Define form and grid widgets

UI Component generation

UI component scheme - `urn:magento:module:Magento_Ui:etc/ui_configuration.xsd`
Magento_Ui, name just like page_configuration

Generation flow:

- \Magento\Framework\View\Layout\Generator\UiComponent::generateComponent - create, prepare, wrap
- create component \Magento\Framework\View\Element\UiComponentFactory::create
  - \Magento\Framework\View\Element\UiComponentFactory::mergeMetadata – components definitions in array format + dataProvider.getMeta
    - metadata = [‘cert_form’ => [‘children’ => dataProvider.getMeta]]
  - for each child[], \Magento\Framework\View\Element\UiComponentFactory::createChildComponent recursively:
    - create PHP class for component, e.g. `new Magento\Ui\Component\DataSource($components = [])`;
  - create PHP class for component, e.g. `new Magento\Ui\Component\Form($components = [children...])`;
- prepare component recursively
  - getChildComponents[].prepare - update data, js_config etc.
- wrap to *UiComponent\Container* as child ‘component’
  - toHtml = render -> renderEngine->render (template + ‘.xhtml’)

AbstractComponent

component PHP class common arguments:

- name
- template -> .xhtml
- layout: generic - default/tabs. DI config Magento\Framework\View\Layout\Pool
- config:
  - value
- js_config:
  - component - JS component class
  - extends - by default context.getNamespace - top level UI component instance name, e.g. ‘cms_block_form’
  - provider - context.addComponentDefinition - adds to “types”
- actions - context.addActions. Broken?
- html_blocks - context.addHtmlBlocks
- buttons - context.addButtons
- observers

What is the difference between component data `config`, `js_config` and others?

- looks like js_config is special - goes to `types` definition
- config - normal JS component config overriding `defaults`?
- other values - only for PHP component class usage?

`provider`, **`extends`**

- PHP Component.getJsConfig - all components automatically get `data.js_config.extends = [top level ui component name]`.
- PHP Component.`prepare`:
  - every component data.js_config is registered in `types` by constant type (*see below*). Definitions for multiple same types (component `column` occurres many times) are merged.
  - when data.js_config.`provider` is set:
    - `extends` is removed. This is TOP LEVEL component.
    - this TOP LEVEL compoment data.js_config is registered by personal name - e.g. “cms_block_form” instead of “form”.

This makes sense - ALL components inherit same top level `provider` via `extends`.

Example:

```
"Magento_Ui/js/core/app": {
    "types": {
        // [component constant type]: [data.js_config],
        // TOP LEVEL specific js_config and required provider
        cms_block_form: {
            provider: 'cms_block_form_data_source'
        },
        // below are generic types by constant type
        form: {
            extends: 'cms_block_form'
        },
        fieldset: {
            component:
'Magento_Ui/js/form/components/fieldset',
            extends: 'cms_block_form'
        },
        input: {
            extends: 'cms_block_form'
        }

    },
    "components": ...
}
```

DataSource, DataProvider

DataProvider interface is NOT in UI module, but in framework. Magento\Framework\View\Element\Uicomponent\DataProvider\DataProviderInterface:

- getName - why???
- getData - [items, totalItems]
- setConfig, getConfigData
- addFilter, addOrder, setLimit
- getPrimaryFieldName - used e.g. to delete row by ID
- getRequestFieldName ???
- getMeta – extends/overrides ui component structure - converted array. Meta format example:

```
[
    '' => [
        attributes => [
            class => 'Magento\Ui\Component\Form',
            name => 'some_name',
        ],
        arguments => [
            data => [
                js_config => [
                    component => 'Magento_Ui/js/form/form',
                    provider =>
'cms_block_form.block_form_data_source',
                ],
                template => 'templates/form/collapsible',
            ],
        ],
        children => [ ...nested components... ]
    ]
]
```

- getFieldMetaInfo, getFieldsMetaInfo, getFieldsetMetaInfo – wtf?
- getSearchCriteria, getSearchResult ???

