If You Were your own BOSS, How Would you Manage Yourself?

Week Breakdown

first — 2+ days Strategy/Biz/growth

then — rest Software dev

1st Strat.

— Kickstarter / short? mid long talk + long Storyboards outline

— What would let me use Romie effectively?

1
- outline that
- And this

- Review Coding outsource sites #1
- ID potential devs

→ Outline what ~~is~~ would be needed to utelize them + Platform, i.e.
- Scope/size of typical Job
- elements of Job spec
- typical form of spec?

Wireframes + detailed google doc

~~Content for Outline~~

2 ~~1st draft of Words~~

A) ~~Words~~

2) Content for Outline
- A) words
- B) semi-to-high detail Hand Mockups

MockNotebook/ ~~notebook~~ atomera.json

Lisa ISO: 17025

Collections

Papers

Paper1 Paper1 Auth @PDF

Paper Title2 Auths @PDF

Book Inventory

ID ISBN title cover @file

Files

- papers
  - PDF1
  - PDF2
  - CoverImages
    - ISBN1.png
    - ISBN2.png
  - Misc
    - Images
    - CSV
      - ISBNS1.CSV
  - .

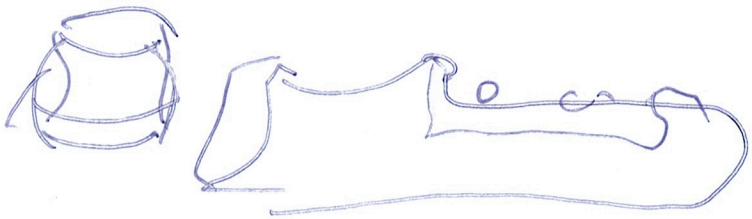
Collections

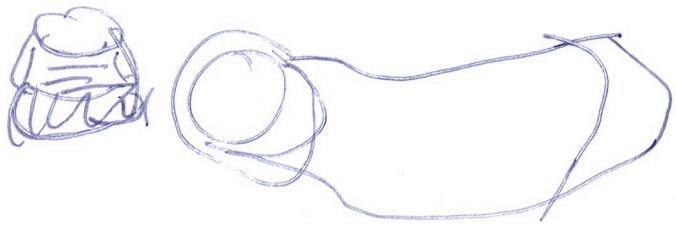
Block#	Collection version	ISBNS
	1 2	
1	3	001: aaaaaa12 002: 6666 00
⋮		
4	1	001: aaaaaa12
	2	002: FOO!
	3	003: New one

Block:1:ISBNS:v3

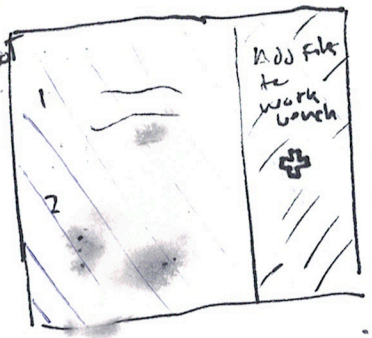
ISBNS Event log / command log

Block:	version:	state / msg
1	1	
1	2	id:001: aaaaaa12
1	3	002: 6666 00
4	1	(get state Block1)
4	2	set 1.002, Foo
	3	add (New one!)





Body
div
#root



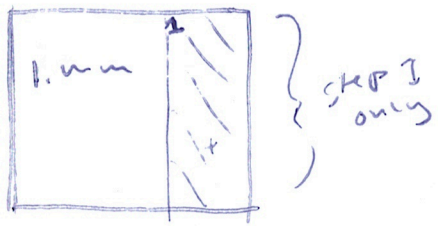
Mobx
UICx
~~dragin~~
show DND :

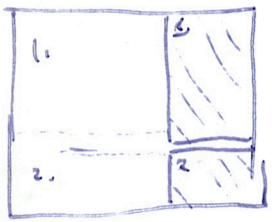
#root > App
onDrag

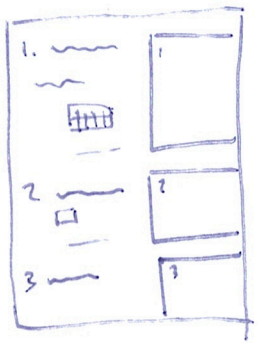
METATEM
ATEMETA

Data vs METADATA
file
structure

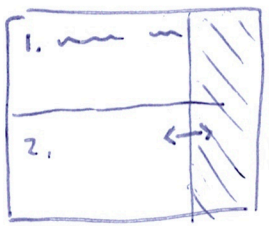
Blurring the two
FLUID
FLUWETA WETAMETA
WETAMETA

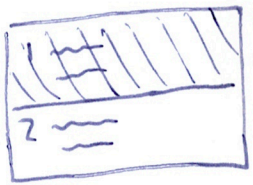


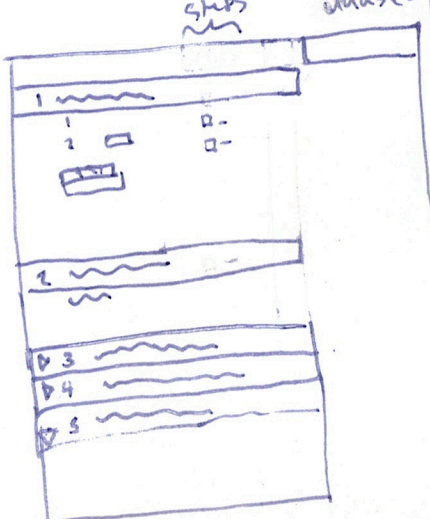




Bench







Notebook.json

Notebook.json

- collections []
- files []
  - ID: xyz1
  - Instances: []
    - Provider
    - Path
    - ID
    - version
    - updatehash / msg ID
  - ~~(FileMeta)~~

FileMeta:
- name
- mtime
- size
- mime
- ext

init

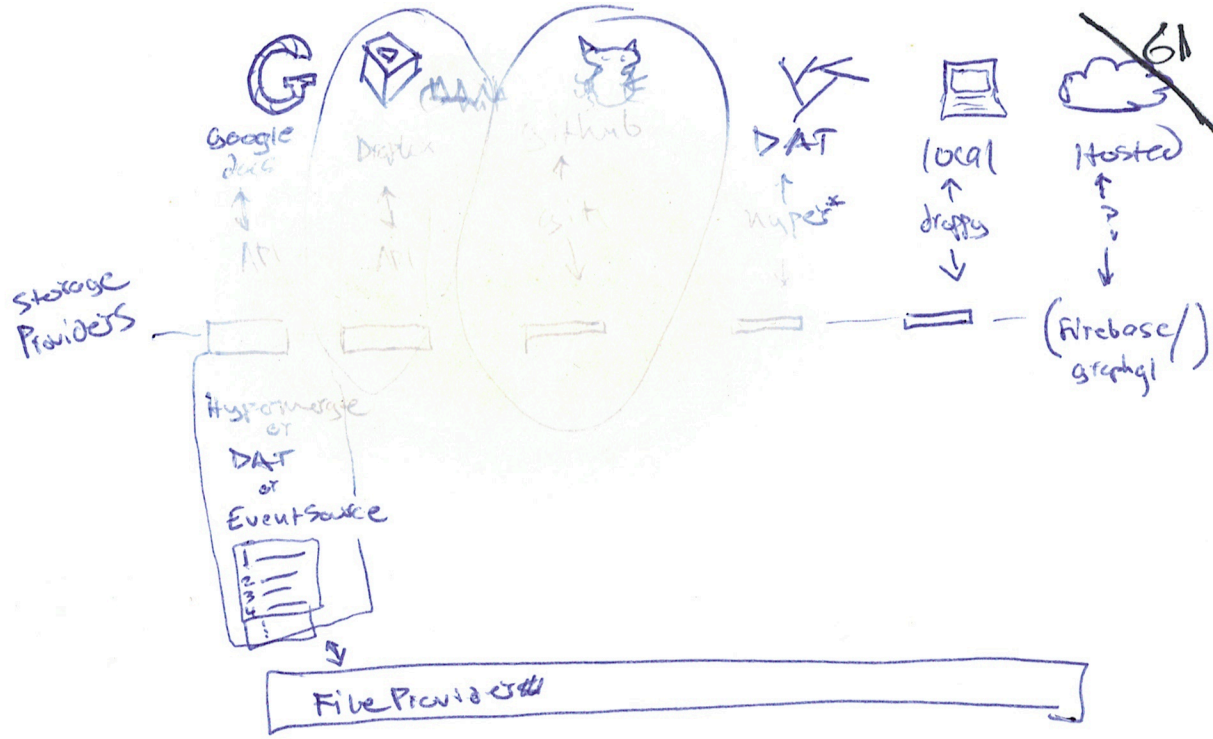
1. Create File instances
   - Put in FileStore
2. Create Collections Instances, → CollectionsStore
3. Create Record Instances in Collections
4. Construct Notebook (Frontend)
   - Blocks
   - Files in Blocks, call getURL()

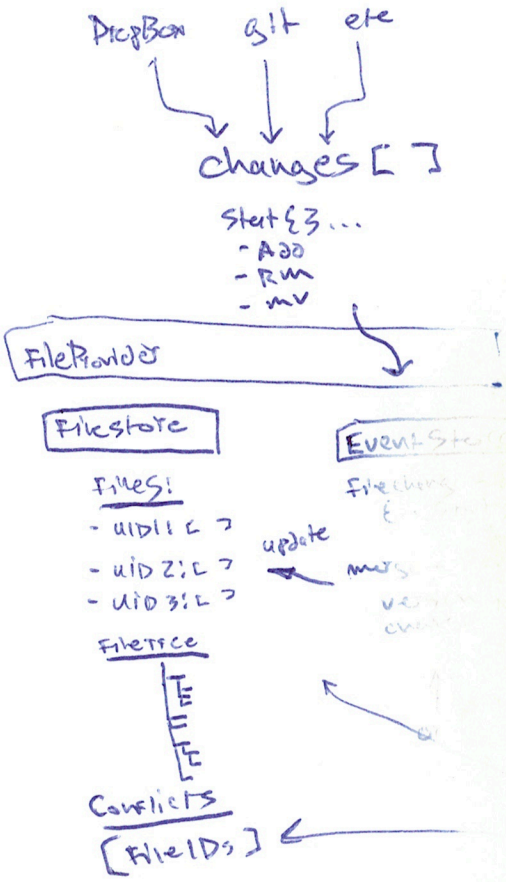
11

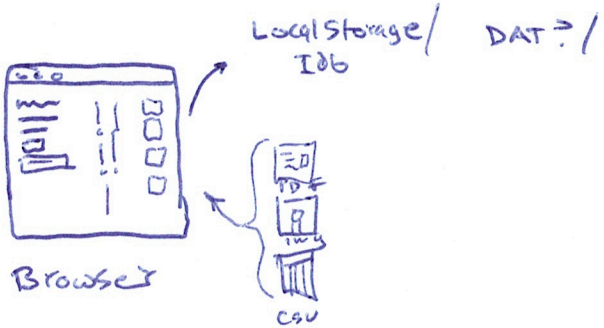
8, 24, 45
45 53, 60

What should happen when File(s) added?

- Put in Temp Files collection
- MAYBE sync to storage
- Suggest file as input to nearest block
- Check if similar file already in FS store
  - if so, ask to update







Browser

Pending Changes

1 per File Entry
- OK: ← undo
- — or —
- conflicts: ← Fix

File Preview
- save()
- autosave(Bool)

DEMO

local
- Block Editor
- collections
- schemas
- validation
- memFs / indexdb

⇓

Create Workflow → Export NB.json
→ share on DB/online

⇓

Hosted @ ~~[illegible]~~ dicect / uuid-friendly

Direct download NB

Post to Dropbox

Fork/EDIT

1

FSstore: collection

File really similar to Existing file?

Yes

2

Fs Store! File

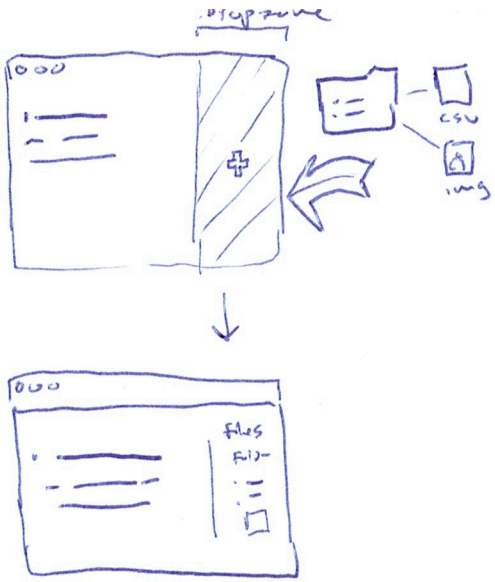
No Constructor

Filetype: guessFiletype

Preview: ()

update()

Forget Directory Explorer

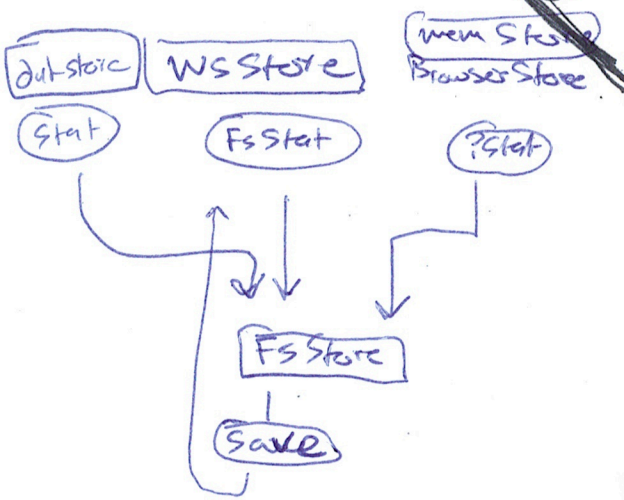


- object
  - FsStore.json
  - BoksStore.json

mobx collections

serialize

deserialize hydrate



A file is added to...

DnD: browser

handled by...

memStore

Then:

- creates URL
- updates fstore

—save?→

remote FS

handled by...

WsStore

Then:

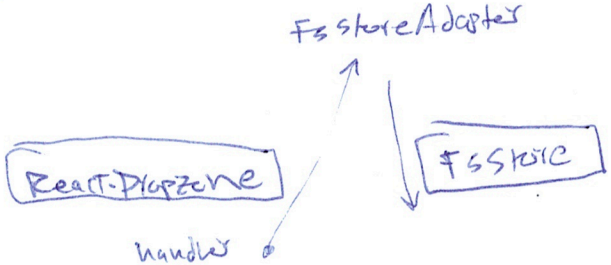
updates fstore

DutStore

stat

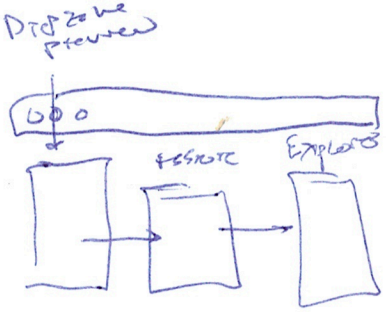
FileStore

UID: path name parent type etc



FsExplorer ~~FsTree~~

- name
- type
- mtime
- size



- See lattice CRDT app for Ex of Mobx

- Dat FS
  - mobx FileProvider collection
  - based on file-system-react
    - manages IO w/ dat hyperdrive
    - optionally:

server.ss:
- scans folder
- generates FS-Stat.sson
- hosts all files

- FileProvider inits w/ FS-Stat.sson.
  resources available over http

HyperMerge For Collections

References to "Files" maintained as hyperdrive/path/version refs

UI can "lock" these to prevent accidental changes

test app:

- Dat SDK
- Preview list of files
- Shows toast when file moved / updated
- New files can be added from browser

Frozen / snapshot / Filtered (nodes?) Entries replicated to a second Archive

Notebook can always revert record from archive

DROP

MemStore

guessFileType

collections/
files

ID
Name
Path
ext
mime
mtime
size
added

media ?
→ URL

text ?
→ string?

csv

xls
blob?

make it bidirectional

or more redux pattern

56

animation ID
rotary position
keyframe

|||

Take Notes + plan Ops

ops block shows template of expected data + config for checking/saving

Note
- date
- m m m

resources
- PDF
- img

resources
PDF
img

① data records containing references

How does editing w/ Table work?

Facts

schemas
A .1 .2
B .3 .4
U
A.1
B.4

② Display data template + validation from schemas

collection
Facts/relations
coral id:
coral type:
DNA seq:

Schemas
checks if given Fact "Bag" validates
- has subset that validate

24 June 19

- mom check
- community Gio essay
- didact plan
  └ split collection view into
    - controller & view
    - use blueprint or other frontend for view

Igem tool

Blockchain, Art christies?

"nonfungible tokens"

fundable token → art world
Immediate ~~buy~~ buy → Marketplace
no demand → demoralized

→ News Feed about founders of nonfungible tokens
"crypto punks"
"crypto cats"
too many lists

Primary Metric: get paid by Investor (raised in April)

Biggest Obstacle: Presenting Event ~~to fit~~
need figure out how to present to non Engineers
↑
solve

Here to Accomplish:

2 July '19

Consider them all strings

Primer:
- id (cuid)
- seq (dnaseq)
- name (str?)
- desc (str?)
- src (any)

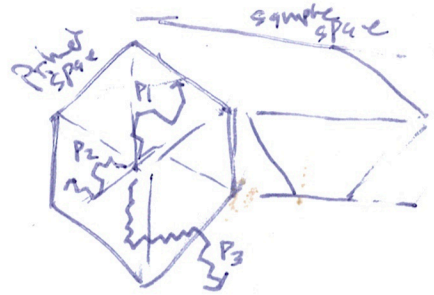
5d seq space

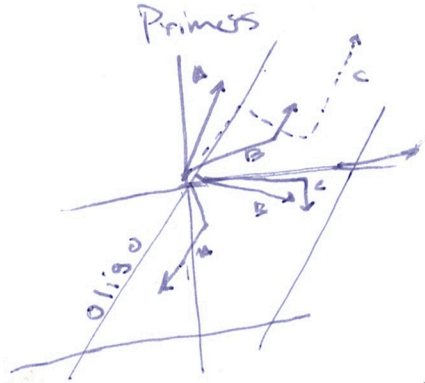
32'd ascii space

seq vector

Oligo:
- *Primer
- *sample
- cuid

Sample:
- format
- storage





Rule of naming

(tagging / defining / collecting)

not meaningful to "tag" or name fundamental types — they already have a sufficient name

~~Max set of~~ types can be reduced w/o understanding coherence of name. It needs the name

Is there SSON-seq or SSON-vec for embedding structured key:val maps or Dicts into highD space?

- can vector be cuid? ⟶ this means
- is there uniform distance metric?
- say for values that are max 32 ascii char?
- at the least it would be awesome if it could always be true that if

$$\|cuid\_A - cuid\_B\| < cuid\_C$$

then we can be sure A:B are more similar than A:C or B:C

Oligo:
- Primer: <cuid> = primerspace vector

not so good for files !

Becomes pass by val

Primers

Samples

mastermix Calc

Primer F
seq
conc

Primer R
seq
conc

Template
seq
conc

Polymerase
conc

Thermocycle program
pre
cycles
cycle
=
hold

PF: Primer forward
PR: Primer Rev
t1: template 1
t2: template 2

gel/
- size
- matrix
- voltage
- dye
- buff
- conc

gel/wells

	Name	seq	vol
1			
2	first	aagt	.22
3	load	ccta	22
4	See	acta	20

gel/image
File:
name
path

Annotate gel image
Expects:
gel/image
gel/wells/
- loc
- size
- name; in: gel.name

gell/wells/
seq ⇒ seq.size
gell/wells/well_no

gel
size
matrix
conc
wells
voltage
dye
:: → image

callbacks
+ gel image
+ ~~per samples~~
-
= annotated gel image

Well #
Sample
- name
- seq > amplicon
- length

Config
- PF
- PR
-

Samples 1
1
2
3
Config

Samples 2
1
2
3
Config

Samples 1 Config

	name	seq	PF	PR	con
1					
2					
3					

defs
- A: Sample (oligo)
- B: Amplicon length (num)
- C: Gel_image (file)

ins
- [A]

outs
- [A, B]
- C

Block

defs

ins

steps [Blocks]

outs

- Meta
  - runcon
  - ready
  - log

state

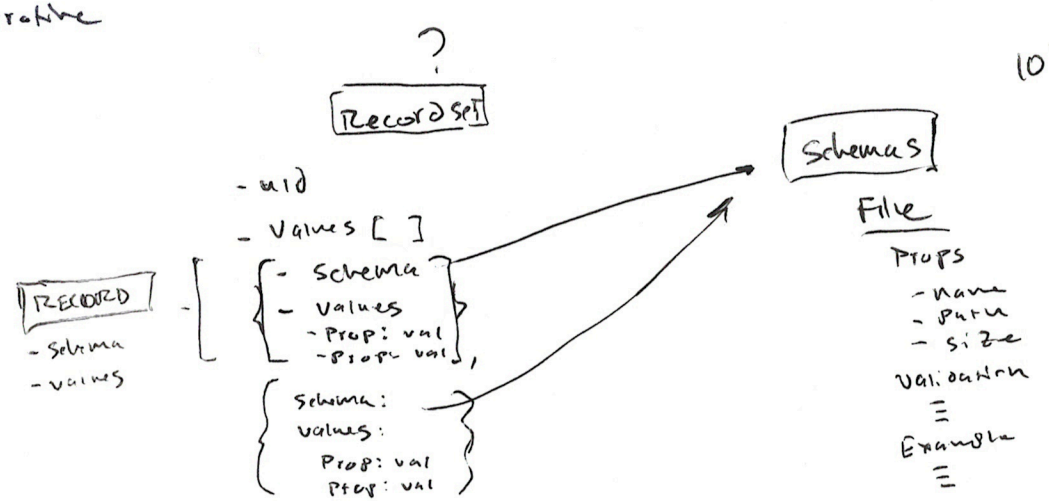
narrative

steps
- OP
- log

Enter content of Block or OP

Block/op resolve ins
- type in scope?
- shape in scope?
- show type template

10 may '19



Record class

@ computed
get Flat ! → { Schema1/prop1 : val, Schema2/prop1 : val }

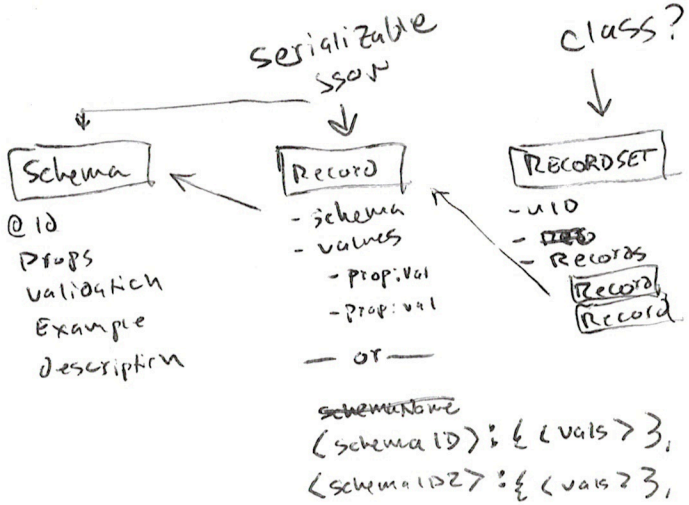
vals
set ~~Map~~ ! → { Schema1: { Prop1 : val }, Schema2: { Prop2 : val } }

@ computed
get Schemas ! → [Schema1, Schema2]

Set vals ?

