Demonstrate ability to utilize themes and the template structure

app/design/frontend/Vendor/themename/

```
- composer.json
- registration.php
- theme.xml - name, parent, logo
- etc/view.xml - currently catalog images configuration
- i18n/
- media/ - pub logo here
- Magento_Checkout/ - extend specific module
---- layout/ - normally layouts are extended
------- override/base/ - special case: completely replace base layout for module
------- override/theme/Magento/luma/ - special case: completely replace specific
theme layout
---- templates/ - replace module files
---- web/ - replace module files
- web/ - replace theme files
---- js/
---- css/
------- source/
---- fonts/
---- images/
---- i18n/en_US/ - locale-specific file replacement (without module)
---- i18n/en_US/Magento_Checkout/ - locale-specific module file replacement
```

- Magento\Theme\Model\View\Design implements View\DesignInterface
- View\Model\Layout\Merge implements View\Layout\ProcessorInterface

Layouts:

Normally layouts are merged, extending parent ones.

To completely replace *original module layout* file, e.g. to replace vendor/magento/module-checkout/view/frontend/layout/default.xml place new file in Magento_Checkout/layout/override/base/default.xml

To completely replace some *parent theme's layout* customization, e.g. to replace vendor/magento/theme-frontend-luma/Magento_Checkout/layout/default.xml place new file in Magento_Checkout/layout/override/Magento/luma/default.xml

It is *not possible* to use override trick inside *module*, only in theme. You cannot create new module app/code/MyVendor/MyModule/view/frontend/layout/override/… and replace layout files.

Locale:

You can replace static files with locale-specific - JS, logo, font etc. You have 2 places:

- non-module specific, e.g. theme asset - web/i18n/en_US/
- from some module - web/i18n/en_US/Magento_Checkout/

how is theme loaded

- plugin magento-store/etc/di.xml
- App\Action\AbstractAction.beforeDispatch
- View\DesignLoader::load

- App\Area::load('design')
- App\Area::_initDesign
    - singleton View\DesignInterface = \Magento\Theme\Model\View\Design
    - design.setArea('frontend')
    - design.getConfigurationDesignTheme
    - design.setDefaultDesignTheme
        - get config `design/theme/theme_id` (website scope if single store mode, otherwise store view scope)
        - EE only: if theme is not configured, use default from DI argument Magento/luma
    - design.setDesignTheme - area, theme model. loads theme model from DB
        - View\Design\Theme\FlyweightFactory::create
        - View\Design\Theme\ThemeProviderInterface::getThemeByFullPath Has 2 implementations, real DB and fake data collection - Can be used to deploy static or in other setup commands, even if Magento is not installed yet.

singleton *View\DesignInterface* = \Magento\Theme\Model\View\Design - holds current area, locale and theme model

- get/setDesignTheme = theme
- get/setArea
- getLocale
- getConfigurationDesignTheme
- setDefaultDesignTheme
- getThemePath
- getDesignParams - area, theme, locale

View\Design\ThemeInterface = \Magento\Theme\Model\Theme - this is also DB model

- getArea
- getCode
- getParentTheme
- getInheritedThemes
- getId
- getThemePath
- getFullPath
- isPhysical

Theme type: physical, virtual, staging

Physical themes

Physical refers to the fact that those themes are defined by files. For example, the blank and luma theme are physically defined under app/design/frontend/

Virtual themes

This is yet unclear but I think virtual themes refer to themes you can create in the backend which extends existing physical themes but it seems like it's not fully implemented yet. You can see that there's two tabs available in the edit section only for virtual themes which let you provides custom CSS and JS for a virtual theme. I reckon a virtual theme would be something you setup temporarily (like Christmas theme) for a short period of theme and which requires only few design changes compared to the physical theme it extends.

Staging

Must be something about EE campaign.

how loadLayoutUpdates works

- Loads ALL layout update files using file name as layout handle name - as XML object.
- Selects only currently relevant handles (e.g. 'default', 'checkout_cart_index').
- Finds instructions, merges them recursively.
- Selected file contents as added to `_updates[]` array as pieces of string - they will be converted to XML object later in generateLayoutXml.

