Coding a concept design

Turning concepts into a running app, by hand and with agents

Eagon Meng

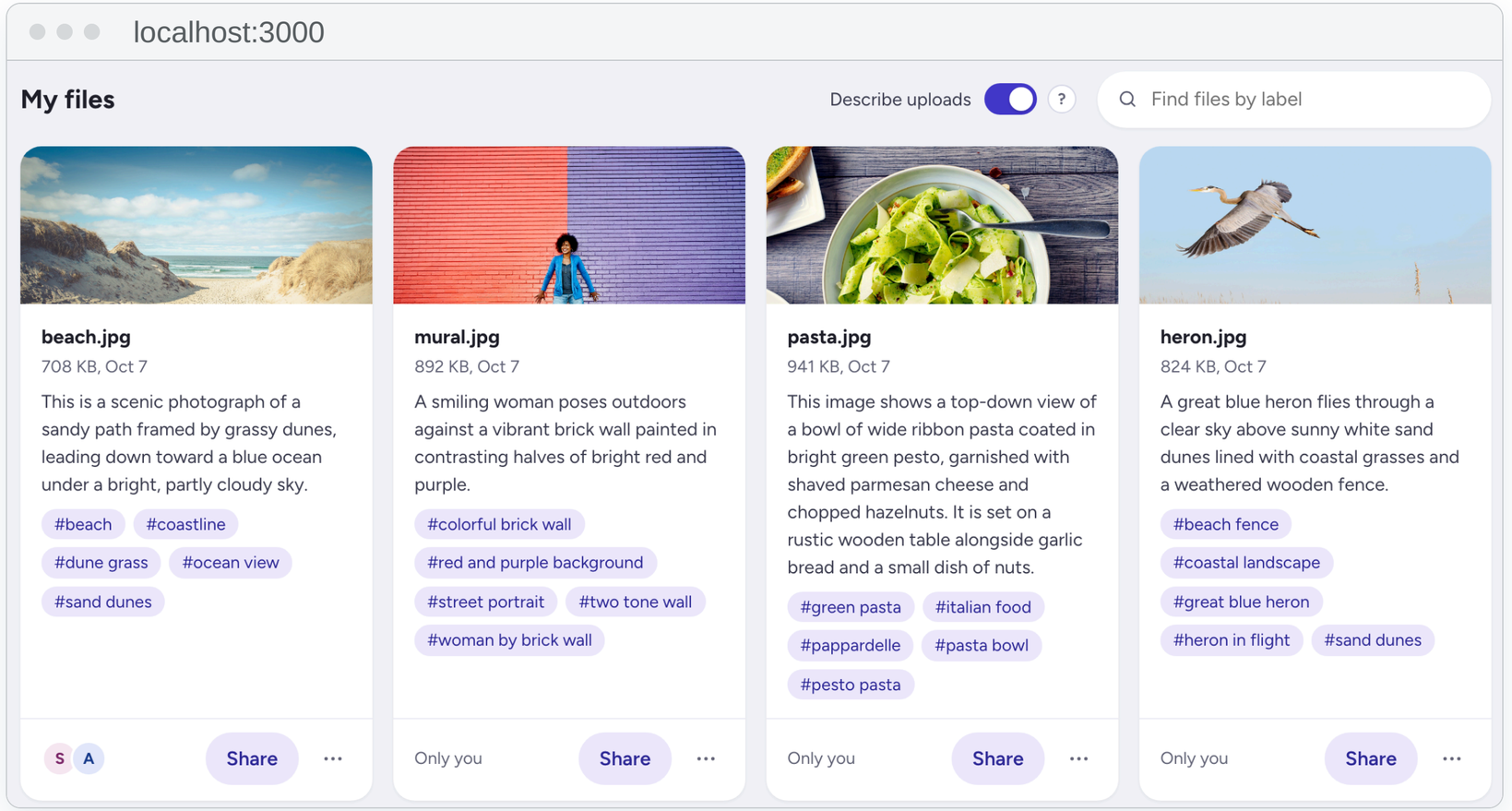
6.1040 Software Design

Part 1

An app made of concepts

What is ConceptBox made of?

Maya



An app made of concepts

Which concepts would you need?

ConceptBox

In almost every app

Authenticating

`register` `authenticate`

Sessioning

`start` `use` `end`

For a file service

Storing

`start` `finish` `delete`

Sharing

`share` `revoke`

A class forum

Authenticating

`register` `authenticate`

Sessioning

`start` `use` `end`

For a forum

Posting

`post` `edit` `delete`

Upvoting

`upvote` `unvote`

An app made of concepts

Deleting a file, the usual way

the usual handler

```
app.post("/files/delete", async (req, res) => {
  const session =
    await db.sessions.findOne({ token: req.body.session });
  if (!session || session.lastUsed < oneWeekAgo())
    return res.status(401).end();
  await db.sessions.updateOne({ _id: session._id },
    { $set: { lastUsed: new Date() } });

  const file = await db.files.findOne({ _id: req.body.file });
  if (!file || file.uploader !== session.user)
    return res.status(404).end();

  await bucket.delete(file.key);
  await db.files.deleteOne({ _id: file._id });
  await db.shares.deleteMany({ item: file._id });
  res.json({ file: file._id });
});
```

1. Check the session
2. Check that Maya owns the file
3. Delete the file
4. Revoke the shares

A forum deletes posts too. Which of these lines would you reuse?

An app made of concepts

Where does each job go?

1. Check the session
2. Check that Maya owns the file
3. Delete the file
4. Revoke the shares

Sessioning

`start` `use` `end`

the composition
code written for ConceptBox

Storing

`start` `finish` `delete`

Sharing

`share` `revoke`

An app made of concepts

Deleting a file in ConceptBox

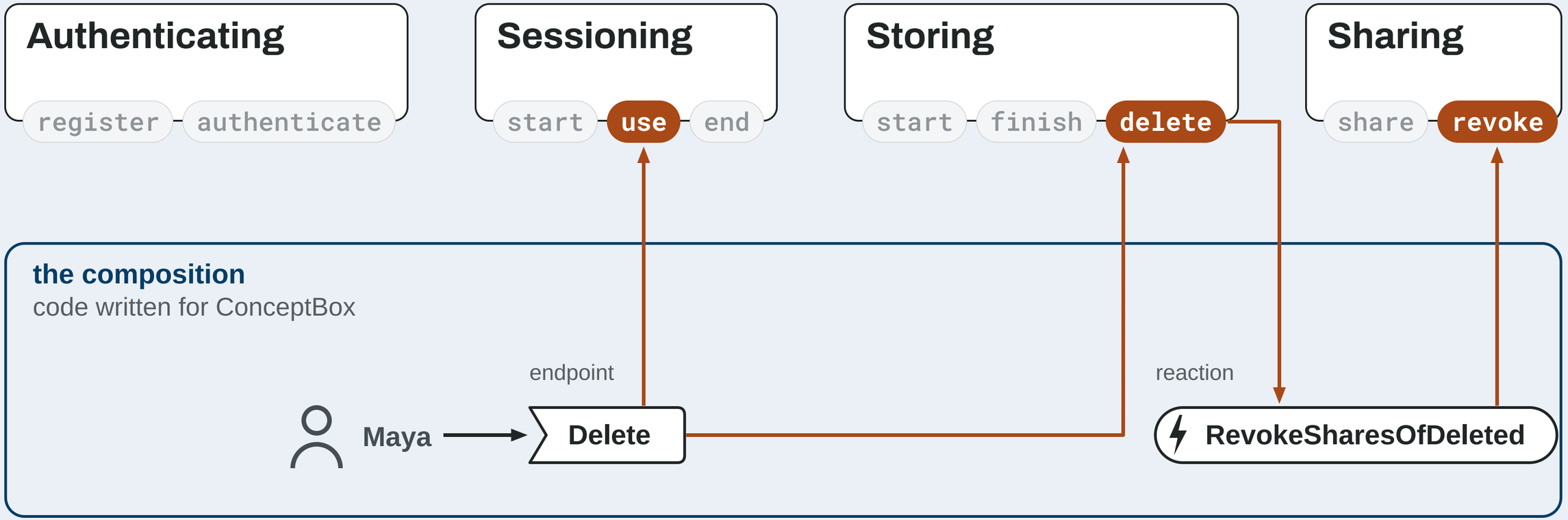
the usual handler

src/compositions/files.ts, first version

```
68  const Delete = endpoint(
    ...
72        .then(Sessioning.use({ session }).responds({ subject: user }))
73        .then(
74          where(owns({ user, file }))
75            .then(Storing.delete({ file }).responds({}))
76            .then(respond({ file }))
    ...
85  const RevokeSharesOfDeleted = reaction(({ file, recipient }) =>
86    when(Storing.delete({ file }).responds({}))
87      .where(Sharing._recipients({ item: file }).is({ recipient }))
88      .then(Sharing.revoke({ item: file, recipient })),
```

1. Check the session
2. Check that Maya owns the file
3. Delete the file
4. Revoke the shares

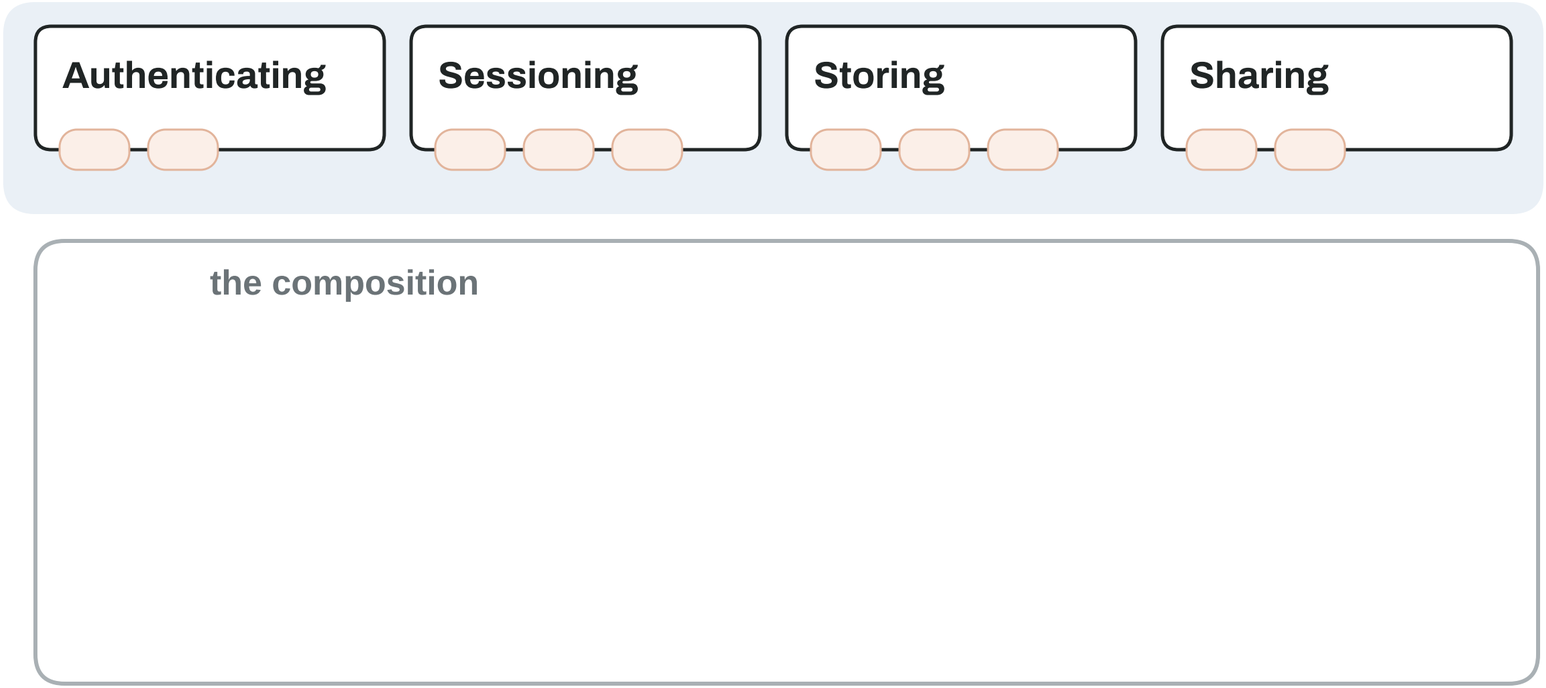
ConceptBox is made of concepts you reuse, joined by endpoints and reactions written for it.



Part 2

Concepts in code

You have a concept specification. How do you write the code?



Concepts in code

Sharing

design/concepts/Sharing.md, summary

Sharing [Item, Person]

State

a set of **Shares** with
- an Item
- a recipient Person
- unique item and recipient

Actions `share` `revoke`

Queries `_recipients` `_sharedWith`

Principle

Maya shares beach.jpg with Sam.

Sharing

Shares

item	recipient
beach.jpg	sam

`share` `revoke`

Maya shares beach.jpg with Ada.

Sharing

Shares

item	recipient
beach.jpg	sam
beach.jpg	ada

`share` `revoke`

Maya revokes Sam's share.

Sharing

Shares

item	recipient	
~~beach.jpg~~	~~sam~~	removed
beach.jpg	ada	

`share` **`revoke`**

Concepts in code

One specification, one class

design/concepts/Sharing.md

```
1  # Sharing
   ...
16 external Item
   ...
19 external Person
   ...
26 a set of Shares with
27   an Item
28   a recipient Person
29   unique item and recipient
```

src/concepts/Sharing/Sharing.ts

```
6  interface ShareDocument<Item, Person> {
7    item: Item;
8    recipient: Person;
9  }
   ...
12 export class SharingConcept<Item extends string,
       Person extends string> {
13   private readonly shares:
         Collection<ShareDocument<Item, Person>>;
   ...
21   #ready() {
22     this.indexes ??= this.shares
23       .createIndex({ item: 1, recipient: 1 }, { unique: true })
```

1. Sharing becomes one class, and Item and Person become type parameters.
2. The set of Shares becomes the collection `shares`.
3. “unique item and recipient” becomes a unique index.