Patch (vals)
upsert

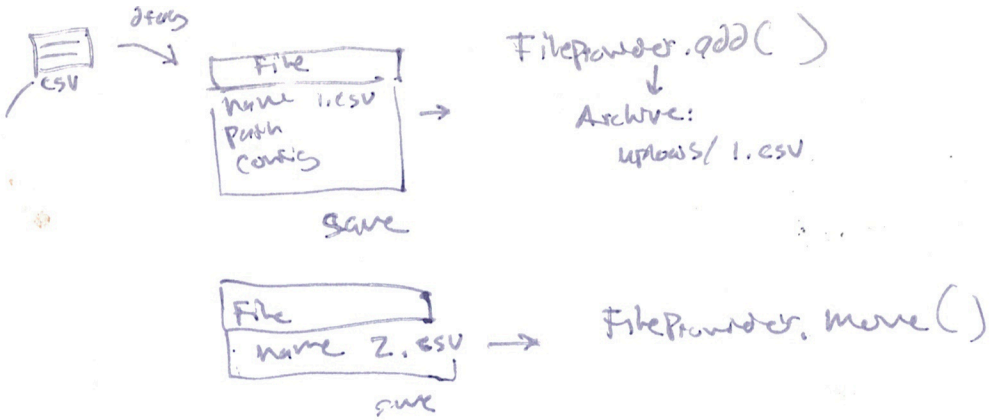
delete (val path)



File Provider

Log (Path)

diff stream



in general

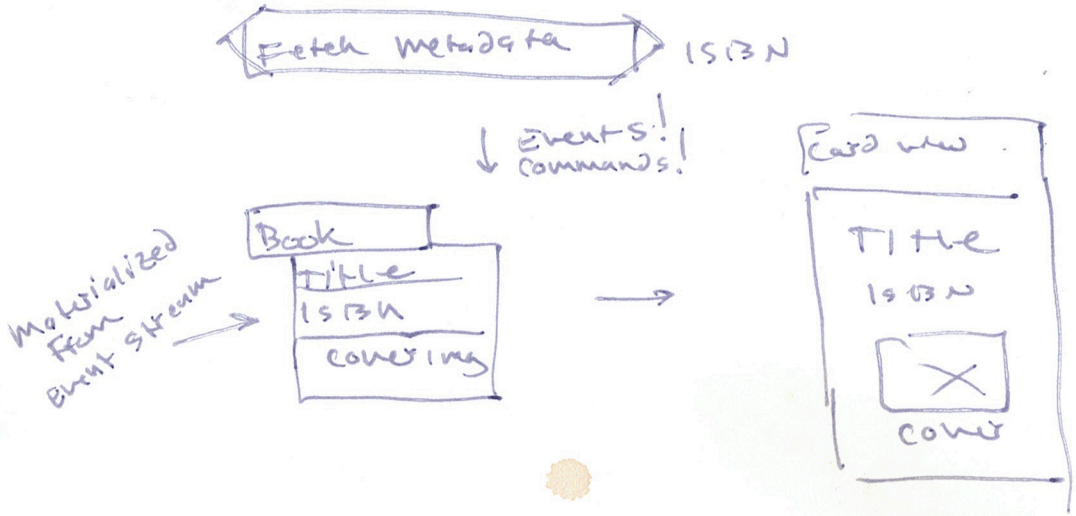
how to pass file

- csv
- pdf
- img
- excel

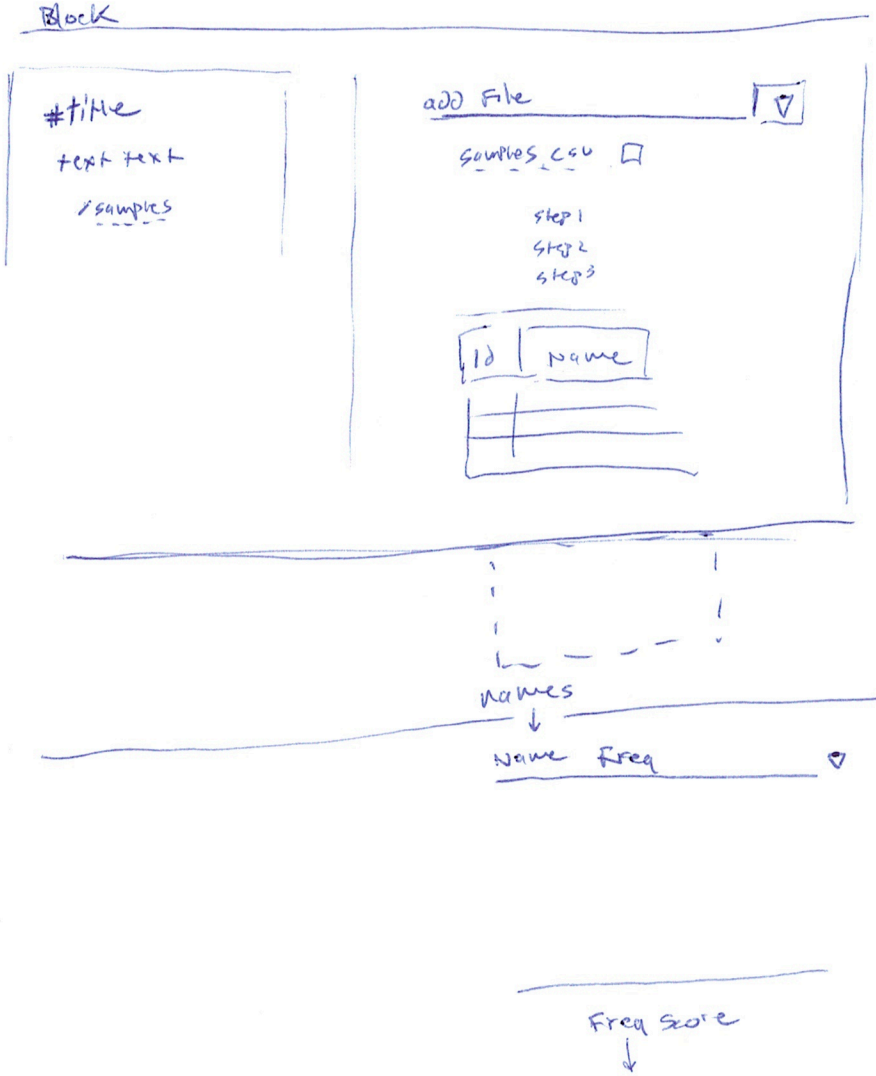
to various ss view modules?

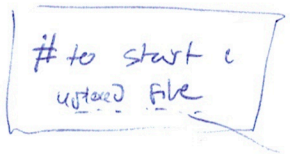
---

how to DnD add file to archive?

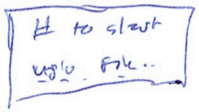


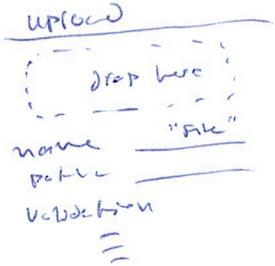
If a picture is worth a thousand words, how many words is an interactive interface worth?

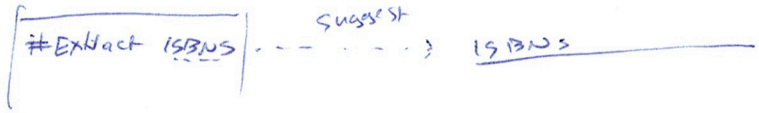


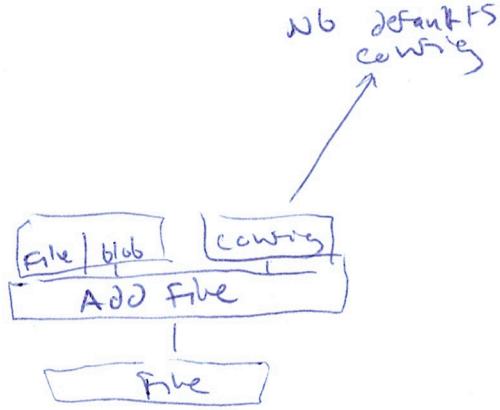


Suggest lists possible daisy op templates









Notebook

Blockgraph

b0 b1
↓ ↙
b2
↓
b3

Blocks

b0
b1
b2
b3

Blocks, steps
ops[ ]

Ops

o1
o2
o3
o3.1
o3.2

Schemas
Files
fetch
csv

Collections
c1
c2
c3

Entities

Unit
~~Unit~~
Unit

derivation
computation
derivation
refinement
production

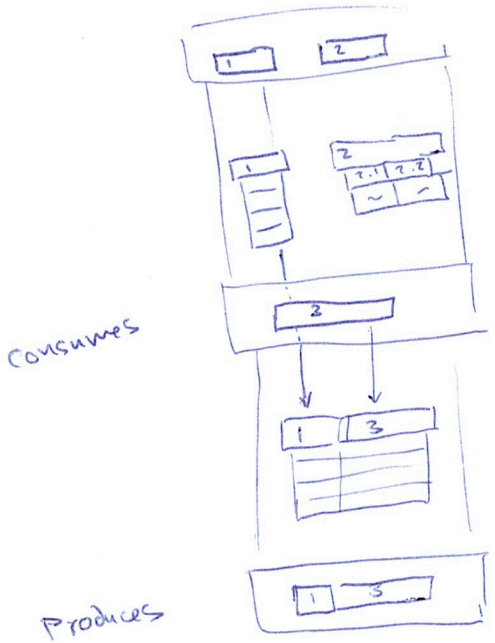
Join
separate collections
single collection
Join

How to represent op that adds a column?

ISBN
op: FetchCover

FetchCover
ins: isbn
outs: fileCover

Img



← UI starts here shows when satisfied

ops

FetchCover

ins:
  isbn
    type: string
    name: isbn

outs:
  cover
    name: cover
    type: File

op:
({isbn}) => { }

or

({...ins}) => fetch(isbn)

Scans.csv

date	isbn

name: "FileCover"
op: FetchCover
ins [isbn]
outs: [~~File~~ cover]

inputs from
outputs to

obs inputs: [isbn: —
(computed) obs outputs: File
(computed) obs ready
obs autoexec
obs runcount
(computed) obs msg

op: "FetchCover :: isbn (string) → cover (File)"

ops: —

↓

ops: "FetchCover"

suggestOp(ins, outs)

resolveOps

opStore

op: FetchCover
ins: isbn
out: File

21 May '19

Cols
- id
- seq
- note
- conc
- name

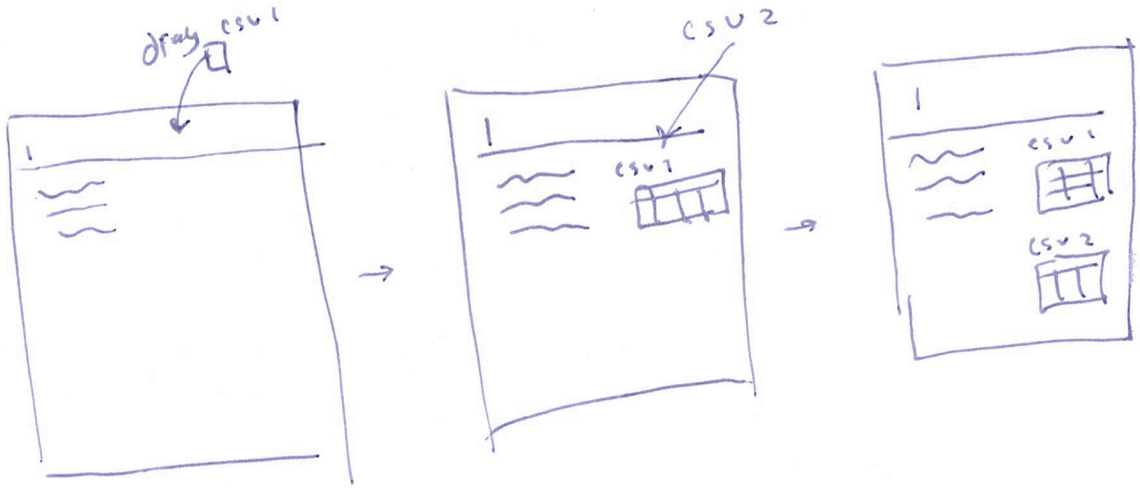
id	seq	note	conc	name

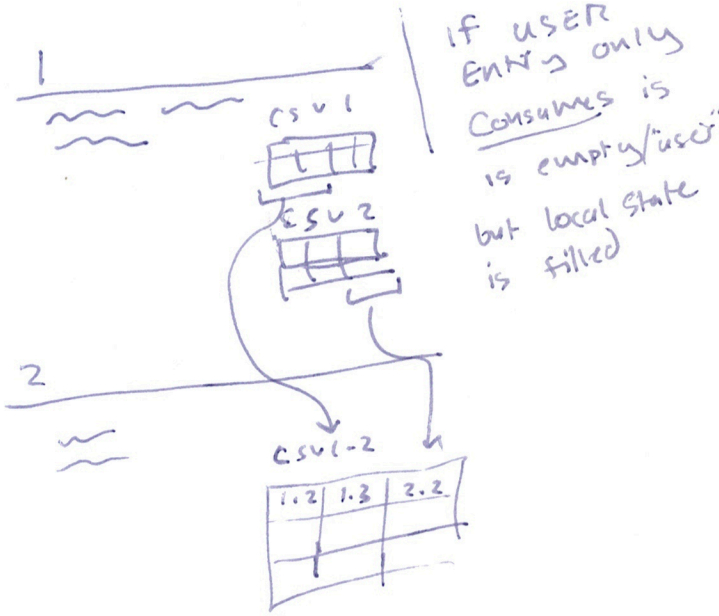
Primer ⟶ [oligos]

Primerset (2 primers)
⟶ [oligos]

oligo

id	name	conc	note	@primer (ref)		
				id	name	seq





If USER ENTRY only
Consumes is
is empty/"user"
but local state
is filled

Can Entity be a facade around JSON value of object
get/set/toss does special stuff to handle
references?

↓
Should work! Object.keys(obj) or Map.entries()
will return POJO ~~of~~

10 May '19

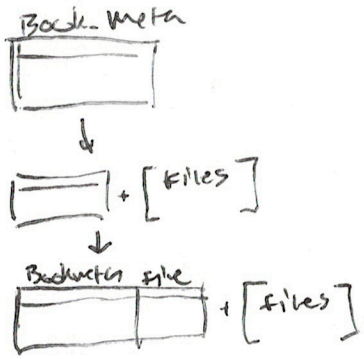
Strategy

1) Rough Mock of Output
   - Folders
   - Files
   - tables (json)

- ISBN (csv)
  ↓
- Book Meta (csv)
  ↓
- BookMeta (csv)
- Images/
  img1.jpg
  img2.jpg
  ↓

2) static UI component representing this Part

Showing



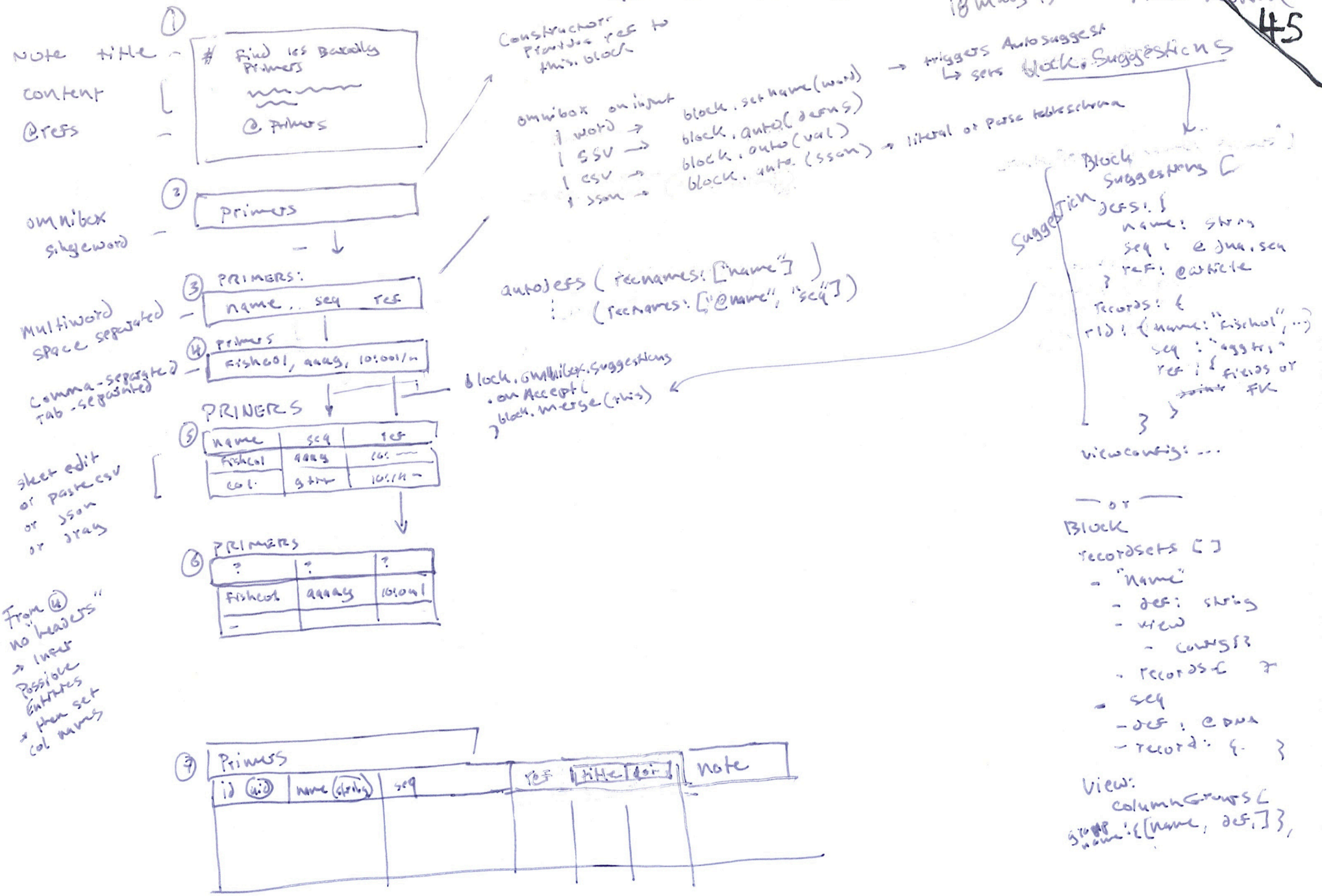
3) then work out what model / store class methods should implement to compute props for UI components

Ear
info @plushcore.com
Attn Dr Crabtree
|
Check on
Fri
↓
mild
otitus
Externa ?

Autosuggest Actions

18 mars 19

Final Action



Column type suggest

From 1) Col title
2) Inferred keywords
3) Data vals

- From computed list of all Property Names
  - return matches closest to col title

- for schema matches, return w/ extra ~~cols~~ cols to be added

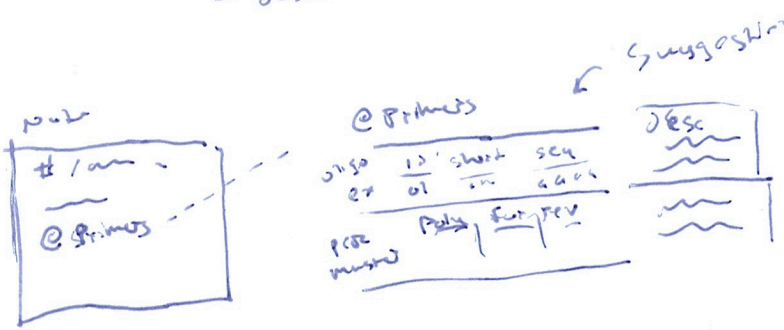
- for more than 1 col
  starting w/ all schemas that have props ≥ # col
  - Col names:
    check if schema contains props named col name + 2 edit

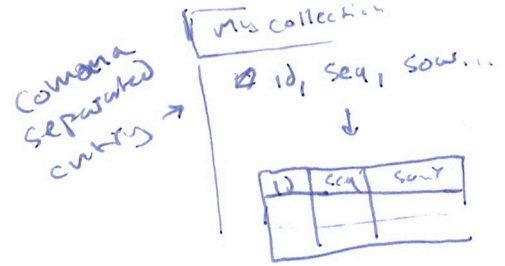
  - Col vals:
    same filtering as ↑
    return if validation passes

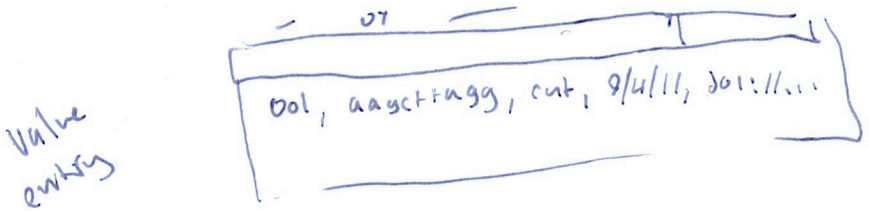
Brute force:
for each col
check each validation
for each schema

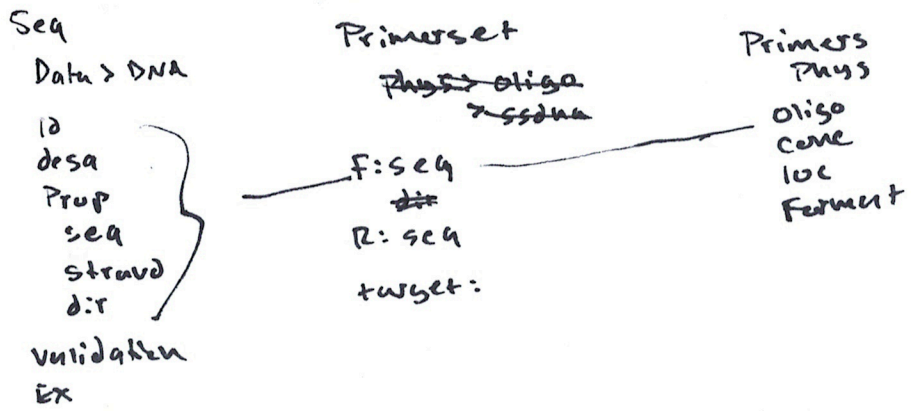
validates as [

Col 1	Col 2	Col 3	Col 4
- Name	- seq	- title	Path
- note	- oligo	- auth	
- comment	- string	- note	
- string			
- UID			
- title			
- auth			
- desc			









Projections

Context
~~Jefs~~
@ seq
@ primers
@ Article

Recordsets

Primdesigns:
Jobs
id
@seq
Note
twset
spec
@Article

It's valid if its parts are valid

If an existing resource is consumed
the consumer must
Need its original schema

If just part, we just need that part

So UI interaction reduces down larger space of CONTEXT

So Parse all known schema such that terminal props & Composites are linked as leaves in graph

use graph to resolve autosuggest
- set siblings

(1) >_

Block
- ins { }
- defs { }
- steps [ ]
- outs { }
- vars { }
- _meta
  - id: 2

(2) > Fetc...

