Unlocking Data-driven Systems

Clojure/conj :: November 20, 2014

Paul deGrandis
@ohpauleez

The value of values

- “Data all the things”
- The spectrum of value-oriented systems
- Outcomes and conclusions
- What’s next?

Consumer Reports®

Consumer Reports

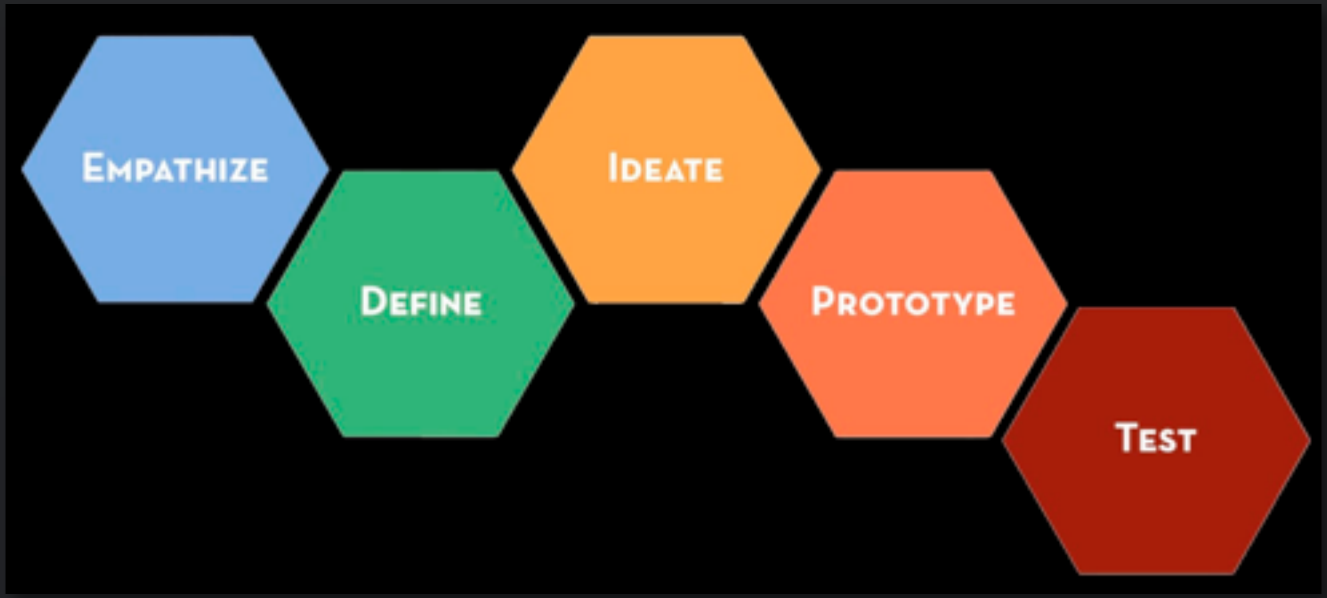
- Construct prototypes with no backend experience
- Free the data
- Produce in hours not days or weeks
- Embrace change

Consumer Reports

Consider the hypernym

Design and thinking

- What would you do?
- How might we ...?
- What do we mean when we say “service”?

