Developer Call - 2013-11-05

On the Call: Daryl Hedley, Rob Moore, Dennis Heaney, Ryan Adams

Actions Arising:

1. Daryl is going to prototype the routing stuff we talked about
2. Ryan is going to write up some thoughts on Rich Text
3. Images needs additional consideration - needs owner
4. ID generation needs considered - (Ryan?)

Single width and full span components - could be put into schema - e.g. spanned:true

Adapt Desktop view is 960px, so a half-width component is 480px or thereabouts (with padding/margins to be taken into account)

Three options were discussed:
1. have a span attribute on a component with the allowed options of 'full' || 'single' which makes adding additional options like 'triple' possible in future.

2. A block having 'slots' that are filled up as components are added

3. Use the CSS Grid columns to define the capacity of the block, a component advertises the number of columns it requires. The Block knows how many columns it has available.

The goal is to limit what the user can add to a block so that the result does not interfere with the responsive layout

Could the editor warn the course creator (visually?) when they've overshot the slot allocation of a block

Proposed Solution

Blocks advertise their capacity so the Authoring Tool can verify the build - suggest an attribute of 'slot-capacity'. Components have two attributes 'slot-size-minimum' and 'slot-size-maximum' which define the number of slots the component consumes.

The Authoring tool validates the block by evaluating the sum of all the component 'slot-size-minimum' values in a block and highlighting if that number exceeds the 'slot-capacity'.

Question: Do we allow the author to edit the slot-size value of a component (limited by the min/max values)? I'd (Ryan) argue that we shouldn't - the Author shouldn't have much control over layout, that's the domain of the component author and/or the theme author.

IDs - discuss best way for these - router issues with long IDs

Current Approach

m05-t05-p05 : gets to page

m05-t05-p05-a05-b05-c05 : get to component

m05/t05/p05 : passed to router

Current Course Structure

```
course
 |
 >> contentObject
   |
   >> article
     |
     >> block
       |
       >> component
```

Notes

The Authoring Tool needs to generate the IDs.

We had a long discussion here about the right approach. The new structure (without Module & Topics) means we can have nested contentObjects in our hierarchy.

The URL approach could then become quite complex and difficult to manage in the router.

We proposed adjusting the router in the following ways

1. only ever have one ID in the URL, something like /#/:id and have the Router lookup the type of thing we're addressing by getting its ID.
2. allow multiple IDs to permit referencing Articles, Blocks or Components, so the URL is of the form /#/:id/:articleID/:blockID/:componentID - in this scenario, the router passes the articleID, blockID and componentID on to the PageView object so it can scroll to that place.
3. encode the type of object into the ID (e.g. an articleID would be of the form "a:1234" where the a signifies that it's an article
4. to save on looking through multiple collections for an ID, an alternative approach may be to generate (in the Authoring Tool) a list of all the objects in the course, their ID, type and parent Page. This reduces the effort required to find the right page to render if we only use a single ID.

Code Snippets

Really simple routing, just redirect everything to the routing function

```
routes: {
        "":"menu",
        "*":"routing"
    },
```

Encode the type of object in the Model object

```
model : {
            type:”page”
}

model: {
            type:”menuItem”
}
```

Generate an Object Index to ease lookups

(this example shows that the object with id is an Article with a parent Page of c04)

```
objectIndex : [
    {
             id: “a01”,
            “type” : “article”
            “page” : “c04”
    },
]
```

A rough (pseudocode) example of the Routing function

```
routing: function(id) {

            // check what type of thing we’re trying to show
            if prefix == a {
                        var currentObject = Adapt.articles.findWhere({id:id});
                        var currentPage = currentObject.searchParent(‘page’);
                        new PageView({model:currentPage});
                        _.defer(function() {
                                    $.scrollTo(“#”+id);
                        })
            }
}
```

An alternative where we just have contentObjects in the URL

```
routing : function (id) {
            var currentObject = Adapt.contentObjects.findWhere({id:id});

            if(currentObject.get(‘type’) == ‘menuItem’) {
                        new MenuItem({model:currentObject});
            } else {
                        new PageView({model:currentObject});
            }

}
```

```
prefix2type:function(pref){ switch(pref){'a': return 'articles' …etc };
routing: funtion(id){
	var m = /([\a-zA-Z]+)([\d]+)/.exec(id);
	Adapt[prefix2type].findWhere({id:id});
}
```

can editor pick up on theme classes? - inherit themes classes

```
theme.json {
	classes:{
		articles:["dark-shadow","light-shadow","grey"],
		blocks:["full-width","padded"]
	}
}

courseThemeSettings
[
	{
		theme:hahah
		articles:[
			{id:a10, class:'dark-shadow'}
		]

	},
	{
		theme:DennisMajestic
		articles:[
			{id:a10, class:'light-shadow'}
		]
	}
]
```

Notes

The requirement here is to be able to apply a theme style to an element in the authoring tool. The theme.json example above is a suggested way for the theme to advertise the allowed styles to the authoring tool.

There is a possible issue where the theme is changed after some styles have been applied. In this scenario the style may not exist in the new theme. We'll need to figure out what to do here in the interface, but the

courseThemeSettings json snippet above is one suggestion of how we might handle it in storage. The publish process would need to apply the appropriate styles to the elements depending on which theme has been applied.

Also discussed but no conclusion drawn was the requirement to add custom styles to an element in the authoring tool. This may fall out of scope of the MVP but will be considered again.

add inline img or html into textareas

Our (the LP team) gut reaction here was a negative one, but it's very important for Kineo. We've agreed to spend some time thinking about this and come up with a good solution.

requested elements that we talked about

- images
- headings
- bold/italic/underline
- links
- lists

mobile/desktop images - graphic component - should only allow mobile size if full spanned

it was nearly lunchtime so we flew through this one. Suggestions here were that the Authoring Tool would automatically produce images at required sizes but that it should be possible to overwrite these with custom images if required.

The required sizes depend on the the theme - how this works is still to be decided.