Dynamic forms - PoC phase

In order to validate assumptions for cleaner usage of Vaadin in Magnolia forms, 3 PoCs were conducted in September 2016. This page archives their requirements and debrief notes.

Introduction

PoC Goals

- Validate several assumptions from Mika
- If all goes well, put it all together at the end, through an Integration phase

Process

- To be carried out by the VN Core Team
- Try to work in pairs (can still switch afterwards), or alternate between group phases and individual work/thinking
- Keep each PoC standalone until the end of the PoC phase
- Target date: 25 Sep 2016

General directions & goals were that of the refactoring initiative—see parent page.

Deliverables: PoCs took the shape of several, independent Vaadin "sandbox" applications, alike e.g.:

- **Vaadin + Magnolia sandbox**, based on 5.5
    - contains only the ACL fields & validation sample UI
    - on branch v2: https://git.magnolia-cms.com/users/mgeljic/repos/vaadin-sandbox/commits?until=refs%2Fheads%2Fv2
    - use this as a skeleton for the PoCs
- **Pure Vaadin Sandbox** (without Magnolia)
    - contains many samples for various vaadin components & fields (grid, selects, theming, fieldgroups)
    - on Mika's github repo: https://github.com/mkgl/vaadin-sandbox

General outcome: ✅ **Confirmation**

- much less custom code, more Vaanilla Vaadin => help reduce technical debt
- e.g. FieldGroups, JS Extensions, declarative layouts
- more Vaadin expertise

PoC #1: New FormPresenter

DEV-279 - Getting issue details... STATUS

Expectations		Outcome

• Reading definitions and building tabs and fields
  • Result: a simple Vaadin application (w/ ui-framework on the classpath)
• 1 Vaadin `FieldGroup` + `FormLayout` for each tab (merge bits from `FormBuilder`)
• **`FormPresenter` invoking `FieldFactories`**
  • for the PoC, you can instantiate factories manually
  • either refactor existing ones, or reimplement a simple TextFieldFactory with less Magnolia dependencies
  • maybe API like `public Field<T> createField(D definition)`
  • so we can reuse a single `FieldFactory` to produce multiple instances of a field
  • **`FieldFactories` no longer doing property data-binding** (`#setPropertyDataSource`)
  • `FormPresenter` doing the databinding instead => `fieldGroup#setItemDataSource`
	✅	• APIs
		  • stateless FieldFactory
		  • stateless FormPresenter
		  • defs and items passed via #start
		• Plain Vaadin
		• (y)

Enable or validate one field from the value of another one	❌	
Not needed for the PoC: • locale and i18n handling • complex fields (only text fields, selects) • visual appearance, no MagnoliaTabSheet needed, just a plain `VerticalLayout` with one-to-many `FormLayouts`		

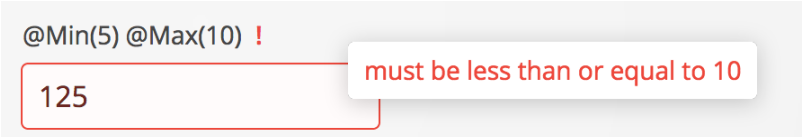
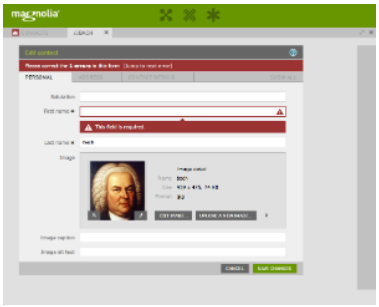
Next steps

- Extensibility
  - Interface over Abstractions
  - Public over Protected (extending should be demoted)
  - 💡 Generate wrapper implementations for component interfaces
- Backwards compatibility

PoC #2: Validation bubbles below field

DEV-280 - Getting issue details... STATUS

Expectations		Outcome

Validation as a Vaadin JS Extension, see the few links below:
 	
• Integrating JavaScript Components and Extensions (Vaadin docs)
• class `com.vaadin.server.AbstractJavaScriptExtension`
• Component Extensions & Component and UI Extensions (the GWT way, Vaadin docs)
• see also `TextAreaStretcher`, `Bidi` (GWT way) or `JCrop`, `RowScroller` (JS way) for more extension examples
 	
