Selection fields

~~MGNLUI-4519~~ - Migrate select fields to Vaadin 8 CLOSED

Goal

- should work with modernised form framework
- should support the current select field functionality
    - single select with pre-defined options
    - single select with items coming from the back-end (JCR or different)
    - multi-selects covering the same functionality as single selects
- design a new definition hierarchy that would be clearer than the current (e.g. the option of connecting to backends is provided in a hacky and restrictive way aka 'remotePath' or smth')

Definition proposal

Items can originate from

- Preconfigured or inline options, i.e. within the select definition itself
- a "remote" JCR location (e.g. a workspace other than config)
- an actual remote data source, e.g. a REST based service

Below it's an example of the two latter definitions

JCR or remote data provider

```
select:
  label: select
  class: info.magnolia.ui.framework.databinding.definition.JcrSingleSelectFieldDefinition
  datasource:
   class: info.magnolia.ui.datasource.jcr.JcrDatasourceDefinition
   workspace: contacts
   sortable: true #true it's the default value actually
   sortByProperties: # if omitted will default to describeByProperty, if the latter is present
    - foo
    - bar
   describeByProperty: bar
selectRest:
  label: selectRest
  class: info.magnolia.ui.framework.databinding.definition.RestSingleSelectFieldDefinition
  datasource:
    class: info.magnolia.restcontentapp.rest.RestDataProviderDefinition
    url: https://restcountries.eu/rest/v2/regionalbloc/eu
    jsonPathPredicate: "$.[*].name"
```

Below an example of inline or preconfigured options: it still uses a `datasource` but it's a "static" or fixed-size one (naming still uncertain)

Preconfigured options are by default sorted alphabetically in ascending order, unless `sortOptions` is set to false

Preconfigured options

```
select:
  label: select
  class: info.magnolia.ui.framework.databinding.definition.PreconfiguredSingleSelectFieldDefinition
  textInputAllowed: true
  filteringMode: startsWith
  datasource:
   class: info.magnolia.ui.framework.datasource.impl.FixedSizeDatasourceDefinition
   properties:
    - name: foo
      value: Foo
    - name: bar
      value: Bar
    - name: qux
      value: Qux
```

A `BaseSelectFieldDefinition` will be shared by selection components with single or multiple selection capabilities

```
public class BaseSelectFieldDefinition<T, DS> extends ConfiguredFieldDefinition<T> {

   ...
       /**
     * An object representing the DataProvider configuration definition.
     */
    private DS datasource;
```

The data provider is obtained through `DatasourceSupport` at `AbstractSelectFieldFactory`

```
public abstract class AbstractSelectFieldFactory<DS, D extends BaseSelectFieldDefinition<T, DS>, T, F extends
Component & HasValue<T>> extends AbstractFieldFactory<D, T, F> {
...


@Inject
public AbstractSelectFieldFactory(ComponentProvider componentProvider, D definition, ..., DatasourceSupport
datasourceSupport) {
    super(definition, componentProvider, locale, i18NAuthoringSupport);
    this.dataSourceBundle = datasourceSupport.getDatasourceBundle(getDefinition().getDatasource());
}
...
protected Optional<? extends IconResolver> getIconResolver() {
    return  dataSourceBundle.lookup(IconResolver.class, getDefinition().getDatasource());
}

protected Optional<? extends ItemDescriber> getItemDescriber() {
    return  dataSourceBundle.lookup(ItemDescriber.class, getDefinition().getDatasource());
}

protected Optional<? extends DataProvider> getDataProvider() {
   return dataSourceBundle.lookup(DataProvider.class, getDefinition().getDatasource());
}
```

Concrete factories implementing this will do something like the following

```
getDataProvider().ifPresent(dp -> select.setDataProvider(dp));
getItemDescriber().ifPresent(describer -> select.setItemCaptionGenerator(item -> describer.describe(item)));
...
```

Converter resolving strategy

We need to come up with a general strategy as to how to provide converters, that is the Vaadin mechanism responsible for translating between model (data type used by the backend) and presentation (data type used by the UI) cause the two may not match (e.g. a TextField displaying a Number, where the presentation type is a String and the model some numeric type).