Fetch (url) => json

op. ~~API~~ HTTP

config
- url
- Auth
- headers

Req

Res
- json
- blob

Insert op...

(3) HTTP/fetch/config

URL http://...

Auth

headers

Block
- steps
  1: http/fetch
  - ins:
    @config/url
  - state
    config/url:
    ~~state:~~ "http://..."

Pees 1:18pm
burst 190

Schema

@id
desc
Props attrib — @id/name, type/defn
Ex
valid
_meta
  desked_in

Collection?

groups / Recorders / destinations
- Bookmeta: <schema>
- Files: @file

Block

5/15/'19

@id : ~~fetch_book_meta~~ _B1
name: Fetch book meta
In: ISBNs URL

jsonSchema makes it kinda tricky to mash-up subsets of two distinct schemas - its not very composable / modular

Stencila figured out a way... using @id "base:prop" ? "from": "schema"

Primer
↑DNA seq
has phys: oligo

PrimerSet
Forward: primer
Rev: primer

Schema ( )
name
prop
:
_meta
/ desked_in

resource
Collection
- Each ~~block~~ gets its own schema?

B/C they may modify it...
or ref to Existing schemas they use?

Fetch/in
Inputs for Fetch"
attrib
Endpoint:
type: URL | string
required: true
test: /^http/
headers:
type: object (JSON)
req: false
test: ——

SchemaStore
registerSchema ( )
_schemas [ ]
Constructor (B)
this.base rootStore = B
~~when (rootstore. blocks)~~
@computed
get Schemas ( )
_schemas.concat (root. blocks. schemas)
get ~~SchemaByName~~ Find ( )
get SchemaById
action updateById ( )
clone

State is result of "op" - is literally op.produces

ops have pointers

ops form tree ins → outs

op can be seen as SOIN - INS: OUTS
or PREDICATE of SPO where S: ins ; O: outs

— or —

State is like Event sourcing Aggregate
- applies latest leaf → root chain
of ops

result

PrepSamples → consumes samples
produces samples:

id	note
1	foo
2	~
3	~

Samples:
op: user Entry, state: [1, 2, 3]

1	~
2	~
3	~

op: pcr , state

op: gel

Ref/pointer to op

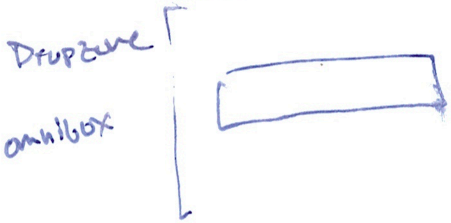
gel → op: gel_electro
consumes
gel_config
samples
Produces
gel_image

Samples: PrepSamples [1,2,3]
config: <filled in by local op>
gel_image: <ref to File/image>

Samples

id	note	"gel"
1	~	<ref>
2	~	<ref>
3	~	<ref>

Block



notebook

Blocks
stuff

Mock

collections
"first"

Block
inputs
"rename": collections.first

state
"rename":

collections
add sson → new cxn
[cid: new entity

cols
rows

...

- 2 mock collections
  - scans
  - authors
- 2 mock blocks
  - empty
  - scans
- Blocklist
  - add block
- Block
  set input
- Collection
  name
  defs?
  data

bercie et al
"Remember me"

- Walker
  3pm

Mom workshop

- Lisa Party scan prints

- Feles one

Produces captures how state will change

name | date | file
string | date | @ref/schem/file

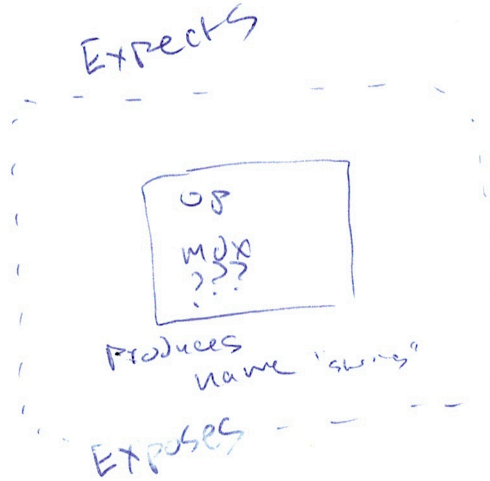
Thing1: {
name: –
date: –
file: {
filename
file type
}
}

Bike
flowingdata
Remote
clean
batteries

BM LEDs

Entity UI

Produces
... state
... Entity



ugle

ops that in/out
this type

Entities that
match this type

Collection
name
desc
schema

Thing2: {
self: {
name:
date
}
file: {
name
path
}
}

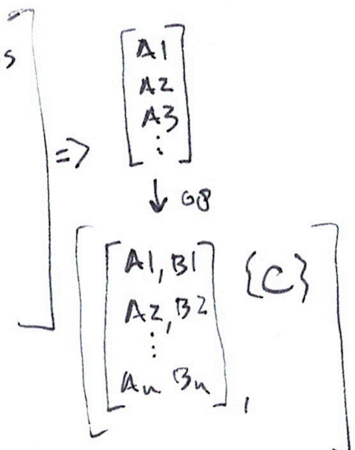
collection of Entities

definitions

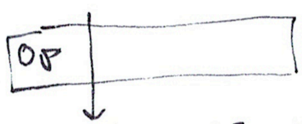
defs
- A: fields
- B: "
- C: "

ins: [A]

outs: - [A, B]
- C



ins - input1



outs - input1, out 2
- out 3

if out type
not in ins then
infer it was gen
by op

name | date | filename | path

mobx - Atom

State

Ident: Isbn
Spec:
Name
Desc
Example
Expect
Value

view / lens

Cxn.Name

ISBN
(value)

Cxn.Name

id ISBN
val value
spec + desc
- example
- Expect

JSON

Browser

Blocks
1
2
3

Block >...

goal is iteratively prompt
For input until it unambiguosly
selects for 1 col in inputs

frame "Scans" CSV File

frame "Scans" > Date Scan type

"Scans" > ISBN

ISBNS > ISBN

addProp (id ~~name~~)

frame "Scans"

Date	Scan	type
scan record →		
→		

"Scan"

Collections

ISBNS

Entities:

ISBN

Entity

ISBN

= Name

= Validation

Example

Author

=

Description

=

Books

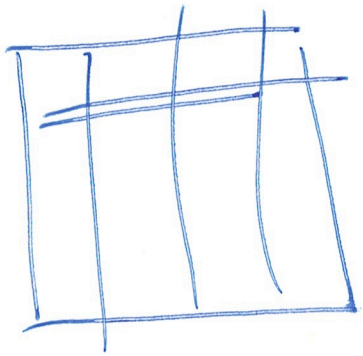
ENTITIES:

ISBN

Autor

Desc

DATE



Name

desc

spec

Block

names of things
their shape
list of operations

Block Store

Imperative:
These is this state:
name schema values

Declarative:
These will be this
From that

name1 schema → values1
name2 schema → values2

op:: inschema → outschema

opExec

New schema 1 → values

What is diff about
suggesting next OP
vs. suggesting next
Block>

Block is State
names
schema
→ values
points to

OP is transformation
of state
schema A
↓
schema B

Subject verb object
subject op object

step:
Take these things
and do this op
to it

names/vars

OP:
interface
function

~~Nouns~~
naming

Spec step
Flow model
Node
Procedure
Production
Interface

literal

① ISBNs — values

Props: isbn: type = — isbn: aaaa, isbn: bbbb, isbn: cccc

② ISBNS2 — values

Props	isbn	title
isbn	aaaa	Foo
title	bbbb	bB
	cccc	cat

indirect

① ISBNS — Props: isbn — rows 1/1: isbn/1, 1/2: isbn/2, 1/3: isbn/3

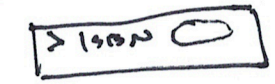
isbn
1: aaa
2: bbb
3: ccc

② ISBNS2 — Props: isbn, title

rows	isbn	title
2/1	isbn/1	title/1
2/2	isbn/2	title/2
2/3	isbn/3	title/3

isbn:
1: aaa
2: bbb
3: ccc

title:
1: apple
2: bat
3: cat

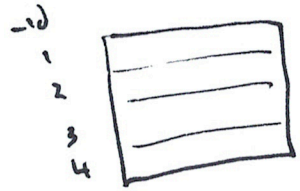


Physical
samples
reagents
inventory

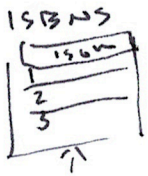
Digital / dbm
image
file
sequence
PCR program

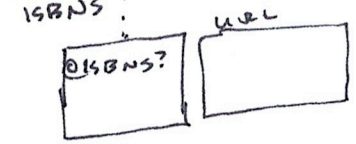
ISBNS — RV — setup config

ISBN
strips
desc
valid



Schemas
core
~~Index~~
local ()?





defs

ISBN: C/docs/isbn

~~API URL: @URL~~

book_meta ~~title~~: @ steps

ins

ison

outs

book_meta: isbn title auth

steps

1 op: http/API_fetch

ins:

- URL

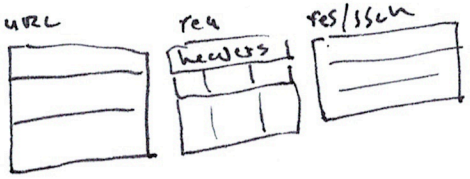
outs:

- req
- res/ssen

steps

1. fetch

URL:

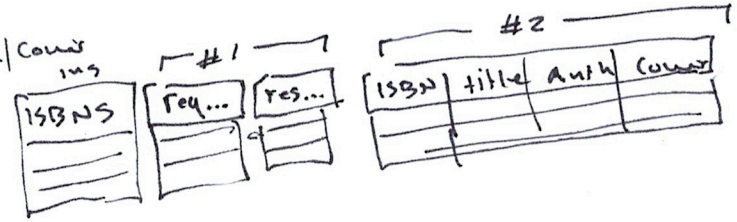


2. ~~select~~

ssen.parse

res/ssen → isbn | title | Auth | Cover

ins



3. ~~select~~ check

ins.ISBN === isbn

4. select / 4. DROP

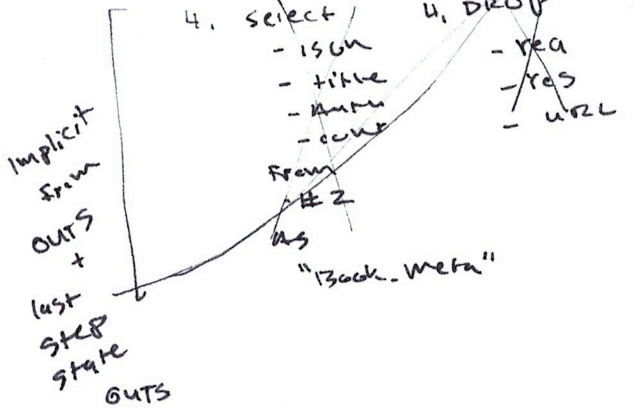
- isbn
- title
- Auth
- cover

from #2

ins

"book_meta"

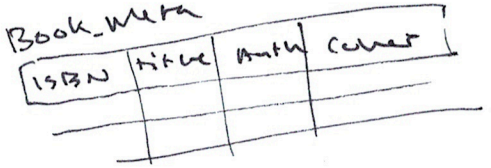
- req
- res
- URL



OUTS

book_meta:

- ISBN
- title



~~REST~~

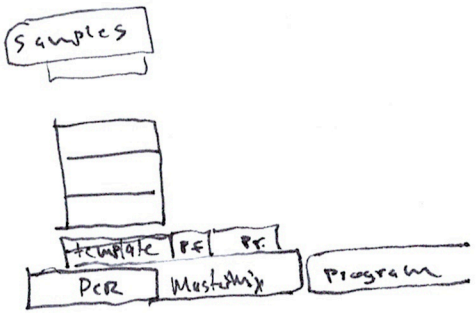
types
- add
- get
- contains
- usedby
- Find
- suggest (prop name)
- infer (extdata)

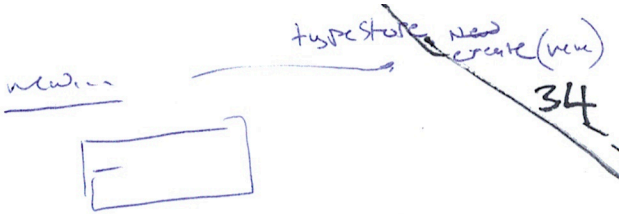
~ usedby(type)

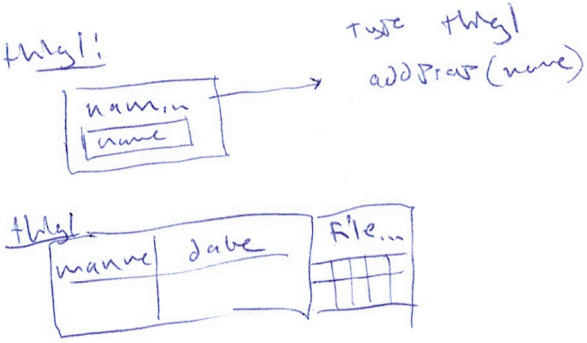
type
- _id
- name
- desc
- props
  - name
  - type
  - Example
- validation
  - Example
  - Tags
- usedby

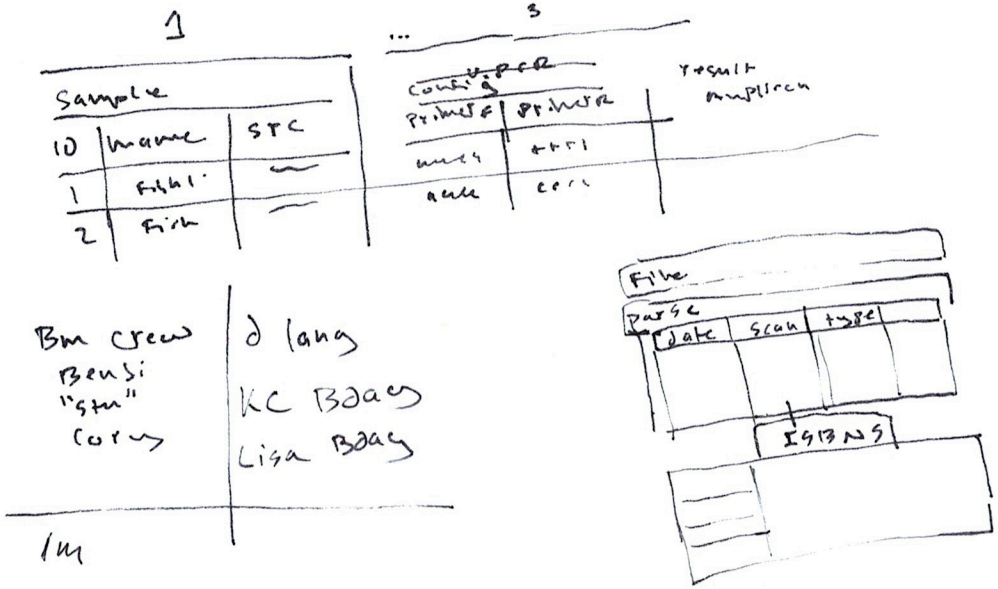
Types
- Base
  - date
  - string
  - int
  - blob
  - url
  - os
  - md
  - ssx
- custom
- Composite

Tag
semantic tags
"stags"









File

File/image

File/pdf

getImage
- set-
- File/image

650 260 8591 (2) mac tab 1

18 May 19

Computed?

RootStore
BlocksStore
SchemaStore
EntityStore

Block
narrative
@c defs/content
@c consumes
@c Produces
data → recordsets: name: recordset
name: recordset
Suggestions
Inputs?
Outputs?
viewconfig

RecordSet
rowid:
· name: entID

Entity
Context:
def
def
def
key | val
key | val

Entity
name Primer
defs Schema
id: type: id
seq: type: @dna
text or file → file: type: File?
vals
_id:
- id: 111
- seq gacatc
- file: id

USE CASES

Enter # item w/ ref to other schema on one line w/ values for ref props

partial defn of ref schema
→ create/show template of missing cols

@oligo
id
conc
comment
@primer.seq

Block state
[Recordsets]
name: oligo
Props:
id : str
conc str
vol str
Primer:
id
seq
File:
id
path

→ recordSet
_rid:
id:
conc:
vol:
"@Primer/id":

join
join

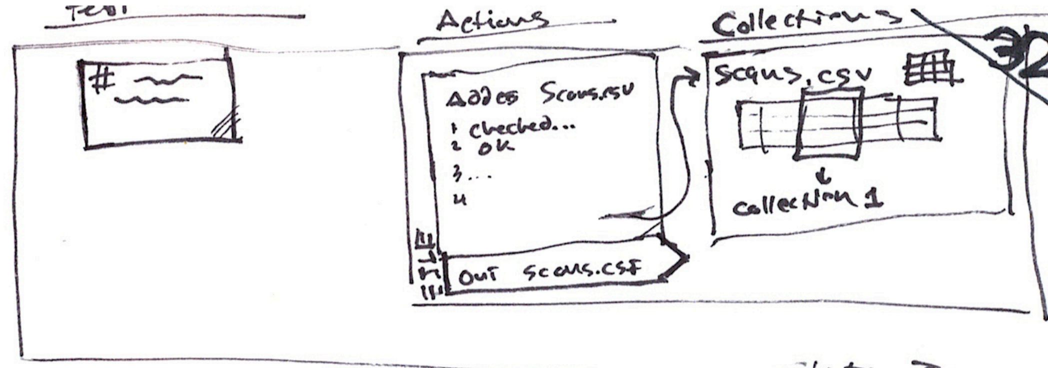
Oligo
id | conc | vol | comment | primer:
id | seq | name | ref
File:
path | blob | type

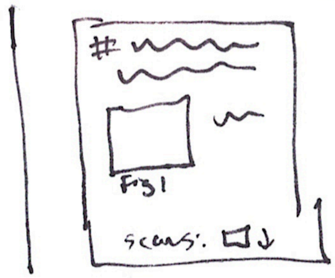
OPLOG

records salient user interactions

- linked-list?
  - Tree?

of STATE
↓
op
↓
state delta





Create Collection
Source
Scans.csv
#collection]

State Tree

collections:
Books:
Source:
"Files.csv"
ref
Block [illegible]

Files:
(uuid: 1 }
filename
path

Does <OBJECT> have DUCK-NATURE?