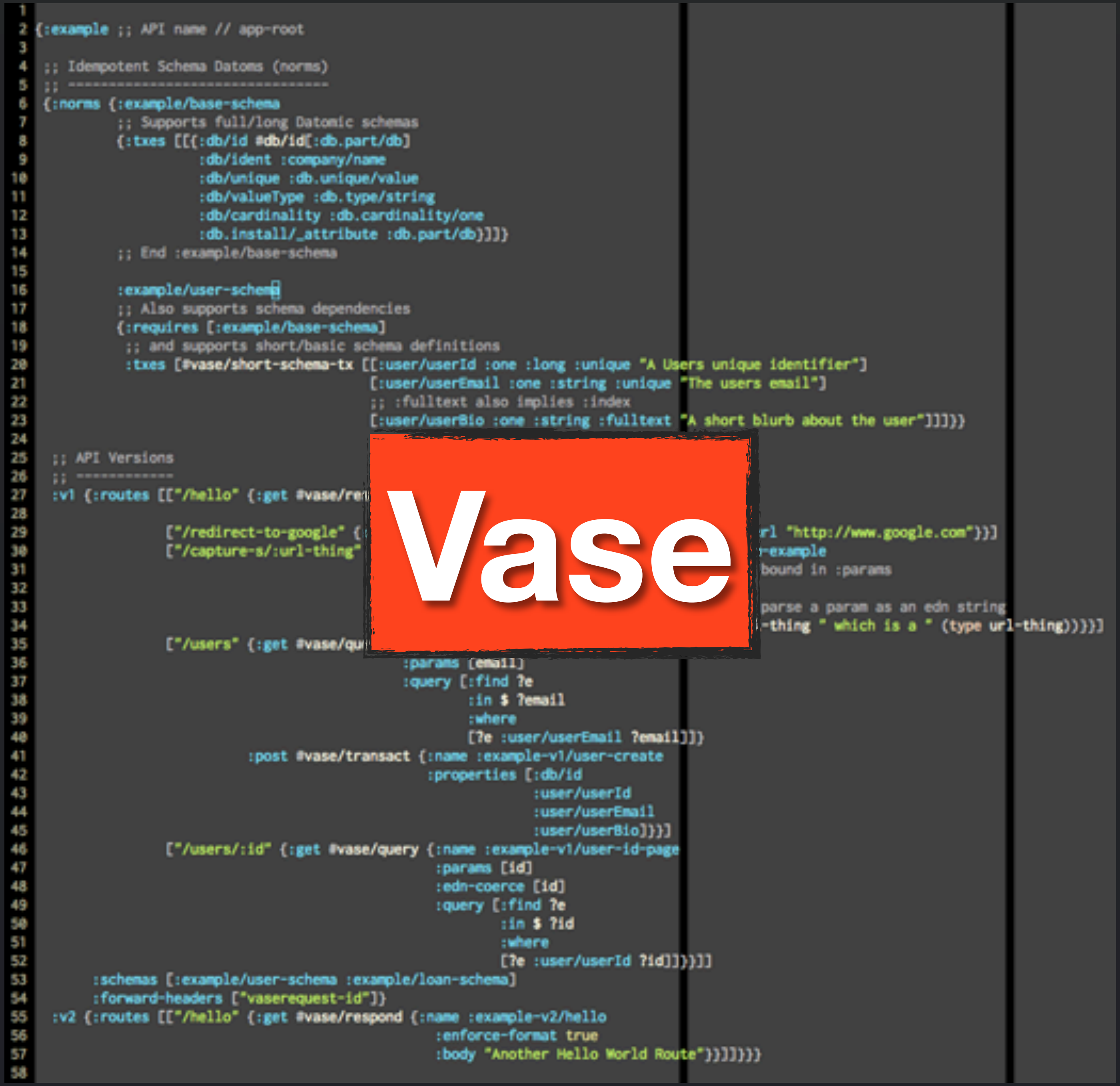


Standford dschool

Container service

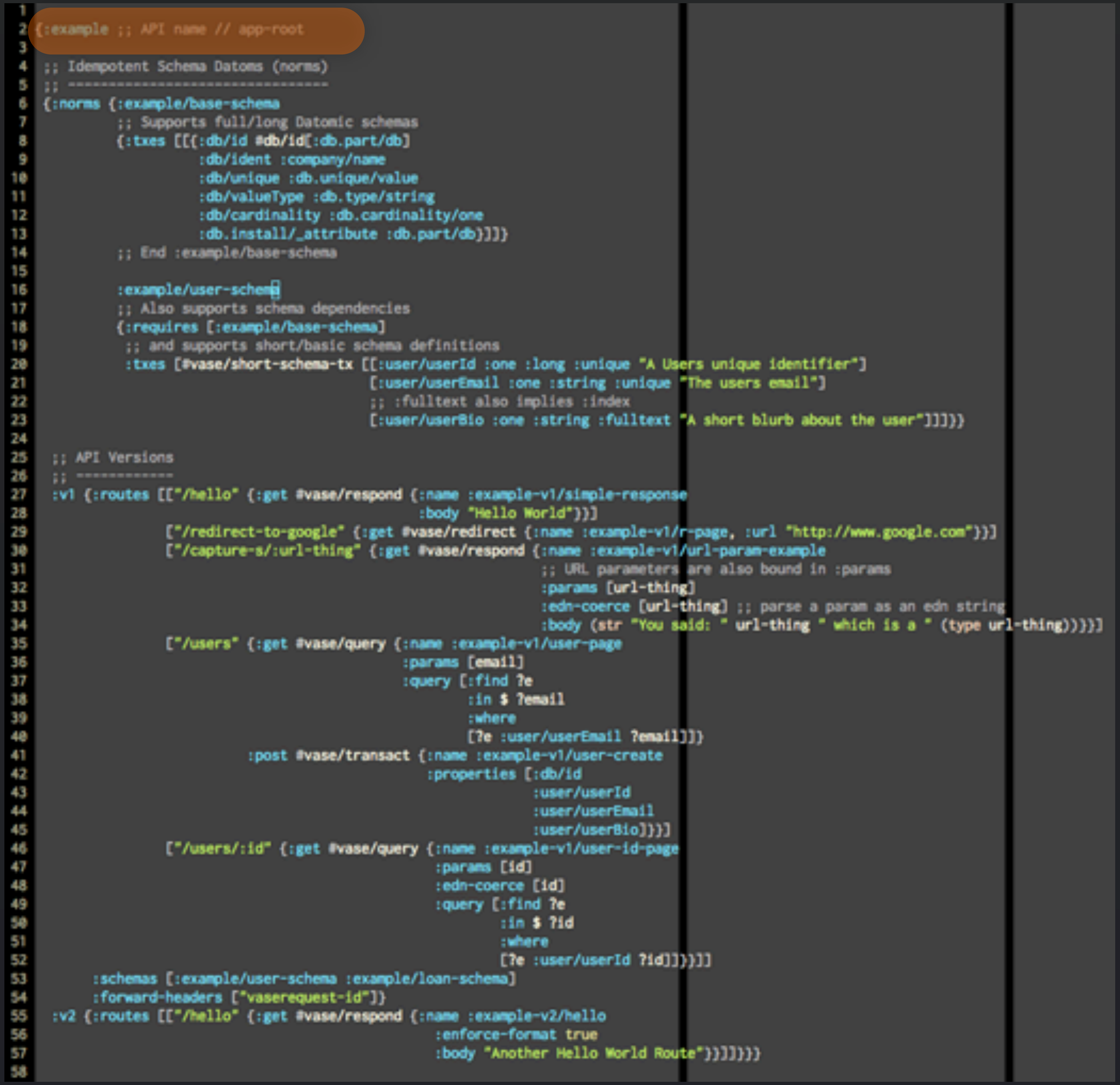
- Describe services using edn
- Contains schema, routes - all versioned
- Constrains services' actions

