



Bonjour Lambda Jam!

Je commence

Functional CRUD

Using 'bureaucracy' to
tame a full-stack
Clojure/ClojureScript app

Sam Roberton – @sroberton
github.com/samroberton/bureaucracy

Testing

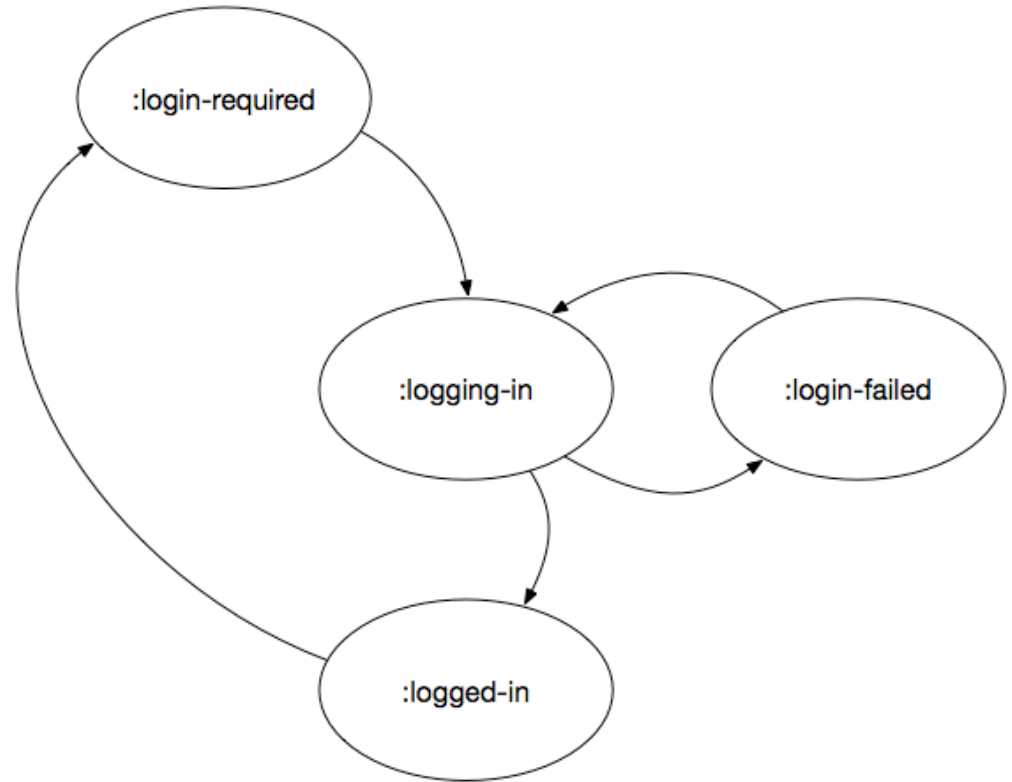
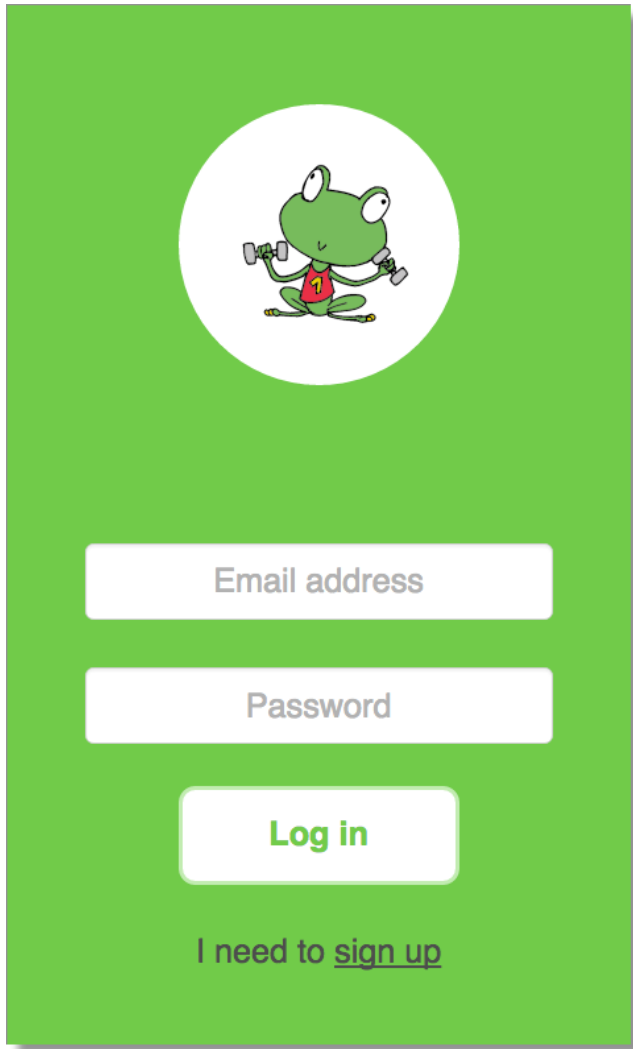