Concepts in code

Each call responds or refuses, and code matches on which

```
Sharing.share({ item: "beach.jpg", recipient: "sam" })
  .responds({ item: "beach.jpg", recipient: "sam" })
```



Concepts in code

Each call responds or refuses, and code matches on which

```
Sharing.share({ item: "beach.jpg", recipient: "sam" })
  .responds({ item: "beach.jpg", recipient: "sam" })
```

```
Sharing.share({ item: "beach.jpg", recipient: "sam" })
```

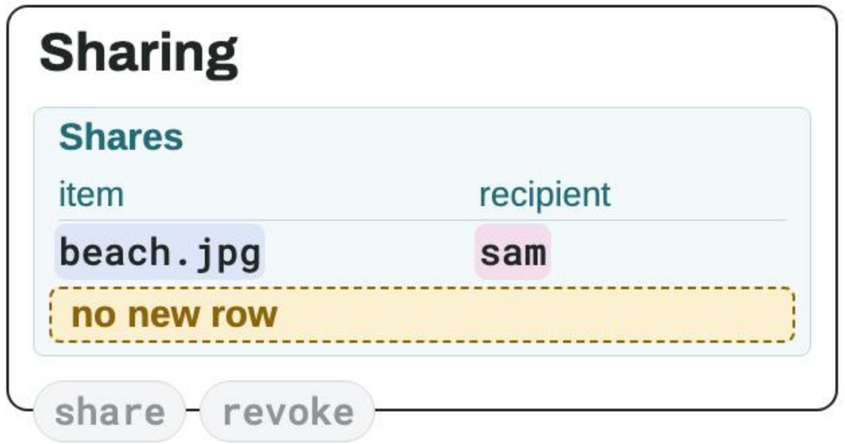


Concepts in code

Each call responds or refuses, and code matches on which

```
Sharing.share({ item: "beach.jpg", recipient: "sam" })
  .responds({ item: "beach.jpg", recipient: "sam" })
```

```
Sharing.share({ item: "beach.jpg", recipient: "sam" })
  .refuses({ error: "ALREADY_SHARED" })
```



Concepts in code

Each call responds or refuses, and code matches on which

```
Sharing.share({ item: "beach.jpg", recipient: "sam" })
  .responds({ item: "beach.jpg", recipient: "sam" })
```

```
Sharing.share({ item: "beach.jpg", recipient: "sam" })
  .refuses({ error: "ALREADY_SHARED" })
```

Sharing

Shares

item	recipient
beach.jpg	sam

no new row

share revoke

a call

`Concept.action({ field: value })`
the action, and what it was given

`.responds({ field: value })`
matches a success with these values

`.refuses({ error: "CODE" })`
matches a refusal with this error

Concepts in code

An action becomes a method

design/concepts/Sharing.md

```
35 share(item: Item, recipient: Person) :
       returns (item: Item, recipient: Person)
36   where a share has item and recipient
37   then
38     refuses ALREADY_SHARED
           "This is already shared with that person."
39   where no share has item and recipient
40   then
41     append a new share with item and recipient
42     returns item, recipient
```

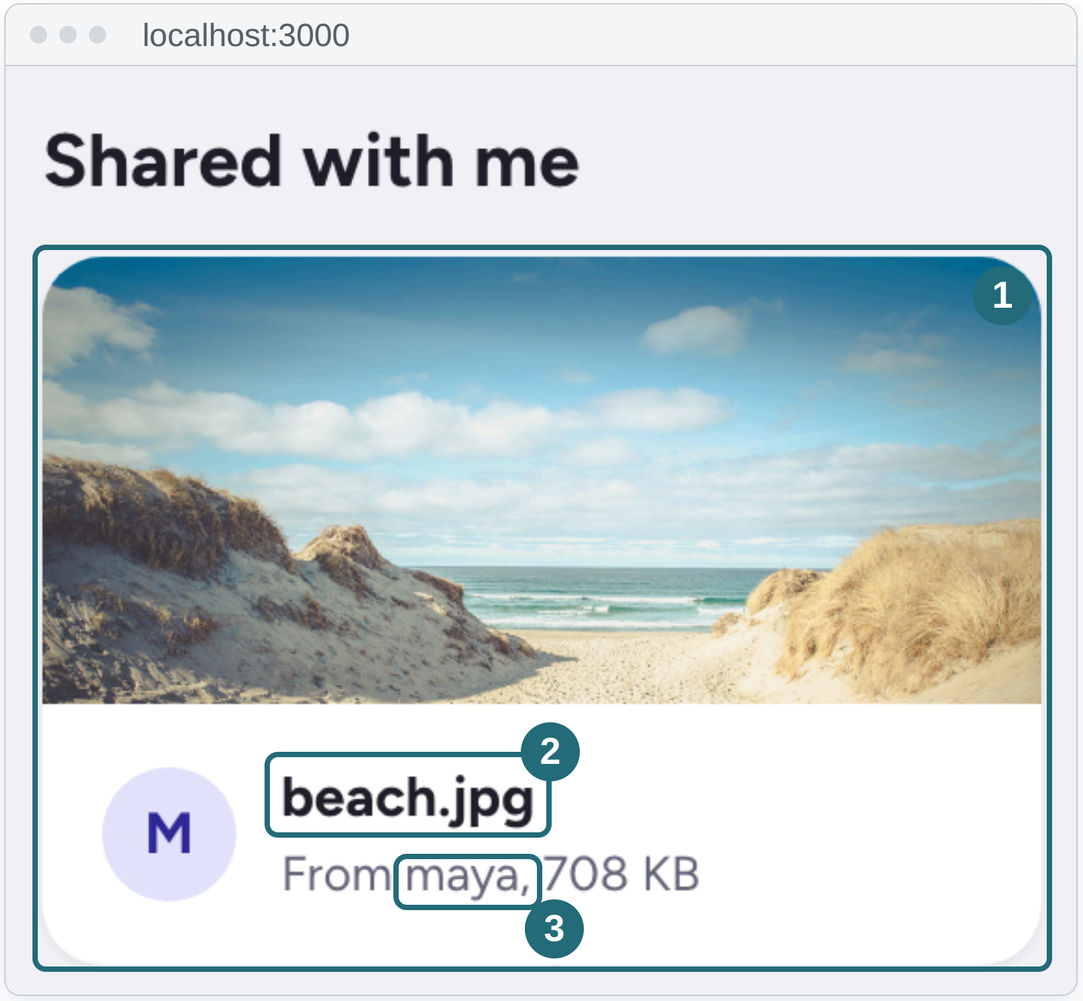
src/concepts/Sharing/Sharing.ts

```
31 async share({ item, recipient }:
       { item: Item; recipient: Person }) {
32   await this.#ready();
33   try {
34     await this.shares.insertOne({ item, recipient });
35   } catch (error) {
36     if (isDuplicate(error)) throw new AlreadyShared(
           "This is already shared with that person.");
37     throw error;
38   }
39   return { item, recipient };
40 }
```

1. The method **`share`** takes one object: `{ item, recipient }`.
2. If the share exists, `insertOne` fails on the unique index and **`share`** throws **`AlreadyShared`**.
3. Otherwise, **`share`** inserts the share and returns `{ item, recipient }`.

Concepts in code

Which concept does each piece come from?



1	This card is in Sam’s Shared with me	**Sharing**
2	beach.jpg	**Storing**
3	the username maya	**Authenticating**

Concepts in code

A query reads the state

design/concepts/Sharing.md

```
1  57 _recipients(item: Item) :
2        many (recipient: Person)
   58   Returns everyone the item is shared with,
          in the order it was shared with them,
          or no rows when it is shared with
          nobody.
```

src/concepts/Sharing/Sharing.ts

```
1  48 async _recipients({ item }: { item: Item }) {
   ...
2  50   const shares = await
            this.shares.find({ item })
            .sort({ _id: 1 }).toArray();
   51   return shares.map(({ recipient }) => ({ recipient }));
   52 }
```

1. A query's name starts with an underscore.
2. **`many`**: **`_recipients`** returns a list of rows, in the order shared, possibly empty.

```
Sharing._recipients({ item: "beach.jpg" })
```

recipient

```
"sam"
"ada"
```

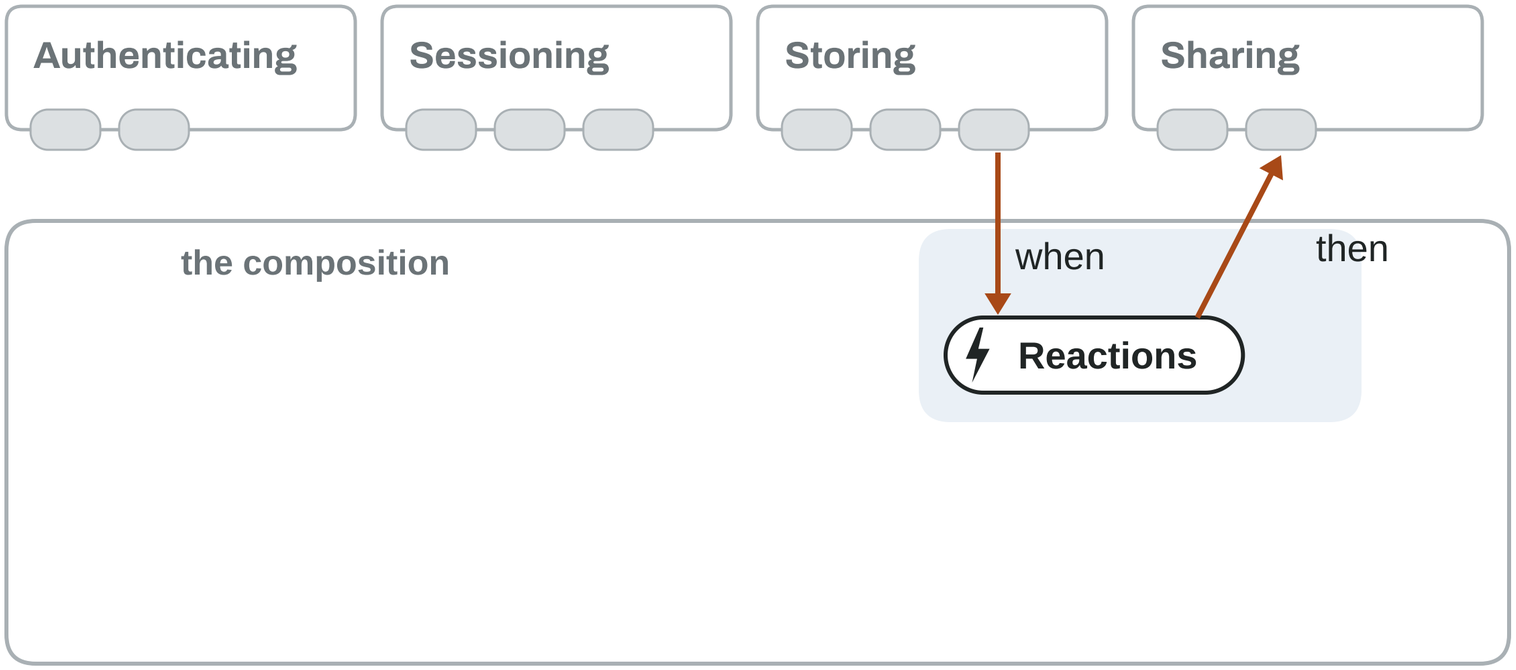
Write the principle as a specification, then each part of it as one piece of code.

the principle	design/concepts/Sharing.md	src/concepts/Sharing/Sharing.ts
	Sharing [Item, Person]	`class SharingConcept`
who each item is shared with	**State** a set of **Shares** (item, recipient) unique item and recipient	**`shares`**: a collection a unique index
Maya **shares** beach.jpg with Sam	**Actions** **`share`**`(item, recipient)` refuses **`ALREADY_SHARED`**	**`share`**`({ item, recipient })` throws **`AlreadyShared`**
Maya **revokes** Sam's share	**`revoke`**`(item, recipient)` refuses **`NOT_SHARED`**	**`revoke`**`({ item, recipient })` throws **`NotShared`**
Sam **finds** what's shared with him	**Queries** **`_sharedWith`**`(recipient)` `many (item)`	**`_sharedWith`**`({ recipient })` returns rows `{ item }`

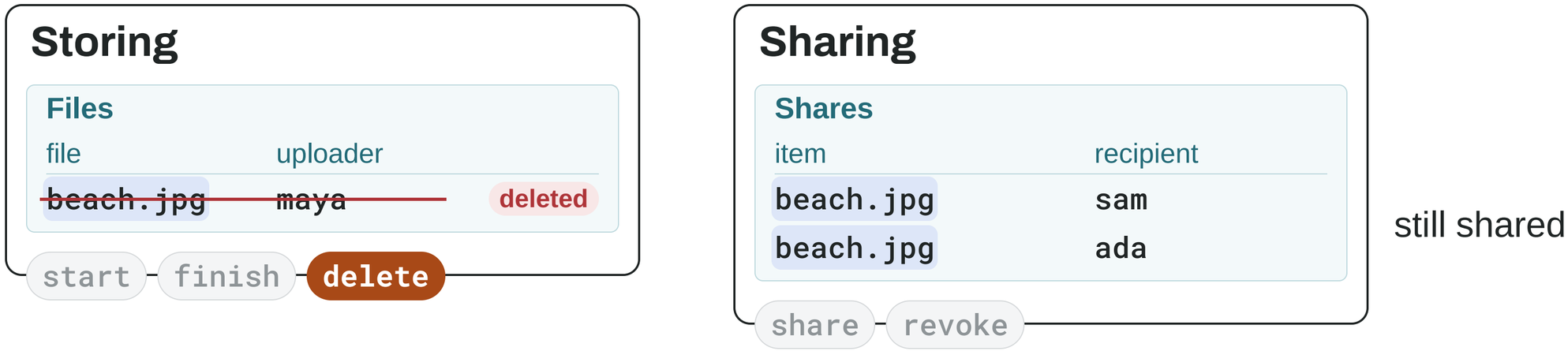
Part 3

Reactions

How do you make an action in one concept run after an action in another?



Maya deletes beach.jpg



Where would you write the code that revokes the shares?

(A) In `Storing.delete` (B) In Sharing (C) Somewhere else: a reaction ✓

RevokeSharesOfDeleted, line by line

generated/conceptbox.md, shortened

```
### files.deleting.RevokeSharesOfDeleted
…
when Storing.delete (file)
where
  Sharing._recipients (item: file)
      has (recipient)
then
  Sharing.revoke (item: file, recipient)
```

src/compositions/files.ts

```
68 const RevokeSharesOfDeleted = reaction(
       ({ file, recipient }) =>
69   when(Storing.delete({ file }).responds({}))


70     .where(Sharing._recipients({ item: file })
           .is({ recipient }))
71     .then(Sharing.revoke(
           { item: file, recipient })),
```

How to write a reaction

reaction

```
const Name = reaction(({ names }) =>
```

a name for the readout, then every name the clauses use, in one object

```
  when(Concept.action({ … }).responds({ … }))
```

the call that sets it off, as a pattern; .responds({}) matches any success

```
    .where(Concept._query({ … }).is({ … }), …)
```

optional: every line must hold; .is gives a name its value, or checks one it already has

```
    .then(Concept.action({ … })));
```

runs once for each row the where clause finds

The then clause runs once per row

generated/conceptbox.md, shortened

```
where
  Sharing._recipients (item: file) has (recipient)
then
  Sharing.revoke (item: file, recipient)
```

where

```
Sharing._recipients({ item: "beach.jpg" })
```

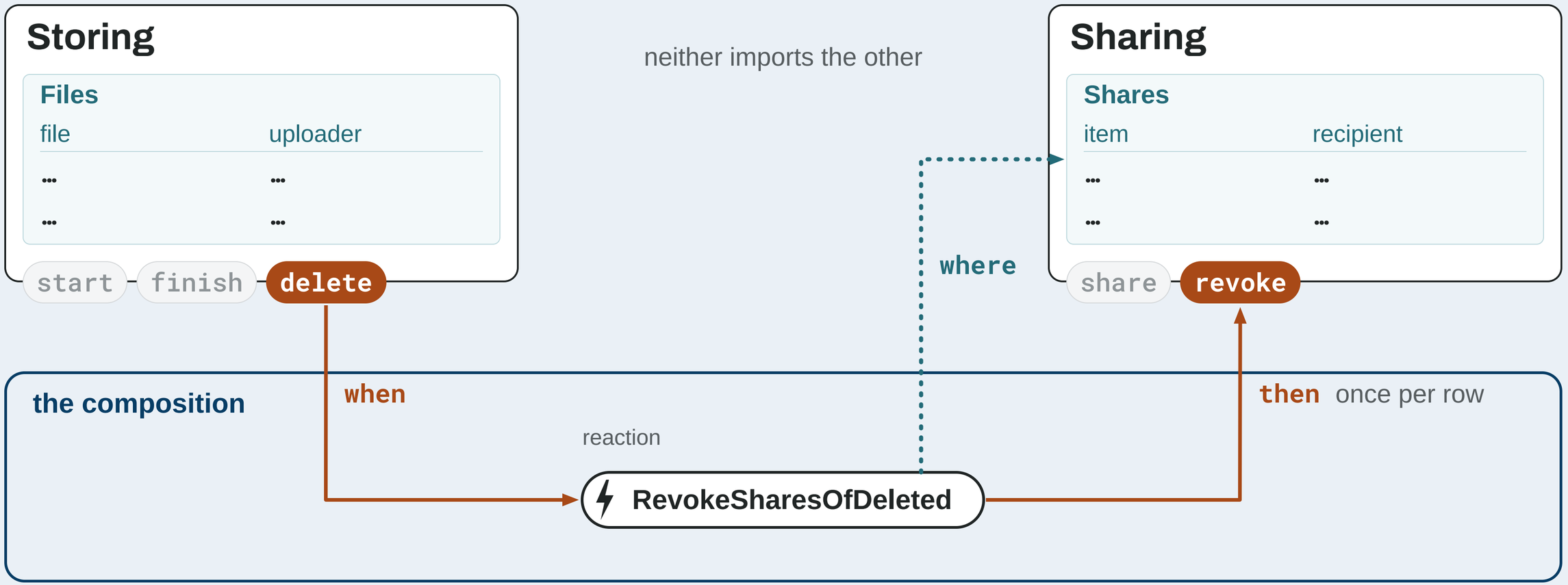
recipient
"sam"
"ada"

then

```
Sharing.revoke({ item: "beach.jpg", recipient: "sam" })
```

```
Sharing.revoke({ item: "beach.jpg", recipient: "ada" })
```

