Demonstrate ability to customize the shopping cart

Describe how to implement shopping cart rules.

DB tables:

- `salesrule`
  - name, description, is_active, is_rss
  - from_date, to_date, conditions_serialized, is_advanced, sort_order, product_ids
  - actions_serialized, simple_action, discount_amount, discount_qty, discount_step, apply_to_shipping
  - stop_rules_processing, uses_per_customer, times_used
  - coupon_type, use_auto_generation
- `salesrule_label` - label translation
  - rule_id, store_id, label
- `salesrule_product_attribute`
  - rule_id, website_id, customer_group_id, attribute_id

Multiple selection:

- `salesrule_website`
- `salesrule_customer_group`
- `salesrule_coupon`
  - rule_id, code, usage_limit, usage_per_customer, times_used, expiration_date, is_primary, created_at, type

Usage tracking:

- `salesrule_coupon_usage`
  - coupon_id, customer_id, times_used
- `salesrule_customer`
  - rule_id, customer_id, times_used
- `salesrule_coupon_aggregated` , `salesrule_coupon_aggregated_updated` , `salesrule_coupon_aggregated_order` - reporting stuff

Notes:

- ui component `sales_rule_form`
- conditions - html content block -*generic form* Edit\Tab\Conditions
- conditions fieldset renderer \Magento\Rule\Block\Conditions.render
- `$element->getRule()->getConditions()->asHtmlRecursive();`
- Magento\SalesRule\Model\Rule extends Magento\Rule\Model\AbstractModel
  - store labels
  - coupon generation
- Conditions model instance - \Magento\SalesRule\Model\Rule\Condition\Combine, extends getNewChildSelectOptions:
  - \Magento\SalesRule\Model\Rule\Condition\Product\Found - Product attribute combination
  - \Magento\SalesRule\Model\Rule\Condition\Product\Subselect - Products subselection
  - Conditions combination - self, recursion

    - \Magento\SalesRule\Model\Rule\Condition\Address - Cart Attribute: base_subtotal, total_qty, weight, shipping_method, postcode, region, region_id, country_id

what is:

- `is_advanced` column - not used
- `product_ids` column - not used
- `salesrule_product_attribute` table - attributes currently used in conditions and actions of rule.

  Used in plugin to \Magento\Quote\Model\Quote\Config: `getProductAttributes` to ensure all needed attributes are loaded with quote.

    - quote.load()
    - quoteResource. `_assignProducts`
    - product collection `->addAttributeToSelect($this->_quoteConfig->getProductAttributes())`.

What is the difference between sales rules and catalog rules?

- Catalog rules update product final price, they are visible in all catalog - category listing, product page.
- Sales rule is visible only in checkout and can require a coupon.
- Sales rules can affect shipping price
- Sales rules can trigger on products combination

How do sales rules affect performance?

Performance is not affected, all calculation happens in totals collectors.

What are the limitations of the native sales rules engine?

Only one coupon can be applied.

Implementing rules:

- Rule model
    - getConditionsInstance - will be used in form as `$model->getConditions()->asHtmlRecursive()`. Usually starting conditions are composite - selector [all/any] and sub-conditions selection.
- Composite conditions model
    - getNewChildSelectOptions - used by Consition\Combine.getNewChildElement

Describe add-to-cart logic in different scenarios.

Add to cart

cart model - used only on frontend, marked deprecated.

- checkout/cart/add [product, qty, related_product]

    - cart model.addProduct
        - filter request here, can register own via argument for \Magento\Checkout\Model\Cart\RequestInfoFilter
        - by default filtered out `form_key` and `custom_price`
    - quote.addProduct
    - product type instance. `prepareForCartAdvanced`
    - get same quote item or create new
    - \Magento\Quote\Model\Quote\Item\Processor::prepare
        - quote item.addQty
        - support request.customPrice -> quote item.customPrice
    - event `checkout_cart_product_add_after`
    - checkout session.setLastAddedProductId
    - cart.save
        - quote collect totals
        - event `checkout_cart_save_after`
        - reinitialize state - remove addresses and payments
    - event `checkout_cart_add_product_complete`
- checkout/cart/updatePost

    - cart.suggestItemsQty => \Magento\CatalogInventory\Model\StockStateProvider::suggestQty - qty increments, min/max qty
    - cart.updateItems
        - events `checkout_cart_update_items_before`, `checkout_cart_update_items_after`
        - quote item.setQty – triggers stock validation
    - cart.save

controller checkout/cart/add:

- cart.addProduct:
    - quote.addProduct: type. `prepareForCartAdvanced`, event `sales_quote_product_add_after`
    - event `checkout_cart_product_add_after`
- event `checkout_cart_add_product_complete` – this event only when normal add to cart
    - withlist observer, marked to be deleted

add to withlist `wishlist/index/add`:

- same buy request
- cart candidates = type instance.processConfiguration -> `_prepareProduct`. Same as with add to cart.
- create/update qty withlist item, just like quote item
- event `wishlist_product_add_after`
- event `wishlist_add_product`

add to cart from wishlist `withlist/index/cart`:

Can override original buyRequest when adding to cart. NO event *checkout_cart_add_product_complete*.

- load withlist item and options
- merge buy request - saved and current.
- wishlist item.addToCart

- `cart.addProduct` with merged buyRequest:
  - events `sales_quote_product_add_after`, `checkout_cart_product_add_after`
- cart.save
  - event `checkout_cart_save_before`
  - event `checkout_cart_save_after`
- quote.collectTotals
- \Magento\Wishlist\Helper\Data::calculate
  - write to customer session
  - event `wishlist_items_renewed`

merge quotes controller `customer/account/loginPost`:

Cart model is not used at all.