Trace:

View\Layout\Builder::loadLayoutUpdates

- event `layout_load_before`
- View\Model\Layout\Merge::load
  - Collects loaded layout files contents into `updates[]` array.
  - Loads from cache when possible, XML contents (e.g. '<block name="someBlock" …> <referenceBlock…>') and separately pageLayout (e.g. '1column')

merge.load(handle)

- merge._merge(handle)
  - Called for every page handle, e.g. 'default', 'checkout_cart_index'.
  - Checks cyclic dependency - logs error in dev mode
  - _fetchPackageLayoutUpdates
  - _fetchDbLayoutUpdates
- merge._validateMergedLayout
  - View\Model\Layout\Update\Validator::isValid with `urn:magento:framework:View/Layout/etc/layout_merged.xsd`

_fetchPackageLayoutUpdates

- merge.getFileLayoutUpdatesXml - loads ALL layouts regardless of handle
  - merge._loadFileLayoutUpdatesXml
    - get physical theme, checking parents until found
    - update files = theme file source.getFiles + page layout file source.getFiles
    - read files one by one
    - merge._substitutePlaceholders - replaces in raw XML string - {{baseUrl}} and {{baseSecureUrl}}
    - adds file contents to result as one of:

```
<layout id="{$fileName}" {$attributes}>{$fileContents}</layout>
<handle id="{$fileName}" {$attributes}>{$fileContents}</handle>
```

    - all are wrapped in outer tag

```
      <layouts xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <layout id="default" {$attributes}>{$fileContents}</layout>
        <handle id="checkout_cart_index" {$attributes}>{$fileContents}
</handle>
        <handle id="checkout_cart_index">{$fileContents}</handle>
        ...
      </layouts>
```

- finds all and with ID *matching handle* that is being loaded

    - merge._fetchRecursiveUpdates - finds xpath , calls _merge recursively
    - merge.validateUpdate plugin \Magento\PageCache\Model\Layout\MergePlugin::beforeValidateUpdate checks that entity-specific update handles don't have `ttl` declarations
    - merge.addUpdate – main goal, adds content to `updates[]`

_fetchDbLayoutUpdates:

- does nothing itself, but there's a plugin \Magento\Widget\Model\ResourceModel\Layout\Plugin::aroundGetDbUpdateString - loads updates from DB by theme/store
- merge._fetchRecursiveUpdates - finds xpath , calls _merge recursively
- merge.validateUpdate
- merge.addUpdate

Layout file source

merge.*fileSource*: View\Layout\File\Collector\Aggregated\Proxy - Source of layout files aggregated from a theme and its parents according to merging and overriding conventions

- add *base files* - modules

    - View\File\Collector\Decorator\ModuleDependency as layoutFileSourceBaseSorted
    - View\File\Collector\Decorator\ModuleOutput as layoutFileSourceBaseFiltered
    - *View\File\Collector\Base* as layoutFileSourceBaseFiltered, subDir = 'layout'
    - Component\DirSearch::collect(type='module', pattern) - uses component registrar to get modules, then searches by pattern

```
    <modules>/view/base/layout/*.xml --- isBase = true, shared between adminhtml
and frontend themes
    <modules/view/frontend/layout/*.xml
```

- get inherited themes

for each theme (e.g. Magento/blank, then Magento/luma):

- add theme files

    - View\File\Collector\Decorator\ModuleDependency as layoutFileSourceThemeSorted
    - View\File\Collector\Decorator\ModuleOutput as layoutFileSourceThemeFiltered
    - *View\File\Collector\ThemeModular* as layoutFileSourceTheme, subDir = 'layout'

```
    <theme>/*_*/layout/*.xml -- sets module context, e.g. Magento_Checkout
```

