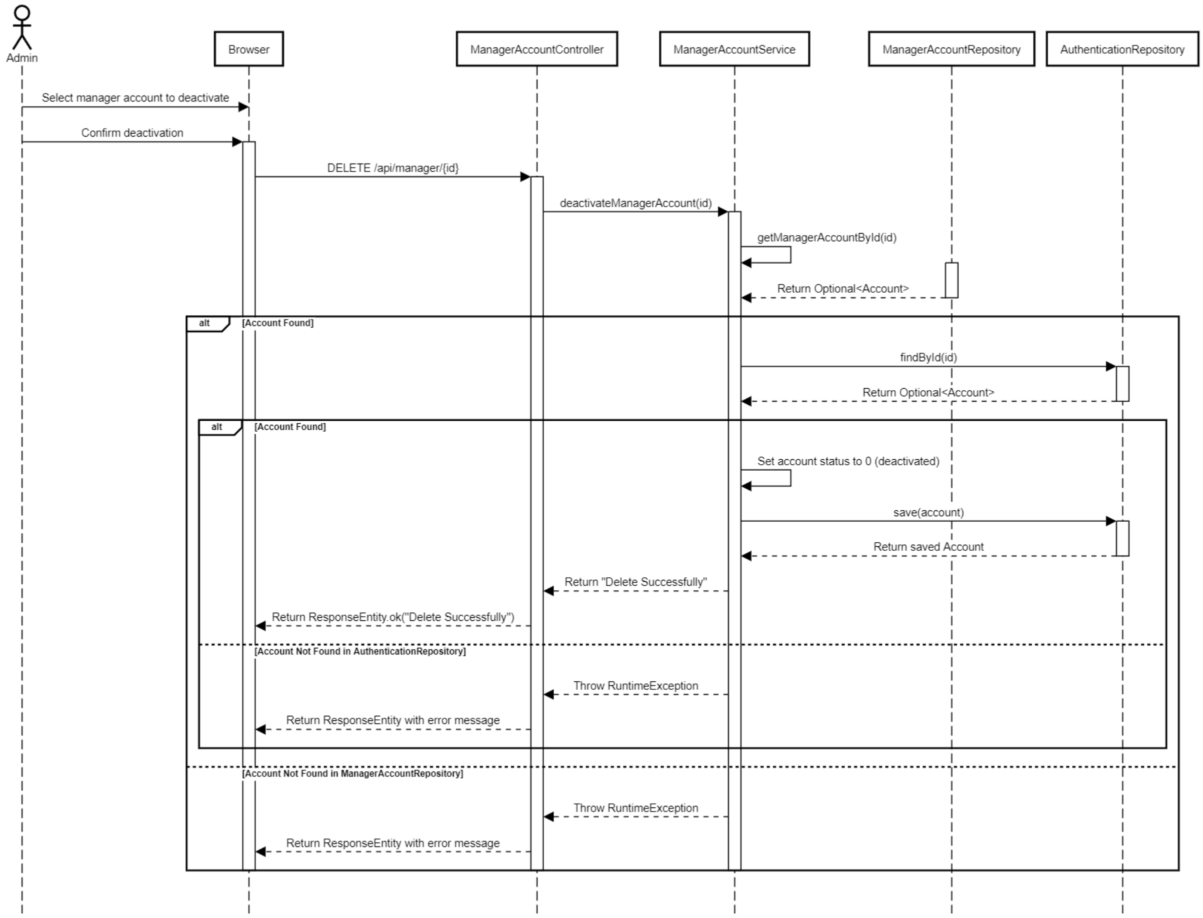
Actor	**Admin**
Trigger	Admin decides to modify details of an existing manager account.
Pre-condition	- Admin is logged into the system. - Manager account to be updated exists in the system.
Post-condition	Changes to manager account details (e.g., role, contact information) are successfully updated.
Assumption	N/A
Dependency	Availability of manager account update tools and admin access rights in the system.


Admin
Browser
ProfileController
ProfileService
AuthenticationRepository
Enter updated manager profile details
Submit update form
PUT /api/managers/{managerId}
updateManagerProfile(managerId, updateManagerRequest)
findById(managerId)
Return Optional<Account>
alt
[Account Found and Role is ROLE_MANAGER]
Update account details
save(account)
Return saved Account
Create UpdateManagerResponse
Return UpdateManagerResponse
Return ResponseEntity.ok(updatedProfile)
[Account Not Found or Not a Manager]
Return null
Return ResponseEntity.notFound()

Use Case: UC-29: Delete manager account

Objective: Allow admin to delete manager account.

