GAP Retail System Use Cases

Use Case Name: Customer Online Ordering

Primary Actor: Customer

ID: UC-1

Importance Level: High

Short Description:
The customer browses the website (navigating categories or using search), selects desired product variants (size, color), adds items to the cart, and completes an online purchase. System tracks customer purchases, processes payments, reserves/updates inventory, applies loyalty rewards/promotions, and records transaction details.

Trigger: The customer decides to purchase an item from GAP online.

Type: Transactional

Preconditions:

- GAP website is functional.
- Customer has an account (logged in) or chooses to check out as a guest.
- Product variant (SKU) is available in inventory.
- Payment system/gateway is functional.
- Customer has a valid payment method.

Major Steps Performed:

Major Steps	Information for Steps
1. Customer navigates GAP website (via menus, search), selects product, and chooses specific variant (size, color).	Input: Product/Variant selection (SKU), Quantity. Database Tables: `products`, `product_variants`, `categories`
2. Selected variant(s) are added to the shopping cart. System checks availability and displays cart subtotal/item count.	Input: SKU availability check. Output: Updated shopping cart view. Database Tables: `shopping_cart`, `shopping_cart_items`, `inventory_levels`
3. Customer proceeds to checkout. System prompts for login or guest checkout option.	Input: Checkout action. Database Tables: `customers`
4. (If logged in) Customer confirms/selects shipping address and payment method from saved options. System checks for applicable loyalty points/rewards.	Input: Customer confirmation. Output: Pre-filled details, available loyalty rewards. Database Tables: `addresses`, `loyalty_accounts`
5. (If guest checkout) Customer provides email, shipping address, and payment information.	Input: Guest details. Database Tables: `customers`, `addresses`
6. Customer chooses delivery/shipping method.	Input: Shipping method selection. Output: Calculated shipping cost, estimated delivery date. Database Tables: `shipments`
7. Customer reviews order (items, address, payment, total cost). Applies any promo codes or loyalty rewards.	Input: Promo code, loyalty redemption choice. Output: Final order summary with adjusted total. Database Tables: `promotions`, `order_promotions`
8. Customer confirms order and submits payment details.	Input: Final confirmation, payment details. Database Tables: `orders`
9. System reserves inventory for ordered items.	Output: Inventory reservation lock. Database Tables: `inventory_levels`
10. System sends payment details to Payment Gateway for authorization.	Input: Payment details, amount. Database Tables: `payments`, `payment_methods`
11. System receives authorization status from Payment Gateway.	Input: Authorization success/failure. Database Tables: `payments`, `payment_status`
12. (If authorized) System captures payment (or places authorization hold based on policy).	Output: Payment capture/hold transaction ID. Database Tables: `payments`
13. System generates Order ID, displays order confirmation on screen, and sends confirmation email.	Output: Order created in system, Order ID, Confirmation screen, Confirmation email. Database Tables: `orders`, `order_tracking`
14. Order details sent to Warehouse Management System for fulfillment (UC-4 trigger).	Output: Order fulfillment request sent. Database Tables: `orders`, `shipments`

Major Steps	Information for Steps
15. System updates sales data and relevant customer loyalty account (points earned).	Output: Sales records updated, Loyalty points updated. Database Tables: `sales_analytics`, `loyalty_accounts`

Alternate Courses / Exceptions:

- **Item becomes unavailable after adding to cart**: Before Step 8, system notifies user in cart, requires removal or 'save for later'. (Tables: `inventory_levels`, `shopping_cart_items`)
- **Address validation fails**: During Step 5 or 6, system prompts user to correct or confirm the provided shipping address. (Tables: `addresses`)
- **Payment Authorization Fails**: At Step 11, payment declined. System prompts customer to re-enter payment details or try a different method. Order is not confirmed. (Tables: `payments`, `payment_status`)
- **Fraud Check Failure**: After Step 12, order flagged for manual review. Status updated, confirmation may be delayed. (Tables: `orders`, `order_tracking`)
- **Customer Chooses In-Store Pickup - Original Alternate Course Modified**:
  - 3a. Customer proceeds to check out and chooses "In-Store Pickup".
  - 3b. System presents nearby stores with stock based on item availability and customer location/preference. Customer selects a store. (Tables: `stores`, `inventory_levels`)
  - Steps 4-5 proceed (address may be needed for verification/billing). Payment info required.
  - Step 13 includes pickup location information in confirmation. (Tables: `shipments`)
  - Step 14 sends order details to the selected store for pickup preparation. (Tables: `orders`, `shipments`)
- **Customer cancels order before shipment**: Customer initiates cancellation via website/CSR. If order not yet processed by warehouse, system voids order, releases inventory reservation, and processes refund/voids payment authorization. (Tables: `orders`, `order_tracking`, `inventory_levels`, `payments`)

Post Conditions:

- Order is recorded in the system with status 'Processing' (or similar). (Tables: `orders`, `order_tracking`)
- Inventory for ordered items is reserved. (Tables: `inventory_levels`)
- Customer receives order confirmation.