You don't have to implement data provider from scratch, choose from 2 implementations:

- in framework next to interface - `Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider`. This provider sets collects all input filters and sorts into *search criteria*. Data is returned like `searchResult = $this->reporting->search($searchCriteria)`. Nice and tidy.
    - static properties are passed in ui component XML in `listing/dataSource/dataProvider/settings`:
        - name ??? - `[YourComponentName]_data_source`
        - primaryFieldName, e.g. `entity_id`
        - requestFieldName, e.g. `id`
    - addFilter, addOrder, setLimit - proxy to search criteria builder

    But who will process our searchCriteria?

    - `\Magento\Framework\View\Element\UiComponent\DataProvider\Reporting` is responsible for returning SearchResult by SearchCriteria.
    - `\Magento\Framework\View\Element\UiComponent\DataProvider\CollectionFactory` creates collection instance by *data provider name*.
    - You register your collection as DI arguments for CollectionFactory:

    ```
    <type
    name="Magento\Framework\View\Element\UiComponent\DataProvider\CollectionFactory">
        <arguments>
            <argument name="collections" xsi:type="array">
                <item name="sales_order_grid_data_source"
    xsi:type="string">Magento\Sales\Model\ResourceModel\Order\Grid\Collection</item>
            </argument>
        <arguments>
    </type>
    ```

- in Magento_Ui - `\Magento\Ui\DataProvider\AbstractDataProvider` This provider works directly with *collection*, you define collection in constructor when extending.
    - as usual, static properties are passed in ui component XML as arguments or settings:

        - name
        - primaryFieldName
        - requestFieldName
        - meta
        - data - data['config'] for your own usage, you can pass some settings, preferences etc.
    - addFilter = collection.addFieldToFilter
    - addOrder = collection.addOrder
    - etc…
    - getData = collection.toArray

Source of confusion - unknown UI component.xml structure, node names. `definition.map.xml`

`definition.map.xml` maps what values come from where! This explains custom node names and attributes that are magically inserted in resulting JS configuration.

`Magento_Ui/view/base/ui_component/etc/definition.map.xml`

E.g. component `dataSource` > argument `dataProvider` -> primaryFieldName = `xpath(dataProvider/settings/requestFieldName)` Example from `definition.map.xml`:

```
    <component name="dataSource">
        <schema name="current">
            <argument name="dataProvider" xsi:type="configurableObject">
                <argument name="data" xsi:type="array">
                    <item name="config" xsi:type="array">
                        <item name="submit_url" type="url"
xsi:type="converter">settings/submitUrl</item>
                        ...
                        <item name="clientConfig" type="item"
xsi:type="converter">dataProvider/settings/clientConfig</item>
                        <item name="provider" type="string"
xsi:type="xpath">@provider</item>
                        ...
                    </item>
                    <item name="js_config" xsi:type="array">
                        <item name="deps" type="deps"
xsi:type="converter">settings/deps</item>
                    </item>
                    <item name="layout" xsi:type="array">
                        <item name="type" type="string"
xsi:type="xpath">settings/layout/type</item>
                        <item name="navContainerName" type="string"
xsi:type="xpath">settings/layout/navContainerName</item>
                    </item>
                </argument>
                <argument name="class" type="string"
xsi:type="xpath">dataProvider/@class</argument>
                <argument name="name" type="string"
xsi:type="xpath">dataProvider/@name</argument>
    ...
```

And `definition.xml`:

```
<dataSource
class="Magento\Ui\Component\DataSource"/>
```

Using definition.map.xml we can deduce full `<dataSource>` definition and what parameter will end

up where:

```
<dataSource class="Magento\Ui\Component\DataSource">
    <aclResource/>
    <settings>
        <submitUrl />
        <validateUrl />
        <updateUrl/>
        <filterUrlParams/>
        <storageConfig/>
        <statefull/>
        <imports/>
        <exports/>
        <links/>
        <listens/>
        <ns/>
        <componentType/>
        <dataScope/>
        <deps/>
        <layout>
            <type/>
            <navContainerName/>
        </layout>
    </settings>
    <dataProvider name="some_name"
class="SomeDataProviderClass">
        <settings>
            <primaryFieldName>entity_id</primaryFieldName>
            <requestFieldName>id</requestFieldName>
        </settings>
    </dataProvider>
</dataSource
```

Fun fact - some parameters are scattered in `settings`, another in `dataProvider/settings`, but in fact ALL of them will be used only for dataProvider object data.