- \Magento\Customer\Model\Session::setCustomerDataAsLoggedIn
- event `customer_login`
- \Magento\Checkout\Observer\LoadCustomerQuoteObserver
- checkout session. `loadCustomerQuote`:
  - event `load_customer_quote_before`
  - finds existing customer quote
  - detects that guest quote <> customer quote
  - customer quote. `merge` (guest quote):
    - event `sales_quote_merge_before`
    - for each quote item. `compare` - product id, options same
    - if items same, qty++
    - if guest item is new, clone quote item, quote.addItem, event `sales_quote_add_item`
    - event `sales_quote_merge_after`
  - *delete guest quote*

reorder controller `sales/order/reorder`:

Like normal add to cart, but NO event*checkout_cart_add_product_complete*.

- load order, ensure can view
- for each order item, `cart.addOrderItem` - converts order item to quote item:
  - load product
  - get item option buyRequest
  - `cart.addProduct` with old buy request
    - events `sales_quote_product_add_after`, `checkout_cart_product_add_after`

Describe how to customize the process of adding a product to the cart.

- plugin over product type `prepareForCartAdvanced`
- event `catalog_product_type_prepare_full_options` - custom options in `_prepareOptions`
- plugin over \Magento\Quote\Model\Quote\Item\Processor::prepare - qty, custom price
- event `checkout_cart_product_add_after`

Which different scenarios should you take into account?

- add to cart from catalog

- add to cart from wishlist
- move all wishlist to cart
- merge quote when existing customer has quote, then shops as a guest and logs in
- admin create order
- admin reorder
- configure added product - change custom options

Render product types

checkout cart index:

- `<update handle="checkout_cart_item_renderers"/>`
- `<update handle="checkout_item_price_renderers"/>`
- block \Magento\Checkout\Block\Cart\AbstractCart::getItems - quote visible items
    - can limit with *pagination*
    - *custom_items* can override
- abstract cart. `getItemHtml` (quote item)
- render via abstract cart. `getItemRenderer` by product type
    - renderer child block `renderer.list` \Magento\Framework\View\Element\RendererList
    - renderer list block.getRenderer(type, template)
    - finds child block by alias = product type
    - \Magento\Checkout\Block\Cart\Item\Renderer
    -

Customizations:

- provide cart block data arguments `<referenceBlock name="checkout.cart.form">`:
    - `renderer_list_name` - use custom renderer list
    - `overriden_templates` - array by product type
    - `renderer_template` - regardless of product type
- in `checkout_cart_item_renderers`, `<referenceBlock name='renderer.list'>`, add child blocks by alias = product type. E.g.

```
<referenceBlock name="checkout.cart.item.renderers">
    <block class="Magento\Checkout\Block\Cart\Item\Renderer"
name="checkout.cart.item.renderers.default" as="default"
template="Magento_Checkout::cart/item/default.phtml">
        <block class="Magento\Checkout\Block\Cart\Item\Renderer\Actions"
name="checkout.cart.item.renderers.default.actions" as="actions">
            <block class="Magento\Checkout\Block\Cart\Item\Renderer\Actions\Edit"
name="checkout.cart.item.renderers.default.actions.edit"
template="Magento_Checkout::cart/item/renderer/actions/edit.phtml"/>
            <block class="Magento\Checkout\Block\Cart\Item\Renderer\Actions\Remove"
name="checkout.cart.item.renderers.default.actions.remove"
template="Magento_Checkout::cart/item/renderer/actions/remove.phtml"/>
        </block>
    </block>
</referenceBlock>
```

Describe the available shopping cart operations.

Configure product in cart - controller `checkout/cart/configure`

Product view and product configure are rendered via same helper \Magento\Catalog\Helper\Product\View.prepareAndRender.

The only difference is params:

- category_id - can be `false`
- configure_mode - optional
- buy_request - optional
- specify_options - error message if missing options

Normal product view:

- params[category_id] = current
- params[specify_options] - from product type
- helper product view.prepareAndRender
  - \Magento\Catalog\Helper\Product::initProduct - load product, check visible/enabled
    - event `catalog_controller_product_init_before`
    - event `catalog_controller_product_init_after`
  - event `catalog_controller_product_view`

Configure product:

- params[`buyRequest`] = quote item.buyRequest
- helper product view.prepareAndRender
  - product helper.initProduct
  - product helper. `prepareProductOptions` - set default selections from buy request, e.g. selected size=XXL
    - product.processBuyRequest, `product type.processBuyRequest`, `product type.checkProductConfiguration`
    - product. `setPreconfiguredValues`
  - product.setConfigureMode – used to show all hidden customization options instantly in edit mode
  - event `catalog_controller_product_view`

product.getPreconfiguredValues is later used:

- default product qty = product.preconfigured values.qty
- custom option values = product.preconfigured values.option_{$id}

How do you add a field to the shipping address?

Quote address extends AbstractExtensibleModel, and so should implement `getCustomAttributesCodes()` to add custom attributes. Custom attributes should load and save automatically, assuming you added a column in migration and registered custom attribute in plugin (see below).

Quote address custom attributes:

- community \Magento\Quote\Model\Quote\Address\CustomAttributeList - empty, use plugin to add
- EE \Magento\CustomerCustomAttributes\Model\Quote\Address\CustomAttributeList::getAttributes

- customer address attributes + customer attributes

Alternatively, you can always register extension attribute and load/save it manually.