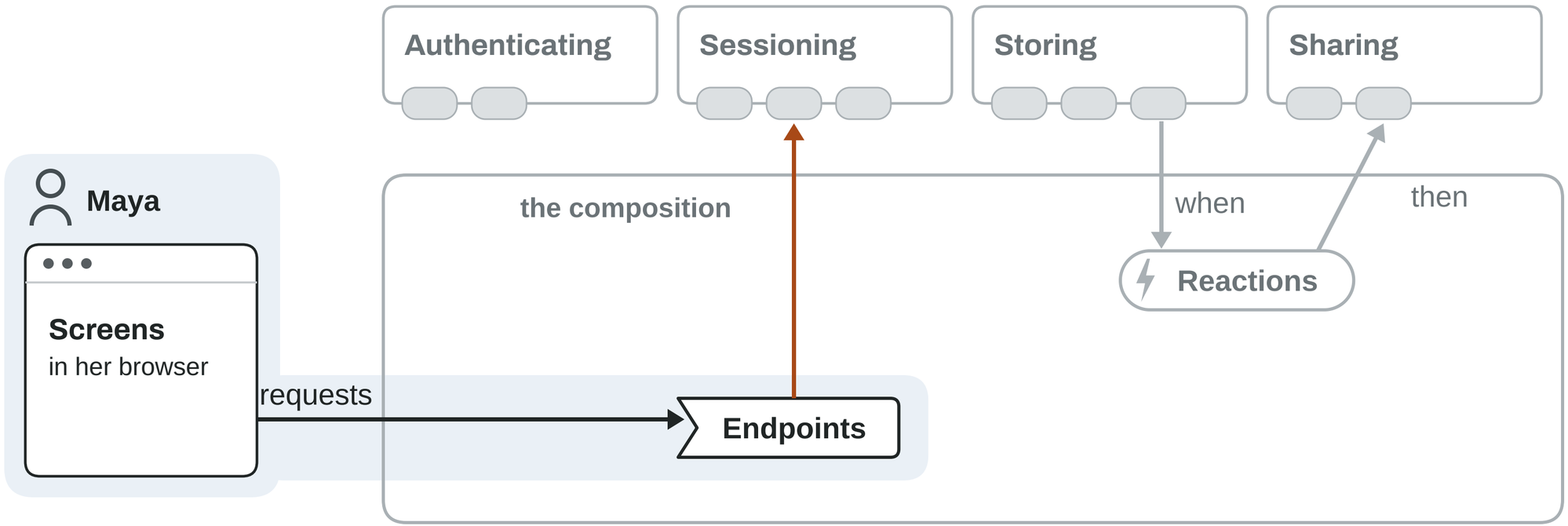
Concepts never call each other: a reaction in the composition connects them.



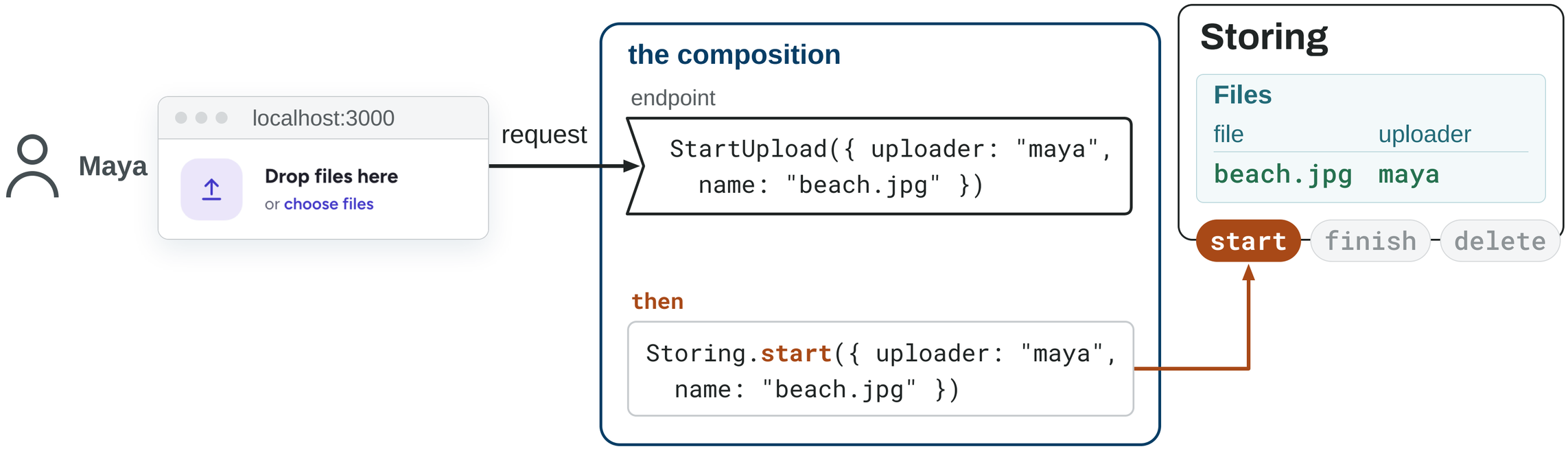
Part 4

Endpoints and flow

A request comes in from the browser. What runs, and who is asking?

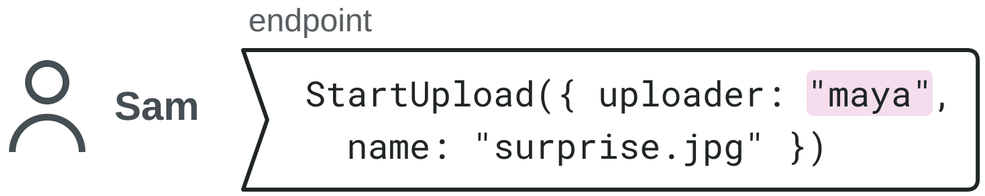


Maya uploads beach.jpg



But who says who is uploading?

The request names the uploader

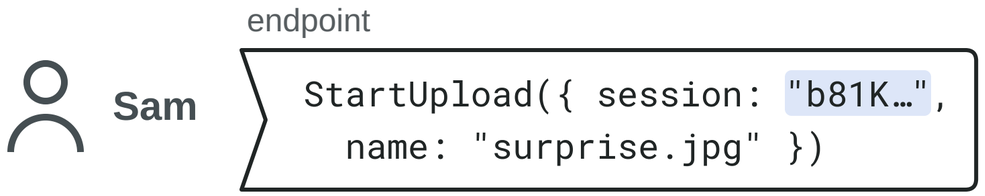


then

```
Storing.start({ uploader: "maya",
  name: "surprise.jpg" })
```

surprise.jpg is in Maya’s box.

The request carries a session



then

```
Sessioning.use({ session: "b81K…" })
  .responds({ subject: "sam" })
```

then

```
Storing.start({ uploader: "sam",
  name: "surprise.jpg" })
```

surprise.jpg is in Sam’s box.

Endpoints and flow

StartUpload, step by step

in words

The endpoint StartUpload, at /files/start,
and every name its steps bind.

1. **Receive** the `session`, name, and media type.
2. Look up the `session`'s `user` in **Sessioning**.
3. Start an upload for that `user`,
   with **Storing**.
4. **Respond** with the `file` and its
   `upload address`.

src/compositions/files.ts

```
 9 const StartUpload = endpoint(
10   "/files/start",
11   ({ session, name, mediaType,
         user, file, uploadUrl }) =>
12     receive({ session, name, mediaType })
13       .then(Sessioning.use({ session })
           .responds({ subject: user }))
14       .then(Storing.start({ uploader: user,
           name, mediaType })
           .responds({ file, uploadUrl }))
15       .then(respond({ file, uploadUrl })),
```

How to write an endpoint

endpoint

```
const Name = endpoint("/path", ({ names }) =>
```

the path the browser calls, and every name the steps use

```
  receive({ fields })
```

what the browser sends; nothing else is read from it

```
    .then(Concept.action({ … }).responds({ … }))
```

the steps, in order; .responds names what the action returns

```
    .then(where(…).then(…).named("case"),
          where(…).then(…).named("case"))
```

optional: one where for each way the request can go

```
    .then(respond({ … })),
```

answers the browser and ends the request

```
  { validators: { input: textInput } });
```

copy as is

Endpoints and flow

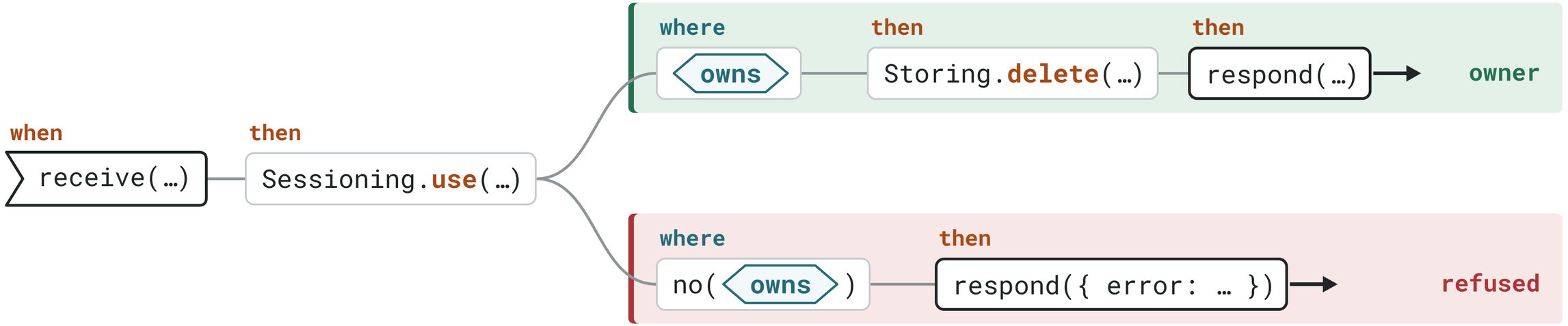
Sam tries to delete Maya’s beach.jpg

src/compositions/files.ts, first version

```
68 const Delete = endpoint(
…
71     receive({ session, file })
72       .then(Sessioning.use({ session }).responds({
             subject: user }))
73       .then(
74         where(owns({ user, file }))
75           .then(Storing.delete({ file }).responds({}))
76           .then(respond({ file }))
77           .named("owner"),
78         where(no(owns({ user, file })))
79           .then(respond({ error: "NOT_FOUND" }))
80           .named("refused"),
81       ),
```



Then clauses chain; where clauses branch



Would you refuse Maya sharing beach.jpg with herself?

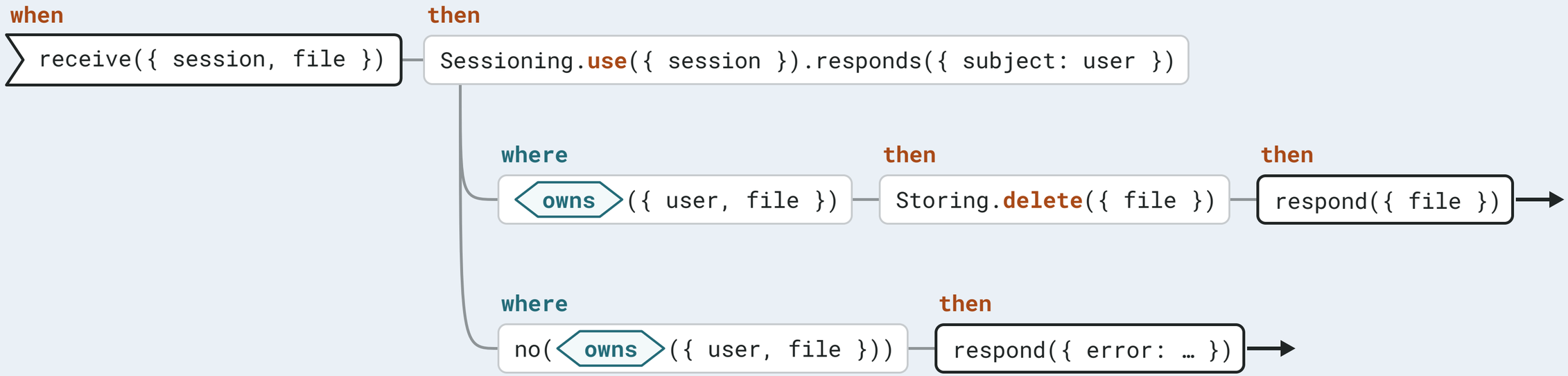
A	In `Sharing.share`	refused in every app that uses Sharing
B	In the Share endpoint	refused in ConceptBox only
C	Allow it	beach.jpg shows in Maya’s “Shared with me”

In ConceptBox: B.

src/compositions/shares.ts, Share

```
12  receive({ session, file, username })
13    .then(Sessioning.use({ session }).responds({ subject: user }))
    …
23      where(owns({ user, file }), Authenticating._byUsername({ username }).is({ user }))
24        .then(respond({ error: "SHARING_WITH_YOURSELF" }))
25        .named("yourself"),
```

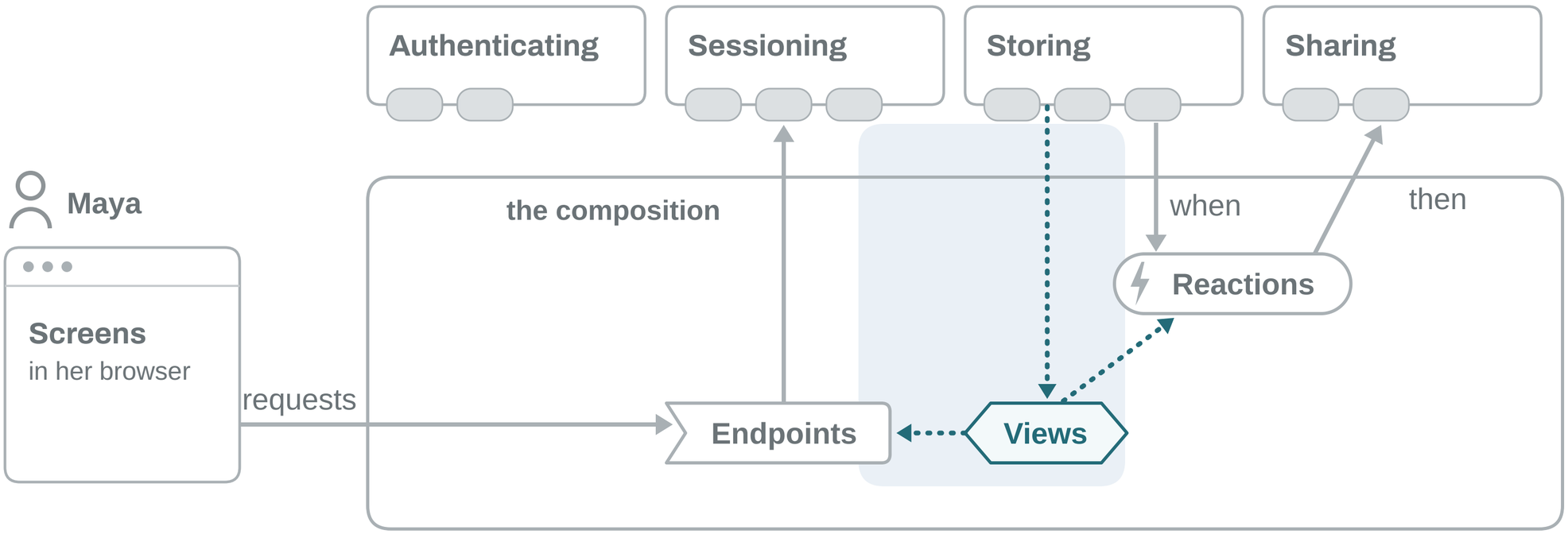
Write a request's whole flow in one place: who is asking, what it checks, what it changes.