API ~~Operat~~ > Plugins > Operators

Obj

Prop Name	Prop def
_meta / uuid, name	meta.schema
coverImg / path, mime, ext	File/image /path /mime /extension

static typing
~~API.http.get~~

API.http
local typedef
~~req~~ Consumes: <name; schema>
url: url schema
– validators [ ]
Produces: <name; schema>
req: req.schema
res: res.schema

API.http.fetch
consumes:
FileURL: url.schema – validators
Produces
FileBlob: file.schema
– validators [ ]

(cont.)
API.http.fetch
duck typer:
FileUrl: (f) =>
should.be: URL

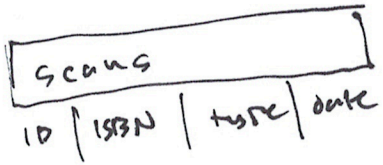
Collection of OBS

_meta		coverImg		
uuid	name	Path	Ext	mime
1	First	~/First	png	image/png
2	Second	~/.	jpg	image/jpg

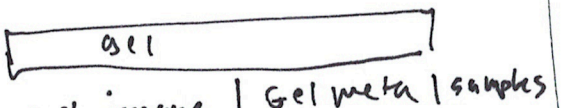
Asv vs Yup

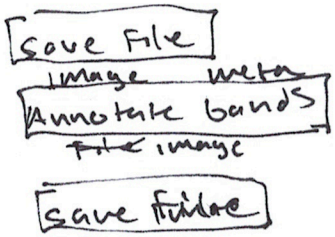
Dispatch - reducer
Plugin typecheck

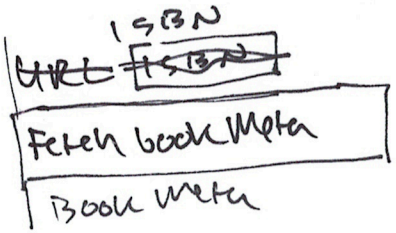
Columnar data store



opManager (op)







operations
Doc. plugins. register | Registry

Plugins

InputTypeMap → (type) ⇒ [Actions]
OutputTypeMap
CompositionGraph (typeName (node))
typename: can Input from / can Output To / Producers Of / ConsumersOf

OpControl (JSX) (OP)
- construct reacts from spec
- attach handlers from opManager

gel_Image

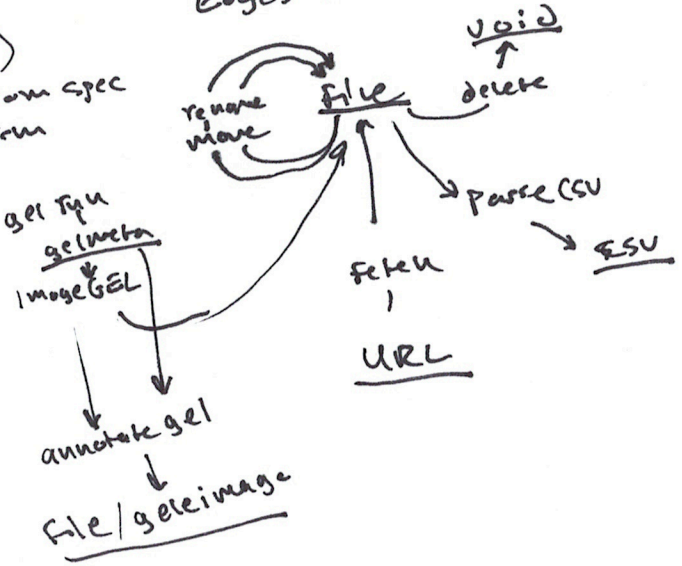
files actions

has gel metadata

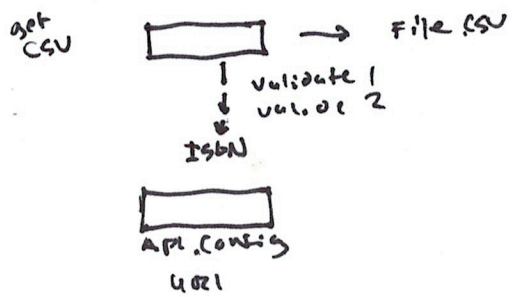
annotate_gel_bands ( )
Suggested after GelImage block

Action Chain Graph

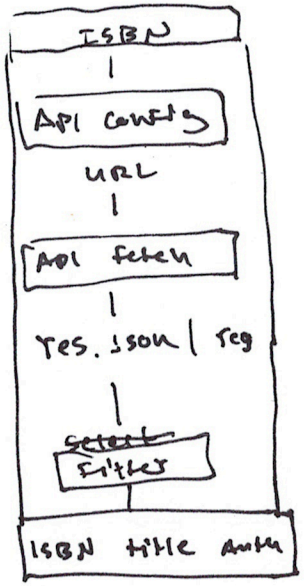
Nodes are types
Edges are actions

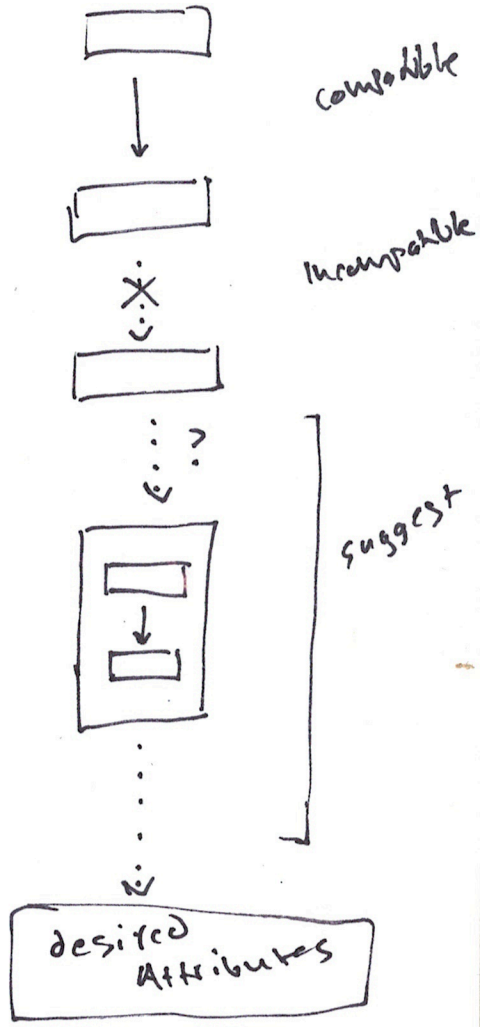


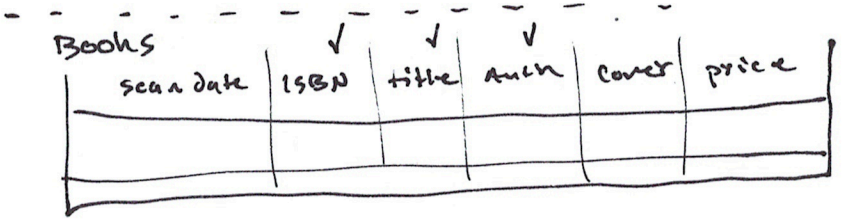
Is block ready



API.fetch







~~Modular~~ Module System / workflow Structure (AST? Grammar?) 29

A lot of this

Minimum:

- Modules define 1 or more actions
- actions defined in a Block
- a block has Inputs [] & Outputs [ ]
- each input & output **MUST have**:
  - ref name (aka variable name)
  - ~~logical type (~~
  - type definition in form of validation rules (runtime)
    or
  - logical type i.e. SS scalar (int, string etc) or obj
- Each input & output **SHOULD have**:
  - description
  - Example or default value
  - more runtime validation logic w/ error messages
  - taxonomy paths ("semantic types")

∈: "member of"

⊂: "subset of"

⊇: "superset of"

∩: "intersection" (and)

∪: "union" (or)

Duckee

T: ~~this~~ list of scalar types $[t_1, t_2, t_3, \cdots t_n]$

S: set of schemas of ~~scalar~~ composite types $[S_1, S_2, \cdots S_n]$

$t_x = \{t.name, -type, -tests\}$

$\{t \in T : t \in S\}$

$S_x = [\,](name: t)$ ordered list of named scalar types

Duckee(S) => ordered list of S where all t in S are in T →

Duckee(K, S) : ⟨compatMap: C⟩ $\{(t \in T, s \in S) : t \in s\}$ →

"The set of all t ~~that are~~ scalar types that are in the composite set S of composite ~~composite~~ types"

in [

K: list [] of named scalars
ex: [filename: string, length: int]

S: set {} of schemas (composite scalars)

]

out [

~~C: Compatible map~~

C: map (S → [K]) of schemas compatible w/ certain K

]

- file
- scalars
- notebook
- HTTP

[... core schemas]

~~"fileSpec. schema"~~

Ex: Duckee([fn: string, ln: int],

{ File: [fn]

Scalars: [in]

~~notebook: null~~

~~HTTP: null~~

}

Operation
Autocomplete/
suggest

ops.compositionGraph
nodes are types
edges (in, out) are ops

Resolve potential operations from untyped values

Match table col to known Schema or "Type"

Schema Matcher

for each column in table
  if column has SchemaId
    and schemaId is in schemaRegistry
      add {schema:[col:id]} to _Types (set)
        score=1
  else if column has name
    and name is in SchemaRegistry, then
      w/ score
  else if SchemaRegistry.FuzzyMatch(name) > 0.75
      w/ score
  else add {unknown schema:[col:id]} to _Types
    score=0

1

Sort of static analysis shortcut

For cols in Types where score < 1

TypeRegistry.map(s) => (
  s.validate(col)
).filter(). → add to _Types(col)

2

Good idea to keep track of validation messages...

Now we know duck type interfaces of each column

col_1: String
  Isbn
  Isbn_10

3

Now we have Typenames for each col (more than 1)

col1	col2	col3	col4	col5
uuid	Isbn	Url	FilePath	Date
int	string	string	@ref	date
string				string

TableTypes & ColTypes or blockOutTypes

~~ops.all.inputs.map(inTypes) =>~~

ops.all.map(op) =>
  op.inputs.types.~~map~~.map(ins)
    ins.map(-in) {
    Block.~~out~~types.indexOf(-in) > -1
    }

Something like this but use sets — see umbrella

Available Ops: [
  "Fetch/Bookmark":
    col1: uuid
    col3: Url
    col2: ISBN
]

Suggestions

Data Suggestor
op Suggestor

Upload → file/add

Barcode.csv → Filename
↳ csv

Narrative / command bar

upload Barcode CSV

select ISBNs

fetch book meta from ISBNs

DATA

name Barcode.csv
type file/csv

op file/upload

config parse CSV
validate

op: Http/fetch
in: ISBNs

Data

Files

Barcode CSV

Filename
type
save true
path 'brcms'

contents

collections

ISBNs

~~Add~~ Join (rows) (rows)

ISBN → URL

in: ISBNs
Create column: URL

~~value~~
values:
table.compute
http:// + ISBN

in: ISBNs

op: table compute
in: ISBN
out: URL

Suggestions
Operations [
{ op: "Http/fetch" }

HTTP/fetch (URL)
desc
config
(JSON) (req) (res)

in: ISBNs

op: Create column
in: [- name: # API-URL
- type: URL

op: compute values
in

Block 3:

uuid	ISBN
aa1	...
aa2	2213
ab2	3331

Perspective

ISBNs CSV

/file/add/CSV

↳ File
↳ isbns

CSV

ISBN →

get metadata

ops suggested based on # title words

OPERATION: Name: Fetch Book Meta

oplog 1

oplog 2 FetchBook

title	Author	ISBN	cover img
...	..	...	..
...	...	..	..

OP:
name: "get_book_meta"

~~Fetch~~

uuid	?	URL
	:	?
		?
		?

OP: # Api.Fetch
In: block3 // url

ops.local.FetchBookMeta (ISBN)

API URL: `http...${isbn}`

↳

uuid	ISBN	URL
aa1	1111	http//...
aa2	1223	"
ab2	3333	"

ops.Api.Fetch (url)

↳

			bookMeta		
uuid	ISBN	URL	title	name	cover
aa1	1111	http	aa	xx	http...
ab6	222	"	...	ss	http...
ccc	23	"		cc	"

~~Fetch~~ depends on URL

Since it isn't in scope, it adds a column in scope

1 —— Bo char ——— 1

upload some.csv

Δfile...

upload some.csv

uploaded

Files

Files

Bscans.csv

upload some.csv

- user added File bscans.csv
- saved to "/files/bscans.csv

upload some.csv

▽ user added File bscans.csv
- saved to "files/bscans.csv

▽ parsed file "bscans.csv"
- validates as csv (ok)
- has scan columns (ok)
- collection Barcode scans

Files

Bscans.csv

click...

▽ Barcode scans

steps
actions code
blocks

config
sub blocks

markdown blocks

card

list

table

Blocks capture
map ins → outs

OPS capture
implementation

Collections
Values
groups of objects

schema universal
unambiguous
semantics of
data shape

2

some step header txt
blah blah md ssmd

add file

▽ collection csv scan

config
validation:
1 csv
2 col name
regex: csv...

More text see
inline config widget
above

3

Need to make JSON AST for table operations...

Schemas
ISBN
name isbn
desc
example
spec

MCUP
Create Column
Insert Column
Compute value
Insert value
Set value

in ISBNS

in schemas
ISBN: @/isbn

op: table/~~compute~~ ~~cell~~ create column (...)

op: table/computeColumn

opIn ISBNS

op: compute Column
isbn → `https://... ${isbn}/scan` ~~for table (isbn)~~

opout: ISBNS

opOut schema:
ISBN
- spec
@val.is.ISBN
- type ??

API_URL
type: url

1

Files
Barcode scans.csv

ISBNS

ISBN
111
222
321

2

Files...

ISBNS

ISBN	API_URL *
111	
222	
321	

operations
1 set API_URL/2 "http://:foo...."
2 <selection>.map row `http${isbn}`

* pops up suggestion to satisfy deps

4

Files...
ISBNS...
Books

title	Auth	date	ISBN	cover

* inserts deps into op out schema
- up to user to find way to satisfy

560 px

Note row

①

TEXT Block

name
(icon)
content

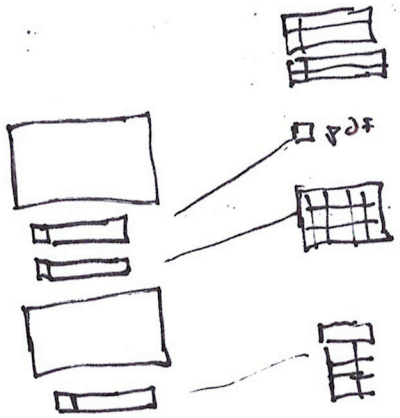
code block
action

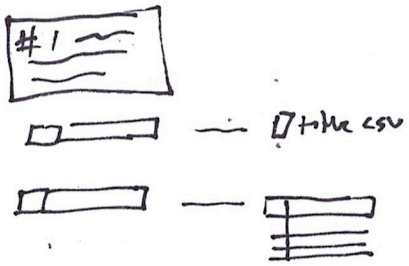
action block 2

data block 2

Editor - md render
Editor - click to edit md
Editor - inline data card
inline data Ref

Collection - list view
card view
table view





React-easy-state etc

21

Mock Table Preview

- ✓ Highlight ISBN col
- Log
  - assuming headers ( )
  - looking for ISBNs
    (~~#~~ string AND
    (length: 10 or
    length: 12)
- Found 12 / 12 rows, 0 invalid

ISBN Collection
Flex display

OP: Fetch meta
openLibrary

ISBN → openLibrary → Book:
(fields...)

Table preview

Fetch covers images

URL → File
Storage: covers / <isbn>_cover.ext)
log [downloaded <name>...>]

[Show Source] Expand ↓

[JSON]

~~OP~~
BlockSequence: Z
desc
note
ins
1. type
   ISBN
   Source
   Scanned Books
outs
1. type
   Book
   values
   BooksCollection
OP
openLibrary.FetchMeta

Table

Result

In | Out
Scanned books | Book
ISBN
an...
2...
3...
...

Table

Blocks

1 In ISBN

type: fn

outType ISBN | URL

outVal

inType URL (ISBN etc)

inVal

2

OUT FILE URL

type:

Preview:

{ "http://s.org/isbn" : {

Component: "<TextArea>"

default-props: { text: "0511-1234-..." }

}

Lisa - Sull

Tracker (expiration date, annual review) → not so good

+ Doc repo must be 21 CFR → propel docs ok

+ electronic sig must be 21 CFR

+ want change control → not so good

Doc operations

- watermark every page
- veeva would archive source doc copy in original format, then provide PDF w/ watermark
- After SOP superseded date, veeva auto~~matically~~ally disables old version, but only after new version has been uploaded & approved

Docs:

PDF Rendition (~~watermark~~ + approved)

Source
V1. Word
V2. Word
V3. PDF

→

V1. PDF
V2. PDF
V3. PDF

Metadata
- watermarks
- deprecation date
- change control

Given 1) name, & 2) Example value,

infer type

ISBN: "24-2177-000-X"

Cover: file/svg image

<Schema>: <Example>

name: ISBN
type: string
Example: "24-2177-000-X"
description: "....."

How does Sanity figure out props to display in list/terse views?

Collections [New]

hover to see

▽ Books 27 items [edit]

Description of collection

Schema: ▽

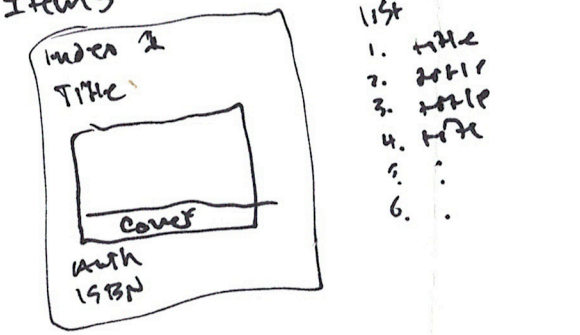
ISBN (primary key) (required) (type: string) validation:

24-177-0019

title

author

Items



List
1. title
2. title
3. title
4. title
5. .
6. .

HI DVO pieces

FTB tax

goal: SPA expt showing

Narrative + collections + oplog + dep suggest

[+] [-]

Files
1. temp.csv

ISBN
1 AAA
2 BBB

[name: isbn
desc: ~~~] collection

schema: block1/csv1

parse csv

Is it ready?
For all inputs
- all input validations pass for > 0 row

No Yes → show exec button

What does it need?

[input group name
col
name
schema]

Any blocks upstream w/ output = needed input

"block1.isbn/1": 2 1 0 0 2 3 (

blocks

Books
- desc
title
isbn [9 0 0 , 0 2 2 ...]

block2.isbns.1:

block6

"blocks.books.isbn.id: 21123

Input ~~Scans CSV~~ : dropzone

todays_books.csv

CSV ~~file~~ file

steps: save to <name>

~~read~~ parse CSV

Output SCANS

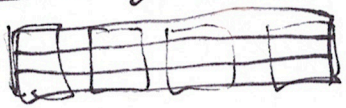
Input SCANS

steps select ISBNS

Output ISBNS

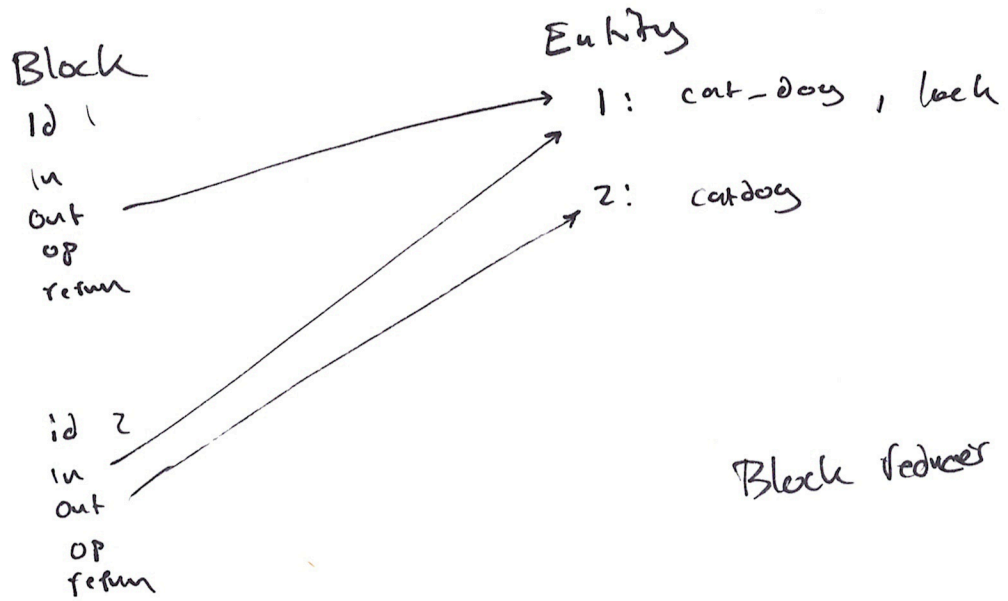
steps fetch meta [status messages]

Output Books tableview cardview



steps download covers [status messages]

Output Books Tableview cardview



Start:

actions 1. deserialize_notebook

For each block:

1. deref input
2. if autorun ∃ !input.waiting:
   run op
   ↓ dispatch running
   resolve out

Block reducer

default:
return deser_ins(state)

better to have toplevel state

- Steps 1-3 → TackBlock
- ~~OpOut~~Presentation: Tag | json | table | tackblockback
- steps [1. ..., 2. ...]

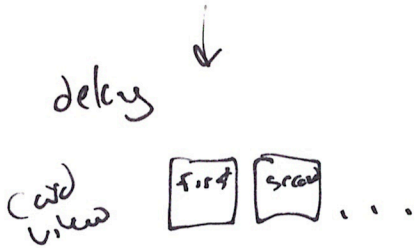
Todo

① generalized ui update
delay animation

② notebook block component
- Table view
- card view

autorun button

List [first second third]



out2 [array]

in3 : out2

op delay map x → x

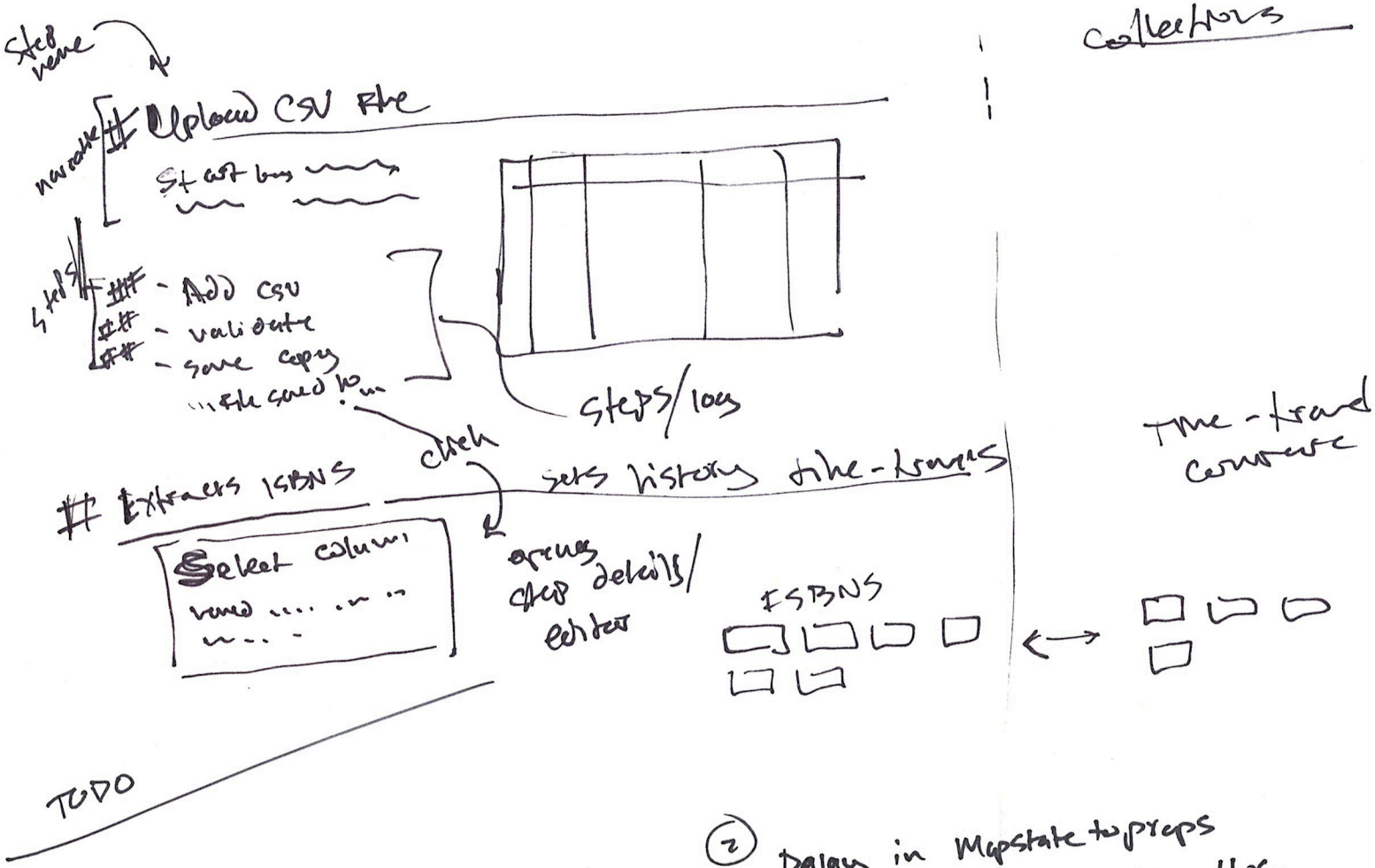
result : cards

out3 : [array]

[input]

Operation : forEach, APICall
ready? finished?
autorun?
running?