- Payment is captured or authorization hold is placed. (Tables: `payments`)
- Fulfillment process (UC-4) is initiated. (Tables: `shipments`)
- Sales analytics updated. (Tables: `sales_analytics`)

Use Case Name: Customer In-Store Purchase

Primary Actor: Customer (interacting with Sales Associate/Cashier)

ID: UC-2

Importance Level: High

Short Description:
Customer makes a purchase at a GAP retail store using a Point of Sale (POS) system.

Trigger: The customer decides to purchase item(s) at a GAP store.

Type: Transactional

Preconditions:

- GAP store is operational.
- POS system and inventory systems are operational.
- Customer has a valid payment method.
- Sales Associate/Cashier is logged into POS system.

Major Steps Performed:

Major Steps	Information for Steps
1. Customer selects items and brings them to the checkout desk.	Database Tables: `products`, `product_variants`
2. Associate scans item barcodes. System displays item details (name, price, SKU) and running total on POS.	Input: ProductID/SKU (via barcode scan). Output: Item details, running total. Database Tables: `products`, `product_variants`
3. Associate asks if customer is a loyalty member and looks up account (e.g., by phone number).	Input: Customer identifier. Output: Loyalty status, applicable rewards/discounts. Database Tables: `customers`, `loyalty_accounts`
4. Customer confirms purchase total and chooses payment method (Cash, Credit, Debit, Gift Card, Mobile).	Input: Payment method selection. Database Tables: `payment_methods`
5. Associate processes payment via selected method(s).	Input: Payment details (card swipe/tap/insert, cash amount). Database Tables: `payments`
6. System validates and processes payment via Payment Gateway (if applicable).	Output: Payment success/failure confirmation. Database Tables: `payments`, `payment_status`
7. System updates inventory levels for the store and records sales data.	Output: Inventory level updated, Sales record created. Database Tables: `inventory_levels`, `sales_analytics`
8. Associate asks customer preference for receipt (printed, emailed, none). System generates receipt.	Output: Receipt (printed/digital/none). Database Tables: `orders`, `order_items`
9. Associate deactivates/removes security tags (if applicable) and bags items.	

Alternate Courses / Exceptions:

- **Item not found/Unpriced**: Associate performs manual lookup or seeks supervisor assistance/override. (Tables: `products`, `product_variants`)
- **Price check/dispute**: Associate verifies price, may need manager approval for adjustment. (Tables: `products`, `product_variants`)
- **Payment method fails**: Customer prompted to retry or use alternative method. If no valid method, transaction terminated. (Tables: `payments`, `payment_status`)
- **Split Payment**: Customer uses multiple methods (e.g., gift card then credit card). Associate processes each sequentially. (Tables: `payments`, `payment_methods`)
- **Gift Card Payment**: System checks balance before applying. (Tables: `payment_methods`, `payments`)

- **Tax Exemption**: Associate applies tax-exempt status after verification. (Tables: `orders`)

Post Conditions:

- Order is recorded in the system. (Tables: `orders`, `order_items`)
- Store inventory is reduced. (Tables: `inventory_levels`)
- Customer receives goods and receipt (if requested).
- Payment is captured. (Tables: `payments`, `payment_status`)
- Loyalty points updated (if applicable). (Tables: `loyalty_accounts`)
- Sales analytics updated. (Tables: `sales_analytics`)

Use Case Name: Inventory and Product Management

Primary Actor: Inventory Manager, Merchandiser (Role may vary by step)

ID: UC-3

Importance Level: High

Short Description:
Manages product lifecycle (creation to discontinuation), monitors and adjusts stock levels across locations (warehouse, stores), and triggers restocking activities.

Trigger: New product introduction, product discontinuation, stock level changes (sales, returns), periodic inventory counts, manual adjustments required.

Type: Operational

Preconditions:

- Product Catalog system is available.
- Inventory Management system is linked to warehouse and store systems.
- Relevant user has appropriate permissions.