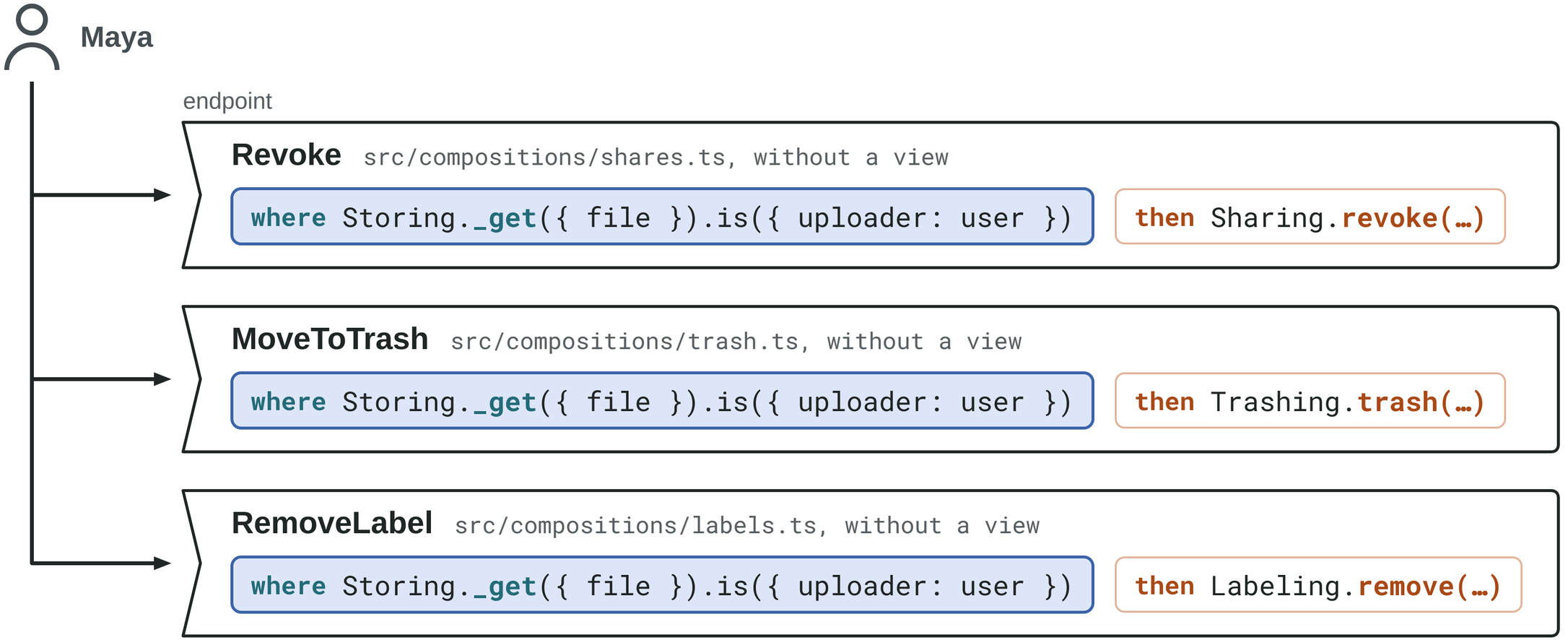
Part 5

Views

How do you write a rule once and use it anywhere?



The same check, three times



Who owns a file?

design/compositions/access.md

A person **owns a file** when they uploaded it, and only the owner may delete a file, share it, or take a share back.

generated/conceptbox.md, shortened

```
(user) owns (file)
    – inputs (user, file); outputs (); ...

  where Storing._get (file)
      has (uploader: user)
```

src/compositions/access.ts

```ts
export const owns = view(
    "(user) owns (file)",
    ({ user, file }) =>
  where(Storing._get({ file })
      .is({ uploader: user })),
).holds();
```

Views

Who can read a file?

design/compositions/access.md

A person **can read a file** when they own it, or when it is still stored, shared with them, ⋯

generated/conceptbox.md, shortened

```
(user) can read (file)
    – inputs (user, file); ⋯

  where view "(user) owns (file)"
      with (file, user)


  where
    Storing._get (file)
    Sharing._recipients (item: file)
        has (recipient: user)
⋯
```

src/compositions/access.ts

```
12 export const canRead =
       view("(user) can read (file)",
       ({ user, file }) => [
13   where(owns({ user, file })),


14   where(
15     Storing._get({ file }),
16     Sharing._recipients({ item: file })
           .is({ recipient: user }),
   ⋯
18   ),
19 ]).holds();
```

Two where clauses: either one can hold

design/compositions/access.md

A person **can read a file** when they own it, or when it is still stored, shared with them, ⋯

generated/conceptbox.md, shortened

```
(user) can read (file)
    – inputs (user, file); ⋯

  where view "(user) owns (file)"
      with (file, user)


  where
    Storing._get (file)
    Sharing._recipients (item: file)
        has (recipient: user)

⋯
```

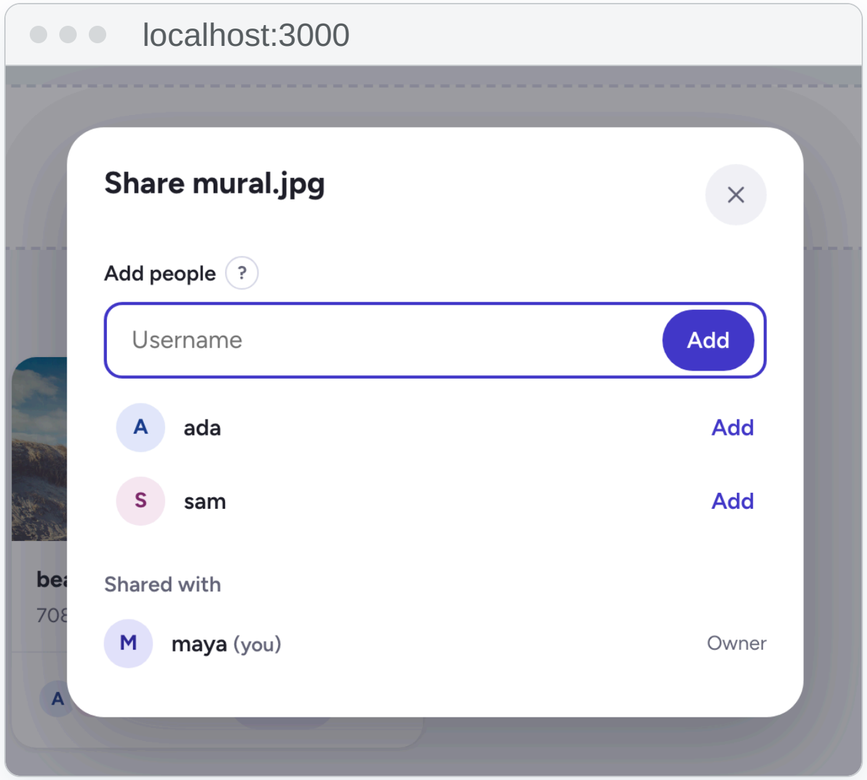
or

src/compositions/access.ts

```
12 export const canRead =
       view("(user) can read (file)",
       ({ user, file }) => [
13   where(owns({ user, file })),


14   where(
15     Storing._get({ file }),
16     Sharing._recipients({ item: file })
           .is({ recipient: user }),
   ⋯
18   ),
19 ]).holds();
```

Whom does Maya share files with?



design/compositions/shares.md

When the owner opens the share dialog, it shows people they already share files with, so they rarely type a whole username. The view **who the user shares files with** matches the recipients of the user's own files and the owners of files shared with the user.

```
sharesFilesWith({ user: "maya" })
```

```
person
"ada"
"sam"
```

Views

sharesFilesWith, line by line

generated/conceptbox.md, shortened

```
(user) shares files with (person)
    – inputs (user); outputs (person);
    bindings (file) ···

  where
    Storing._uploadedBy (uploader: user) has (file)
    Sharing._recipients (item: file)
        has (recipient: person)



···
```

src/compositions/shares.ts

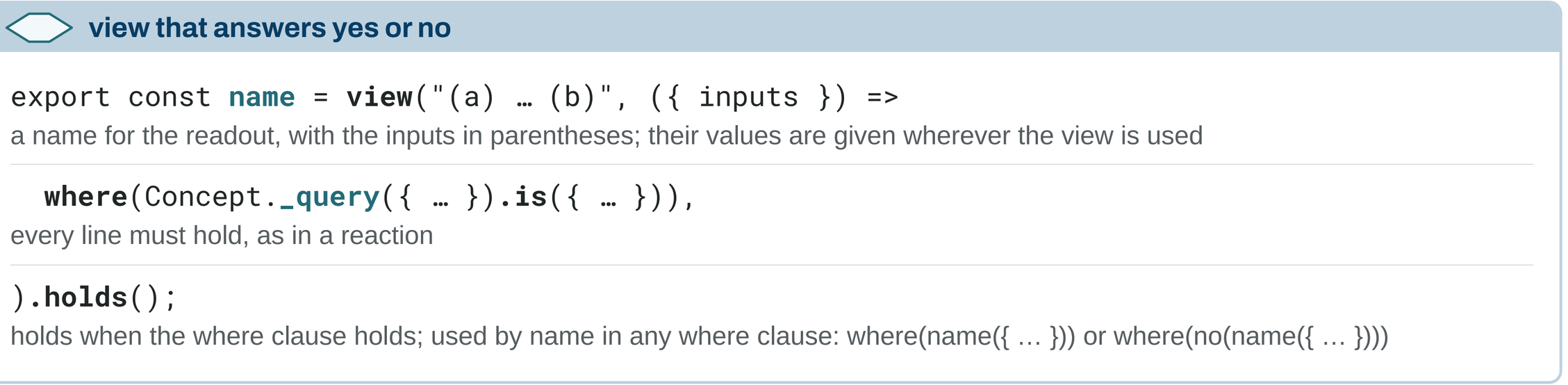
```
54 const sharesFilesWith = view(
55   "(user) shares files with (person)",
56   ({ user }, { person }, { file }) => [

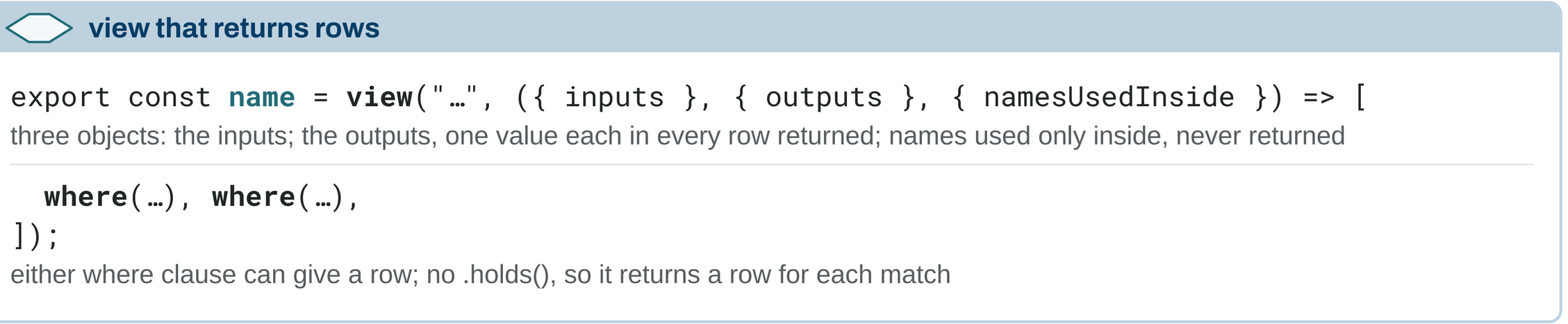
57     where(
58       Storing._uploadedBy({ uploader: user })
              .is({ file }),
59       Sharing._recipients({ item: file })
              .is({ recipient: person }),
60     ),

   ···

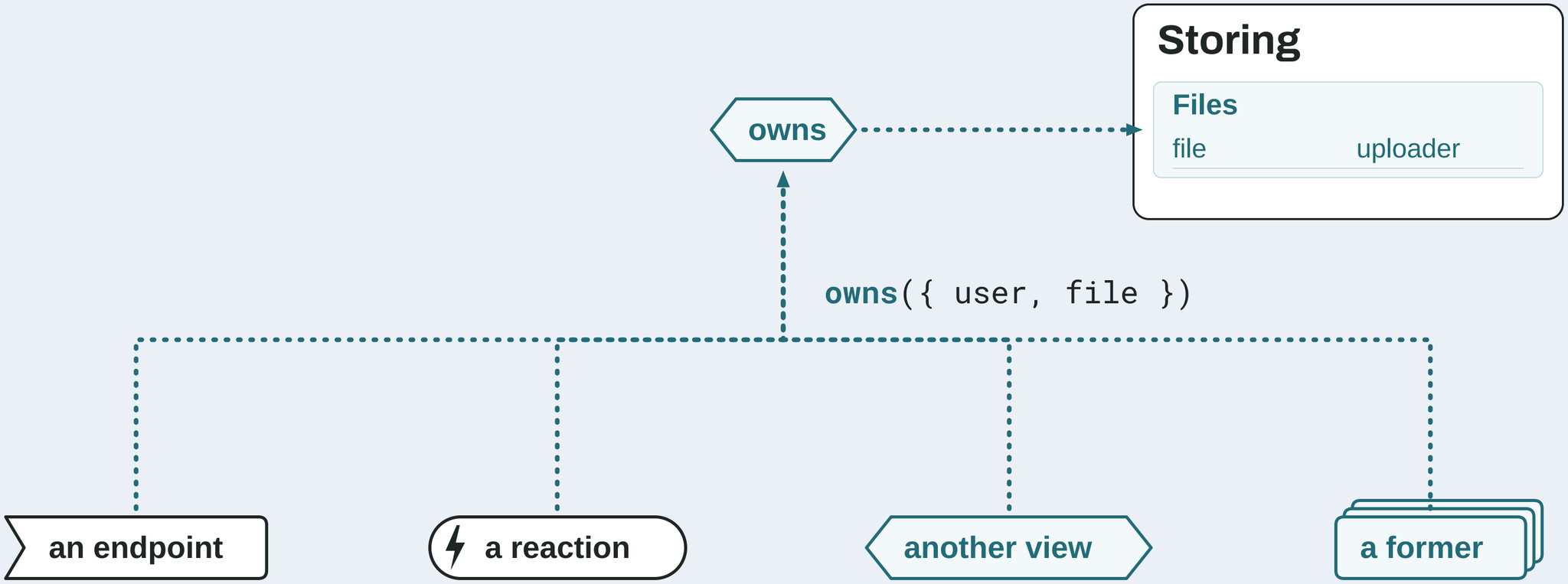
65   ],
66 );
```

How to write a view





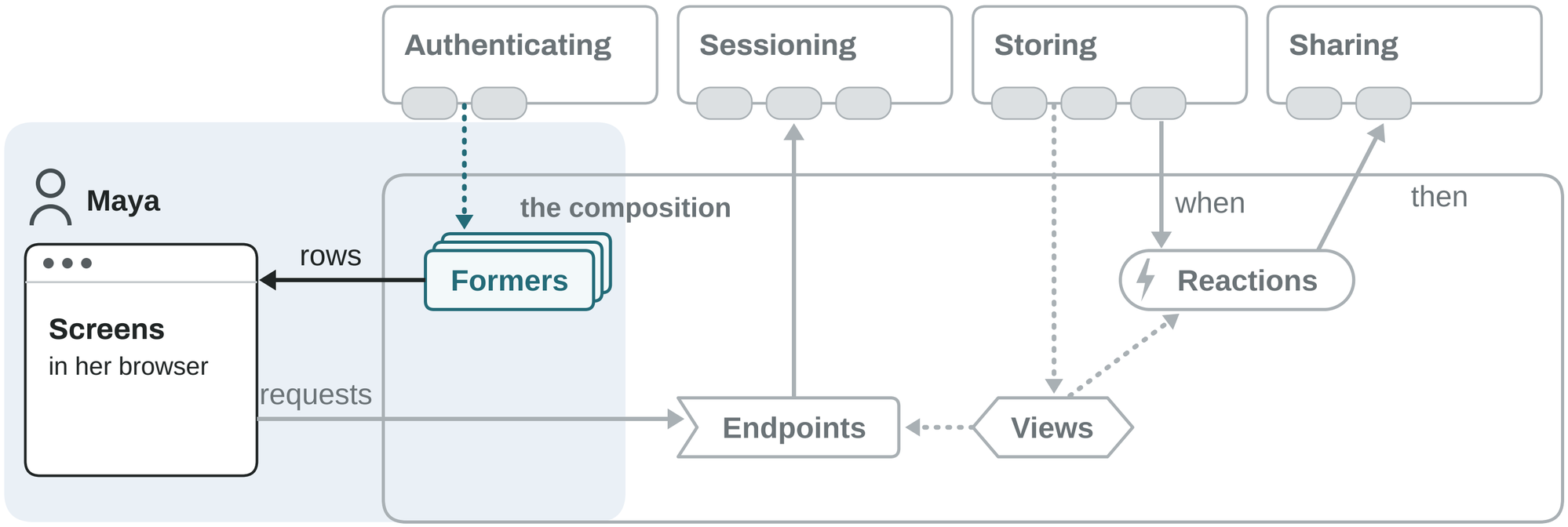
Write a rule once as a view, and check it by name everywhere.