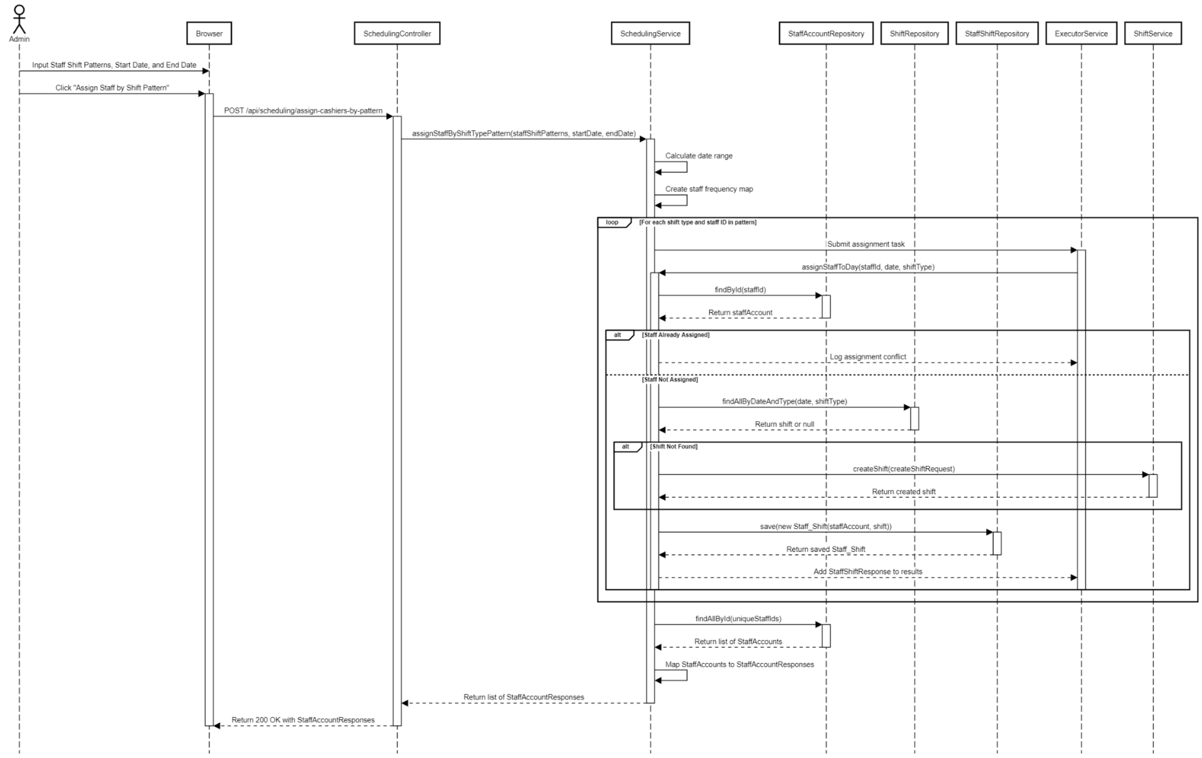
Actor	Admin
Trigger	Admin decides to remove an existing manager account from the system.
Pre-condition	- Admin is logged into the system. - Manager account to be deleted exists in the system.
Post-condition	Manager account is successfully deleted and no longer accessible in the system.
Assumption	N/A
Dependency	Availability of manager account deletion tools and admin access rights in the system.


Admin
Browser
ManagerAccountController
ManagerAccountService
ManagerAccountRepository
AuthenticationRepository
Select manager account to deactivate
Confirm deactivation
DELETE /api/manager/{id}
deactivateManagerAccount(id)
getManagerAccountById(id)
Return Optional<Account>
alt
[Account Found]
findById(id)
Return Optional<Account>
alt
[Account Found]
Set account status to 0 (deactivated)
save(account)
Return saved Account
Return "Delete Successfully"
Return ResponseEntity.ok("Delete Successfully")
[Account Not Found in AuthenticationRepository]
Throw RuntimeException
Return ResponseEntity with error message
[Account Not Found in ManagerAccountRepository]
Throw RuntimeException
Return ResponseEntity with error message