Default Vaadin validation is as follows:
 	
• warning icon
• tooltip when hovering the field
 	
 
 	
Extension should primarily **add** the bubble below
 	
• Same styles as now
  
  • see
  • bubble width is the same as field width with border
  • arrow tip should be around 30–40px from the left
• Result: small Vaadin application
  • Apply to a TextField (e.g. using standard Vaadin email validator)
  • If using the Magnolia theme, then field border should become red too
• Currently tooltip is hidden via CSS in Magnolia
  • we can still do this the same way, maybe as well for the warning icon
	✅	• As GWT Extension
		  • stateChange
		  • errorMessage
		  • + validationVisible
		  • ❓ triangle
		  • Test drive validation on a CompositeField
		  • pluggable to any Vaadin form, not just Magnolia
		  • (y)
		• As JavaScript Component
		  • vs. JavaScript Extension
		  • addComponent?
		  • JS Connector + HTML template (y)
		  • valueChange server-side
		  • moving away from GWT

Next steps

- Combine the two approaches
- Additional style enhancements / animations

PoC #3: Complex Fields refactoring

DEV-281 - Getting issue details... STATUS

Expectations		Outcome
• Get rid of Magnolia-specific components & defs inside field implementations • Get rid of AbstractCustomMultiField abstraction	✅	

<table>
<tr>
<td>
<h3>Pure Vaadin CompositeField</h3>

close to <code>AccessControlField</code>
Except fields are not hard-coded

pass sub-fields into constructor?

maybe as map with field-names as keys?

would help us keeping FieldFactories outside of field implementations

i.e. get rid of infamous <code>AbstractCustomMultiField#createLocalField</code>

also bind fields internally via <code>FieldGroup</code>?
reuse <code>#setValidationVisible</code> and handling of valueChange on sub-fields
Result: simple Vaadin application

operating over a <code>PropertysetItem</code> or <code>BeanItem</code>
part 1. using a CompositeField for First name and Last name

laid out horizontally
has an API to set direction (minimum requirement)

part2. also API to set an arbitrary layout

if feasible, but not a must-have
or maybe using a Vaadin declarative layout?

check out Sasha's talk from mconf16
Designing UIs declaratively (Vaadin docs)
icing on top of the cake really

</td>
<td></td>
<td rowspan="3">

Good

not bound to factory
move away from AbstractCustomMultiField / createLocalField

valueChange propagated

? exposing getFieldById/Name

? BeanItem multifield APIs

need more conventions?

Declarative-layout based composite field (y)

</td>
</tr>
<tr>
<td>
<h3>Pure Vaadin MultiValueField</h3>

close to <code>AccessControlListField</code>
reuse some form of <code>EntryFieldFactory</code> (maybe also NewEntryHandler if applicable)
Result: simple Vaadin application

operating over a <code>PropertysetItem</code> or <code>BeanItem</code>

i.e. <code>extends CustomField<I extends Item></code>
or <code>extends CustomField<C extends Collection></code> (maybe more in line with Vaadin 8?)

with a property of type Collection or List, then one entry per entry in the collection
field needs to create Vaadin properties for each entry in the list, and bind it internally to sub-fields
also bind fields internally via <code>FieldGroup</code>?
reuse validation and valueChange logic from <code>AccessControlListField</code>
steps:

a MultiField of TextFields (can be hard-coded)
a MultiField of a CustomField type (First name + Last name)
a MultiField with the pluggable NewEntryHandler (closer to use cases in Magnolia where subfields are configured)

</td>
<td></td>
</tr>
<tr>
<td>
<h3>Pure Vaadin LinkField</h3>

Inspired from <code>AccessControlField$PathChooserHandler</code>
Result: small Vaadin application

see <code>info.magnolia.dev.acls.AccessControlFieldUI</code> in Mika's sandbox
opens a Vaadin Window

field inside the window shares property data-source with text field

</td>
<td>✗</td>
</tr>
</table>

Next steps

- do some dynamic relationship on form level
- refine generic value typing for complex fields
  - ideally move away from Items, in favor of domain types
  - Challenging with JCR adapters in particular