Rich UIs & JavaScript

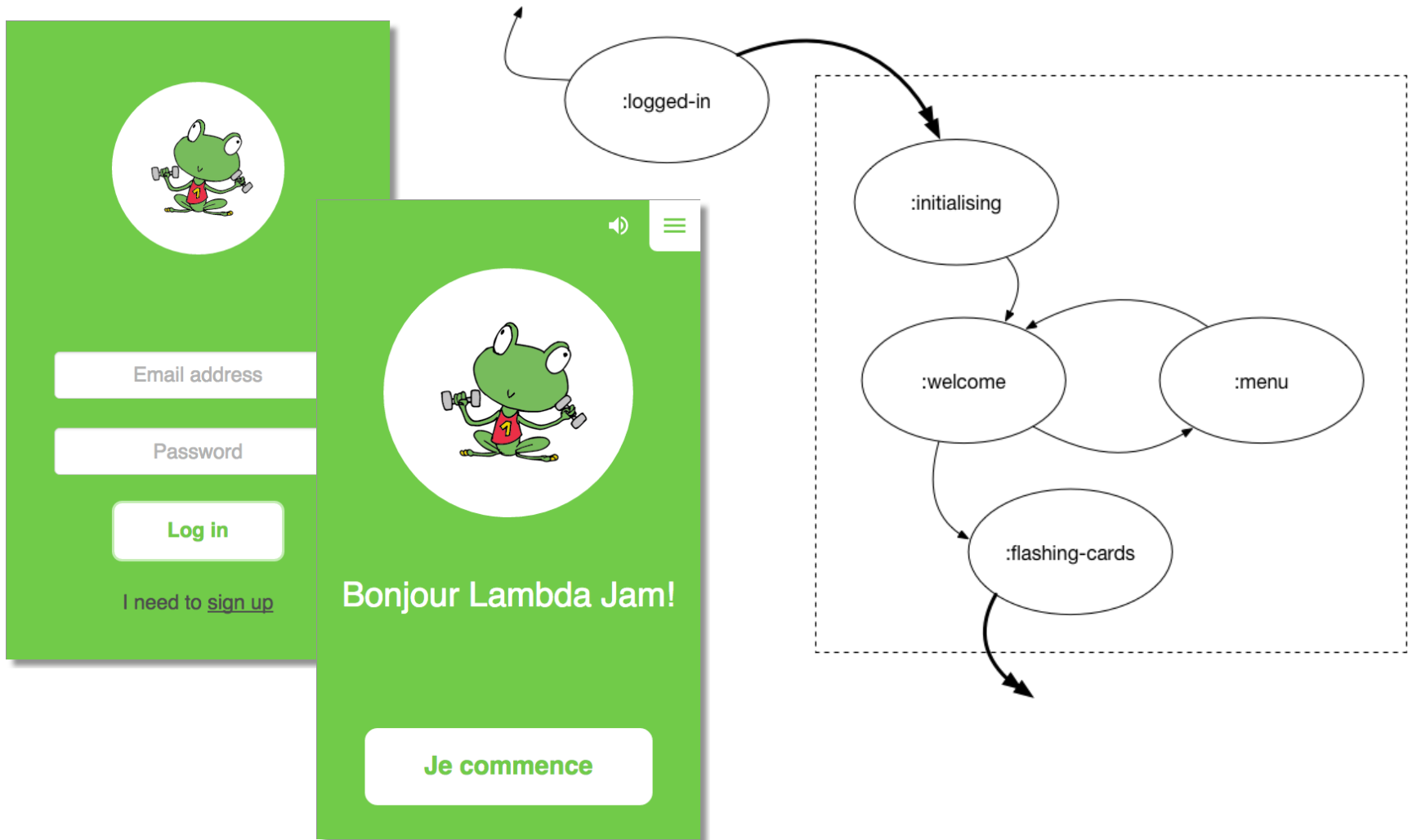


State Machines

State machines



Composeable state machines



Composeable state machines

Exercises

CECR level School level Category Card type Lesson # Exercise name

Level A1-1
Level A1-2
Level A2-1
Level A2-2
Level A2-3

585 Pronoms COD - COI (Audio QCM)
584 De plus en plus (Audio PE)
583 Il y a / depuis (Audio QCM)
582 Ne ... que, ne ... plus, non plus (Aud...
581 Pronom EN (Audio PE)
580 Pronom EN (Audio QCM)
579 Préciser la quantité (Audio PE)
578 Préciser la quantité
529 Lexique - Exprimer des règles (Ima...
528 Lexique - Exprimer des règles (Ima...
527 Phonie : [u] et [w]
526 Phonie : [u] et [w] (Répétez)
525 Phonie : [u] et [w]
524 Lexique - Habitats alternatifs (Image..
523 Lexique - Habitats alternatifs (Image..
522 Lexique - Annonces immobilières
521 Lexique - Transformations maison (...
520 Lexique - Transformations maison (...
519 Lexique - Meubles (Images)
518 Lexique - Meubles (Images)
517 Lexique - Pièces de la maison (Ima...
516 Lexique - Pièces de la maison (Ima...
515 Avant / Maintenant - Imparfait / Prés...
514 Plus - Prononciation

Exercise 527

Phonie : [u] et [w]

CARD TYPE **MC audio -> text**

CECR LEVEL SCHOOL LEVEL

LESSON REFERENCE

CATEGORY

NAME

INSTRUCTIONS

TAGS

Card #1

QUESTION huit, lui

CORRECT Identique

INCORRECT Différent

Card #2

QUESTION trois, bois

CORRECT Identique

Écoutez et sélectionnez la bonne réponse.

Identique Différent

How do we win?

- portable Clojure
 - test code which targets the browser alongside code which targets the server
- *complete* separation of view from state and behaviour
 - view is a *very simple* pure function of state
 - view effects change by dispatching events with minimal information:
 - (dispatch :update :username "sam")**
 - (dispatch :submit)**
 - behavioural tests don't need the view



Sélectionnez la bonne réponse.

Bonjour ! Comment ça va ?

Ça va bien,
merci !

Je m'appelle Bob

J'adore le
chocolat !

Je déteste la
végémite !