~~Display~~
Presentation (input)
table | card | namepill



① Schema → Preview
card table list
→ editor
card
Table

② Delay in Mapstate to props
Delay Enhancer or Hoc

③ OpEngine as Middleware

④ download file op

note ref ↓

note w/ underlined reference to attachment

Email / send ↓

scans.csv

added to <path>

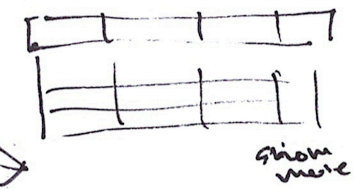
name "scans.csv"

type <mimetype

meta Local Attachments

Attachment details

table cards



show more

medium preview

"ISBNscanned"

name

Schema Editor

ISBN

type string

rules

13 ~~number~~

length 10 or 13

ISBN Fetch

In: ISBN

out: Book

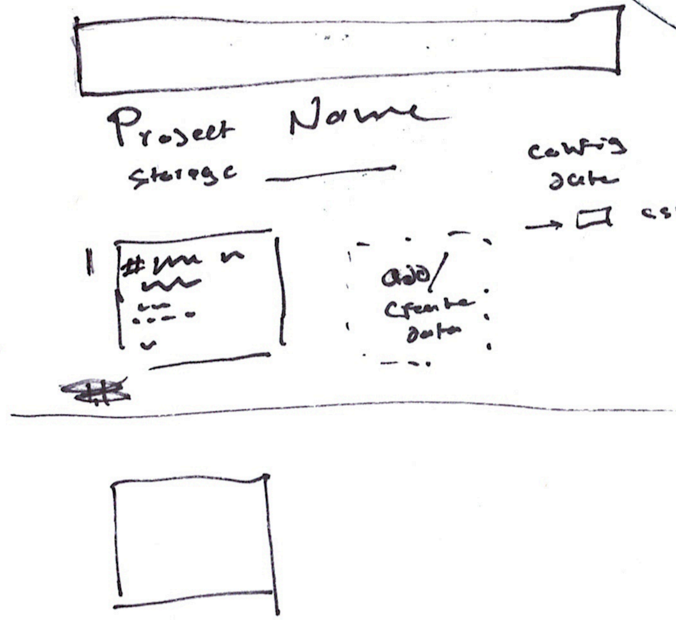
"1234567891c"

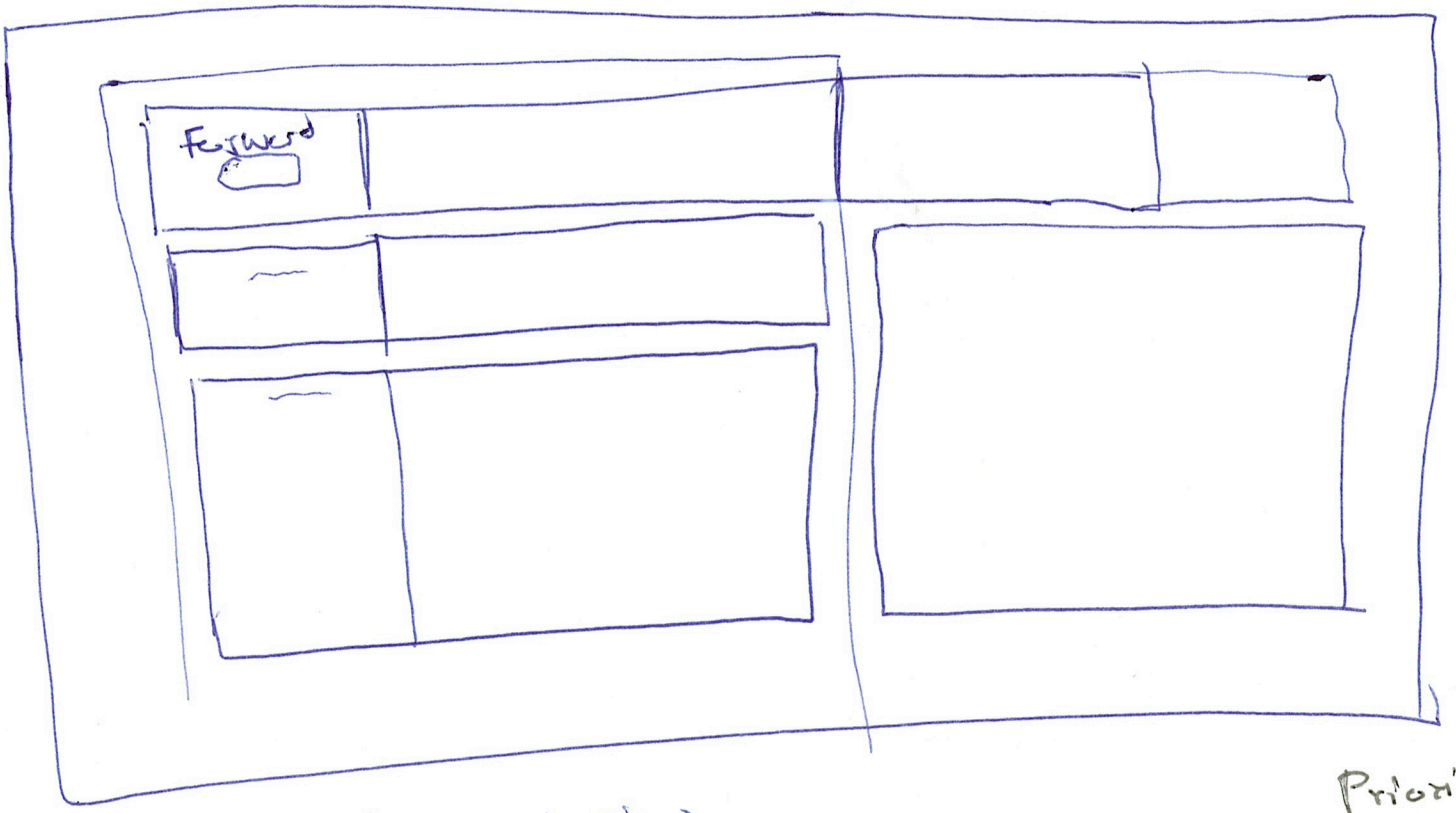
Title

Name

JS Abr

Date

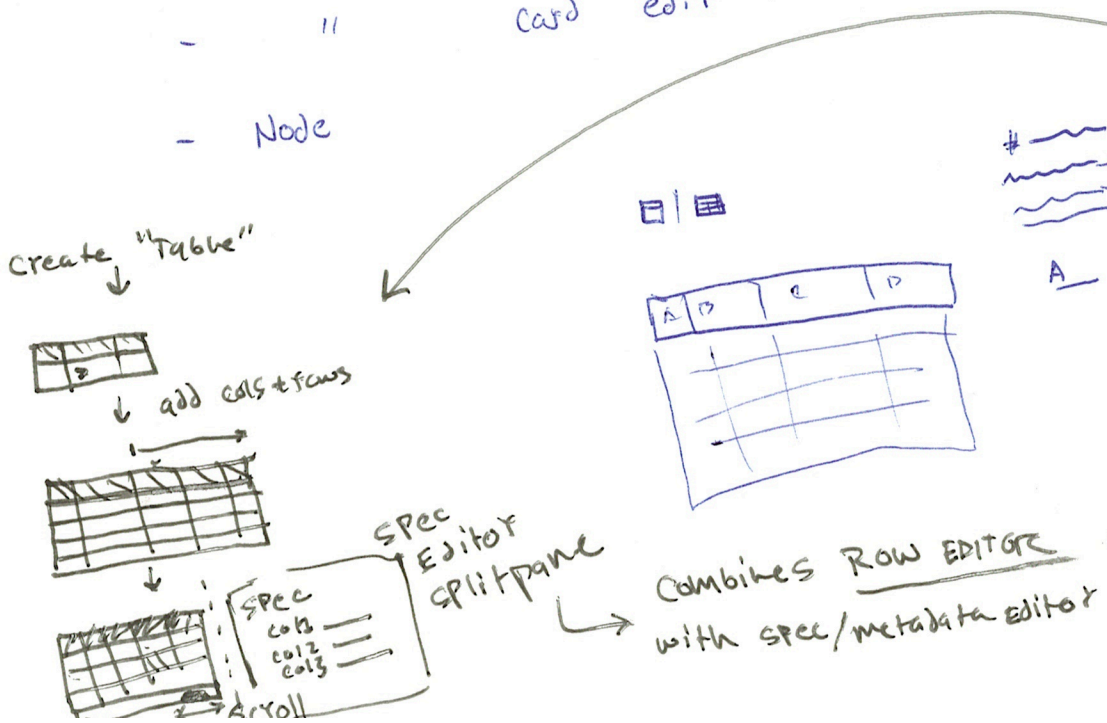




- CSS grid card view
- " card edit
- Node

Priorities

- Table Editor
- CSV import
- Data shape edit/infer @ table UI
- Reference + Aggregation for DB
- Data Flow + Exec of Frames
- Op 3; spec autosuggest



Notes

Things (schema)

operations

names (references/ vars)

instances (data)

(1) How does interact stencil CELL pick react component for output cell ?

schema Editor

File: AI 2008.pdf

Going to add some
@papers ...
(1) create collection →

(2) PAPERS

grid view · card view

id

name

file
- Path
- type
- added
- Filename

settings
- storage format
- seen.db
- path
- "file"
- #date + #name

Scraper
- Query (string)
- search Engine
- result num
- result type
- (image, paragraph)
- url

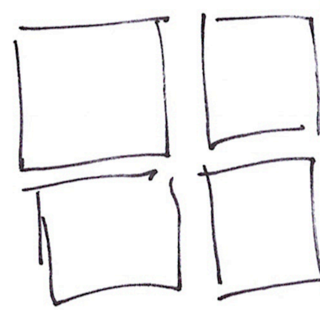
Card view

NAME

schema :

Propname : type

[+]



File Preview
Small

filename.csv
saved to attachments/ 1.filename.csv

details

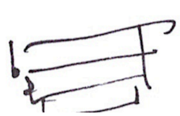
filename.csv

cell	2	34	5

Parse details

Column
- name: "isbn"
- validation:
- =
- type ISBN

DeBUGs



Schema
+
validations
+
storage rules

meta

separate from

Instances of data
scans.csv

instance

Help users organize & compose & generate data

10

num	string	json	code	"repo"	Book
1	"cat"	{ "a" : "b" }	''' $ $=7() '''	git://agcm...	ISBN for the author

users can contribute schemas & components & operations for arbitrary Data types

(1) users can contribute schemas & components & operations for arbitrary Data types

(2) join diff types in compositions — tuples — (i.e. rows)

(3) capture / process ~~that~~ of Document composition (i.e. joins)

schema yup

name : ISBN

validation : - number
- regex?

validation req : bool

users implicitly / transparently defines schema ~~when~~ from data viewer (scans.csv)

DROP_FILE :

(1) name
mimetype
othermeta
original src
saved | false | path

→ uploadscollection

(2) current note
add to attachments?
ref: collection
uploads
- id

(3) render Preview view component

Literal
vs
Statement "performs an action" (if loop)
vs
Expression
- produces a value
- "computed literal"

workflow

1. save file w/(name) in (path)
2. fetch metadata — save covers/local
3. verify
4. add price
5. add to (inventory) collection

scans.csv

Date	scan	format
	2dd1	

select SCAN as ISBN from (scans.csv)

→ now NAMED collection in doc scope

ISBN
list [2001232?, 0012m.....]

Fetch Meta
input "ISBN"
type "ISBN"
out "BookMeta"

How does workflow get exported?

- How do new runs of workflow effect existing collections?
- are collections also exported?

A | workflows have CONTEXT which links to FS, collections, Parent

Save local image

Joined collections table view at bottom →

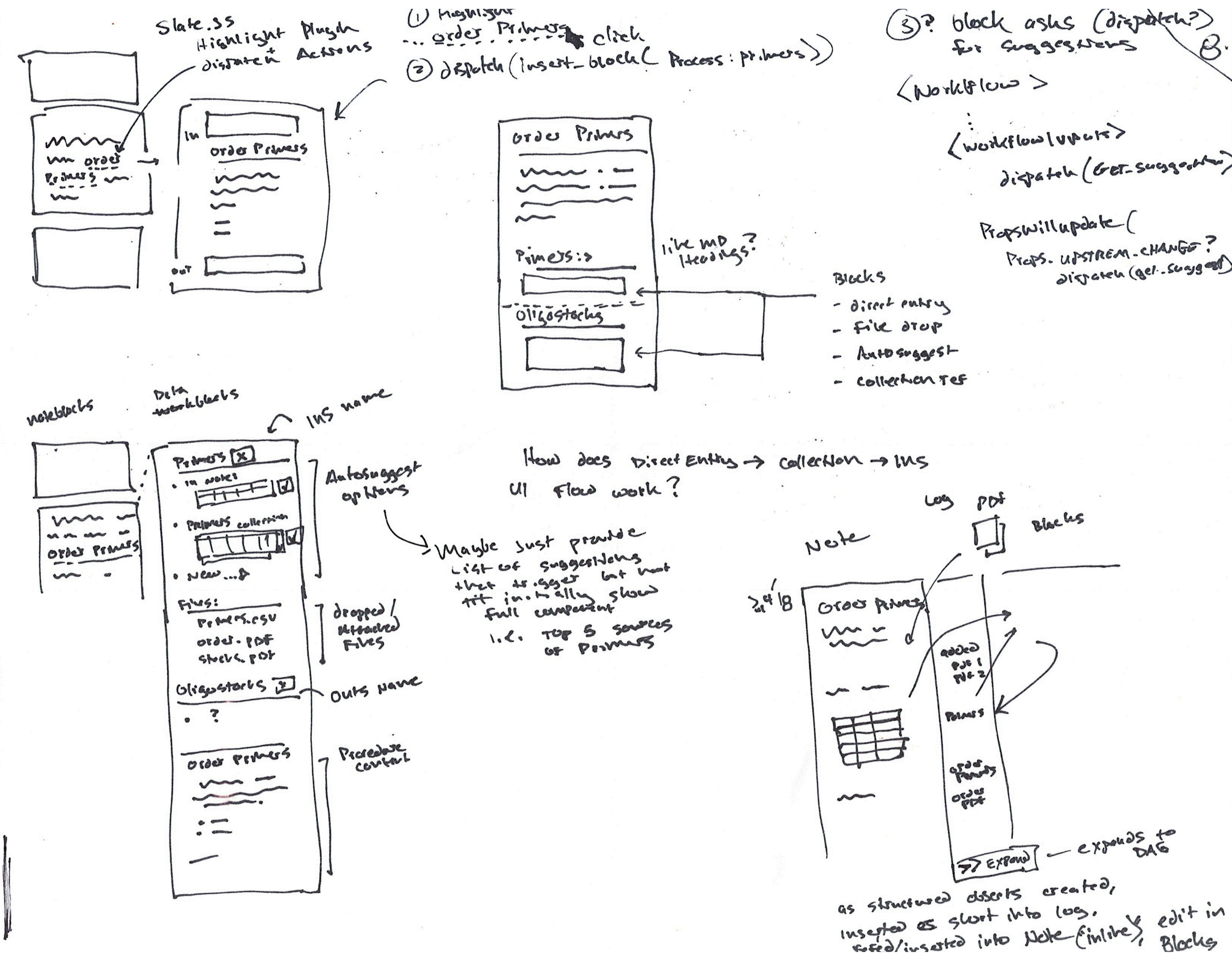
User "x"es columns not wanted in output

Books

ISBN	Title	(x) ISBN	ISBN 13	cover ...	(x) local cover
x1227:	cats	x1234	xxxxxxx	http://www	cats.jpg

OUT "Books"

ISBN	title	ISBN13	localCover	scanDate


Slate.js
Highlight Plugin + Actions
dispatch
(1) Highlight ... order Primers ... click
(2) dispatch(insert_block(Process: primers))
(3)? block asks (dispatch?) for suggestions
<Workflow>
<workflowInputs>
dispatch(Get_suggestions)
PropsWillUpdate(
Props.UPSTREAM_CHANGE?
dispatch(get_suggest)
In
Order Primers
Out
Order Primers
Primers:
Oligostocks
like MD Headings?
Blocks
- direct entry
- file drop
- Autosuggest
- collection ref
noteblocks
Data workblocks
order Primers
INS name
Primers
• in notes
• Primers collection
• New ...
Files:
Primers.csv
order.PDF
Stocks.pdf
Oligostocks
Order Primers
Autosuggest options
Maybe just provide list of suggestions that trigger but not initially show full component
i.e. top 5 sources of Primers
dropped/attached Files
Outs name
Procedure content
How does Direct Entry → collection → INS
UI Flow work?
Log PDF
Blocks
Note
Order Primers
Added PDF 1 PDF 2
Primers
Order Primers
Order PDF
Expand
— expands to DAG
as structured objects created, inserted as short into log, refed/inserted into Note (inline), edit in Blocks

ΔXT

order Primers

ins:
type: seq
name: sequences

or ins:
<type>: <name>

or ins:
<name>: <type>
"sequences": ["seq"]

outs:
type: ~~ssDNA primers~~
name: primer stocks

block?

op: "manual"

validation: { } block?

ops
content:
type "md"
content "~~~" block?

block?

<WorkBlock>

<OperationContainer>

~~<op>~~

load "empty" op-component

Trigger InputHelper

Provide <inputs>,
<controls>,
<validation>

---

or

create input placeholders components → dispatch suggest ins
update ins

create op component

create outs placeholder components → dispatch update outs

~~state~~ props

op: "@protocols/orderPrimers"

ins:
"sequences": [-ids] block?

outs:
"primer stocks": [-ids] block?

goals

(1) Implicitly capture NAMES + shape of I/O for set of operations

(2) Node can be reused easily w/ different I/O - facilitated by NAMEING convention

∴ data

(3) Computational deps are unambiguously referenced

Workable

Plan
- graph / timeline
↓
PlanBuilder

Block
Name?

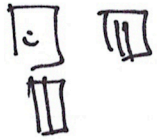
OP:
- Inputs — Terms → Semantic types / Data types
- Out? Schema?
- ExecutionCont ϕ?
- Note
- ID

Log
- added file
- updated block
- executed block output →

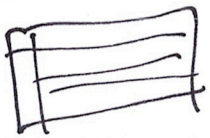
Collections View
Files

csv

Img



Inventors

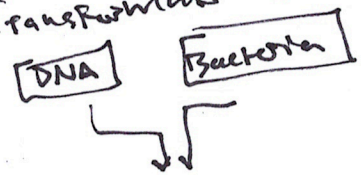


Artifact
- Queryable
- Provenance

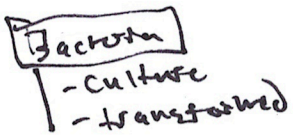
Terms / Semtypes : think MESH taxonomy?

Validation

TransRowMerson



Schema



Naming + Saving + loading Conventions

Scenarios

- write out
- create wireframes for two

Frontend

- Demo simplest scenario
- use traditional server
- Uppy for FS manager?
- try to cut out blob stuff from atlaskit Provider

  or

  implement w/o atlaskit

Backend

- Dat content provider in worker or service worker

Alice - scenarios

1 "living doc" for adding inventory

2 cousin bob automates (ss) fetching metadata based on alices doc

Emily - sell CD collection on ebay

imports csv from scanner
didact suggest workflow

James' plumbing

(1) develop RFQ Estimate

capture photos, notes

checklist

↓

PDF + report files

(2) Itemized work completed
Photo of finished work
time + ~~estimate~~
parts + cost

Mike - researcher

keeps track of primers ← papers
orders
use

Mike - sales

sentiment analysis
- uses didact to record experiments
- shares working process w/ team
- manual process → dashboard for management
- Figures out how to publish as private automatic service + dashboard

Alice

Runs small used bookstore

uses industry specific book ~~sales~~ POS to manage inventory shipping & integrate w/ ~~pos~~ stripe

uses barcode scanner to add used book purchases to ~~bookeems~~ POS

10% of these books are not recognized & require manual entry.

scenario 1

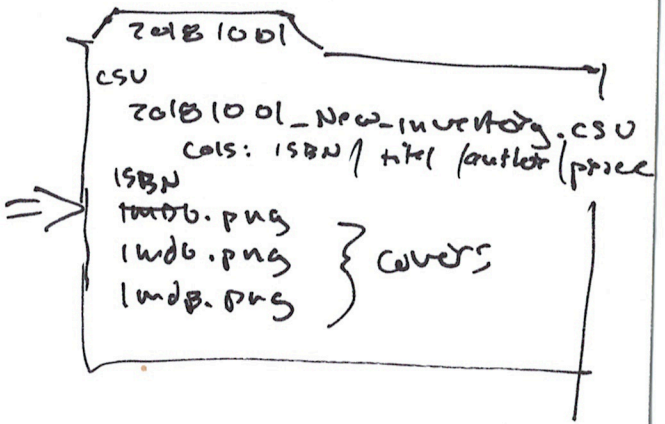
Alice uses didact to capture notes as she develops a workflow for ~~adding~~ manually adding books to bookems pos

~~outcomes~~ value:

- Process captured in "living document" single source of truth
- didact encourages use as ~~of~~ Primary source of truth by keeping output of process well-organized — Preventing Context-rotting Copies

Outcome

- ID data fields needed to import to Book POS in Didact
- Describe & set up naming & saving scheme for organizing results of workflow



scenario 2

Alices notebook shows her that it

takes a CSV as input, ~~with columns~~ with columns <scan_out>

and produces an object: Book

with properties

~~tmdb~~ <ISBN> <title> <cover> <author> <price>

and saves the result to a ~~file~~ CSV file in a ~~folder~~ new folder named <Date>

~~and saves~~ along with attached <cover> images

click something ...