PHP data source creation equivalent:

```
$dataProvider = $objectManager->create(`SomeDataProviderClass`, [
    'name' => 'some_name',
    'primaryFieldName' => 'entity_id',
    'requestFieldName' => 'id',
    'data' => [
        'config' => [
            'submit_url' => '',
            'validate_url' => '',
            'update_url' => '',
            'filter_url_params' => '',
            'clientConfig' => '',
            'provider' => '',
            'component' => '',
            'template' => '',
            'sortOrder' => '',
            'displayArea' => '',
            'storageConfig' => '',
            'statefull' => '',
            'imports' => '',
            'exports' => '',
            'links' => '',
            'listens' => '',
            'ns' => '',
            'componentType' => '',
            'dataScope' => '',
            'aclResource' => '',
        ],
        'js_config' => [
            'deps' => [],
        ],
        'layout' => [
            'type' => '',
            'navContainerName' => '',
        ],
    ],
]);
$dataSource = $objectManager->create('Magento\Ui\Component\DataSource',
[
  'dataProvider' => $dataProvider,
]);
```

With this sacred knowledge we can add/override data provider parameters explicitly in addition to normal config:

```
<dataSource name="...">
    <argument name="dataProvider" xsi:type="configurableObject">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="referer_url" xsi:type="url"
path="sales/archive/shipments/index"/>
            </item>
        </argument>
    </argument>
    <dataProvider class="..." name="..">
        <settings>...</settings>
    </dataProvider>
</dataSource>
```

Okay, ui component *data source* doesn't get any configuration, it all goes to *data provider* - 'submit_url' etc. Data provider can only return data and modify meta, how does it pass these JS values for JS components?

Here's how:

- View\Layout\Generic::build (initial page open) or View\Element\UiComponent\ContentType\Json::render (load AJAX data)
- context.getDataSourceData – merges
  - component.getDataSourceData
  - data provider.getConfigData – returns data['config'] – all the params we set as arguments

UI Form

Caveats:

- default form data.template is `templates/form/default`. But its spinner works only with `tabs` layout. `<div data-role="spinner" data-component="{{getName()}}.areas"`. Normal form does not have `areas`, and so this spinner will work forever. Solution:
  - either set arguments.data.template = `templates/form/collapsible`: `<div data-role="spinner" data-component="{{getName()}}.{{getName()}}"` - this matches normal form
  - or set form.settings.layout.type = `tabs`. It will init `areas` suitable for default template. See \Magento\Ui\Component\Layout\Tabs::initAreas.

component Form.getDataSourceData:

- read ID from request, e.g. current entity ID
- filter data provider by primaryField = $requestedValue
- get row data by requested primary ID
  - note that normal collections don't return rows indexed by ID and you will receive an error
  - Magento data provider override getData and index by ID manually, e.g. \Magento\Cms\Model\Page\DataProvider::getData

Working with form fields:

Example: `<field name="something" formElement="input">`.

Magic unraveled:

- see definition.xml `<field class="Magento\Ui\Component\Form\Field"/>`
- \Magento\Ui\Component\Form\Field::prepare - creates new child ui component `wrappedComponent` with type = `$this->getData('config/formElement')`.
- how to set config/formElement? See definition.map.xml, search for `component name="field"`: `<item name="formElement" type="string" xsi:type="xpath">@formElement</item>`.

`formElement` creates arbitrary ui component and inherits same data as main `field`.