Réessayer

Voir la
réponse



Model

```
{:state :flashing-cards  
  :username "sam"  
  :nickname "Sam"  
  :card {:question "Bonjour, comment ça va?"  
    :right-answer "Ça va bien, merci!"  
    :wrong-answers ["Je m'appelle Bob"  
      "Je suis australien"  
      ...]}  
  :incorrect-attempts ["Je m'appelle Bob"]  
  :remaining-cards [{...}  
    {...}]}
```


Controller

```
(defmachine state-machine
  {:start      :question
   :transitions
   {:question  {::right-answer :correct
                ::wrong-answer :incorrect
                ::submit-answer [#{:correct :incorrect}
                                   submit-answer-next-state]
                ::skip-card    :skipping
                ::finish-session :done}
    :correct   {::next-card   [#{:question :done}
                               next-card-next-state]}
    :incorrect {::show-answer  :show-answers
                ::try-again   :question
                ::skip-card    :skipping
                ::finish-session :done}
    :skipping  {::next-card   [#{:question :done}
                               next-card-next-state]}
    :show-answers {::right-answer :correct
                  ::submit-answer [#{:correct :incorrect}
                                   submit-answer-next-state]
                  ::skip-card    :skipping
                  ::finish-session :done}
    :done      {}}
  :transition-fn transition})
```

Controller (cont)

```
(defmulti transition (fn [db event] (:id event)))
```

```
(defmethod transition ::right-answer  
  [{:keys [state-db] :as db} _]  
  (-> db  
    (update-in [:state-db :stats  
               (-> (:card-and-exercise state-db)  
                   :exercise  
                   :category)  
               (if (:incorrect-attempts state-db)  
                   :incorrect  
                   :correct)]  
              (fn nil inc 0))  
    (update :state-db  
            dissoc  
            :student-answer  
            :incorrect-attempts  
            :has-shown-answer?)))
```

View

```
(defn [{:keys [dispatcher]} _ model]
  [:div
   [:span (:instructions model)]
   [:span (:question-text model)]
   [:button {:on-click (dispatcher :right-answer)}
    (:right-answer-text model)]
   [:button {:on-click (dispatcher :wrong-answer)}
    (first (:wrong-answer-text model))]
   [:button {:on-click (dispatcher :wrong-answer)}
    (second (:wrong-answer-text model))]])
```

How do we win?

- test behaviour

```
(let [system (...)]  
  (input! system :update :username "sam")  
  (input! system :update :password "pass")  
  (input! system :submit)  
  (is (= :logged-in  
        (current-state system []))))
```

- “test” views
 - call render code, but without viewing result
 - Devcards

Devcards x

localhost:3449/devcards#!/nanny.web_client.devcards.cards

devcards / nanny.web_client.devcards.cards

mc-fill-the-blank-card-grammar

Here are the instructions

Le boulanger fait du pain

Right Wrong 1

Wrong 2 Wrong 3

Here are the instructions

Le

Right Wrong 1

Wrong 2 Wrong 3

Réessayer Voir la réponse

Here are the instructions

Le boulanger fait du pain

Right Wrong 1

Wrong 2 Wrong 3

Here are the instructions

Le

Passer cette carte

Finir la session

Autoriser audio

Right Wrong 1

Wrong 2 Wrong 3

mc-fill-the-blank-card-vocabulary

Here are the instructions

Le boulanger fait du pain

Right Wrong 1

Wrong 2 Wrong 3

Here are the instructions

Le

Right Wrong 1

Wrong 2 Wrong 3

Réessayer Voir la réponse

Here are the instructions

Le boulanger fait du pain

Right Wrong 1

Wrong 2 Wrong 3

Here are the instructions

Le

Passer cette carte

Finir la session

Autoriser audio

Right Wrong 1

Wrong 2 Wrong 3

mc-fill-the-blank-card-phonetics

Here are the instructions

Le boulanger fait du pain

Right Wrong 1

Here are the instructions

Le

Right Wrong 1

Wrong 2 Wrong 3

Here are the instructions

Le boulanger fait du pain

Right Wrong 1

Wrong 2 Wrong 3

Here are the instructions

Le

Passer cette carte

Finir la session

Autoriser audio

Right Wrong 1

Wrong 2 Wrong 3

What are we testing?

- the whole app

```
(def db (atom {...}))  
(def state-machine ...)  
(def view-tree ...)
```

```
(add-watch db (view-renderer view-tree))
```

```
(defn init! []  
  (swap! db #(start state-machine % nil)))
```

- only **'db'** changes, in response to discrete inputs
 - it is a simple succession of values over time

Yay, FP is so easy! ...