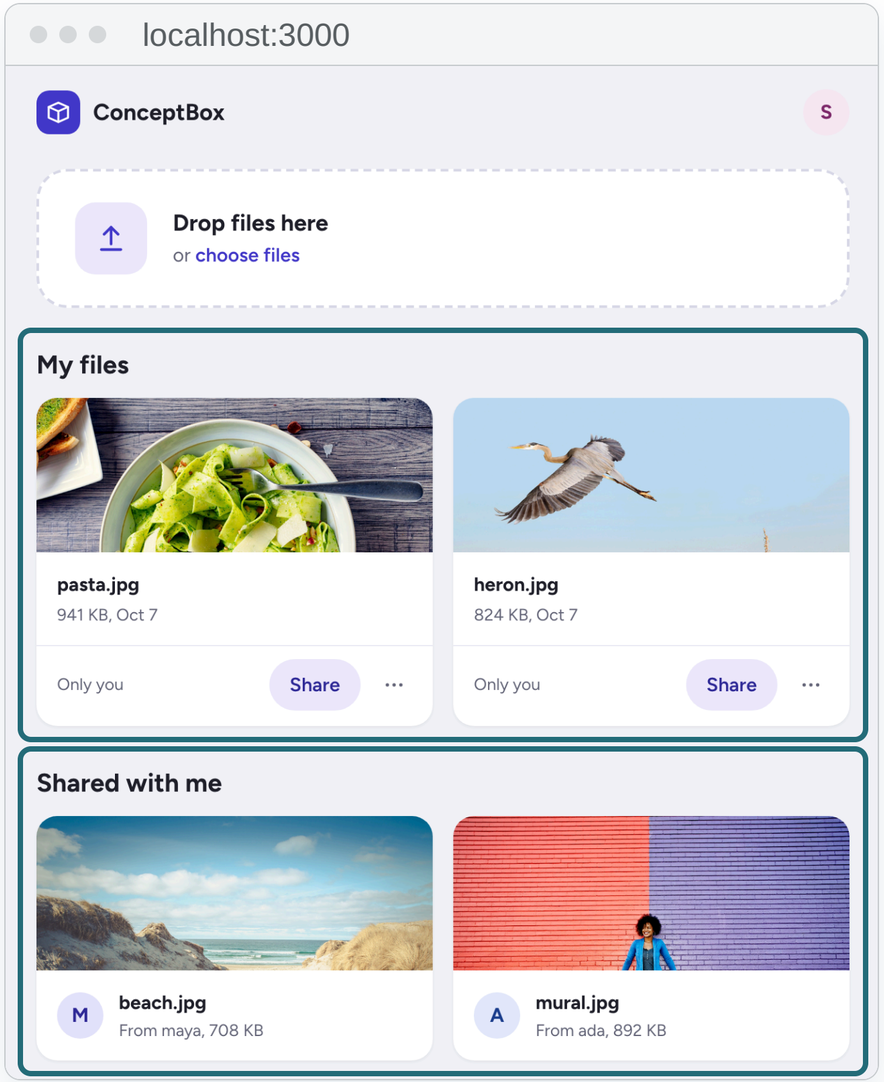
Part 6

Formers

Where does the data on a screen come from?

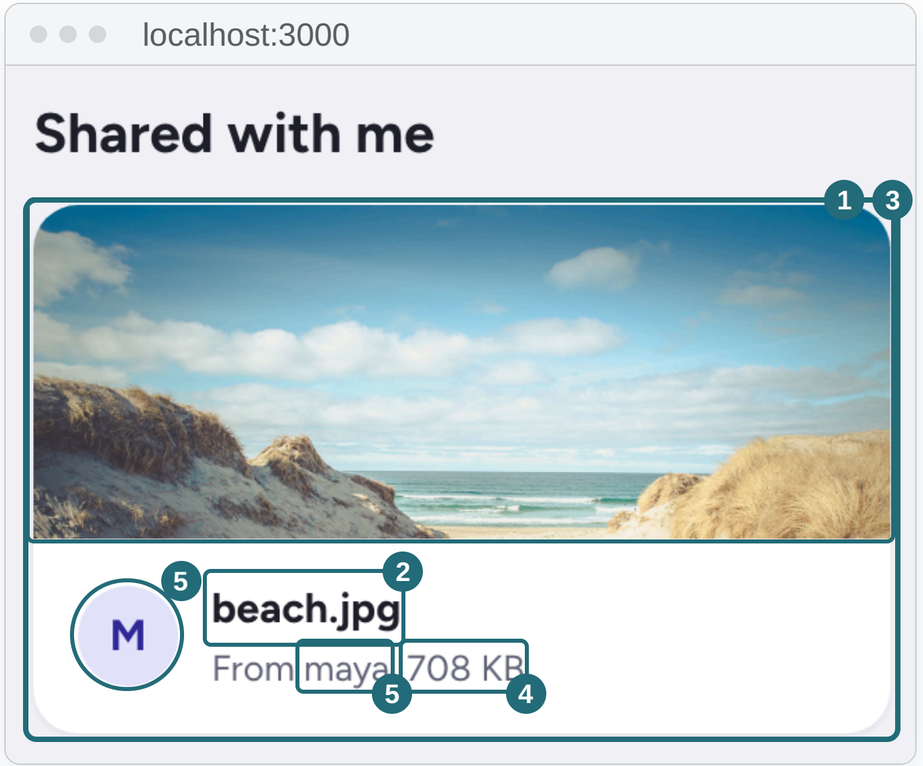


Where does the data on Sam’s screen come from?




```
sharedWithMe: ?
```

From the screen to the JSON




```
sharedWithMe: [   which rows?
  { file: "f7c2…" 1,
    name: "beach.jpg" 2,
    mediaType: "image/jpeg" 3,
    size: 708341 4,
    uploadedAt: "2026-10-07T…",
    ownerName: "maya" 5 },
  { … }
]
```

Where do the rows and the fields come from?

Sharing

Shares

item	recipient
beach.jpg	sam
mural.jpg	sam

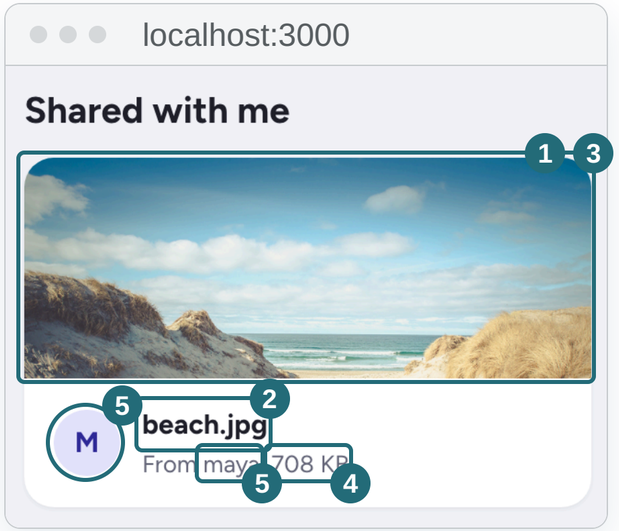
canRead

```
sharedWithMe: [
  { file: "f7c2…",
    name: "beach.jpg",
    mediaType: "image/jpeg",
    size: 708341,
    uploadedAt: "2026-10-07T…",
    ownerName: "maya" },
  { … }
]
```

Storing

Authenticating

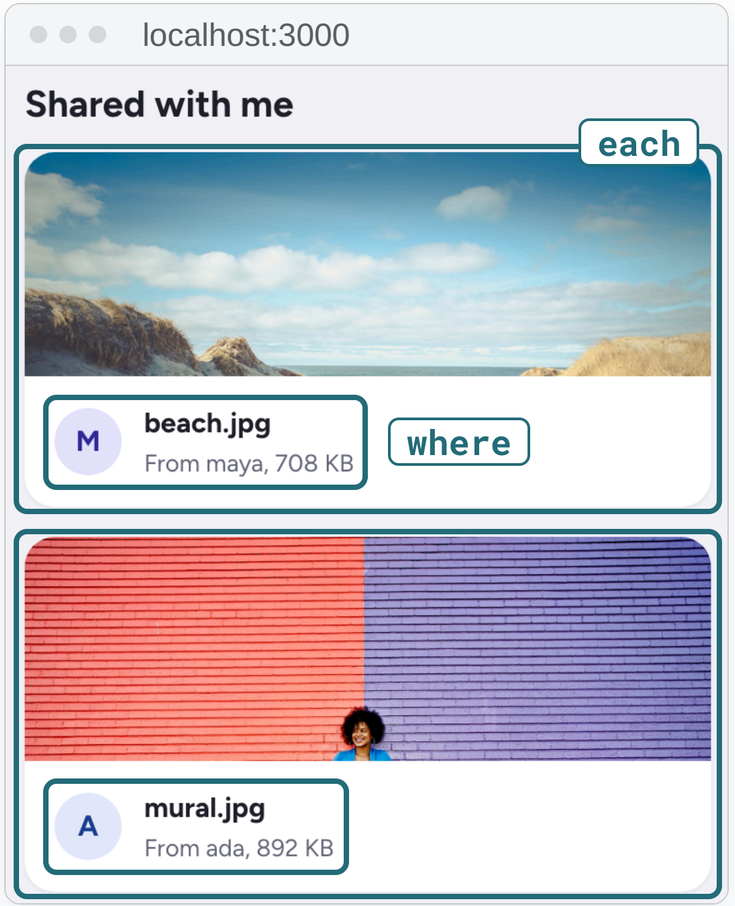
sharedWithMe, line by line



src/compositions/box.ts

```
28 const sharedWithMe = former(
29   "the files shared with (user)",
30   ({ user }, { file, name, mediaType, size, uploadedAt, owner, ownerName }) =>
31     each(Sharing._sharedWith({ recipient: user }).is({ item: file })) // 1
32       .where(
33         canRead({ user, file }),
34         Storing._get({ file }).is({ uploader: owner, name, mediaType, size, // 2 3 4
35             uploadedAt }),
36         Authenticating._username({ user: owner }).is({ username: ownerName }), // 5
37       )
38       .form({ file, name, mediaType, size, uploadedAt, ownerName }),
39 );
```

How to write a former



former

```
const name = former("description", ({ inputs }, { names }) =>
```

a name for the readout, the inputs, then every name used inside; unlike a view's, form picks what is sent

```
  each(Concept._query({ … }).is({ … }))
```

one row for each row the query returns

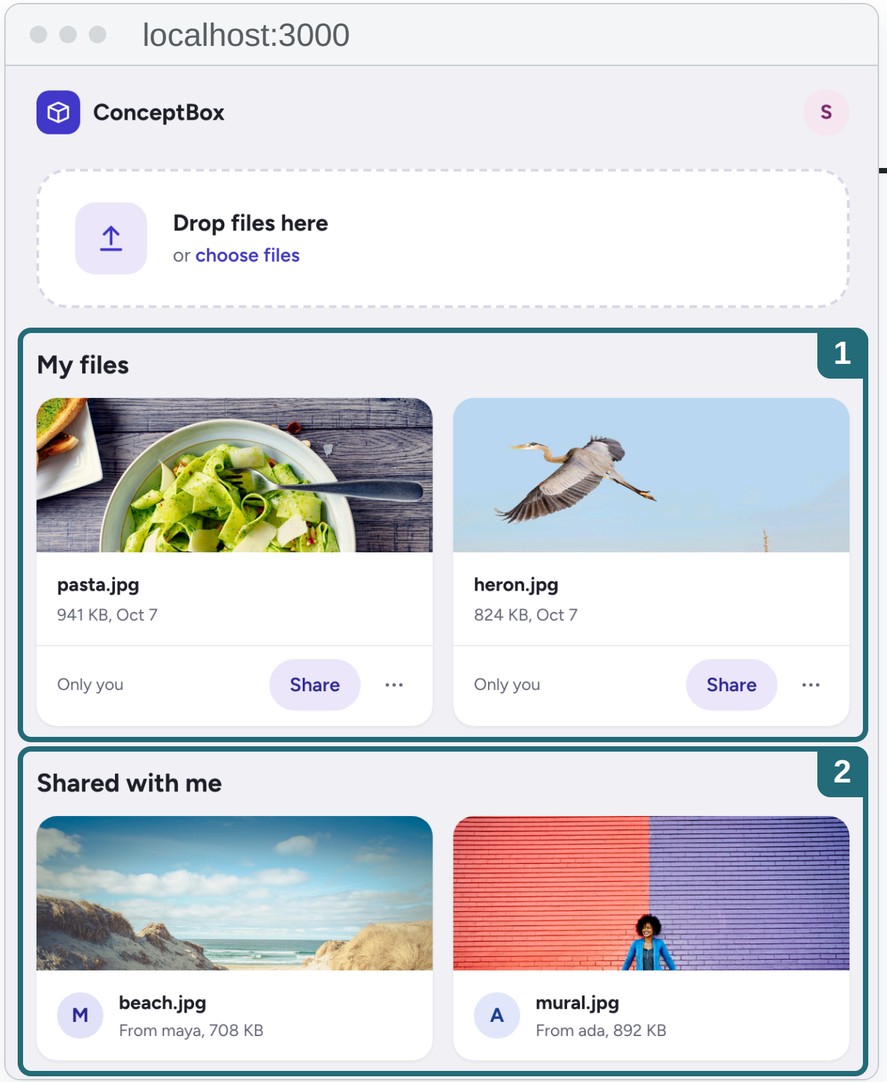
```
    .where(view({ … }), Concept._query({ … }).is({ … }))
```

keeps a row only if every line holds; .is adds fields to the row

```
    .form({ fields }));
```

the fields each row sends; a field can be another each(…) list

One endpoint sends a whole screen, one former per part

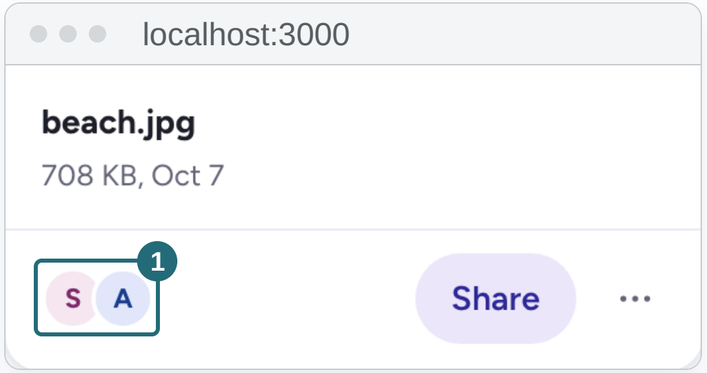


the composition

ShowBox

```
respond({
  1 myFiles:    myFiles ({ user }),
  2 sharedWithMe: sharedWithMe ({ user })
})
```

myFiles: a list inside each row

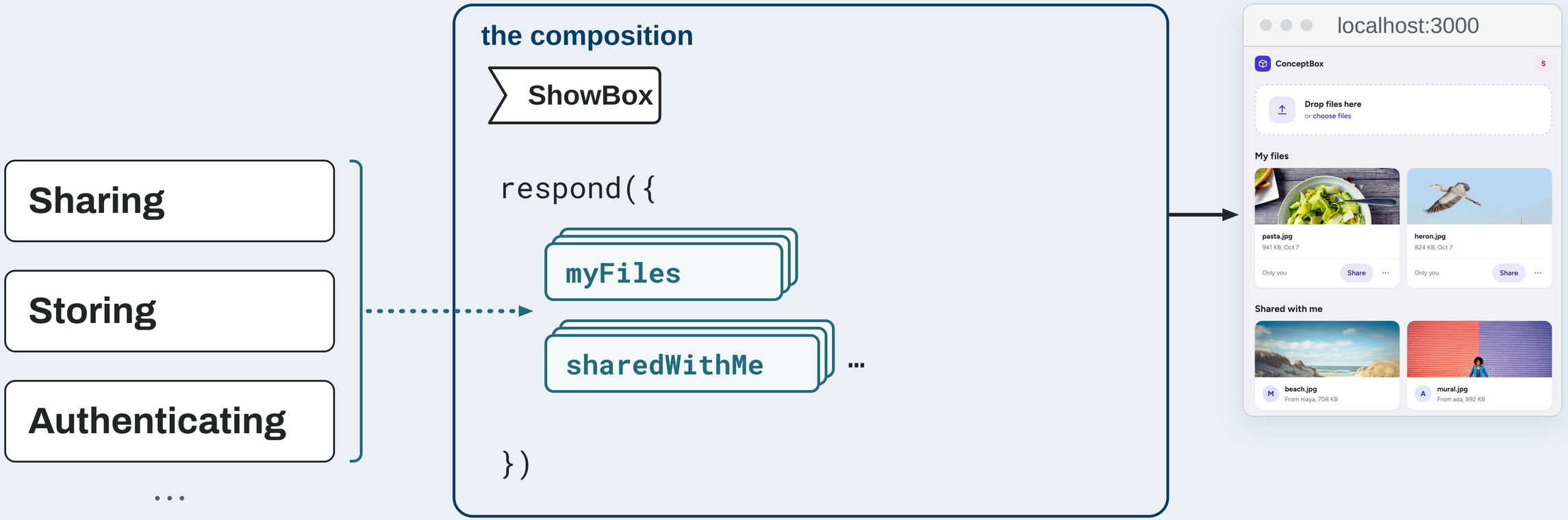



```
myFiles: [
  { file: "f7c2…", name: "beach.jpg", …,
    sharedWith: [
      { recipient: "u5a2…", username: "sam" 1 },
      { recipient: "u9c1…", username: "ada" 1 }
    ] } ]
```

src/compositions/box.ts, the former myFiles

```
10  const myFiles = former(
    …
21          sharedWith: each(Sharing._recipients({ item: file }).is({ recipient }))
22            .where(Authenticating._username({ user: recipient }).is({ username }))
23            .form({ recipient, username }),
24        }),
```