Container service



```
{:example ;; API name // app-root

 ;; Idempotent Schema Datoms (norms)
 ;; --------------------------------
 {:norms {:example/base-schema
          ;; Supports full/long Datomic schemas
          {:txes [[{:db/id #db/id[:db.part/db]
                    :db/ident :company/name
                    :db/unique :db.unique/value
                    :db/valueType :db.type/string
                    :db/cardinality :db.cardinality/one
                    :db.install/_attribute :db.part/db}]]}
          ;; End :example/base-schema

          :example/user-schema
          ;; Also supports schema dependencies
          {:requires [:example/base-schema]
           ;; and supports short/basic schema definitions
           :txes [#vase/short-schema-tx [[:user/userId :one :long :unique "A Users unique identifier"]
                                         [:user/userEmail :one :string :unique "The users email"]
                                         ;; :fulltext also implies :index
                                         [:user/userBio :one :string :fulltext "A short blurb about the user"]]]}}

  ;; API Versions
  ;; ------------
  :v1 {:routes [["/hello" {:get #vase/re

                ["/redirect-to-google" {                                                rl "http://www.google.com"}}]
                ["/capture-s/:url-thing"                                                -example
                                                                                         bound in :params

                                                                                         parse a param as an edn string
                                                                                         -thing " which is a " (type url-thing))}}]
                ["/users" {:get #vase/qu
                                             :params [email]
                                             :query [:find ?e
                                                     :in $ ?email
                                                     :where
                                                     [?e :user/userEmail ?email]]}
                          :post #vase/transact {:name :example-v1/user-create
                                                :properties [:db/id
                                                             :user/userId
                                                             :user/userEmail
                                                             :user/userBio]}}]
                ["/users/:id" {:get #vase/query {:name :example-v1/user-id-page
                                                 :params [id]
                                                 :edn-coerce [id]
                                                 :query [:find ?e
                                                         :in $ ?id
                                                         :where
                                                         [?e :user/userId ?id]]}}]]
       :schemas [:example/user-schema :example/loan-schema]
       :forward-headers ["vaserequest-id"]}
  :v2 {:routes [["/hello" {:get #vase/respond {:name :example-v2/hello
                                                 :enforce-format true
                                                 :body "Another Hello World Route"}}]]}}}
```