Major Steps Performed:

Major Steps	Information for Steps
1. (Product Introduction) Merchandiser defines new product attributes (name, desc, category), variants (SKUs, size, color, price), assigns Supplier, uploads images.	Input: Item lifecycle details, attributes, variant specifics. Output: New Product/SKUs created in system. Database Tables: `products`, `product_variants`, `product_categories`
2. (Product Discontinuation) Merchandiser updates product lifecycle status (e.g., to 'Clearance' or 'Discontinued'). System flags item as non-replenishable.	Input: Product ID/SKU, new status. Output: Updated lifecycle status. Database Tables: `products` (ProductLifecycleStatus)
3. System monitors inventory levels across all locations in near real-time.	Input: Store and warehouse inventory data feeds. Database Tables: `inventory_levels`
4. System automatically adjusts stock levels based on confirmed sales (UC-1, UC-2) or processed returns (UC-5).	Input: Sales and return transaction data. Output: Updated QuantityOnHand. Database Tables: `inventory_levels`
5. System detects low stock (below reorder point) for a SKU at a location.	Input: Current stock level vs. reorder point. Database Tables: `inventory_levels`, `product_variants` (LowStockThreshold)
6. System automatically generates restock request (to trigger UC-6 Supplier Order or internal Stock Transfer).	Output: Restock request / Purchase Order generated. Database Tables: `supplier_orders`, `warehouse_stock_transfer`, `store_inventory_replenishment`
7. (Stock Adjustment) Inventory Manager performs manual stock adjustment based on cycle count or discrepancy investigation.	Input: SKU, Location, Adjusted Quantity, Reason Code. Output: Updated QuantityOnHand, Adjustment Logged. Database Tables: `inventory_levels`

Alternate Courses / Exceptions:

- **Inventory count discrepancy**: System flags mismatch during cycle count. Requires investigation (step 7) and potential adjustment. (Tables: `inventory_levels`)
- **Product attribute update**: Merchandiser modifies details (e.g., price, description) for an existing product/SKU. (Tables: `products`, `product_variants`)
- **Clearance Strategy**: For discontinued products still in stock, manager defines clearance pricing/promotions. (Tables: `products`, `promotions`)
- **Inter-store/Warehouse transfer needed**: Manager manually initiates a Stock Transfer request between locations. (Tables: `warehouse_stock_transfer`, `store_inventory_replenishment`)

Post Conditions:

- Product lifecycle status reflects current phase. (Tables: `products`)

- Inventory records are current and accurate. (Tables: `inventory_levels`)
- Restock requests generated for low-stock items. (Tables: `supplier_orders`, `warehouse_stock_transfer`, `store_inventory_replenishment`)
- Manual adjustments are logged. (Tables: `inventory_levels`)

Use Case Name: Warehouse Fulfillment and Shipping

Primary Actor: Warehouse Staff (Picker, Packer, Shipper roles)

ID: UC-4

Importance Level: High

Short Description:
Handles received online order fulfillment including picking, packing, generating shipping labels, and dispatching orders via couriers.

Trigger: Order details received from Order Management System (from UC-1).

Type: Transactional

Preconditions:

- Order is successfully placed and payment authorized/captured.
- Warehouse has necessary stock allocated/reserved for the order.
- Warehouse Management System (WMS) is operational.

Major Steps Performed:

Major Steps	Information for Steps
1. WMS assigns order to appropriate warehouse based on stock availability, customer location, etc.	Input: OrderID, Customer Location. Output: Warehouse assignment. Database Tables: `warehouses`, `orders`, `inventory_levels`
2. WMS generates pick list for assigned order(s).	Input: Order details (SKUs, quantities). Output: Optimized pick list. Database Tables: `orders`, `order_items`, `product_variants`
3. Picker retrieves items from warehouse locations, confirming picks via scanner.	Input: Pick list. Output: Items collected, Pick confirmation scans. Database Tables: `inventory_levels` (for location)
4. Items moved to packing station. Packer verifies items against order, selects packaging, packs items.	Input: Picked items, Order details. Output: Packed box. Database Tables: `orders`, `order_items`
5. Packer generates shipping label (selects courier/service level, inputs weight/dims). System assigns tracking number.	Input: Order (shipping address), Package details. Output: Shipping label with Tracking Number. Database Tables: `shipments`, `addresses`
6. Packed order is moved to dispatch area. Courier picks up shipment.	Output: Order status updated to 'Shipped'. Database Tables: `orders`, `shipments`, `order_tracking`
7. System sends shipment confirmation and tracking details to customer (via email/account update).	Output: Tracking details notification sent. Database Tables: `shipments`, `customers`
8. System updates order status and decrements inventory levels.	Output: Order status 'Shipped', Inventory levels updated. Database Tables: `orders`, `order_tracking`, `inventory_levels`

Alternate Courses / Exceptions:

- **Item not found during picking**: Picker flags discrepancy in WMS. Initiates inventory investigation (UC-3 Alt Course). Order may be short-shipped or delayed. (Tables: `inventory_levels`, `orders`, `order_items`)
- **Order requires split shipment**: If items come from different warehouses or stock requires backorder, generate multiple shipments/labels linked to the same OrderID. (Tables: `shipments`, `shipment_items`)
- **Shipping delay by warehouse/courier**: System updates order status; customer may be notified. (Tables: `orders`, `order_tracking`, `shipments`)
- **Packaging constraints**: Packer follows special procedures for oversized/fragile items. (Tables: `product_variants` for dimensions/weight)

Post Conditions:

- Order is shipped to the customer. (Tables: `orders`, `shipments`)
- Order tracking is available to customer. (Tables: `shipments`, `order_tracking`)
- Inventory levels accurately reflect shipped stock. (Tables: `inventory_levels`)
- Order status is updated to 'Shipped'. (Tables: `orders`, `order_tracking`)

Use Case Name: Returns and Refunds Management

Primary Actor: Customer, Customer Service Representative (CSR), Sales Associate, Warehouse Staff

ID: UC-5

Importance Level: High

Short Description:
Handles customer requests for product returns or exchanges via multiple channels (online, in-store), processes refunds, and manages returned inventory.

Trigger: Customer initiates a return/exchange request (online, in-store, phone).

Type: Transactional

Preconditions:

- The original order exists in the system.
- Return request is within the defined return policy window/conditions.

Major Steps Performed:

Major Steps	Information for Steps
1. (Initiation) Customer requests return/exchange via chosen channel (online form, in-store visit, call to CSR).	Input: OrderID, Item(s) to return, Return Reason. Database Tables: `orders`, `order_items`
2. System checks eligibility against return policy (original order date, item eligibility, etc.).	Input: Return request details, Policy rules. Output: Eligibility Approved/Denied. Database Tables: `orders`, `order_items`
3. (If Approved Online/Phone) System generates return shipping label for customer.	Output: Return shipping label provided to customer. Database Tables: `returns`
4. (If In-Store) Associate accepts item(s), verifies against order/policy.	Input: Returned item(s), Order details. Database Tables: `orders`, `order_items`
5. (Item Receipt) Warehouse staff receives mailed return / Associate receives in-store return.	Input: Returned package/item(s). Database Tables: `returns`
6. Staff inspects returned item condition (resellable, damaged, defective).	Input: Returned item(s). Output: Condition assessment. Database Tables: `return_items` (ItemCondition)
7. Staff updates system with received status and inspection result.	Output: Return status updated, Inventory location updated. Database Tables: `returns`, `return_items`, `inventory_levels`
8. System determines appropriate inventory action (restock sellable item, mark down/dispose of damaged/defective).	Output: Inventory update request sent to Inventory system. Database Tables: `inventory_levels`
9. System processes refund (to original payment method, store credit) or initiates exchange order.	Output: Refund transaction processed, Exchange order created. Database Tables: `payments`, `orders` (if exchange)
10. System notifies customer of return completion / refund / exchange shipment.	Output: Customer notification. Database Tables: `returns`, `customers`

Alternate Courses / Exceptions:

- **Return window expired / Item ineligible**: Return request is denied. (Tables: `orders`, `returns`)
- **Product damaged beyond policy / Missing parts**: Partial or no refund given based on policy; customer notified. (Tables: `returns`, `return_items`, `payments`)
- **Return without receipt / Order lookup failed - In-Store**: Associate follows policy for non-receipted returns (e.g., store credit only, manager approval). (Tables: `returns`, `payments`)
- **Gift Return**: Refund typically issued as store credit/gift card. (Tables: `returns`, `payments`)

- **Exchange for Different Item**: Process return (steps 1-8), issue credit/refund (step 9), then initiate new sale (UC-2 or UC-1) for the different item. (Tables: `returns`, `orders`, `payments`)

Post Conditions:

- Returned product is restocked, marked down, or disposed of. (Tables: `inventory_levels`)
- Customer receives refund or exchange item. (Tables: `payments`, `orders` if exchange)
- Relevant inventory and financial records are updated. (Tables: `inventory_levels`, `payments`, `sales_analytics`)
- Return status is marked as 'Completed'. (Tables: `returns`)

Use Case Name: Promotions Application

Primary Actor: Promotion System

ID: UC-6

Importance Level: High

Short Description:
This use case describes how the system automatically evaluates and applies eligible promotions to a customer's shopping cart or order during checkout, based on promo codes, product SKUs, order value, and customer eligibility.

Trigger: A customer proceeds to checkout or updates their shopping cart.