Jan 25 2020

Blocks

- Contain Notes

or point to
- define "specs" of desired data elements

- store "Frames" which are sets of IDs to CTs at particular versions - these are the "state" of desired data at the timepoint of the block

Or, "Frames" + "Frame Tree" is parallel structure that references each Block with CT state
- Serialization cleaner separation B/w Design + Execution

- store Oplog (Not CT?) of Operations that create / Edit / Join

A) the data contained in other blocks?

or

B) data in Causal Tree?

or

C) maps ~~log of~~ user's notebook operations on data to ranges each produced in Causal Tree
- CT captures minimal semantics (CRUD); Block captures specifics + user intent

→ list of operations = DAG of Blocks
i.e. Block can contain Blocks
Block Tree is really a special CT?

Causal Tree(s)

Captures values & changes to data elements
- update
- join
- delete

or CRUD?

~~Frames~~ 3

Frames

~~View~~ / cursors into CTs at particular Time & Place

~~special Frame shows~~
- ~~Frame~~ View / collection
View component that shows data w/ highlights indicating correspondance / dependence to place in BlockTree
contains / points to SPEC

→ MaterializedFrame
returns state of Data for given Frame

Specs

Schema / type / definition of shape of dataset

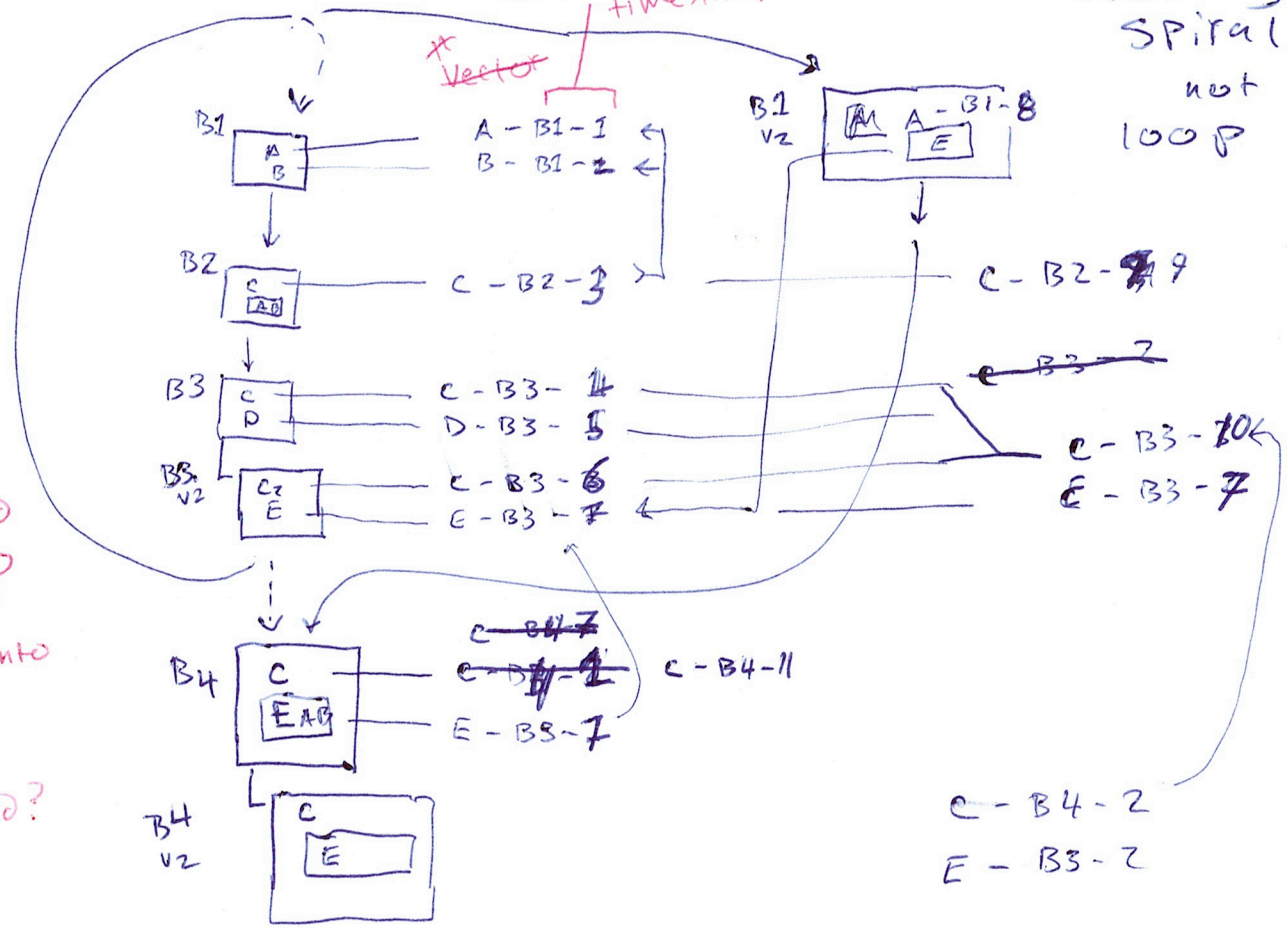
Jan 24 2020

thing id

Notebook id - ~~block id~~ - block id - clock (lamport(ish) timestamp

~~* vector~~

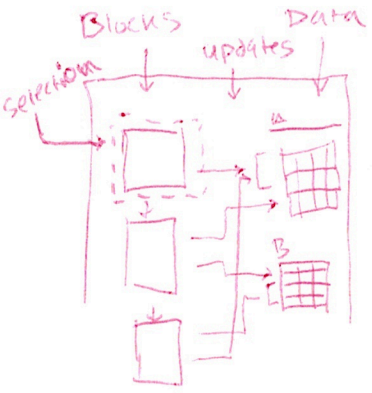
Causality Spiral not loop



What happens if Downstream Data is inserted Back upstream? B3's "E" inserted into B1's "A".

Can it be modeled? How to model?

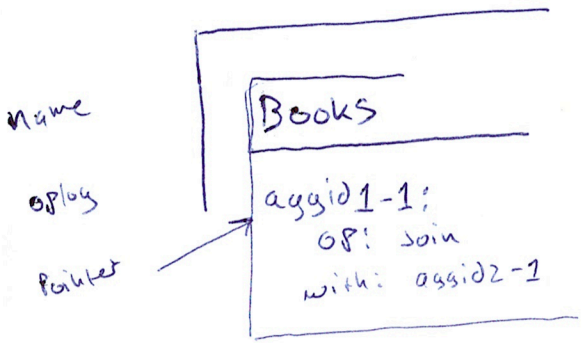
use case is:



Data column shows latest values of data, which user may wish to integrate into Earlier Block

Jan 24 2020

Frame



Frames have SET of ops = oplog

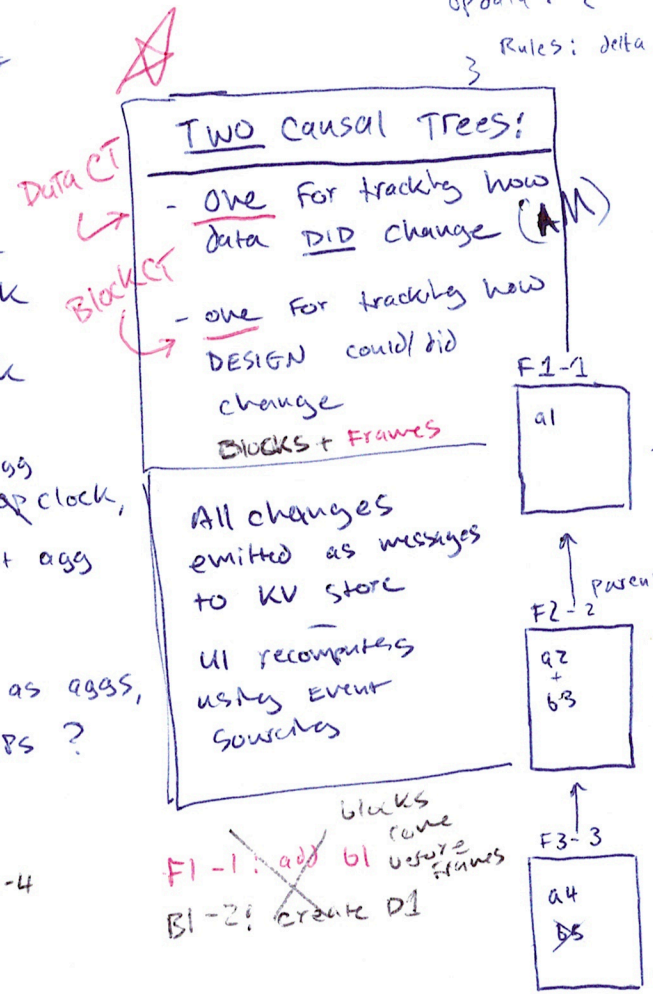
ops take aggs
aggs have a clock
Frames have a clock
~~Ops~~ aggs have a clock

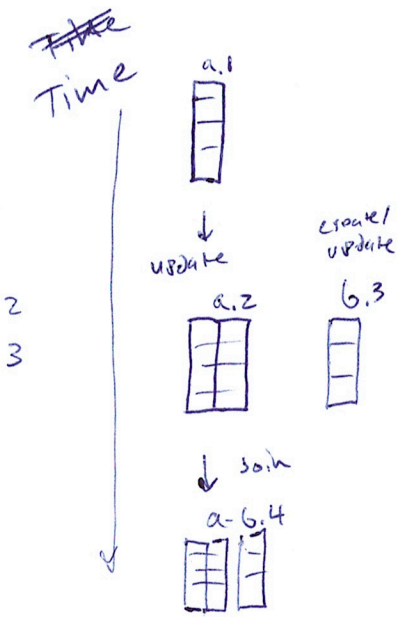
IF Frame clock < ~~op~~ aggs clock, op hasn't run on that agg

Frames are the same as aggs, Except they have OPs ?

B1-1: Create ~~Frame~~ F1-2
F1-3: ~~Free~~ create data D1-4
F2-5: create ~~data D2-6~~ data D2-6
B2-7: Join D1-4 + D2-5
F1-8: update name, meta

Parent: agg2-2
id : agg1-3
op : update
Opdata : {
Rules: delta
}

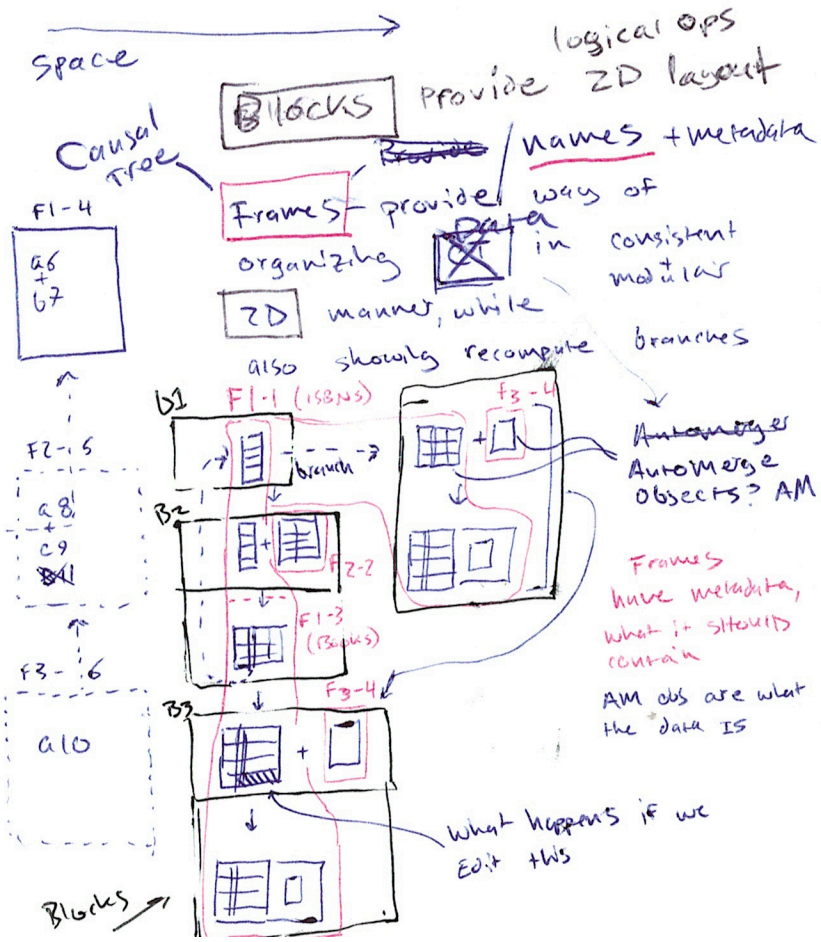




Think about how adding a column later causes upstream Frames to recompute

(1)

Technique = Frame



Goals

26 - Oct - 2019

- Review desired UI
- Identify Focus for Pairing
  - Block + Frame + Spec CRUD
- sketch wireframes
- "Pair-allel" development sprint
  - using react-mobx- boilerplate

- Discuss software Project Management best Practices
- sketch out "Modules" of MVP
- get advice on how to parallelize Dev
  - Maybe write "stories"

Start some magic happen! 2

1.input: upload csv

Choose File book_barcodes_short.csv

steps

1. save file at *inventorynotebook/<date>/<date>_bookscans.csv*
2. store in notebook as **book_scans** collection

success! saved book_barcodes_short.csv to disk as inventorynotebook/2018-11-20/2018-11-20_bookscans.csv

1.output: book_scans

2.input: book_scans

	datetime	scan	format
1	2017-04-20T11:32:0...	055338256X	UPC-A
2	2017-04-20T11:32:3...	0-449-23949-7	UPC-A
3	2017-04-20T11:34:0...	595371086	UPC-A

steps

1. select *scans* or *ISBNs* column from **book_scans**
2. store in notebook as **isbns** collection

2.output: isbns

055338256X 0-449-23949-7 595371086

3.input: isbns

steps

1. run *Fetch metadata from OpenLibrary.org* workflow with **isbns** as input
2. store in notebook as **books** collection

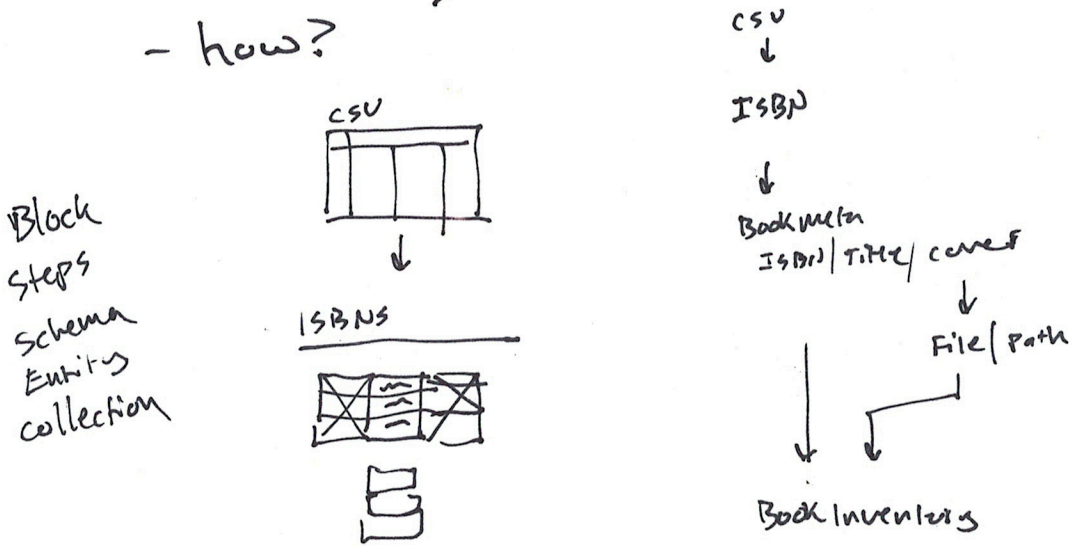
3.output: books

```
055338256X0-449-23949-7595371086
```

Draw state including schema + collections

add step joining bookmeta w/ files (covers)

- how?



PRINCIPLES

(can)

things have a name

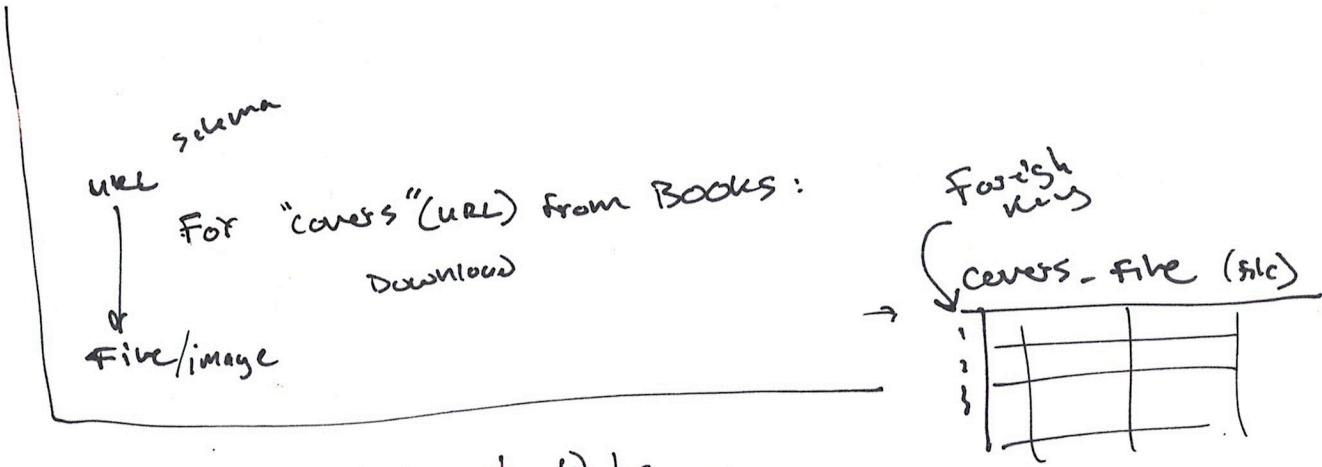
USE names to scaffold goals

OPERATIONS do things to THINGS

~~Names~~ things have schema of names

shape of thing suggests possible operations

Sequence of ~~names~~ (schemas) imply operations (scaffold)



The closest name should be tried first

storage:
default
validate:
etc

① IDB

↔ memstore

Entities

-ID: {a b c}

thing defaults

view

validation

name

collection → thing

mesa

schema ← vals

colview -collection

checks

get rows

② Class thing

new thing()

collection?