In M5 the logic setting a converter and creating a default value is bound to the component itself (see https://git.magnolia-cms.com/projects/PLATFORM/repos/ui/browse/magnolia-ui-form/src/main/java/info/magnolia/ui/form/field/factory/AbstractFieldFactory.java)

With Vaadin 8 this concern has moved to a separate `Binder` class. A special case of converter in Magnolia is `IdentifierToPathConverter` which is used e.g. to convert an id in the form `workspace:uuid` into a corresponding path. This used to resolve e.g. assets in dam or to select items in the UI with a chooser dialog.

Since we want to be JCR-agnostic we'd like our definitions to be free from JCR specific terms, such as `workspace`. Also, since we now have introduced datasource in select field definitions, it would be nice to reuse such feature without additional configuration.

Proposal

Data source implementations know how to resolve complex items (such as a JCR Node) into simpler objects (a Node's String identifier for instance) back and forth. Data source bundles will eventually register such converters. For instance

```
/**
 * JCR-specific implementation of {@link Converter}. Turns a Node (presentation type) into its identifier
(model type or a String) and vice versa.
 */
public class JcrItemToLinkConverter implements Converter<Node, String> {
    ...

    public JcrItemToLinkConverter(JcrDatasourceDefinition definition, Provider<Context> contextProvider) {
        this.definition = definition;
        this.contextProvider = contextProvider;
    }

    @Override
    public Result<String> convertToModel(Node value, ValueContext context) {
        if (value == null) {
            return Result.ok(null);
        }
        return Result.ok(value.getIdentifier());
    }

    @Override
    public Node convertToPresentation(String id, ValueContext context) {
        if (id == null) {
            return null;
        }
        try {
            return new LazyNodeWrapper(getSession().getNodeByIdentifier(id));
        } catch (RepositoryException e) {
           log.warn("Could not get a JCR node by identifier " + id, e);
           return null;
        }
    }
...
```

After registration, the new `Converter` will be available via `DataSourceSupport`

```
public JcrDataSourceBundle(Provider<Context> contextProvider) {
    super(JcrDatasourceDefinition.class);
    ...
    register(Converter.class, def -> new JcrItemToLinkConverter(def, contextProvider));
}
```

A new interface `ItemToLinkConverterResolverStrategy` will be introduced and injected into `ConfiguredBinder`. The converter resolver strategy will take care of resolving, if possible, the appropriate converter.

```
/**
 * Attempts to resolve an item to link {@link com.vaadin.data.Converter} for a given field based on its
datasource configuration.
 * Typically item to link converters are used to represent a complex object as a simple object that references
the complex one.
 * For instance, a JCR Node may be represented by its identifier, that is a plain String.
 * In this example, an item to link converter will be able to transform a Node to its identifier and vice versa.
 *
 * @see DefaultItemToLinkConverterResolverStrategy
 * @see MultiItemToLinkConverter
 */
public interface ItemToLinkConverterResolverStrategy<PRESENTATION, MODEL> {

    /**
     * @return the appropriate item to link converter based on a field's datasource configuration.
     */
    Optional<? extends Converter<PRESENTATION, MODEL>> resolveFromDatasource(Object datasource);
}
```

A default implementation may look like this

```
public class DefaultItemToLinkConverterResolverStrategy implements ItemToLinkConverterResolverStrategy {
   ...
    @Inject
    public DefaultItemToLinkConverterResolverStrategy(DatasourceSupport datasourceSupport) {
        this.datasourceSupport = datasourceSupport;
    }

    @Override
    public Optional<? extends Converter> resolveFromDatasource(Object datasource) {
        if (datasource == null) {
            return Optional.empty();
        }
        return datasourceSupport.getDatasourceBundle(datasource).lookup(Converter.class, datasource);
    }
}
```

Finally `ConfiguredBinder` will bind the converter to a field based on the latter's `datasource` (or `converterClass`) configuration.

Git PR

https://git.magnolia-cms.com/projects/PLATFORM/repos/ui/pull-requests/624/overview