Type: System Initiated

Preconditions:

- The customer has an active shopping cart with at least one product.
- There are active promotions in the system.
- The current date falls within the promotion's valid date range.

Major Steps Performed:

Major Steps	Information for Steps
1. Customer updates shopping cart or proceeds to checkout.	Input: Cart contents, optional promo code. Output: Updated cart data. Database Tables: `shopping_cart`, `shopping_cart_items`
2. System checks for active promotions that apply to cart (based on SKU, category, or customer).	Input: Cart items, customer ID, promotions table. Output: List of matching promotions. Database Tables: `promotions`, `products`, `product_categories`
3. System evaluates if each promotion's conditions are satisfied (e.g., minimum order value).	Input: Promotion rules, cart subtotal, item quantities. Output: List of eligible promotions. Database Tables: `promotions`, `shopping_cart_items`
4. If promo code is used, system checks if it matches a valid, active promotion requiring a code.	Input: PromoCode entered by customer. Output: PromoCode valid or rejected. Database Tables: `promotions` (PromotionCode)
5. System calculates the discount or applies BOGO logic to the applicable items.	Input: DiscountValue, BOGO rules, eligible SKUs. Output: Discounted prices or free items added. Database Tables: `promotions`, `shopping_cart_items`
6. System updates the cart and order summary to reflect the applied promotions.	Input: Cart object, updated item prices. Output: Updated cart total and visual indication of savings. Database Tables: `shopping_cart`
7. Applied promotions and details are recorded in the order record upon payment confirmation.	Input: Finalized order with promotions. Output: Order saved with promotion references. Database Tables: `orders`, `order_promotions`, `promotion_application`

Alternate Courses / Exceptions:

- **Promo code is invalid or expired**: System displays error message, no discount applied. (Tables: `promotions`)
- **Cart does not meet conditions**: System does not apply promotion. (Tables: `shopping_cart`, `shopping_cart_items`)
- **BOGO items not fully satisfied**: Free item not added or discounted. (Tables: `shopping_cart_items`, `promotions`)

Post Conditions:

- Cart and checkout summary reflect the applied promotions and total discount. (Tables: `shopping_cart`, `shopping_cart_items`)
- Final order record contains promotion ID(s) for reporting and auditing. (Tables: `orders`, `order_promotions`)

- Customer sees updated pricing and applied promo breakdown before confirming payment.

Use Case Name: Process Payments

Primary Actor: Customer

ID: UC-7

Importance Level: High

Short Description:
This use case outlines how a customer completes a payment for an order using a selected method and how the system handles and records that payment.

Trigger: Customer proceeds to checkout and submits payment information.

Type: External

Preconditions:

- The customer has an order ready for checkout.
- Payment gateway integration is active.
- Valid payment methods are configured in the system.

Major Steps Performed:

Major Steps	Information for Steps
1. Customer reviews their order at checkout.	Input: Items in cart, shipping information. Output: Order summary displayed. Database Tables: `shopping_cart`, `shopping_cart_items`, `addresses`
2. Customer selects a payment method.	Input: Select Credit card, Debit Card, PayPal, Gift Card. Output: Payment method chosen. Database Tables: `payment_methods`
3. Customer enters payment details.	Input: Card info, PayPal login or gift card code. Output: Payment information captured. Database Tables: (temporarily stored, not persisted)
4. System submits payment to the payment gateway.	Input: Payment info/total amount. Output: Payment authorization request. Database Tables: `payment_methods`
5. Gateway responds with success or failure.	Input: Gateway transaction result. Output: Success/failure message. Database Tables: `payment_status`
6. System updates payment status and saves record.	Input: Payment result. Output: Record in payment tables. Database Tables: `payments`, `payment_status`
7. System generates order confirmation.	Input: Payment confirmed. Output: Order ID and confirmation shown to customer. Database Tables: `orders`, `order_tracking`

Alternate Courses / Exceptions:

- **Invalid or expired payment method**: Error message and return to step 2. (Tables: `payment_methods`, `payments`)
- **Payment gateway failure**: System retries or asks user to re-enter info. (Tables: `payments`, `payment_status`)
- **Delayed payment (e.g., e-transfer)**: System marks as "Pending". (Tables: `payments`, `payment_status`, `orders`)

Post Conditions:

- Payment status is stored as "Paid", "Pending", or "Failed" in the database. (Tables: `payments`, `payment_status`)
- Order is either confirmed (on success) or flagged for retry (on failure). (Tables: `orders`, `order_tracking`)
- Payment method is saved in the system for reporting and reconciliation. (Tables: `payments`, `payment_methods`)
- Transaction reference is stored for future lookup. (Tables: `payments`)