Use Case: UC-30: Assign staff to shift

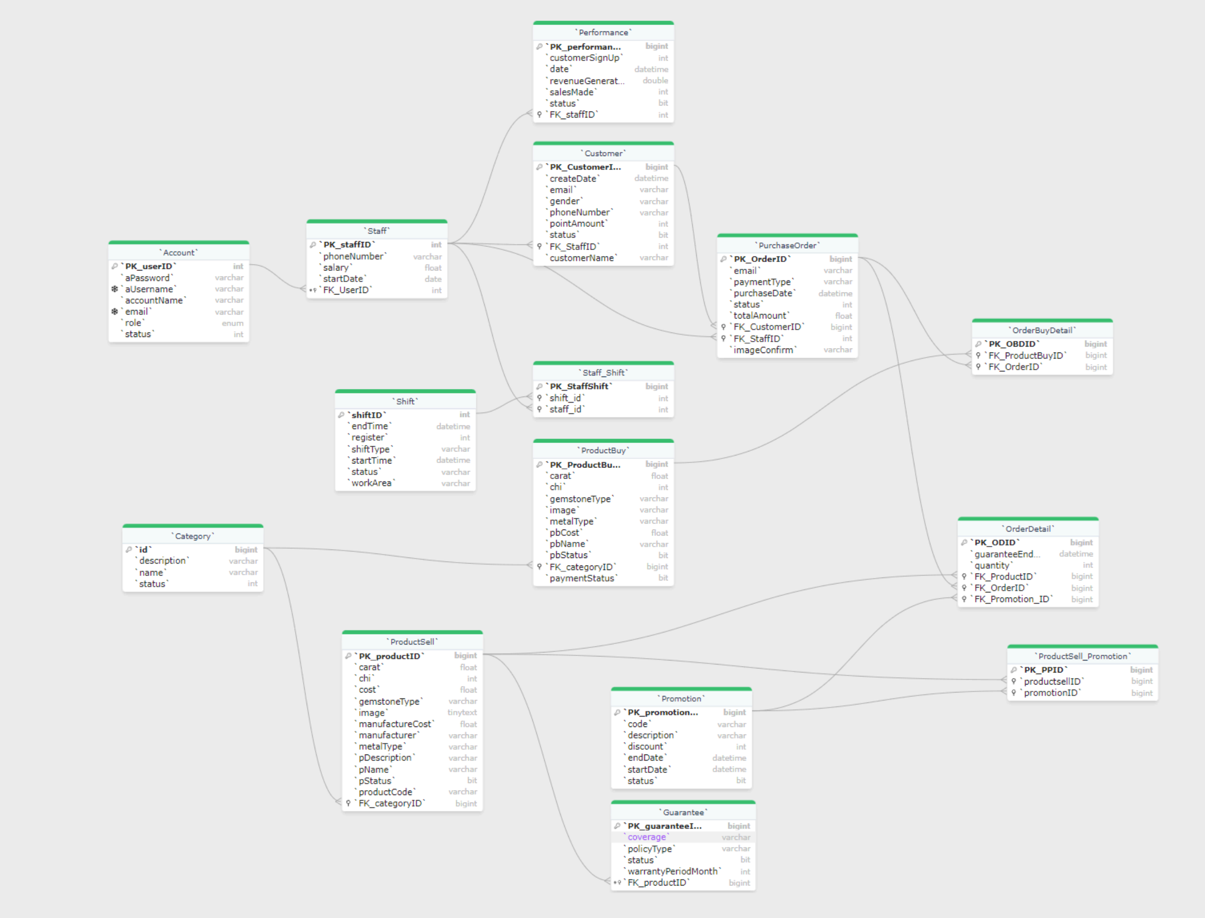
Objective: Allow admin to assign staff members to shifts.

Actor	Admin
Trigger	Admin decides to assign staff members to specific shifts.
Pre-condition	- Admin is logged into the system. - Shift scheduling functionality is available.
Post-condition	Staff members are successfully assigned to specified shifts.
Assumption	N/A
Dependency	Availability of shift assignment tools and admin access rights in the system.