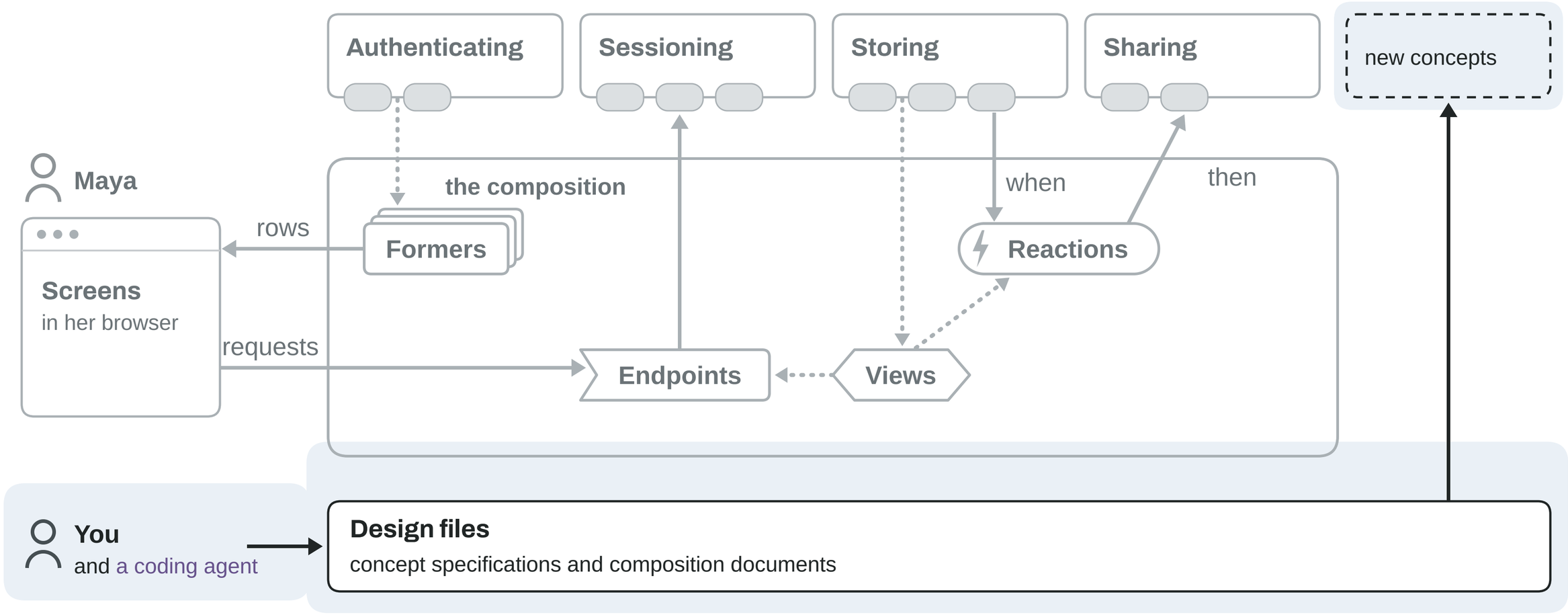
Build each screen’s data in a former, so no concept changes for a screen.



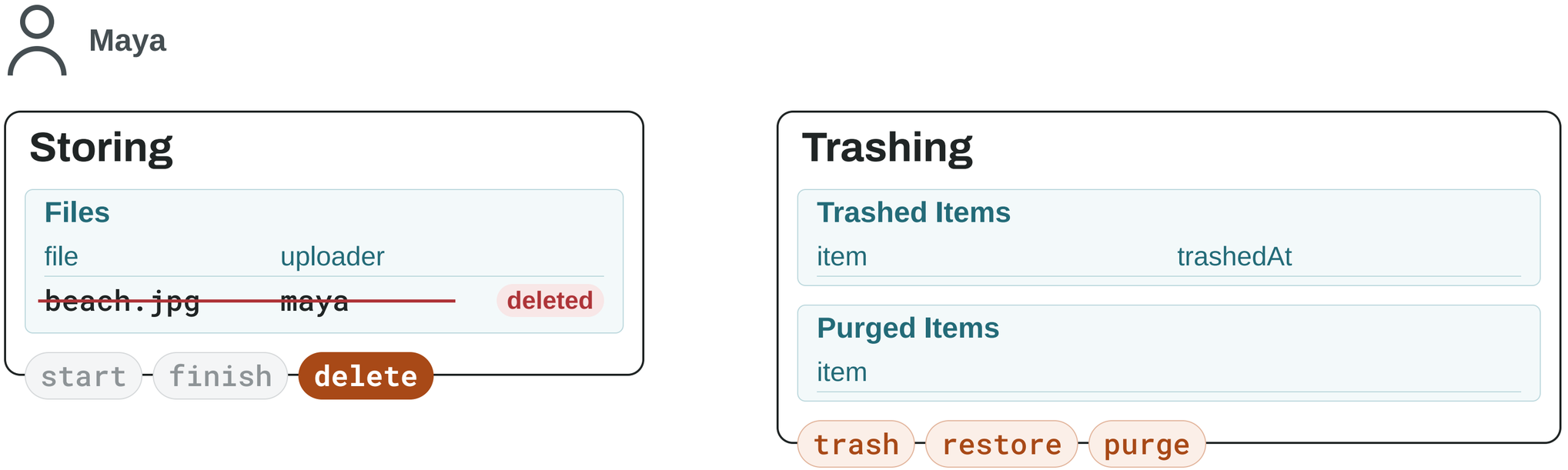
Part 7

Changing the app

How do you add a feature?



Trashing is added. What do you do next?



design/concepts/Trashing.md
src/concepts/Trashing/Trashing.ts
tests/concepts/Trashing.test.ts

Adding the trash: a new concept, new pieces, and a few changed lines

Authenticating
no change

Sessioning
no change

Storing
no change

Sharing
no change

new
Trashing
+66

new
Trash.vue
+120

the composition

new
- MoveToTrash +16
- Restore +16
- Purge +19
- RecordPurge +5
- myTrash +6

changed
- canRead +1
- myFiles +2 −2
- ShowBox +1 −1

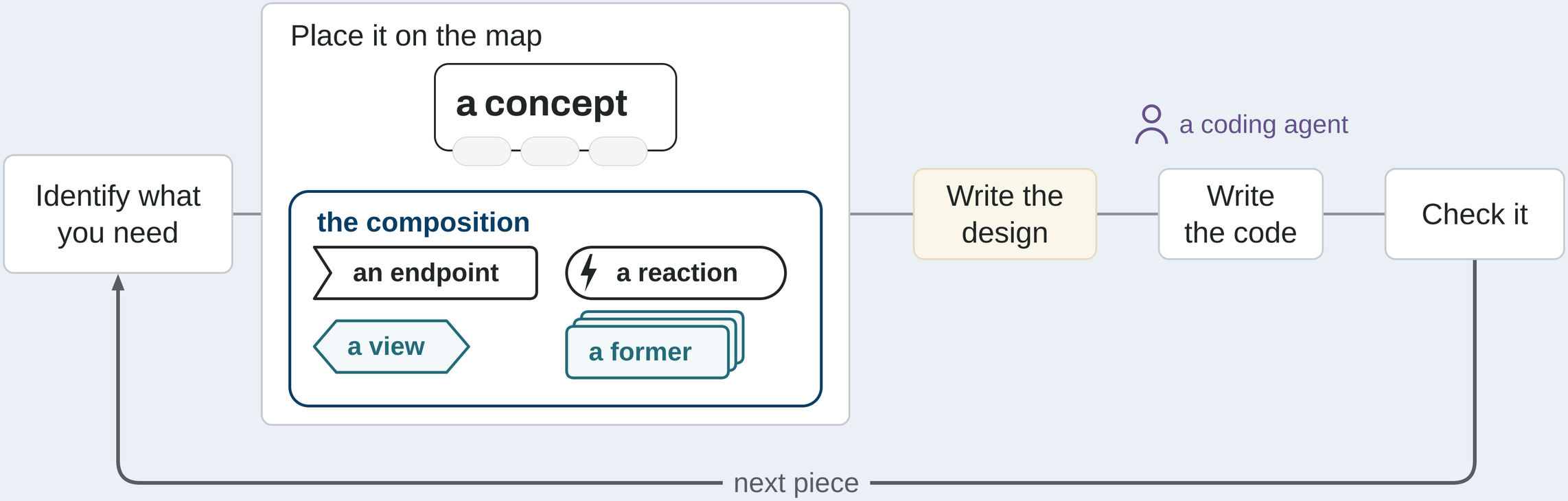
removed
- ~~Delete~~ −16

Changing the app

One piece at a time

Identify what you need	Place it on the map	Write the design	Write the code (a coding agent)	Check it
Take back a delete	**Trashing**	`Trashing.md`	`Trashing.ts`	✓ 7 tests
Move a file to trash	the composition: **MoveToTrash**	`trash.md`	`trash.ts`	✓
See the trash	**myTrash**	`trash.md`	`trash.ts, Trash.vue`	✓
Get a file back	**Restore**	`trash.md`	`trash.ts`	✓
Delete it for good	**Purge**, **RecordPurge**	`trash.md`	`trash.ts`	✓
Others can't read it	**canRead**	`access.md`	`access.ts`	✓

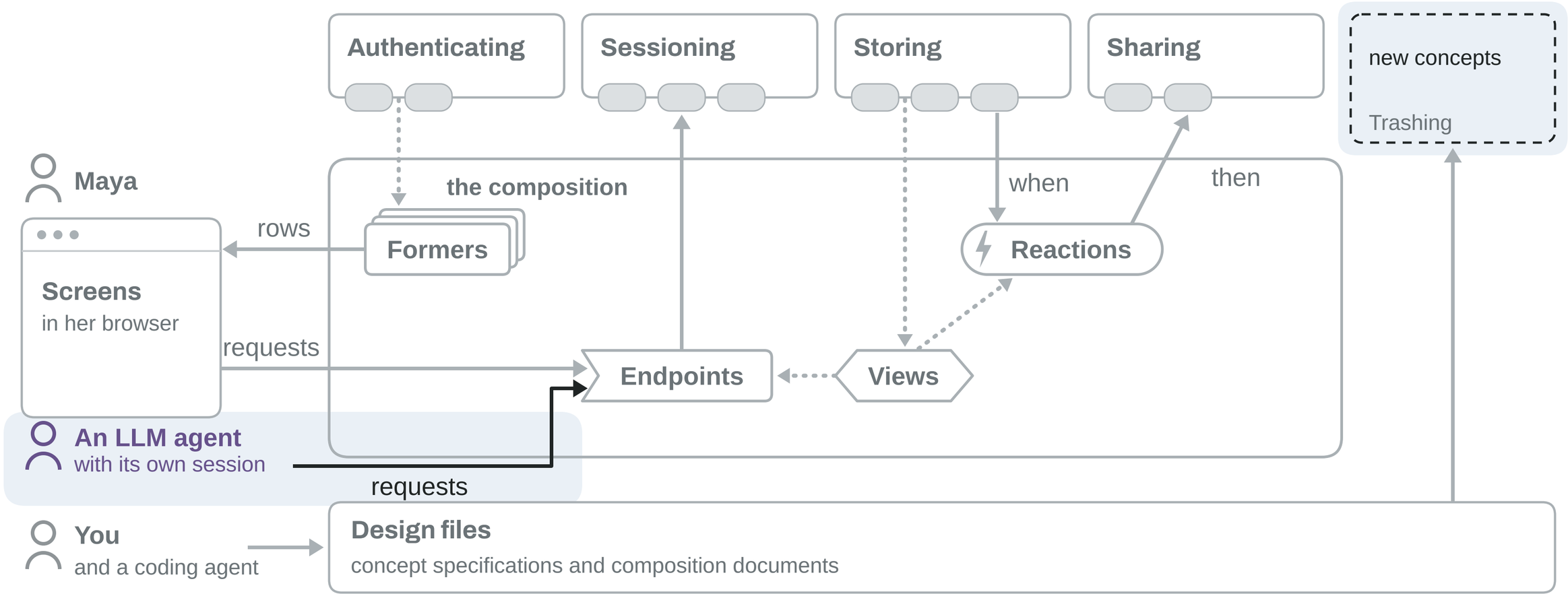
Add a feature one piece at a time: a concept written and tested alone, then each piece connected and checked.



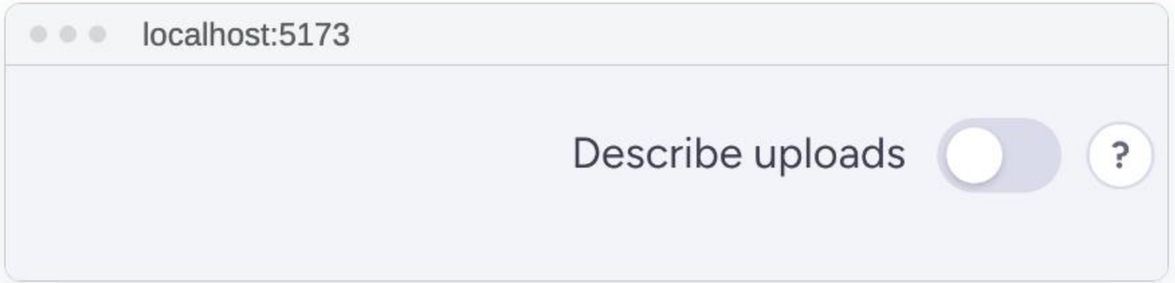
Part 8

Adding AI to your app

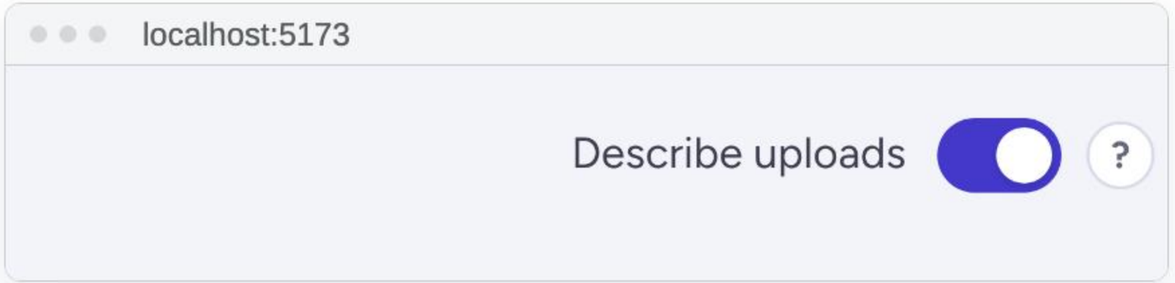
How do you add AI to your app?