→ generate

~~add~~

update ( )

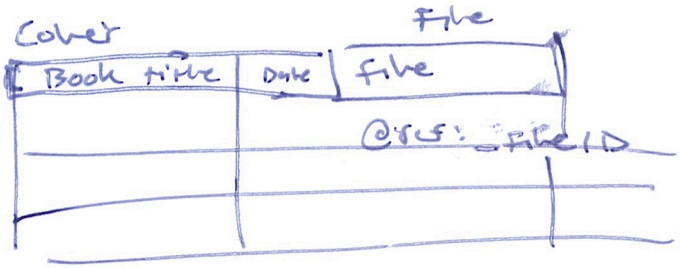
in do it

... then call / autorun

collection validate

③ JSON

↓



show table

1. mock static vals
2. w/ add new
3. hooked up to dropstore

⑤ Schema Editor

schema → fields

→ edit fields

→ defaults

→ validation

④ Pitch IDEA

let me tell you 3 stories

1. Bookshop
2.
3. search for subpath

11 Super Spreadsheet

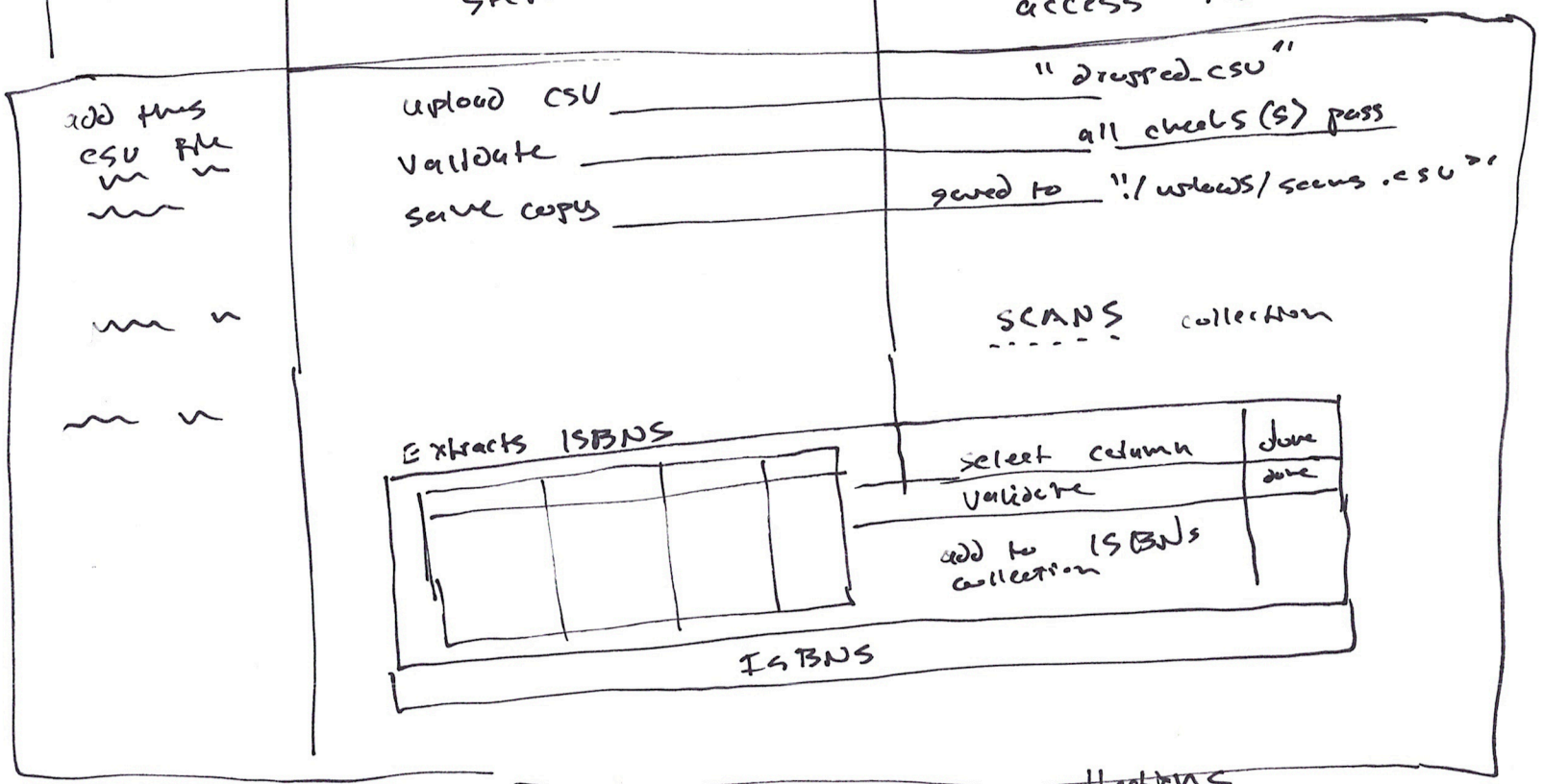
notes explain

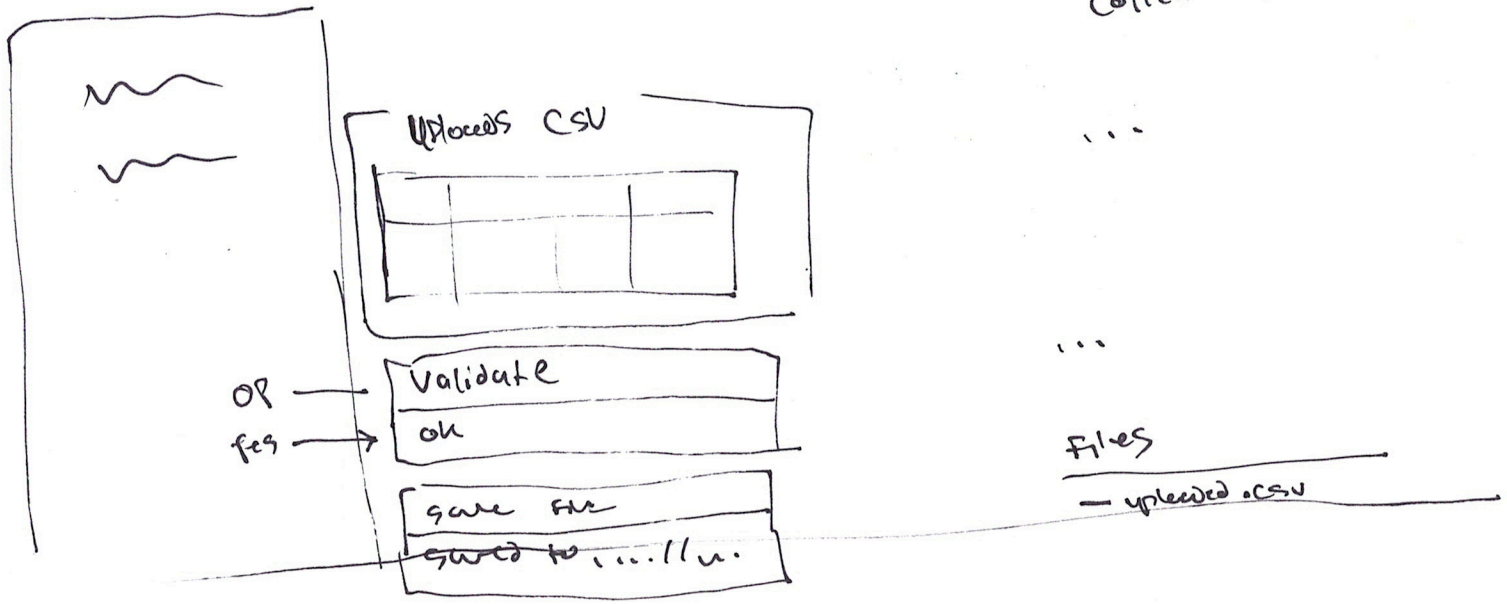
operate on data steps

changes workspace

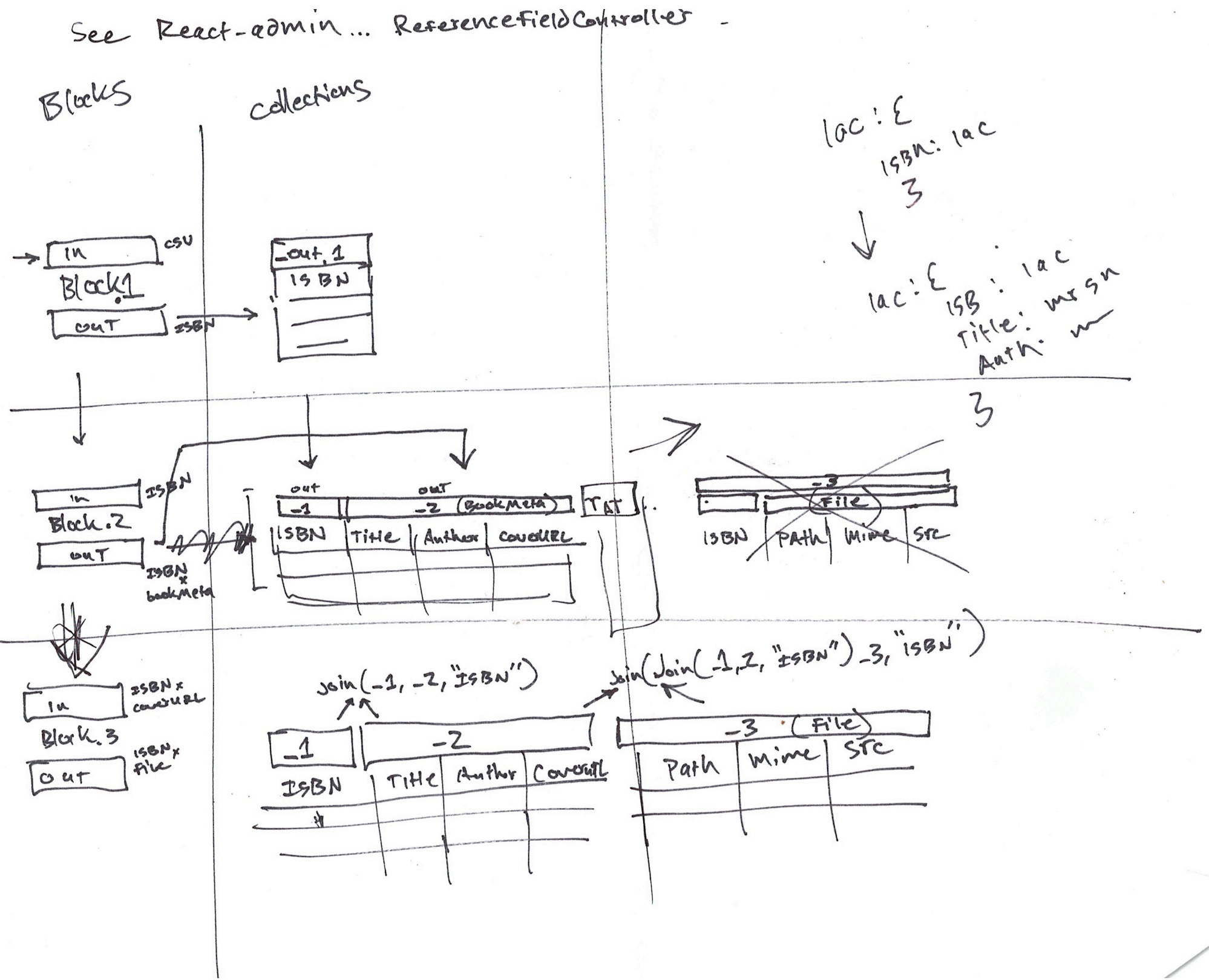
name store collect access

log output results





See React-admin... ReferenceFieldController -



DESIGN

Simulate →

← Edit

REPEAT

LOG

JSON-schema: shape & validation of json data

JSON-LD: unambiguous / semantic identifiers for data

schema + meta ← collection

rows

columns

derived state:

```
collection.<name>.validation = ($) =>
  cols.map (col => row => col.row.map =>
    col.row.valid = collection.name.validate.
```

— or —

state.schemas. collection/meta | file | book | schema

state.collections. book

```
name
schema
meta
rows [ // validation
```

1 CSV

add ISBNs via CSV file

ISBNs # Validate ISBNs

Fetch Books

DO it

Fetch covers

2

Book Inventory

CSV

data

saved to <path> name.csv

ISBNs

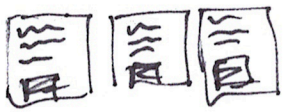
waiting....

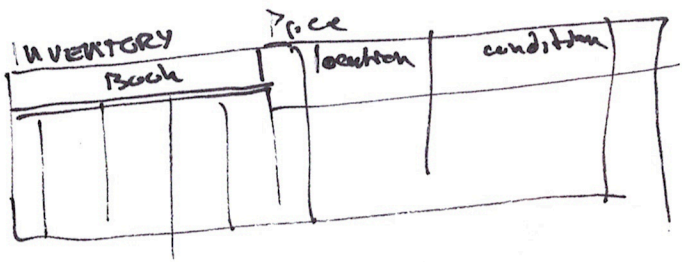
Found 27/27 ISBNs

0001 001231 ...

0012 001231

Fetch Books





Vampire Weekend Unbelievers

db:
cuid: { db }

Collection
- id
- name
- desc
- tags
- example
- validation
- schema
- data

Schema
- jsonSchema

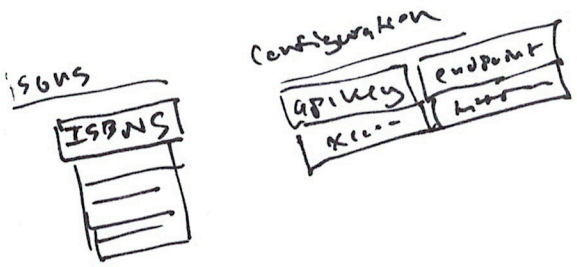
Blocks
- consumes
- Produces
- inputs → collections
- outputs →
- collections

list of Blocks

list of collections

Drop file
csv
image

JSON-PATCH



Primers

IS_A:
data. DNA. seq

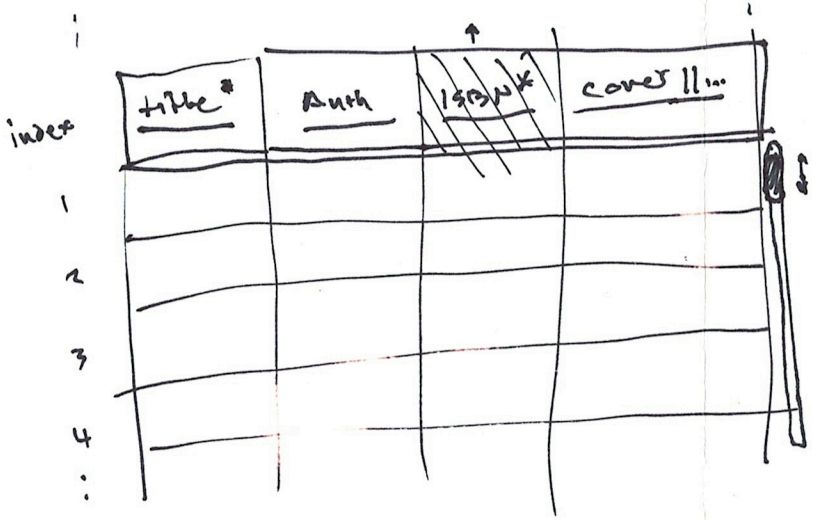
oligos

IS_a?
~~physical. DNA~~
physical. oligo. OF:
- DNA
- rna
- ssRNA
- ssDNA

Books

Configuration for Properties schema →

ISBN
(Primary key)
type string
Example value: x1234
validation rules:
~~mask~~ length: 9 or length: 13
must | should



~~Physical~~ Books

Books

Save

Configuration for collection schema →

Description....

singular: Book
Plural: Books
IS_A: <taxonomy>

~~Book~~

Primary Key: ISBN

Show in listviews

Elements:
- title
- Auth
- ISBN
- cover
  add

Storage rules:
- format json | csv
- schema? ☑
- filename: _ location

Upload scans csv ... uploaded

validate CSV ... ok

Save File
- File saved to "./../pics/<id>.csv"

Store in BookScans collection ... ok

collections / outputs

validated

File
scans.csv

Book_scans

Presentation

Select ISBNs ... ok

1 FROM Book_SCANS
Select column ("scans" OR "isbn")

2 validate
is string

3 store as ISBNS

click
highlights details

ISBNs
0511781 1 2
449-1781-2
77121 3

support direct copy-paste in table view

list table / card

Fetch Book metadata

status (0/3)

For Each ISBN:
Run API.OPENLIBRARY.API.Fetch

store as Books

Books (waiting...)

file > img

BookMeta
Title
Auth
Cover
Pub

API

From Schema

_4:

OP: BookFetch
opconfig:
opin: "ISBNs"
opout: "Books"

collections!

"Books" (name)

Schema: ISBN Title Auth
source:
values: [ ]

ISBN	Title	auth	Desc	URL

Autogen from # save covers

_5:

OP: selection

opconfig:
or: [
colname is "cel val"
and [colname nlp "covers"
celltype validates URL
]
]

OPIN: "Books"

OPOUT: "covers"

collections:

Indented O/C selection

"covers"

Schema:
"cover": URL

values: [{ }]

Books.id	cover (url)
1	
2	
3	

or Books.url

Replaces

"covers"

Schema:
"cover": URL
"file": File

values:

books id	cover (url)	file (file)		
#	cover	folder	name	type
1	http://			
2	h			
3				

_6:

OP: Download_file

opconfig:
folder: `now().isoDate`
name: `@ISBN - @title`
auto: True

opin:
blockid:
collection_path:

inferred?

OPIN:
URL: "covers"

collections!

OPOUT: {
...book,
} file

? or

opout:
covers: file

OP:
name: "Download"
related_names: ["save", "fetch",]
in: URL
out: File

Schemas:
URL:
name: URL
related: address, http
validator:
string: s
s → isURL(s)

File:
... : File

collection
schema
rowOrder: <field>

denormalized rows →

colOrder
data [
{f1: 1, id: 1, name: "first"}
]

collection
schema:
data: 2darray
or
data: {
rowId → 1: [ ]
2: [ ]
3: [ ]
}

state — overwind

schemas {
id: { obs }
id: { obs }
}

collections {
etc
}

~~get~~: (collection) =>
dataMapper()

Just for
UI action/effect:
set:

module

dataMapper module

Autojoin — hmmm

two collections have
- schema element in common (column)
- 3 identical values at that col

Books:

name	isbn	cover_url
	XX1	
	aab	

Join

coverFiles:

isbn	filePath
XX1	some/path.sps
aab	path2.sps

① replace Books by direct insert value of Join

or

② Provide "Materialized" View of Join

or... Both?

collection

name: "books"
schema:
fields:
"name",
"isbn",
"url"
PK: isbn

data: {

}

schema: {
- self: {
name
isbn
url
},
ref: {
schema: "file"
obs: "isbn"
}
}

data { "column sets"

}

in "Books"

↓

~~opt~~

_2: selection
col name is "cov_url"
contains "covers" or
type and → validate URL

OUT
~~Type: URL~~
~~name: covers~~
~~values: [ ]~~

"covers":
Type: URL
values: [ ]

Things

NAME
storage
validation
members
schema

Subname
schema
storage
validation
members
etc

_3!

OP: Download_File

opconfig:
folder Now()
name @ISBN-@title
overwrite
autorun

OP IN:
Selection (block OPId)
or
"covers"

Books!

title	Auth	cover_URL

opout:
"covers":

URL →	File || folder	Filename	type
@cover.URL	~~Folder~~ JS: now()	@ISBN	—
http://www...	2018-10-11_	2A11122	JPG.
https://...	2018-10-11_	2200113	JPEG

OR?

← SELECTION

Cover_URL
contains "cover"
matches URL validate

opout!
"covers":
schema [
_id: @covers.URL
file

foreign key

reuse of collection name implies update

of
opout:
"Books":
Book.
file

implies JOIN

save covers
to ...

synonym for fetch
implies op (fetch/save)
implies thing
(~~file~~) URL

AUTOGEN

~~Covers: name path filetype~~

↓

Fetch (URL)

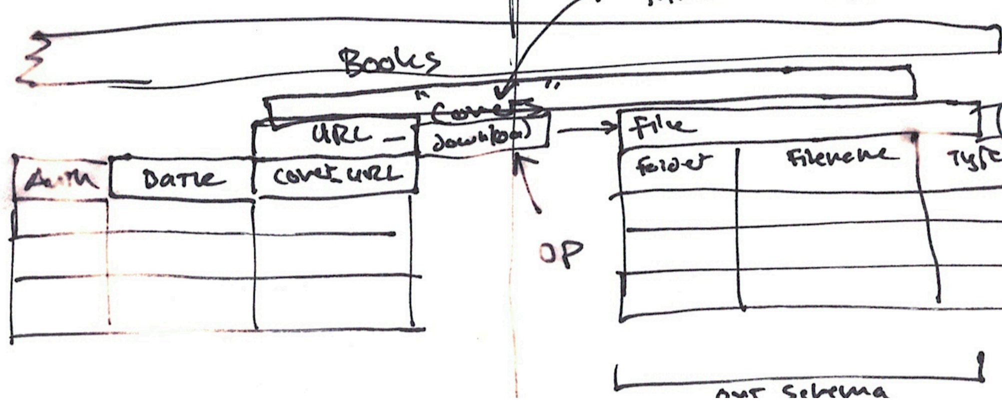
config: -name
~~Folder~~ -folder

↓

File
name: <title>
path
type

collection extension →

inner collection "covers"



Running / Finished Current step: 3 | Start | Stop | Auto |

New schema
Edit schema

placeholder from schema

UX Goal?
Make spreadsheet front + center
running workflows
shows spreadsheet automation

UPDATE/Invoices

1. Drop csv here

2. Screen scrapes

3.