Admin
Browser
SchedulingController
SchedulingService
StaffAccountRepository
ShiftRepository
StaffShiftRepository
ExecutorService
ShiftService
Input Staff Shift Patterns, Start Date, and End Date
Click "Assign Staff by Shift Pattern"
POST /api/scheduling/assign-cashiers-by-pattern
assignStaffByShiftTypePattern(staffShiftPatterns, startDate, endDate)
Calculate date range
Create staff frequency map
loop
[For each shift type and staff ID in pattern]
Submit assignment task
assignStaffToDay(staffId, date, shiftType)
findById(staffId)
Return staffAccount
alt
[Staff Already Assigned]
Log assignment conflict
[Staff Not Assigned]
findAllByDateAndType(date, shiftType)
Return shift or null
alt
[Shift Not Found]
createShift(createShiftRequest)
Return created shift
save(new Staff_Shift(staffAccount, shift))
Return saved Staff_Shift
Add StaffShiftResponse to results
findAllById(uniqueStaffIds)
Return list of StaffAccounts
Map StaffAccounts to StaffAccountResponses
Return list of StaffAccountResponses
Return 200 OK with StaffAccountResponses

4/ Database

4.1/ Data design



4.2/ Table Description

Entity Tables

No	Table	Description
01	Account	Describe account in the system
02	Category	Describe category of product in the system
03	Customer	Describe customer information in the system
04	Guarantee	Describe guarantee details in the system
05	OrderBuyDetail	Describe order buy detail in the system
06	OrderDetail	Describe order detail in the system
07	Performance	Describe performance metrics in the system
08	ProductBuy	Describe purchased product details
09	ProductSell	Describe sold product details

10	ProductSell_Promotion	Link products with promotions
11	Promotion	Describe promotion details in the system
12	PurchaseOrder	Describe purchase order information
13	Shift	Describe shift details
14	Staff	Describe staff information
15	Staff_Shift	Link staff with shifts

Attribute Data Dictionary

Entity	Attribute	Description	Domain	Null
Account	PK_userID	Primary key for user account	int	No
	aPassword	User's password	varchar(255)	Yes
	aUsername	User's username	varchar(255)	Yes
	accountName	Account name	varchar(255)	Yes
	email	Email associated with account	varchar(255)	Yes
	role	Role of the user	enum	Yes
	status	Status of the account	int	Yes
Category	id	Primary key for category	bigint	No
	name	Category name	varchar(255)	Yes
	description	Category description	varchar(255)	Yes

	status	Status of the category	int	Yes
Customer	PK_CustomerID	Primary key for customer	bigint	No
	createDate	Date when customer was created	datetime	Yes
	email	Customer's email	varchar(255)	Yes
	gender	Customer's gender	varchar(255)	Yes
	phoneNumber	Customer's phone number	varchar(255)	Yes
	pointAmount	Amount of loyalty points	int	No
	status	Status of the customer	bit	No
	FK_StaffID	Foreign key referencing staff	int	Yes
	customerName	Customer's name	varchar(255)	Yes
Guarantee	PK_guaranteeID	Primary key for guarantee	bigint	No
	coverage	Coverage details	varchar(255)	Yes
	policyType	Type of policy	varchar(255)	Yes
	status	Status of the guarantee	bit	No
	warrantyPeriodMonth	Warranty period in months	int	Yes
	FK_productID	Foreign key referencing product	bigint	Yes

OrderBuyDetail	PK_OBDID	Primary key for order buy detail	bigint	No
	FK_ProductBuyID	Foreign key referencing product buy	bigint	Yes
	FK_OrderID	Foreign key referencing purchase order	bigint	Yes
OrderDetail	PK_ODID	Primary key for order detail	bigint	No
	guaranteeEndDate	End date of guarantee	datetime	Yes
	quantity	Quantity of product	int	Yes
	FK_ProductID	Foreign key referencing product sell	bigint	Yes
	FK_OrderID	Foreign key referencing purchase order	bigint	Yes
	FK_Promotion_ID	Foreign key referencing promotion	bigint	Yes
Performance	PK_performanceID	Primary key for performance	bigint	No
	customerSignUp	Number of customer sign-ups	int	No
	date	Date of performance measurement	datetime	Yes
	revenueGenerated	Revenue generated	double	No
	salesMade	Number of sales made	int	No

