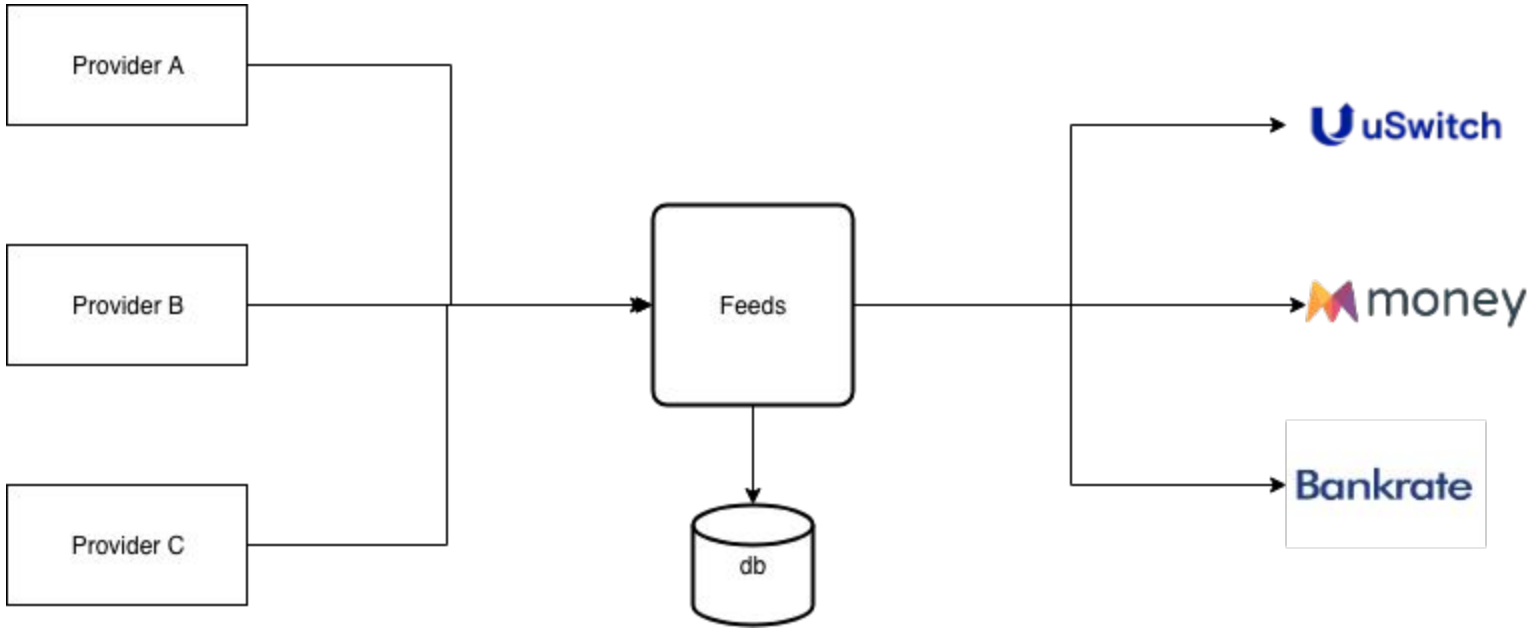
Property based testing for Stateful apps

Akshay Karle

1

Context

Automated Product Ingestion



uSwitch money Bankrate RV

Example based testing

```
(testing "should return a valid created event when mortgage doesn't exist already"
(let [raw-mortgage {:source_id 999
                    :name "Mortgage you would repay until death"
                    :mortgage_type "standard"
                    :provider_name "A Mortgage"
                    :provider_id 999
                    :apr 1.1}
      command {:type :create :source "Provider A" :domain :mortgage :data raw-mortgage}
      {:keys [state event]} (process command nil)]
  (is (= :created (:status event)))
  (is (= (-> command :data :name) (:name state)))))
```

uSwitch money Bankrate RV

2

Introduction of Specs & Generators

Adding specs

```
(s/def ::name string?)
(s/def ::source_id number?)
(s/def ::mortgage_type #{"standard" "fixed_rate" "discount" "equity_release"})
(s/def ::provider_id number?)
(s/def ::provider_name string?)
(s/def ::positive-number (and number? #(< 0 %) #(not= Double/POSITIVE_INFINITY %)))
(s/def ::apr ::positive-number)
(s/def ::mortgage (s/keys :req-un [::name
                                   ::source_id
                                   ::mortgage_type
                                   ::provider_id
                                   ::provider_name]
                          :opt-un [::apr]))

(s/def :command/type #{:create :update :publish :unpublish :archive})
(s/def :command/source string?)
(s/def :command/domain #{:mortgage})
(s/def :command/data ::mortgage)
(s/def ::command (specs/keys :req-un [:command/type :command/source :command/data :command/domain]))
```