Adding AI to ConceptBox: describing uploads

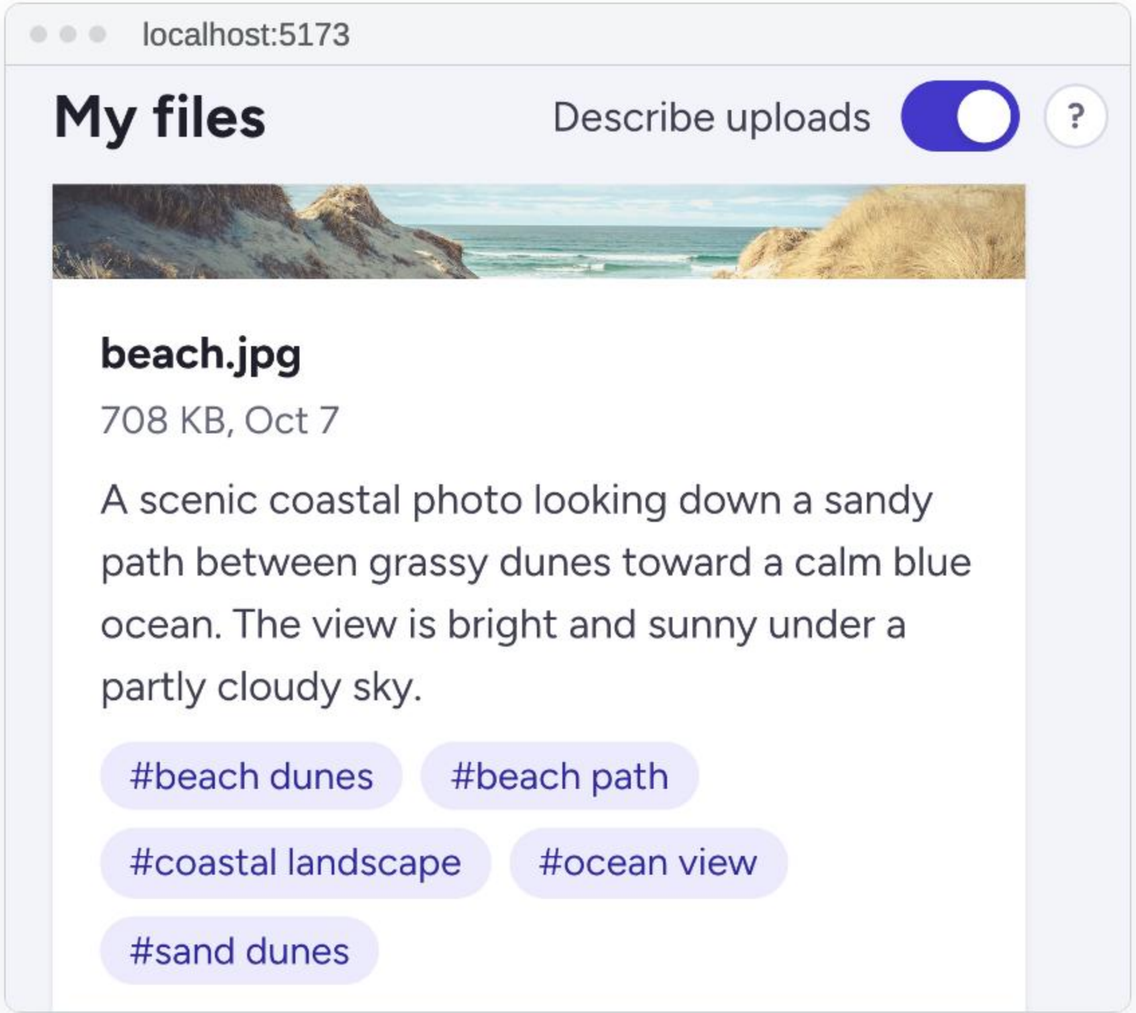


Adding AI to ConceptBox: describing uploads

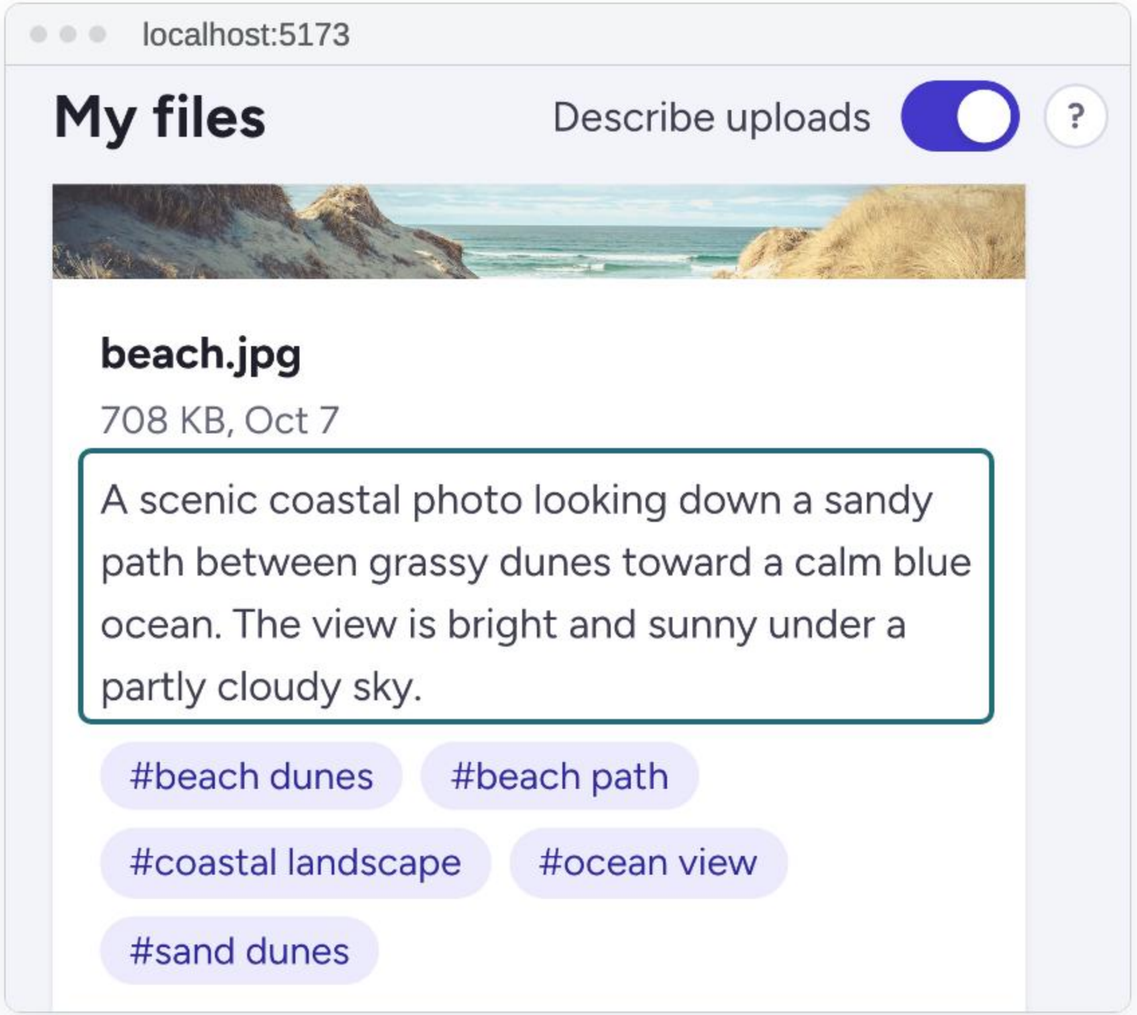


`Consenting.consent`

Adding AI to ConceptBox: describing uploads

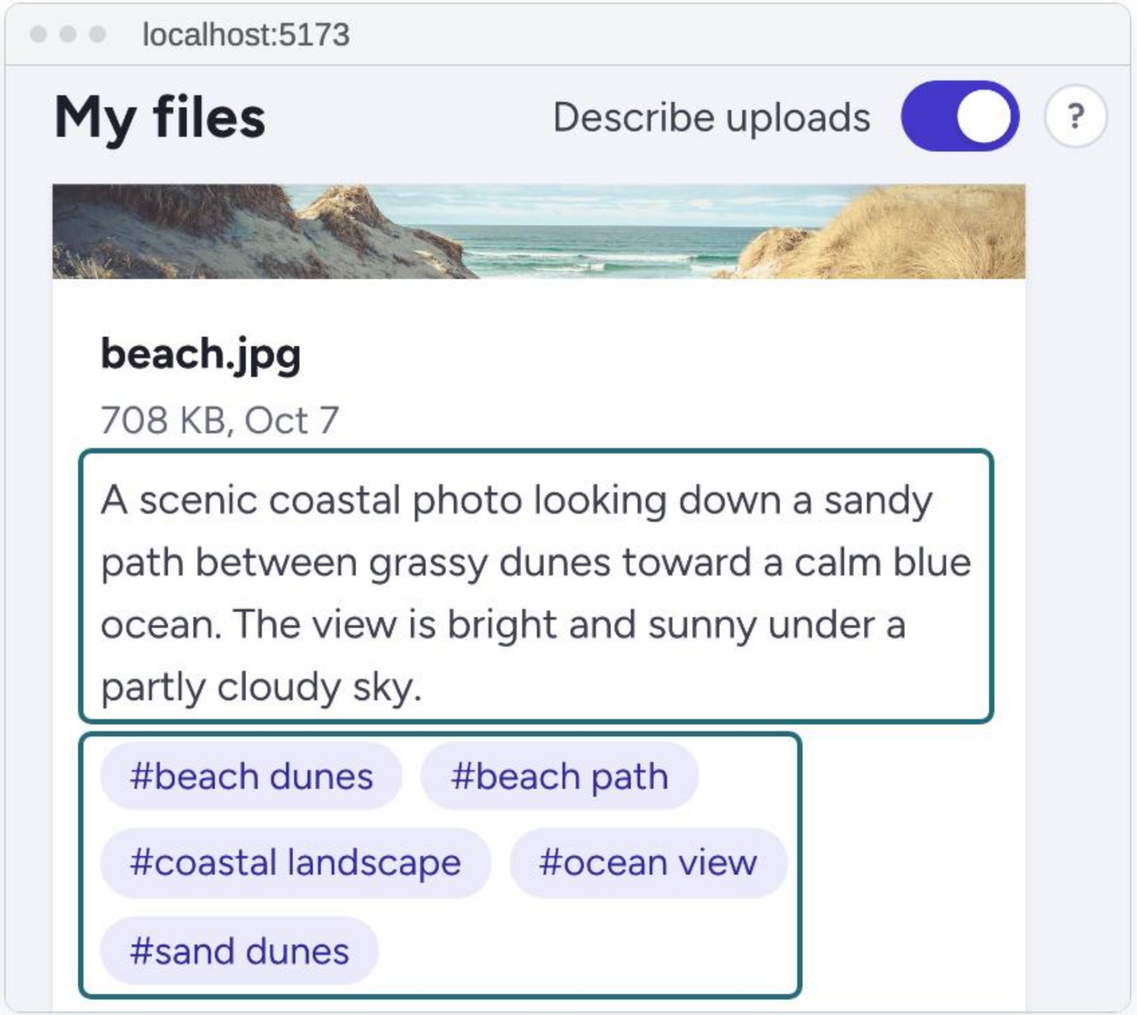


Adding AI to ConceptBox: describing uploads



Who wrote the description?

Adding AI to ConceptBox: describing uploads



Who wrote the description?

Who called `Labeling.label`?

A question that takes time to answer



Each prompt is a question about a subject, and its answer so far.

A question that takes time to answer

Reasoning

Prompts

subject	ask	reasoner	stage
beach.jpg	"Describe this file in one or two short sentences…"		WAITING
answer	""		
beach.jpg	"Suggest up to five short labels…"		WAITING
answer	""		

`prompt` `take` `extend` `conclude` `abandon`

`prompt` runs once per question. Each answer starts empty, and nobody has taken either yet.

A question that takes time to answer

Reasoning

Prompts

subject	ask	reasoner	stage
beach.jpg	"Describe this file in one or two short sentences…"	describer	FORMING
answer	""		
beach.jpg	"Suggest up to five short labels…"		WAITING
answer	""		

prompt — **take** — extend — conclude — abandon

take records the describer as the reasoner. No other reasoner can answer this prompt.

A question that takes time to answer

Reasoning

Prompts

subject	ask	reasoner	stage
beach.jpg	"Describe this file in one or two short sentences…"	describer	FORMING
answer	"A scenic coastal photo looking down a sandy path"		
beach.jpg	"Suggest up to five short labels…"		WAITING
answer	""		

prompt · take · **extend** · conclude · abandon

extend appends the first piece of the answer, as the model writes it.

A question that takes time to answer

Reasoning

Prompts

subject	ask	reasoner	stage
beach.jpg	"Describe this file in one or two short sentences…"	describer	FORMING
answer	"A scenic coastal photo looking down a sandy path between grassy dunes toward a calm blue ocean. …"		
beach.jpg	"Suggest up to five short labels…"		WAITING
answer	""		

prompt · take · **extend** · conclude · abandon

extend appends the next piece. The answer grows with each call.

A question that takes time to answer

Reasoning

Prompts

subject	ask	reasoner	stage
beach.jpg	"Describe this file in one or two short sentences…"	describer	CONCLUDED
answer	"A scenic coastal photo looking down a sandy path between grassy dunes toward a calm blue ocean. …"		
beach.jpg	"Suggest up to five short labels…"		WAITING
answer	""		

prompt – take – extend – **conclude** – abandon

conclude closes the prompt. The answer doesn't change after this.

A question that takes time to answer

Reasoning

Prompts

subject	ask	reasoner	stage
beach.jpg	"Describe this file in one or two short sentences…"	describer	CONCLUDED
answer	"A scenic coastal photo looking down a sandy path between grassy dunes toward a calm blue ocean. …"		
beach.jpg	"Suggest up to five short labels…"		WAITING
answer	""		

prompt - take - extend - conclude - abandon

former

```
describingOf({ file: "beach.jpg" })
```

localhost:5173

A scenic coastal photo looking down a sandy path between grassy dunes toward a calm blue ocean. The view is bright and sunny under a partly cloudy sky.

