Demonstrate ability to process URLs in Magento

Describe how Magento processes a given URL.

urlBuilder - \Magento\Framework\UrlInterface: getUrl

Instances:

- \Magento\Framework\Url
- \Magento\Backend\Model\Url

1. preprocess route params

   \Magento\Framework\Url*RouteParamsPreprocessorInterface*::execute

   - composite - delegates to many instances
   - staging - Preview\RouteParamsPreprocessor
     - if frontend and preview, adds `?___version`
     - if store code in URL, adds `?___store`
2. Disable caching if object found in route params

3. *createUrl* (may be cached in memory)

   - stash all parameters `_fragment`, `_escape`, `_escape_params`, `_query`, `_nosid`
   - \Magento\Framework\Url::*getRouteUrl* - without any system params
   - then compile ?query and #fragment params
4. run modifier

   \Magento\Framework\Url\ModifierInterface::execute - mode entire/base

   - composite - delegates to many instances
   - staging - Url\BaseUrlModifier - only base mode In frontend and preview, replaces host part with `$_SERVER[HTTP_HOST]`

getRouterUrl:

- `_direct` option = baseUrl + `_direct`
- $routeName/$controllerName/$actionName/$param1/$value1/…

How do you identify which module and controller corresponds to a given URL?

- first part is route name. Search routes.xml for route with matching*ID*
- modules for this ID are sorted with "before" and "after"
- controller/action classes is searched in matched modules

What is necessary to create a custom URL structure?

- register custom router, e.g. Magento\Robots\Controller\Router
- create rewrite record for specific URL

Describe the URL rewrite process and its role in creating user-friendly URLs.

Router `urlrewrite`:

- `?____from_store` param, redirect to new URL if necessary.

  Example:

  - on English store category page /shoes switching to Norwegian store
  - _/no/shoes?___*from_store=1*
  - find new rewrite for norwegian store
  - 302 redirect to */no/sko*
- find rewrite by request path

- redirect if necessary

- return forward action - mark request not dispatched, force continue router loop

How is getUrl('catalog/product/view/id/1') replaced with rewrite?

- Product->getProductUrl
- Product\Url->getProductUrl
- Product\Url->getUrl
- UrlFinderInterface->findOneByData
- new Url->getUrl – `_direct` if rewrite found = baseUrl . requestPath

Rewrite is not used with regular getUrl, only when module uses explicitly (catalog, CMS).

CMS URL rewrite

- On event `cms_page_save_after`, if identifier or store changed, deletes and creates new rewrites.
- Doesn't create redirect rewrite for renamed redirects.
- CMS page opens with UrlRewrite router (priority 20), not CMS router (priority 60).

How are user-friendly URLs established, and how are they customized?

Module UrlRewrite:

- \Magento\UrlRewrite\Model\UrlPersistInterface::deleteByData
- \Magento\UrlRewrite\Model\UrlPersistInterface::replace

Product:

- event `catalog_product_save_before` - generate URL key by product name (if url key wasn't provided)

  - ProductUrlKeyAutogeneratorObserver
  - \Magento\CatalogUrlRewrite\Model\ProductUrlPathGenerator::getUrlKey
- event `catalog_product_save_after` - generate and replace URL rewrites (when changed url_key, categories, websites or visibility)

  - ProductProcessUrlRewriteSavingObserver
  - \Magento\CatalogUrlRewrite\Model\ProductUrlRewriteGenerator::generate
  - deleteByData, replace

Category:

- event `catalog_category_save_before` - generate URL key, update child categories
    - CategoryUrlPathAutogeneratorObserver
    - \Magento\CatalogUrlRewrite\Observer\CategoryUrlPathAutogeneratorObserver::updateUrlPathForChildren
    - \Magento\CatalogUrlRewrite\Observer\CategoryUrlPathAutogeneratorObserver::updateUrlPathForCategory
    - \Magento\CatalogUrlRewrite\Model\CategoryUrlPathGenerator::getUrlPath
    - child category.url_path
- event `catalog_category_save_after` - when changed (key, anchor, products)
    - CategoryProcessUrlRewriteSavingObserver
    - \Magento\CatalogUrlRewrite\Observer\UrlRewriteHandler::generateProductUrlRewrites
    - … lots of logic

Describe how action controllers and results function.

App\Action\Action::dispatch:

- event `controller_action_predispatch`
- event `controller_action_predispatch_$routeName`, e.g. `..._checkout`
- event `controller_action_predispatch_$fullActionName`, e.g. `..._checkout_cart_index`
- stop if FLAG_NO_DISPATCH
- *execute* - all action controllers implement this
- stop if FLAG_NO_POST_DISPATCH
- event `controller_action_postdispatch_$fullActionName`
- event `controller_action_postdispatch_$routeName`
- event `controller_action_postdispatch`
- if action doesn't return result, response object is returned – action can just modify response object

How do controllers interact with another?

- Controller\Response\Forward - changes request params, marks request not dispatched, so front controller will match again and new controller will be executed
- Controller\Response\Redirect - another controller URL

How are different response types generated?

\Magento\Framework\Controller\ResultInterface:

- renderResult
- setHttpResponseCode
- setHeader

Controller\AbstractResult:

- *renderResult* - required by interface - applies headers and calls*render*. children must implement this
- setHttpResponseCode
- setHeader
- setStatusHeader

Controller\Result\Raw:

- setContents
- *render* - set response body

Controller\Result\Json:

- setData - array
- setJsonData - string
- *render* - processes inline translations, sets application/json header and response body json string

Controller\Result\Forward:

- setModule, setController, setParams
- *forward* - does the trick, modifies request object, marks request not dispatched
- *render* - does nothing, forward must be called manually

Controller\Result\Redirect:

- setUrl, setPath - custom address
- setRefererUrl, setRefererOrBaseUrl - go back function

View\Result\Layout: - renders layout without `default` handle and page layout (1-column etc.)

- *renderResult*
    - event `layout_render_before`
    - event `layout_render_before_$fullActionName`, e.g. `..._checkout_cart_index`
    - render
- *render* - layout->getOutput, translate inline, set response body

- addDefaultHandle = $fullActionName, e.g. `checkout_cart_index`

- addHandle, addUpdate

View\Result\Page: - wraps layout into page layout

- same events as above
- *render* - renders layout, assigns vars and renders outer page template
- assign - values into viewVars property. default Default: requireJs, headContent, headAdditional, htmlAttributes, headAttributes, bodyAttributes, loaderIcon, layoutContent
- addDefaultHandle = $fullActionName + `default`