Now using generators from the specs

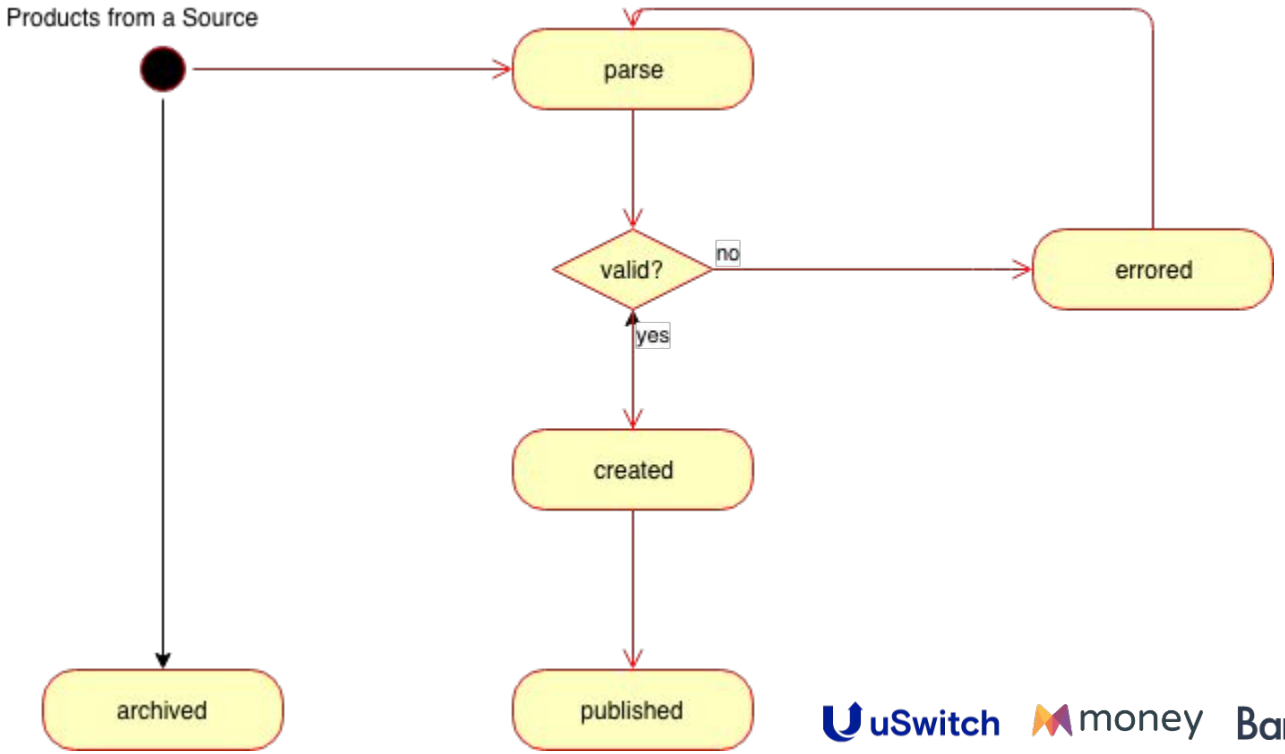
```
(def raw-mortgage-generator (s/gen ::mortgage))
(def raw-mortgage (gen/generate raw-mortgage-generator))
(def command-generator (s/gen ::command))

(defn- type-gen [types]
  (gen/generate (gen/elements types)))
(defn modify-command [raw type c]
  (merge c {:data raw :type type}))
(defn- gen-command-with [raw type]
  (gen/fmap (partial modify-command raw type) command-generator))
(defn- mortgage-commands-for [types]
  (gen-command-with raw-mortgage (type-gen types)))

(testing "should return a valid created event when mortgage doesn't exist already"
  (let [command (mortgage-commands-for [:create])
        {:keys [state event]} (process command nil)]
    (is (= :created (:status event)))
    (is (= (-> command :data :name) (:name state)))))
```

uSwitch money Bankrate RV

3

State machine

Different product transitions



uSwitch money Bankrate RV

Property tests for the product state transitions

```
(defspec create-new-mortgages
  100
  (prop/for-all
   [command (mortgage-commands-for [:create])]
   (let [{:keys [state event]} (process command nil)]
     (is (= :created (:status event)))
     (is (= (-> command :data :name) (:name state))))))

(defspec error-always-changes-status
  100
  (prop/for-all
   [any-commands  gen/commands
    error-command (gen/gen-command-with gen/gen-bad-raw :update)]
   (let [all-results (process-all (conj any-commands error-command) gen/raw-mortgage)]
     (is (= (:status (:event (last all-results)))
            :errored))))
```

uSwitch money Bankrate RV

Summary

- Evolving data models
- Checking system boundaries
- Makes sense when building complex stateful apps
- Encourages thinking of general properties of the system

Thank you!

We are hiring!

github.com/akshaykarle
twitter.com/akshay_karle