Books from from screen

Active step control
  status bar or popover
  control button

log: step action name: result
  Brief
  detailed

Block details
  show log
  show output brief list

Output / collection item detail card

Block in ? out: show schema
  placeholder

Features

Schema

Editor
- Property name - prop tags
- Prop type
- validation
- default
- foreign key

Collections
- name
- List view
- card view
- table view

+ Placeholder Editor

2016 json-schema json-ld primers

```
// primer-json-schema.js

// FISHCO1LBC
// TCAACYAATCAYAAAGATATYGGCAC
//
// FISHCO1HBC
// ACTTCYGGGTGRCCRAARAATCA

{
  "id": 1,
  "name": "FISHCO1LBC",
  "description": "Fish CO1 barcoding primer",
  "sequence": "TCAACYAATCAYAAAGATATYGGCAC",
  "orientation": [
    "forward",
    "reverse"
  ]
}

{
    "type": "object",
    "properties": {
        "id": {
            "type": "string",
            "description": "unique id",
            "minLength": 0,
            "maxLength": 255
        },
        "name": {
            "type": "string",
            "description": "short name",
            "minLength": 4,
            "maxLength": 32
        },
        "description": {
            "type": "string"
        },
        "sequence": {
            "type": "string",
            "description": "sequence"
        },
        "orientation": {
            "type": "string",
            "default": "forward",
            "enum": [
                [
                    "forward",
                    "reverse"
                ]
            ]
        }
    },
    "required": [
        "id",
        "name",
        "sequence",
        "orientation"
    ]
}


{
  "$id": "http://example.com/example.json",
  "type": "object",
  "definitions": {},
  "$schema": "http://json-schema.org/draft-06/schema#",
  "properties": {
    "id": {
      "$id": "/properties/id",
      "type": "integer",
      "title": "id",
      "description": "id string",
      "default": ""
    },
    "name": {
      "$id": "/properties/name",
      "type": "string",
      "title": "The Name Schema",
      "description": "An explanation about the purpose of this instance.",
      "default": "",
      "examples": [
        "FISHCO1LBC"
      ]
    },
    "description": {
      "$id": "/properties/description",
      "type": "string",
      "title": "The Description Schema",
      "description": "An explanation about the purpose of this instance.",
      "default": "",
      "examples": [
        "Fish CO1 barcoding primer"
      ]
    },
    "sequence": {
      "$id": "/properties/sequence",
      "type": "string",
      "title": "The Sequence Schema",
      "description": "An explanation about the purpose of this instance.",
      "default": "",
      "examples": [
        "TCAACYAATCAYAAAGATATYGGCAC"
      ]
    },
    "orientation": {
      "$id": "/properties/orientation",
      "type": "array",
      "uniqueItems": false,
      "items": {
        "$id": "/properties/orientation/items",
        "type": "string",
        "title": "orientation",
        "description": "forward or reverse strand",
        "default": "forward",
        "examples": [
          "forward"
        ],
        "enum": [
          "forward, reverse"
        ]
      }
    }
  },
  "required": [
    "id",
    "name",
    "sequence"
  ]
}

const mobx = require("mobx")
const { types } = require('mobx-state-tree');
const jsonSchemaToTypes = require('jsonschema-to-mobx-state-tree')(types);

const eventSchema = {
  type: "object",
  title: "Event",
  properties: {
    title: {
      type: "string"
    },
    public: {
      type: "boolean",
      default: false
    },
    time: {
      type: "object",
      properties: {
        start: {
          type: "string",
          format: "datetime"
        },
        end: {
          type: "string",
          format: "datetime"
        }
      },
      required: ["start"]
    }
  },
  required: ["title", "public"]
};

const primer = {
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "description": "unique id",
    },
    "name": {
      "type": "string",
      "description": "short name",
      "minLength": 4,
      "maxLength": 32
    },
    "description": {
      "type": "string"
    },
    "sequence": {
      "type": "string",
      "description": "sequence"
    },
    "orientation": {
      "type": "string",
      "default": "forward",
      "enum": [
        [
          "forward",
          "reverse"
        ]
      ]
    }
  },
  "required": [
    "id",
    "name",
    "sequence",
    "orientation"
  ]
}

const eventModel = jsonSchemaToTypes(primer);
console.log(eventModel);

// https://npm.runkit.com/jsonschema-to-mobx-state-tree
```

json-ld_simple_protocol_flow.json

```
<script type="application/ld+json">
[{
  "@context":
  {
    "schema": "http://schema.org",
    "name": {
      "@id": "http://schema.org/name",
      "@type": "@id"
    },
    "input": {
      "@id": "http://schema.org/url",
      "@type": "@id"
    },
    "content": {
      "@id": "http://schema.org/code",
      "@type": "@id"
    }
  },
  "name": "calculate transformation efficiency",
  "input": "http://fab.bio/100ideas/pcr/1/1sjf2h2",
  "content": "content",
  "dateCreated": "2013-02-14T13:15:03-08:00"
}]
</script>


/*######################*/
```

```
<script type="application/ld+json">
{
   "@context": "http://schema.org",
   "@type": "Recipe",
   "name": "calculate transformation efficiency",
   "url": "https://www.example-petstore.com/",
   "dateCreated": "2013-02-14T13:15:03-06:00"
}
</script>

/*######################*/

<script type="application/ld+json">
[{
  "@context":
  {
    "name": {
      "@id": "http://schema.org/name",
      "@type": "@id"
    },
   "operation": "http://schema.org/Recipe",
   "input": {
      "@id": "http://schema.org/url",
      "@type": "@id"
    },
    "content": {
      "@id": "http://schema.org/code",
      "@type": "@id"
    }
  },
  "@type": "operation",
  "name": "calculate transformation efficiency",
  "input": "http://fab.bio/10@ideas/pcr/1/1sjf2h2",
  "content": "content",
  "dateCreated": "2013-02-14T13:15:03-08:00"
}]
</script>


/*######################*/


{
  "@context": {
    "name": "http://fab.bio/schema/0.1/name",
    "homepage": {
      "@id": "http://xmlns.com/foaf/0.1/workplaceHomepage",
      "@type": "@id"
    },
    "Person": "http://xmlns.com/foaf/0.1/Person"
  },
  "@id": "http://me.example.com",
  "@type": "Person",
  "name": "John Smith",
  "homepage": "http://www.example.com/"
}

/*######################*/
```

```
note:
  timestamp: 2012-10-03 11:13:12.10 -7
  intent: "test epibio quickextract kit with fish tissue"
  project: "genelaser_test1"
```

Testing out genomic purification using EpiBio QuickExtract kit. Trying 6 samples to start:

```
samples[epibiotest1]:
  lc: "left cheek, rubbed 10 sec"
  ham: "hamachi sample from Moshi Moshi takeout order"
  mak: "maki sample from Moshi Moshi takeout order"
  neg: "negative control, nothing swabbing"
  col: "positive control, 100 mg of frozen tuna tissue from sample:2122"
  tab: "table top, 5cm streak rubbed 30 sec"
  _file: "samples/2012-10-03_epibiotest1.csv"
```

First protocol! woot. This content is still part of the first block.

Above see sample data stored inline. Should this be the canonical place for it? Or should it reside in a more central json file and be included by reference? Which is less likely for users to screw up...?

`[img:"data/2012-10-03_epibiotest1_sushi_samples.jpg"]`


```
op[1]:
  timestamp: 2012-10-03 11:22:01.33 -7
  protocol: Epibio-quickextract-1@ceba48e
  samples: epibiotest1
  qc: op[2]
```

A new note block! This one is associated with the `Epibio-quickextract-1` operation block above. This is where we would write any notes about how the protocol went.

```
op[2]:
  timestamp: 2012-10-03 11:40:12.53 -7
  protocol: nanodrop_OD-260-280
  in: op[1]
  out: "data/qc/2012-10-03.csv"
```

Another Note (or is it operation?).

`[table:"data/qc/2012-10-03.csv"]` ← viewer renders inline

Quickextract seems to work; deviation in gDNA concentration might be a problem. Their whitepaper suggests a range of 7 - 25 ng / uL when it's working properly. `[ref:2012Weight]`.

Also see `[doi:10.1007/978-1-61779-591-6_6]`.

Notebook files have two kinds of top-level metadata blocks: <operation> and <note>. <note>s are associated with the closest preceding <operation>. <note>s can contain data tables, sample tables, protocol definitions, etc., represented inline or by reference via json or yaml blocks.

<note>s begin with `timestamp: <isodate>` metadata block. Additional metadata is inherited from the project.json file

render samples as table view;
TODO how is it automagically stored in samples/2012-10-03_genelaser-test1.csv?
organize content by experiment? i.e.

```
experiments/
├── 2012-10-03_genelaser-test1/
├──── index.md
├──── samples.csv
├──── images/
├──── data/
├──── notes/
├── 2012-10-06_another-test1/
├──── index.md
├──── data/
├──── notes/
index.md
private/
project.json
protocols/
plugins/
references/
├── references.bib
├── 2010 weight DNA Barcoding Fishes.pdf
```

explicit in/out tags (no typerange tho) in <op> definition

```
op:
  _id: 1
  timestamp: 2012-10-03 11:22:01.33 -7
  protocol: Epibio-quickextract-1@ceba48e
  in: cells.tissue # <-- implicit
  out: DNA.genomic # <-- implicit
  samples: [epibiotest1, epibiotest1_rep2]
  qc: op[2]
```


materials

type	description	

operation

description

notes

samples

id#	short_name	

2018-03-13 Antha, Transcriptic, AutoLims, & other computable workflow tools/apps/standards

compare **"projects"**, **"types,"** **"operations,"** **"protocols,"** **"runs,"** **"units"**, **"data"** in Antha, autoprotocol, Autodesk Wet Lab Accelerator, aquarium, Raik's rotmic lims.

TODO

- respond to https://github.com/scottbecker/autolims/issues/2#issuecomment-372840007
- summarize architecture; org notes on autoprotocol / antha / autolims / aquarium
  - emphasis on extensible & shareable Types and "op" data standard that wraps arbitrary documentation / protocols / workflows with metadata defining conceptual "function signature" based on said Types
  - goal is to enable modular, composable human-centric workflow design and sharing w/ data
  - not strictly computable (i.e. execution will require human in the loop initially)
  - iff traction, then deeper integration into / on top of existing computable workflow languages / standards.s
- respond again to scottbecker
- "walking skeleton" of typseschema + forms + notes
- connect w/ Brian Naughton / BooleanBiotech http://www.hexagonbio.com

---

jupyter for protocols & labwork

- https://github.com/dacarlin/robot-bagel
- http://nbviewer.jupyter.org/github/BjornFJohansson/ypk-xylose-pathways/blob/master/index.ipynb
- http://autoprotocol-python.readthedocs.io/en/latest/protocol.html#protocol-as-dict
- https://github.com/autoprotocol/autoprotocol-python/blob/master/autoprotocol/protocol.py

---

Autoprotocol & Transcriptic

autoprotocol: https://github.com/autoprotocol/autoprotocol-python/blob/master/autoprotocol/protocol.py

transcriptic: https://github.com/transcriptic/transcriptic/blob/master/transcriptic/english.py#L43

thoughts: autoprotocol.org/specification/#protocols identifies key parameters for 19 autoprotocol `ops` or `instructions`. Didact protocol metadata should be able to model these. Port them if licensing allows. Actual code for `ops` is defined in transcriptic/english.py.

autoprotocol "in the wild":

- "simpler" example of autoprotocol https://github.com/scottbecker/transcriptic101/blob/master/Transcriptic 101.ipynb
- great example of an autoprotocol metaprotocol http://blog.booleanbiotech.com/puc19_pcr_amplification.html

core objects jupyter module (docs; code)

- **project**: manages runs
- **run**: manage Instructions, Datasets, "monitoring data"
- **container**: container Type from the Transcriptic LIMS & aliquots present in the container
- **aliquots**: DataFrame of aliquots in the container, along with aliquot name, volume, and properties
- **dataset / data**: DataFrame of well-indexed data values. "*Note that associated metadata is found in attributes dictionary*" ??
- **job_tree**: "A Job Tree visualizes the instructions of a protocol in a hierarchical structure based on container dependency to help human readers with manual execution. Its construction utilizes the algorithm below, as well as the Node object class (to store relational information)"

key concepts:

refs
the set of containers that will be used in the protocol

instructions
Instructions are the unit operations of a protocol; the list of instructions to be performed

runs
A run is a specific instance of the execution of a given protocol. A run is composed of a sequence of instructions, which will be executed in order (with some parallelization where possible).

manifest.json
"A `manifest.json` file contains metadata about protocols required when uploading a package to Transcriptic. A package can contain many protocols but for our example it will contain just one. The "inputs" stanza defines expected parameter types which translate into the proper UI elements for that type when you upload the package to Transcriptic."
...
"It can be thought of as a markdown language you can use to create a graphical user interface for your protocols so that they can be parameterized and launched easily through the web app."

input types (manifest.json): https://developers.transcriptic.com/v1.0/docs/input-types
"An input refers to an editable field on a protocol within the protocol browser, they will allow you to replace the parameters you hard-coded into the preview once the protocol is uploaded as a package to the web app."

example:

```
# source
# blog.booleanbiotech.com/puc19_pcr_amplification.html

# (Python 3 setup cell omitted)
# https://developers.transcriptic.com/docs/how-to-write-a-new-protocol
# https://secure.transcriptic.com/_commercial/resources?q=water

import json
import autoprotocol
from autoprotocol.protocol import Protocol
p = Protocol()

# 3 cols: 0=template+primers+mastermix, 1=primers+mastermix, 2=water
# 3 rows: A, B, C repeated
experiment_name = "puc19_m13_v1"
```

```

inv = {}
inv['SensiFAST SYBR No-ROX'] = "rs17knkh7526ha"
inv['water'] = "rs17gmh5wafm5p"
inv['M13 Forward (-20)'] = "rs17tcpupe7fdh"
inv['M13 Reverse (-48)'] = "rs17tcph6e2qzh"
inv['pUC19'] = "rs17tcqmncjfsh"

#--------------------------------------------------------
# Provisioning things for my PCR
#
# Provision a 96 well PCR plate (https://developers.transcriptic.com/v1.0/docs/containe
# Type      Max       Dead    Safe    Capabilities
# 96-pcr    160 µL    3 µL    5 µL    pipette, sangerseq, spin, thermocycle, incubate, 
#
pcr_plate = p.ref("pcr_plate", cont_type="96-pcr", storage="cold_4")

#--------------------------------------------------------
# SYBR-including mastermix
# http://www.bioline.com/us/downloads/dl/file/id/2754/sensifast_sybr_no_rox_kit_manual.
# Instructions: 10ul mastermix + 0.8ul primer (400nM) + 0.8ul primer (400nM) + ⩽8.4ul 
#

#mastermix_tube = p.ref("mastermix_tube", cont_type="micro-2.0", storage="cold_20")
for well in ["A1", "B1", "C1", "A2", "B2", "C2"]:
    p.provision(inv['SensiFAST SYBR No-ROX'], pcr_plate.wells(well), "10:microliter")


#--------------------------------------------------------
# M13 primers
# I choose m13 (-20) and (-48) because of similar Tm. This amplifies ~110bp including p
#
# 100pmol == 1ul, since Transcriptic dilutes the 1300-1900pmol into 13-19ul (depending 
# I want 400nM in the final 20ul according to the SensiFAST documentation (==8pmol in 2
# 1ul primer in 12ul total equals 8pmol/ul
#

# http://www.idtdna.com/pages/products/dna-rna/readymade-products/readymade-primers
# Name                sequence                        Tm    Anhyd.  pmoles in 10ug
# M13 Forward (-20)   GTA AAA CGA CGG CCA GT          53.0  5228.5  1912.6
# M13 Forward (-41)   CGC CAG GGT TTT CCC AGT CAC GAC 65.5  7289.8  1371.7
# M13 Reverse (-27)   CAG GAA ACA GCT ATG AC          47.3  5212.5  1918.3
# M13 Reverse (-48)   AGC GGA TAA CAA TTT CAC ACA GG   57.2  7065.7  1415.2

primer_tube = p.ref("primer_tube", cont_type="micro-2.0", storage="cold_20")
p.provision(inv['M13 Forward (-20)'], primer_tube.wells(0), "1:microliter") # fwd -20
p.provision(inv['M13 Reverse (-48)'], primer_tube.wells(0), "1:microliter") # rev -48
p.provision(inv['water'], primer_tube.wells(0), "10:microliter") # water


#--------------------------------------------------------
# pUC19
# 1000ug/ml → 1ul = 1ug == 1000ng. Add 1ul to 49ul to get ~20ng/ul
# Then I can transfer 5ul to get 100ng total
#
template_tube = p.ref("template_tube", cont_type="micro-2.0", storage="cold_20")
p.provision(inv['pUC19'], template_tube.wells(0), "1:microliter")
p.provision(inv['water'], template_tube.wells(0), "49:microliter") # water


#-------------------------------------------------------------
```

```

# Move all the reagents into the pcr plate
# The "dispense" command does not work because it needs ⩾10ul per dispense
# in increments of 5ul
#
for wells, ul in (["A1","B1","C1"], 4), (["A2", "B2", "C2"], 9), (["A3", "B3", "C3"], 20
    for well in wells:
        p.provision(inv['water'], pcr_plate.wells(well), "{}:microliter".format(ul))

p.transfer(template_tube.wells(0), pcr_plate.wells(["A1", "B1", "C1"]), "5:microliter")
p.transfer(primer_tube.wells(0), pcr_plate.wells(["A1", "B1", "C1", "A2", "B2", "C2"]),

#--------------------------------------------------------
# Thermocycle, with a hot start (95C for 2m)
# Based on http://www.bioline.com/us/downloads/dl/file/id/2754/sensifast_sybr_no_rox_kit
# I also found http://www.environmental-microbiology.de/pdf_files/M13PCR_13jan2014.pdf
# p.seal before thermocycling is enforced by transcriptic
#
p.seal(pcr_plate)
p.thermocycle(pcr_plate, [{
      "cycles": 1, "steps": [{
          "temperature": "95:celsius",
          "duration": "2:minute"}]
    }, {
      "cycles": 40, "steps": [{
          "temperature": "95:celsius",
          "duration": "5:second"}, {
          "temperature": "60:celsius",
          "duration": "20:second"}, {
          "temperature": "72:celsius",
          "duration": "15:second", "read": True}]
    }],
    volume="20:microliter", # volume is optional
    dataref="qpcr_{}".format(experiment_name),
    # Dyes to use for qPCR must be specified (tells transcriptic what aborbance to use?
    dyes={"SYBR": ["A1", "B1", "C1", "A2", "B2", "C2", "A3", "B3", "C3"]},
    # standard melting curve parameters
    melting_start="65:celsius",
    melting_end="95:celsius",
    melting_increment="0.5:celsius",
    melting_rate="5:second")

#--------------------------------------------------------
# Run a gel
# agarose(8,0.8%): 8 lanes, 0.8% agarose 10 minutes recommended
# 10 microliters is used in the example documentation
# ladder1: References at 100bp, 250bp, 500bp, 1000bp, and 2000bp.
# The gel already includes SYBR green
#
p.gel_separate(pcr_plate.wells(["A1", "B1", "C1", "A2", "B2", "C2", "A3", "B3"]),
               "10:microliter", "agarose(8,0.8%)",
               "ladder1", "10:minute", "gel_{}".format(experiment_name))


#-------------------------------------------------------------
# Analyze and output the protocol
#
jprotocol = json.dumps(p.as_dict(), indent=2)
print(jprotocol)
open("protocol.json",'w').write(jprotocol)
uprint("Analyze protocol")
!echo '{jprotocol}' | transcriptic analyze
```

Antha

Code Walkthrough - Antha OS Documentation

Every element in the Antha environment needs a unique element name. For this element, the name is "Aliquot".

The Aliquot element takes nine pieces of data as its Parameters. **Parameters are the non-physical inputs for an element**, like temperature, duration, or volume. The physical inputs for an element are described in the Inputs block.

The **Data block of an Antha element defines the information produced by the element as a data output**. These include things like final sample volume, number of aliquots performed, or thaw-time required.

For this Aliquot element, there is a single data output WellsUsed. The WellsUsed data output is as the name suggests a list of wells in a specific plate that have been used by this element. The map structure is used here to associate the list of well coordinates that will contain an aliquot to the specified output plate name.

The **Inputs block of an element file defines the physical materials required by the element**. These include things like solution sample, DNA part, or multi-well plate.

The Outputs section lists the physical things that are generated by an element.

The Setup block is performed the first time that an element is executed. This can be used to perform any configuration that is needed globally for the element, and is also used to define any special setup that may be needed for groups of concurrent tasks that might be executed at the same time. Any variables that need to be accessed by the Steps function globally can be defined here as well, but need to be handled with care to avoid concurrency problems.

At this current time, the Setup block is not supported by Antha, but will be in future releases.

The heart of an Antha element is **the Steps block, which defines the actual steps taken to transform a set of Parameters and Inputs into Data and Outputs**. The Steps block is a kernel function, meaning it shares no information for every concurrent sample that is processed, and defines the workflow to transform a single block of inputs and samples into a single set of outputs, even if the element is operating on an entire array (such as micro-titre plate of samples at once).

```
1  // https://github.com/antha-lang/elements/tree/master/an/AnthaAcademy/Lesson1_Commands/l
2
3  // Example protocol demonstrating the use of the Sample function.
4  // The Sample function is not sufficient to generate liquid handling instructions alone,
5  // We would need a Mix command to instruct where to put the sample
6  // We can either modify the code to add this or wire the output Sample into the Lesson1F
7  // Any comment placed here directly above the protocol name will appear in AnthaOS as th
8  //
9  // Concepts covered:
10 // Anatomy of an Antha element
11 // types
12 // Volume
13 // Comments and AnthaOS
14 // LHComponent
15 // Sampling
16 // Reading Code
17 // imports
18 // functions
19
20 protocol Lesson1A_Sample // this is the name of the protocol Lessonthat will be called i
21
22 // the mixer package must be imported to use the Sample function
23 import (
24   "github.com/antha-lang/antha/antha/anthalib/mixer"
25 )
26
27 // Input parameters for this protocol (data)
28 Parameters {
29   // antha, like golang is a strongly typed language in which the type of a variable mus
30   // In this case we're creating a variable called SampleVolume which is of type Volume;
31   // the type system allows the antha compiler to catch many types of common errors befo
32   // the antha type system extends this to biological types such as volumes here.
33   // functions require inputs of particular types to be adhered to.
34   // Any text written above any of the parameters, Data, Inputs and Outputs variables
35   // will appear in AnthaOS as annotations.
36   SampleVolume Volume
37 }
38
39 // Data which is returned from this protocol, and data types
40 Data {
41   // Antha inherits all standard primitives valid in golang;
42   //for example the string type shown here used to return a textual message
43   Status string
44 }
45
46 // Physical Inputs to this protocol with types
47 Inputs {
48   // the LHComponent is the principal liquidhandling type in antha
49   // the * signifies that this is a pointer to the component rather than the component i
50   // most key antha functions such as Sample and Mix use *LHComponent rather than LHComp
51   Solution *LHComponent
52 }
53
54 // Physical outputs from this protocol with types
55 Outputs {
56   // An output LHComponent variable is created called Sample
57   Sample *LHComponent
58 }
59
60 Requirements {
61
62 }
63
64 // Conditions to run on startup
65 Setup {
66
67 }
68
69 // The core process for this protocol, with the steps to be performed
70 // for every input
71 Steps {
72
73   // Programming is typically made up of a series of functions.
74   // Functions, like mathematical functions and like the antha elements themselves,
75   // are black boxes which process some input arguments to produce outputs.
76
```

```
77    // In this line of code we have a variable on the left called Sample.
78    // We initialised this variable as an LHComponent above in the Outputs section.
79    // Because the Sample is to the left of an = sign,
80    // the value of Sample will be updated as the product of the mixer.Sample function to
81    // At the top of the element file we can see that we import a library which ends with
82    // Here we are using a function called Sample from the mixer library.
83    // This demonstrates one use of a full stop when reading code: accessing code stored i
84    // In Antha, as with Golang, any code which is imported from a package will always sta
85    // We can tell Sample is a function here since it is proceded by parenthesis( ).
86    // The contents of the parentheses, Solution and SampleVolume, are the input
87    // arguments to the function. We can find out what a specific function requires as inp
88    // In the mixer library the function signature can be found,
89    // here it is:
90    // func Sample(l *LHComponent, v Volume) *LHComponent {
91    // The function signature  shows that the function requires a *LHComponent and a Volu
92    Sample = mixer.Sample(Solution, SampleVolume)
93
94    // The Sample function is not sufficient to generate liquid handling instructions alon
95    // We would need a Mix command to instruct where to put the sample
96    // We can either modify the code to add this or wire the output Sample into the Lessor
97
98    // we can also export data only outputs.
99    // In this case we'll use quotations to write a message as a string like this:
100   Status = "Lesson 1A_Sample has been a success, now wire the corresponding output into
101
102 }
103
104 // Run after controls and a steps block are completed to
105 // post process any data and provide downstream results
106 Analysis {
107 }
108
109 // A block of tests to perform to validate that the sample was processed
110 //correctly. Optionally, destructive tests can be performed to validate
111 //results on a dipstick basis
112 Validation {
113
114 }
```