The describer is another actor

ask

Maya

endpoint

```
FinishUpload({ session: "Xq3f…",
  file: "beach.jpg" })
```

```
Storing.finish({ file: "beach.jpg",
  uploader: "maya" })
```

reaction

```
AskForLabels
```

```
Reasoning.prompt({ subject: "beach.jpg",
  ask: "Suggest up to five short labels…" })
```

answer

describer
an LLM agent

```
Take({ session: "Hk2v…",
  prompt: … })
```

```
Reasoning.take({ prompt: …,
  reasoner: "describer" })
```

```
Extend({ session: "Hk2v…",
  prompt: …, text: … })
```

```
Reasoning.extend({ prompt: …,
  reasoner: "describer", text: … })
```

```
Conclude({ session: "Hk2v…",
  prompt: … })
```

```
Reasoning.conclude({ prompt: …,
  reasoner: "describer" })
```

use the answer

```
LabelFromAnswer
```

```
Labeling.label({ item: "beach.jpg", name: … })
```

Applying the answer

generated/conceptbox.md, shortened

```
### describing.applying.LabelFromAnswer
…

when Reasoning.conclude (prompt,
    ask:
    "Suggest up to five short labels…",
    subject: file)

where
  Storing._get (file)
  Reasoning._lines (prompt)
      has (line, position)
  position is less than 5

then
  Labeling.label (item: file,
      name: line)
```

src/compositions/describing.ts

```
153 const LabelFromAnswer = reaction(
        ({ prompt, file, line, position }) =>

154   when(Reasoning.conclude({ prompt })
        .responds({ subject: file,
        ask: LABELS_ASK }))

155     .where(Storing._get({ file }),
          Reasoning._lines({ prompt })
          .is({ line, position }),
          is.lt(position, MOST_LABELS))

156     .then(Labeling.label({ item: file,
          name: line })),
```

only the label question

only the first five lines

An LLM agent is another user: a reaction asks, the agent answers, a reaction uses the answer.

ask

when

…

then

```
Reasoning.prompt({
  subject, ask })
```

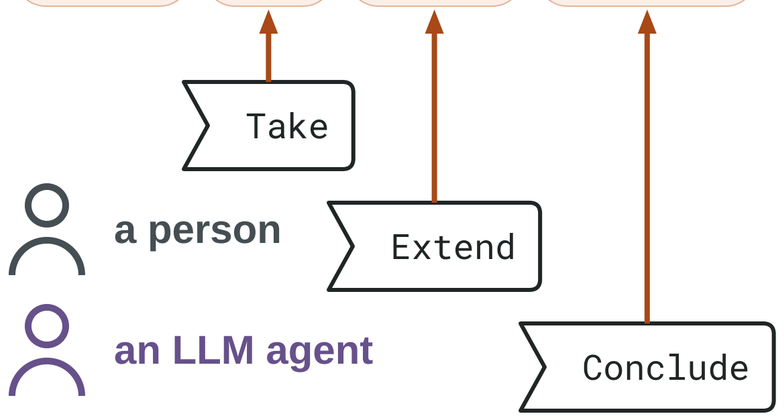
answer (reusable)

Reasoning

Prompts

subject	ask	reasoner	answer	stage

`prompt` `take` `extend` `conclude`



use the answer

when

```
Reasoning.conclude({ prompt })
  .responds({ subject, ask })
```

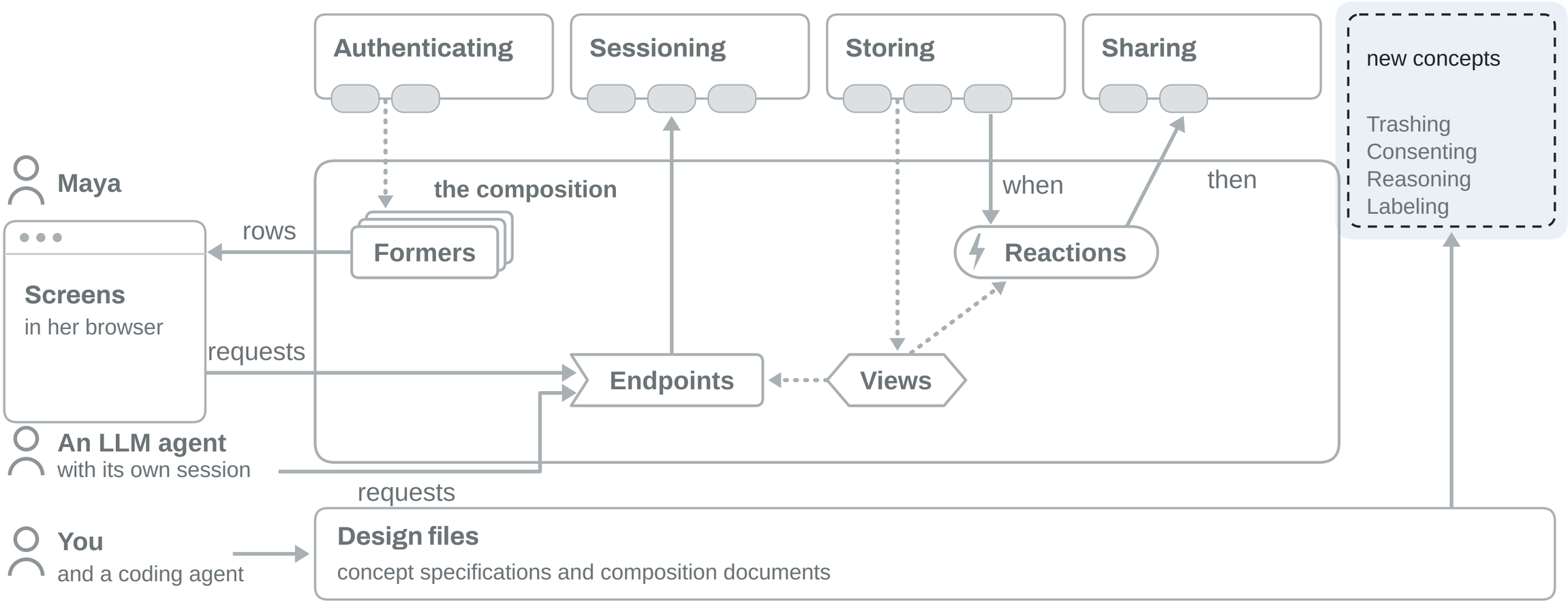
then

…

Part 9

Signing in

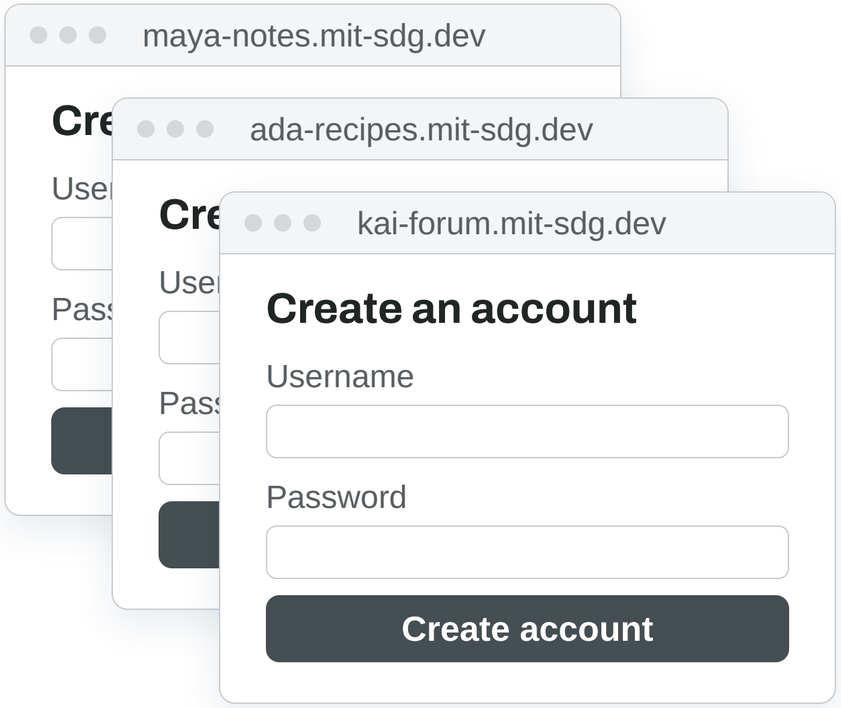
How do people get into your app?



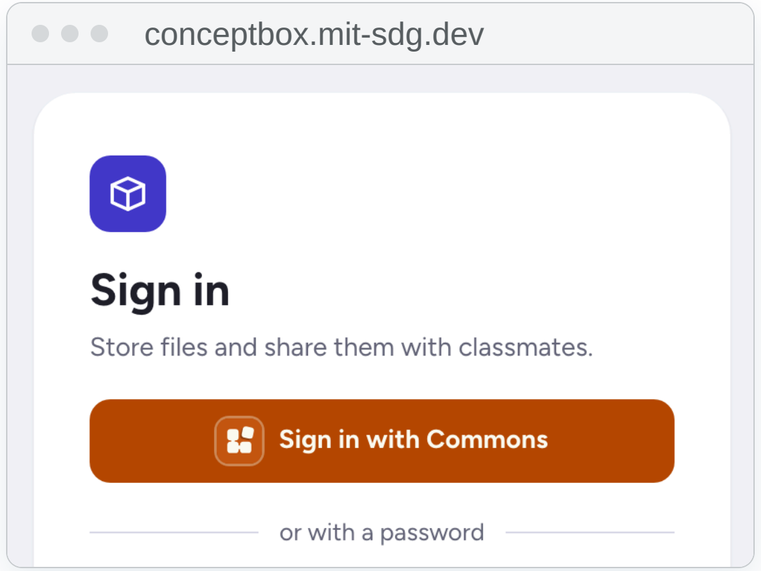
Signing in

An account for every app?

Sam



Sign in with Google



Signing in

Which concept stores the users?

before

Authenticating

Users

user	username	verifier
e21b	ada	••••
3c7d	maya	••••

`register` `authenticate`

after

Registering

Users

user	username
e21b	ada
3c7d	maya
9a4c	sam

`register`

Authenticating

Users

user	verifier
e21b	••••
3c7d	••••

no row for `9a4c`

`set` `authenticate`

Federating

Links

subject	user
u7f3	9a4c

`start` `finish` `link`

Signing in

How does Sam sign in with Commons?

Sign in with Commons

```
CommonsFederating.start({})
  .responds({ address: "https://class.mit-sdg.dev/…" })
```

on Commons **Allow**

```
CommonsFederating.finish({ … }).responds({ username: "sam" })
```

returning

where

```
CommonsFederating._user({ … })
```

user `"9a4c"`

then

```
Sessioning.start({ subject: "9a4c" })
```

new

where

```
CommonsFederating._user({ … })
```

no row

then

```
Registering.register({ username: "sam" })
  .responds({ user: "9a4c" })
```

then

```
CommonsFederating.link({ …, user: "9a4c" })
```

then

```
Sessioning.start({ subject: "9a4c" })
```

Signing in

Passwords, Commons, or both?

Registering

Sessioning

Authenticating

Federating

the composition

`accounts`

- Me
- SignOut

`passwords`

- Register
- SignIn

`commons`

- StartCommons
- FinishCommons
- ChooseUsername

passwords only
tests/compositions/signing-in-modules.test.ts

```
composition: { reusable: { accounts, passwords } },
```

Commons only
tests/compositions/signing-in-modules.test.ts

```
composition: { reusable: { accounts, commons } },
```

both, as ConceptBox
src/application.ts

```
reusable: { accounts, passwords, commons, answering },
```

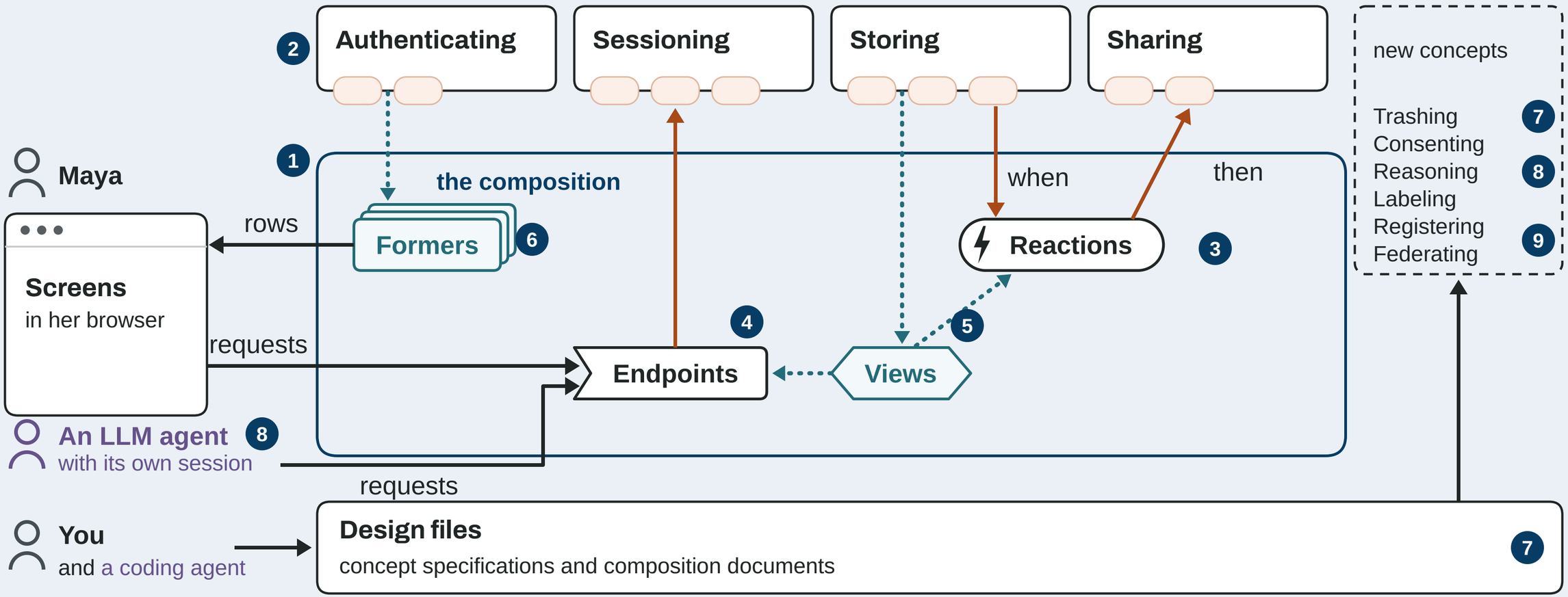
Keep who a user is in one concept; each way to sign in is one more concept and one reusable piece.

who a user is	a password	Google	Commons
Registering	**Authenticating**	**Federating**	**Federating**

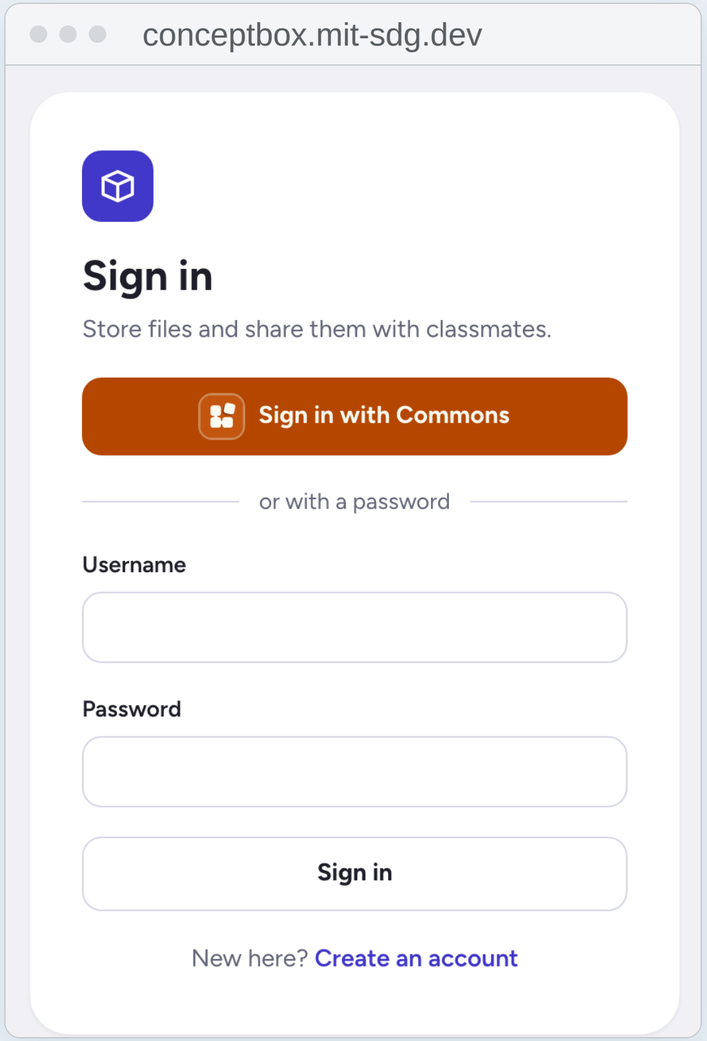
the composition

`accounts`	`passwords`	`google`	`commons`

ConceptBox, part by part



Try ConceptBox, and deploy your own app next week



try it

`conceptbox.mit-sdg.dev`

copy from it

`github.com/mit-sdg/conceptbox`

What address will your app have?

claim it

`_____.mit-sdg.dev`

sign in to each other's apps

Sign in with Commons

post what you want on the forum

`class.mit-sdg.dev`

Thank you, Barish: the original agent



Barish translated all our designs to code, and is the gold standard for agents.

`barish.me`