- replace override base files

    - View\File\Collector\Decorator\ModuleDependency as layoutFileSourceOverrideBaseSorted
    - View\File\Collector\Decorator\ModuleOutput as layoutFileSourceOverrideBaseFiltered
    - *View\File\Collector\Override\Base* as layoutFileSourceOverrideBase, subDir = 'layout/override/base'

```
    <theme>/*_*/layout/override/base/*.xml
```

- View\File\FileList::replace
- View\File\FileList\Collator::collate - replaces by matching keys: is base, theme, module, filename

- replace override theme files

    - View\File\Collector\Decorator\ModuleDependency as layoutFileSourceOverrideThemeSorted
    - View\File\Collector\Decorator\ModuleOutput as layoutFileSourceOverrideThemeFiltered
    - *View\File\Collector\Override\ThemeModular* as layoutFileSourceOverrideTheme, subDir = 'layout/override/theme'

```
    <theme>*_*/layout/override/theme/*/*/*.xml
```

Page layout file source

All same as layout, but subDir 'page_layout':

```
    <modules>/view/base/page_layout/*.xml --- isBase = true, shared between
adminhtml and frontend themes
    <modules/view/frontend/page_layout/*.xml
    <theme>/*_*/page_layout/*.xml -- sets module context, e.g. Magento_Checkout
    <theme>/*_*/page_layout/override/base/*.xml
    <theme>*_*/page_layout/override/theme/*/*/*.xml
```

View\Layout\ProcessorInterface = View\Model\Layout\Merge

- $updates - array of all found string XML file contents E.g.

```
 updates[] = '<body><block name="someBlock"...'
 updates[] = '<referenceBlock name="someBlock">...'
```

- addUpdate(string) - string XML from file

Demonstrate the ability to customize the Magento UI using themes.

When would you create a new theme?

- create new theme from scratch without parent when design is very different from existing
- inherit new theme to add smaller customizations - move, hide, reorder elements, change block arguments, html attributes
- new theme can be assigned to specific store view, for example for b2b store
- theme can apply dynamically based on browser user agent as exception - enter regexp in *Content* > *Design* > *Implementation* > *[Edit]* > *Design Rule* > *User Agent Rules*Full page cache and design exception:

```
    plugin magento-store/etc/di.xml:
    Magento\Framework\App\Action\AbstractAction.beforeDispatch:
    \Magento\Framework\View\DesignLoader::load
      \Magento\Framework\App\Area::load('design')
        \Magento\Framework\App\Area::_initDesign
      \Magento\Framework\App\Area::detectDesign
        \Magento\Framework\App\Area::_applyUserAgentDesignException
        \Magento\Framework\View\DesignExceptions::getThemeByRequest

    plugin \Magento\PageCache\Model\App\CacheIdentifierPlugin::afterGetValue -
checks rule design exception
    View\DesignExceptions::getThemeByRequest
```

- etc/view.xml file *doesn't merge* with parent themes, it either inherits completely from parent or your theme has new version of it

How do you define theme hierarchy for your project?

theme.xml - parent

Determine theme hierarchy of existing project:

- Go to *Content* > *Design* > *Configuration*
- Check "Theme Name" column on row with store view under question
- find paths to all available themes in app/etc/design/frontend/* or vendor/*theme-* (conventionally) or find programmatically:

```
  (new \Magento\Framework\Component\ComponentRegistrar())->getPaths('theme');
```

- open theme path/theme.xml, check `<parent>` node and so on

Demonstrate the ability to customize/debug templates using the template fallback process.

How do you identify which exact theme file is used in different situations?

How can you override native files?

- theme layout override module file: /layout/override/base/*.xml
- theme layout override one of parent theme layouts:/layout/override/Magento/luma/*.xml
- static asset for theme, e.g. /web/js/theme.js
- static asset for module, e.g. /Magento_Checkout/web/js/view/minicart.js
- static asset for locale:

```
  <theme>/web/i18n/en_US/Magento_Checkout/
  <theme>/web/i18n/en_US/
```