Case Study: Batch Update
A User Driven Approach

What we talk about when we talk about design thinking

Karen Shaw
Adam Arling

Northwestern
The NUL dev team is refining a new workflow
What did we do before?

- Developers built and delivered finished product
- Not iterative, product was hard to change once delivered
- Resulting user experience lacking
Design Thinking

- Empathy
- Expansive Thinking
- Experimentation

Prct ion

Northwestern
Empathy

Users should be the #1 focus.

By hearing needs, feelings and motivations, one can create meaningful solutions to actual problems.

https://www.thinkwithgoogle.com/future-of-marketing/creativity/design-thinking-principles/
Expansive Thinking

aka “Brainstorming”

One perfect solution

Look at problem from all conceivable angles

Not 10% better, 10X better. Go big.
Experimentation

*Quickly* figure out which ideas *do* and *don’t* work

Prototype *early-stage* versions

Feedback
Building a better batch edit
What problem are we looking to solve?
“We want to ingest our assets so that they are tracked in our preservation system, and then be able to describe them in large batches so that we can quickly publish them for end user and researcher consumption.”
“We want an application that ….”

- Has a simple, intuitive UI
- Scales well. (A batch based system by default)
- Is transparent - ability to queue jobs, monitor state, success, failure and progress, see meaningful errors
# Initial Prototypes (Wireframes)

## Batch Dashboard

<table>
<thead>
<tr>
<th>Started</th>
<th>ID</th>
<th>User</th>
<th>Nickname</th>
<th>status</th>
<th>Works updated</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-02-02</td>
<td>12352</td>
<td>David Schaber</td>
<td>remove bad genre from Guther</td>
<td>done</td>
<td>1000</td>
<td>view details</td>
</tr>
<tr>
<td>2020-02-02</td>
<td>12354</td>
<td>Laura Alogna</td>
<td>remove bad genre from Guther</td>
<td>done</td>
<td>500</td>
<td>view details</td>
</tr>
<tr>
<td>2020-02-02</td>
<td>12341</td>
<td>Michael Klein</td>
<td>Change Title</td>
<td>done</td>
<td>250</td>
<td>view details</td>
</tr>
<tr>
<td>2020-02-02</td>
<td>12345</td>
<td>Michael Klein</td>
<td>Change description</td>
<td>error</td>
<td>0</td>
<td>view details</td>
</tr>
<tr>
<td>2020-02-02</td>
<td>123415</td>
<td>Michael Klein</td>
<td>Change description</td>
<td>error</td>
<td>0</td>
<td>view details</td>
</tr>
<tr>
<td>2020-02-02</td>
<td>342415</td>
<td>Laura Alogna</td>
<td>Change collection on works</td>
<td>in progress</td>
<td></td>
<td>view details</td>
</tr>
<tr>
<td>2020-02-02</td>
<td>429382</td>
<td>David Schaber</td>
<td>Update Title</td>
<td>queued</td>
<td></td>
<td>view details</td>
</tr>
<tr>
<td>2020-02-02</td>
<td>231231</td>
<td>David Schaber</td>
<td>Update Title</td>
<td>queued</td>
<td></td>
<td>view details</td>
</tr>
</tbody>
</table>

**Note:**
The goal is to give users a quick overview of running and completed batches. Ideally we index the batch’s unique ID so that users can quickly link to a search page showing the works updated using the “XXX Works Updated” link. View details would link to a page with the original query, and number of works updated. We could also do this in a single table if we collapse/expand on the errors.
Initial Prototypes (Wireframes)
Batch Edit Screens
White boarding

How/where do we select batch items?

How do we pass 10,000 + records around?

How do we track job status?

How does the UI know what's going on?

Goals
- Begin ingest w/minimally described records
  Preservation First
  ➔ get files ocf hard drives/network shares, etc.
- Reduce Friction – Production staff
  ➔ Automated Grouping of files works (based on automated workflows)
  ➔ Start w/production staff needs/process using tools they already use

⇒ SYSTEM IS PRODUCTION PIPELINE
- Other teams don’t need to know other process
  Don’t hurt human transformations
- Rely on industry standard file/metadata storage (SI)
Prototyping - Early and Often

- Wireframes & Whiteboarding
- API Design
- API Mocking
API Design

- How do we pass the search result set to the backend? What if there are thousands of id’s?
- Should we query through GraphQL or Elasticsearch? What about a lag between the two?
- When someone updates a field, is that an append or replace operation?
- How do we prevent race conditions?
- Who is responsible for scrolling Elasticsearch results? The front or the back end?
- Do we validate for required fields during a batch operation?
API Mocking

batchUpdate

- `add`: BatchAddInput
- `delete`: BatchDeleteInput

Start a batch update operation

replace: BatchReplaceInput

Replace replaces existing values (single and multi valued fields)

type Details

Input fields for batch replace operations

descriptiveMetadata:

BatchReplaceDescriptiveMetadataInput

Type Details

Input fields available for batch replace operations on works descriptive metadata

type BatchReplaceDescriptiveMetadataInput:

- `abstract`: [String]
- `alternateTitle`: [String]
- `boxName`: [String]
- `boxNumber`: [String]
- `caption`: [String]
- `catalogKey`: [String]
- `description`: [String]
- `folderName`: [String]
- `keywords`: [String]
- `license`: CodedTermInput
- `notes`: [String]
- `physicalDescriptionMaterial`: [String]
- `physicalDescriptionSize`: [String]
- `provenance`: [String]
- `publisher`: [String]
- `relatedMaterial`: [String]
Keep on prototypin’

Iterate on modular prototypes
Search driven selection?
Manually select items?
“Broad stroke” editing?

```javascript
{ query: {
  ...items in {
    ...{ "color": "blue" }
  }
}
```
Add/replace/remove in batch?
Get UI in front of users

Is the flow intuitive?

What else might they want?

FINAL CONFIRMATION SCREEN?  PREVIEW ITEMS?
You are batch editing the following 2663 items.
Back end work in parallel

As the result of our process, we knew:

- Our batches would take place within a **transaction** - an entire batch succeeds or fails.
- We needed a simple batchUpdate **mutation** in the API with **replace**, **add** (append) and **delete** inputs, that also takes an Elasticsearch query.
- We needed to Iterate over the Elasticsearch **scroll** and apply changes to each page of work IDs within one transaction.
Back end iterations

- add/delete controlled terms
- add/replace non-controlled terms
- collection assignment
- queueing/batch management

*Replacing the mocked responses with live responses keeps the API stable and should be seamless for the front end.*
Ending thoughts...
Do Design Thinking techniques aid our team’s workflow?