Suitable `formElement` types:

- input
- textarea
- fileUploader
- date
- email
- wysiwyg
- checkbox, prefer radio/toggle

```
<formElements>
    <checkbox>
        <settings>
            <valueMap>
                <map name="false" xsi:type="number">0</map>
                <map name="true" xsi:type="number">1</map>
            </valueMap>
            <prefer>toggle</prefer>
        </settings>
    </checkbox>
</formElements>
```

- select:

```
<formElements>
    <select>
        <settings>
            <options
class="Magento\Config\Model\Config\Source\Design\Robots"/>
        </settings>
    </select>
</formElements>
```

- multiselect

UI Listing

`listing` structure:

- argument data.js_config.provider
- `dataSource`
  - `dataProvider`
- `filterbar`
  - `filters`
  - `bookmark`
  - `export`
  - `massaction`
- `columns`
  - `column`
- `listing` - just container, server template ‘templates/listing/default.xhtml’
- `columns` - no server processing, all juice here:
  - JS component Magento_Ui/js/grid/listing
- `column`
  - JS component Magento_Ui/js/grid/columns/column
  - don’t forget to include config/dataType = settings/dataType

Implementing listing checklist:

- set settings/spinner = “columns”
  - Listing template ‘templates/listing/default.xhtml’ has spiner

`data-component="{{getName()}}.{{getName()}}.{{spinner}}"`
    - `columns` js component - `Magento_Ui/js/grid/listnig` - hides data when loaded - `loader.get(this.name).hide()`.
- dataSource must specify provider in order to load data propertly `<dataSource provider="Magento_Ui/js/grid/provider"/>` This provider handles loading data via AJAX.
- set `dataSource/settings/updateUrl` = `mui/index/render`. This will return grid data in JSON format. When this parameter is missing, AJAX is made to current URL

```
<column>
    <settings>
        <dataType></dataType> <!-- creates wrapped UI component, column config =
merge(wrapped.config + column.config) -->
    </settings>
</column>
```

DataType:

- text - default

Listing toolbar:

```
<listingToolbar>
    <bookmark />
    <columnsControls />
    <filterSearch />
    <filters />
    <massaction>
        <action name="delete">
            <settings>
                <label/>
                <type/> -
delete/edit/...
                <url path=""/>
                <confirm>
                    <message/>
                    <title/>
                </confirm>
                <callback>
                    <target />
                    <provider />
                </callback>
            </settings>
        </action>
    </massaction>
    <paging />
</listingToolbar>
```

Column types:

- selection

```
<selectionsColumn name="ids">
    <settings>
        <indexField>block_id</indexField>
    </settings>
</selectionsColumn>
```

- select

```
<column name="is_active" component="Magento_Ui/js/grid/columns/select">
    <settings>
        <options class="Magento\Cms\Model\Block\Source\IsActive"/>
        <filter>select</filter>
        <editor>
            <editorType>select</editorType>
        </editor>
        <dataType>select</dataType>
        <label translate="true">Status</label>
    </settings>
</column>
```

- …

column/settings/filter:

- textRange
- text
- select
- dateRange
- …

UNSORTED

Magento\Framework\Config\DataInterfaceFactory = \Magento\Ui\Config\Data::get('cert_form') - evil class that converts all the obscure UI component conventions into real understandable data array. E.g. `<settings>`, `<formElement>` and the like.

Magento_Ui/view/base/ui_component/etc/definition.map.xml
Magento_Ui/view/base/ui_component/etc/definition.xml

- config data.initData
- new config reader \Magento\Ui\Config\Reader('cert_form.xml')
- config reader.read:
    - find all UI files with this name
    - reader.readFiles
        - convert file contents to DOM \Magento\Ui\Config\Reader\Dom(file content) - new \DOMDocument.loadXML
        - convert DOM to array - \Magento\Ui\Config\Converter::convert(DOMDocument)
        - `config converter.toArray` – all conventions start here
            - merges default data by node name from `definition.xml`

\Magento\Ui\Component\Filters: text, textRange, select, dateRange

page layout - `urn:magento:framework:View/Layout/etc/page_configuration.xsd`