- ... but real apps aren't pure functions of an initial DB + user inputs
 - real apps have AJAX calls
 - or local databases
 - or audio to play, or `js/setTimeout`, or ...
- we want to test our app's interactions with the real world, too
 - we need a way to model and reason about effects

Testing the real world?

- keep our state machines pure
 - state machine transitions are side-effect free
 - but they can produce an “output” data structure

```
:outputs [{:id :submit-login  
           :payload {:username “sam”  
                    :password “password”}]}
```

- in the app, this is an AJAX call
- in tests, we have options
 - maybe we assert its contents
 - maybe we give it to the server, but in-process

Why bother?

- why go to all this effort to make state machines so pure and theoretical?
 - easier to reason about
 - ***put the stuff we might get wrong (without noticing) where we can test it most easily***

Why? It's all good already, no?

The screenshot shows the 'Get Fluent' web application interface. The browser address bar indicates 'localhost:3449/teacher'. The navigation menu includes 'SESSIONS', 'EXERCISE LIBRARY', and 'SIGN OUT'. The left sidebar contains filters for 'Exercises' (CECR level, School level, Category, Card type, Lesson #, Exercise name) and a list of exercises categorized by level (A1-1, A1-2, A2-1, A2-2, A2-3). The main content area displays the 'Exercise 527' form, which is titled 'Phonie : [u] et [w]'. The form includes fields for 'CARD TYPE' (MC audio → text), 'CECR LEVEL' (A2), 'SCHOOL LEVEL' (A2-3), 'LESSON' (14), 'REFERENCE' (195), 'CATEGORY' (phonetics), 'NAME' (Phonie : [u] et [w]), 'INSTRUCTIONS' (Écoutez et sélectionnez la bonne réponse.), and 'TAGS' (Tags). Below the form are two cards: 'Card #1' and 'Card #2'. A red circle highlights a pencil icon on the top right of 'Card #1', with a red arrow pointing to it and the text 'DOESN'T WORK' in red. To the right of the cards are two mobile device mockups showing the exercise interface.

Exercise 527

Phonie : [u] et [w]

CARD TYPE MC audio → text

CECR LEVEL A2 SCHOOL LEVEL A2-3

LESSON 14 REFERENCE 195

CATEGORY phonetics

NAME Phonie : [u] et [w]

INSTRUCTIONS Écoutez et sélectionnez la bonne réponse.

TAGS Tags

Cancel Save exercise

DOESN'T WORK

Card #1

QUESTION huit, lui

CORRECT Identique

INCORRECT Différent

Card #2

QUESTION trois, bois

CORRECT Identique

Test client + server in-process

```
(with-rolled-back-db-tx [tx db-spec]
  (let [server (create-server tx)
        client (create-client
                  {:output-handler
                   (mock-ajax server)})]
    (input! client :update
              :username "sam")
    ...))
```

What do we get?

- ability to test as much or as little as we want
 - comprehensive system-level tests invoking a real server system with a live database
 - or unit-level tests of a single component with test-supplied (mocked) responses from server
- fast tests
 - no more extended light-saber fights while Selenium pokes at a browser
- succinct tests

What do we get? (bonus)

- a model of our system that matches the way we think and talk about it
 - “user enters username”
(input! system :update :username “sam”)
 - “user clicks submit”
(input! system :submit)
 - “user is logged in”
**(is (= :logged-in
 (current-state system [])))**
- loose coupling of tests to implementation

What else? (more bonus?)

- dispatcher tracking: know what inputs your view is capable of producing
 - report test coverage
 - random input generation / fuzzing
 - property-based testing?
- re-use “behaviour” for different views
 - React for the web
 - React Native for a native phone app

Merci !



Questions?

Sam Robertson – @sroberton
github.com/samroberton/bureaucracy