	status	Status of performance measurement	bit	No
	FK_staffID	Foreign key referencing staff	int	Yes
ProductBuy	PK_ProductBuyID	Primary key for product buy	bigint	No
	carat	Carat weight of product	float	Yes
	chi	Chi value of product	int	Yes
	gemstoneType	Type of gemstone	varchar(255)	Yes
	image	Image of the product	varchar(255)	Yes
	metalType	Type of metal	varchar(255)	Yes
	pbCost	Cost of product buy	float	Yes
	pbName	Name of product buy	varchar(255)	Yes
	pbStatus	Status of product buy	bit	Yes
	FK_categoryID	Foreign key referencing category	bigint	Yes
	paymentStatus	Payment status	bit	Yes
ProductSell	PK_productID	Primary key for product sell	bigint	No
	carat	Carat weight of product	float	Yes

	chi	Chi value of product	int	Yes
	cost	Cost of product sell	float	Yes
	gemstoneType	Type of gemstone	varchar(255)	Yes
	image	Image of the product	tinytext	Yes
	manufactureCost	Manufacturing cost	float	Yes
	manufacturer	Manufacturer of the product	varchar(255)	Yes
	metalType	Type of metal	varchar(255)	Yes
	pDescription	Product description	varchar(255)	Yes
	pName	Product name	varchar(255)	Yes
	pStatus	Status of product	bit	Yes
	productCode	Product code	varchar(255)	Yes
	FK_categoryID	Foreign key referencing category	bigint	Yes
ProductSell_Promotion	PK_PPID	Primary key for product sell promotion	bigint	No
	productsellID	Foreign key referencing product sell	bigint	Yes
	promotionID	Foreign key referencing promotion	bigint	Yes

Promotion	PK_promotionID	Primary key for promotion	bigint	No
	code	Promotion code	varchar(255)	Yes
	description	Promotion description	varchar(255)	Yes
	discount	Discount percentage	int	Yes
	endDate	End date of promotion	datetime	Yes
	startDate	Start date of promotion	datetime	Yes
	status	Status of promotion	bit	No
PurchaseOrder	PK_OrderID	Primary key for purchase order	bigint	No
	email	Customer's email	varchar(255)	Yes
	paymentType	Payment type	varchar(255)	Yes
	purchaseDate	Date of purchase	datetime	Yes
	status	Status of purchase order	int	Yes
	totalAmount	Total amount of purchase	float	Yes
	FK_CustomerID	Foreign key referencing customer	bigint	Yes
	FK_StaffID	Foreign key referencing staff	int	Yes
	imageConfirm	Confirmation image	varchar(255)	Yes

Shift	shiftID	Primary key for shift	bigint	No
	endTime	End time of shift	time	Yes
	shiftDate	Date of shift	datetime	Yes
	startTime	Start time of shift	time	Yes
	status	Status of shift	bit	Yes
	FK_StaffID	Foreign key referencing staff	int	Yes
Staff	PK_staffID	Primary key for staff	bigint	No
	address	Staff address	varchar(255)	Yes
	startDate	Staff’s employment date	datetime	Yes
	email	Staff email	varchar(255)	Yes
	fullName	Staff’s full name	varchar(255)	Yes
	gender	Staff gender	varchar(255)	Yes
	image	Image of staff	varchar(255)	Yes
	phoneNumber	Staff’s phone number	varchar(255)	Yes
	salary	Staff’s salary	float	Yes
	status	Status of staff	bit	Yes

	FK_positionID	Foreign key referencing position	int	Yes
	staffStatus	Employment status	varchar(255)	Yes

5/ Software Testing Documentation

5.1/ Overall Description

5.1.1 Test Scope

Team has a plan to set up test case for main flow

5.1.2 Test Model

Team use Incremental testing for some approaches:

- Each module is required a role to use clearly
- Each module could test at the early stage of the develop process

5.1.3 Testing Levels

- Testing Level: User Acceptance Test
- Testing Types: Functional Testing, User Interface Testing

5.2/ Test Plan

5.2.1/ Test Stages

Type of Test	Test Level			
	Unit	Integration	System	Acceptance
Function Testing				

User Interface Testing				

6/ Non-Functional Requirement

6.1 External Interfaces

6.1.1 User Interfaces

- All the texts, labels and should be written in Vietnamese.

6.2 Quality Attributes

6.2.1 Usability

- Admin, staff should need less than one week of training to be productive in using the system.

6.2.2 Reliability

- Admin, staff should need training to be productive in using the system.

6.2.3 Performance

- All common response time is under 25 seconds

6.2.4 Design constraints

- Application use Vietnamese

6.2.5 Security

- Each role of user has a specific permission to interact with system