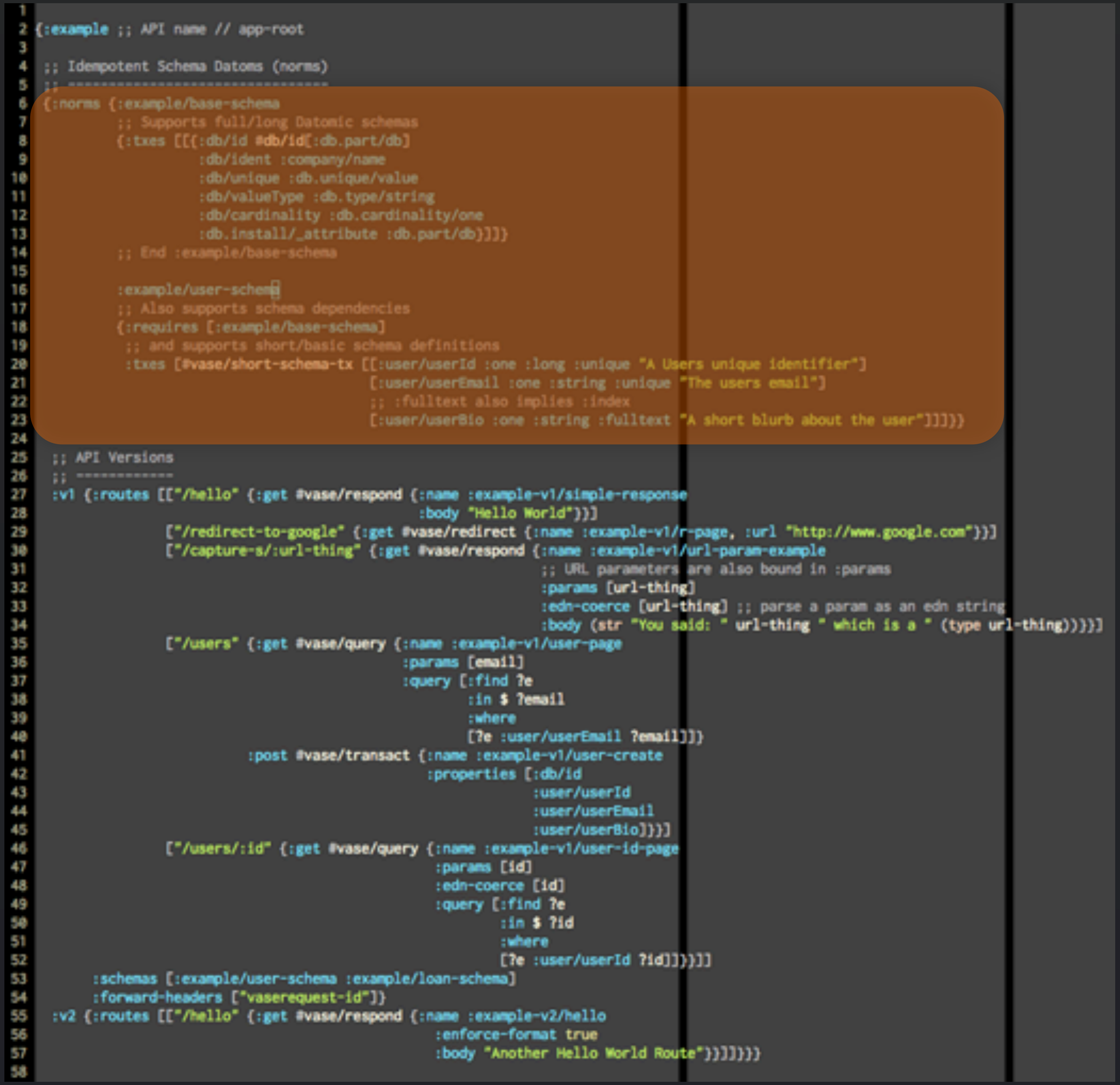
Container service



```
{:example ;; API name // app-root

 ;; Idempotent Schema Datoms (norms)
 ;; --------------------------------
 {:norms {:example/base-schema
          ;; Supports full/long Datomic schemas
          {:txes [[{:db/id #db/id[:db.part/db]
                    :db/ident :company/name
                    :db/unique :db.unique/value
                    :db/valueType :db.type/string
                    :db/cardinality :db.cardinality/one
                    :db.install/_attribute :db.part/db}]]}
          ;; End :example/base-schema

          :example/user-schema
          ;; Also supports schema dependencies
          {:requires [:example/base-schema]
           ;; and supports short/basic schema definitions
           :txes [#vase/short-schema-tx [[:user/userId :one :long :unique "A Users unique identifier"]
                                         [:user/userEmail :one :string :unique "The users email"]
                                         ;; :fulltext also implies :index
                                         [:user/userBio :one :string :fulltext "A short blurb about the user"]]]}}

  ;; API Versions
  ;; ------------
  :v1 {:routes [["/hello" {:get #vase/respond {:name :example-v1/simple-response
                                               :body "Hello World"}}]
                ["/redirect-to-google" {:get #vase/redirect {:name :example-v1/r-page, :url "http://www.google.com"}}]
                ["/capture-s/:url-thing" {:get #vase/respond {:name :example-v1/url-param-example
                                                              ;; URL parameters are also bound in :params
                                                              :params [url-thing]
                                                              :edn-coerce [url-thing] ;; parse a param as an edn string
                                                              :body (str "You said: " url-thing " which is a " (type url-thing))}}]
                ["/users" {:get #vase/query {:name :example-v1/user-page
                                             :params [email]
                                             :query [:find ?e
                                                     :in $ ?email
                                                     :where
                                                     [?e :user/userEmail ?email]]}
                           :post #vase/transact {:name :example-v1/user-create
                                                 :properties [:db/id
                                                              :user/userId
                                                              :user/userEmail
                                                              :user/userBio]}}]
                ["/users/:id" {:get #vase/query {:name :example-v1/user-id-page
                                                 :params [id]
                                                 :edn-coerce [id]
                                                 :query [:find ?e
                                                         :in $ ?id
                                                         :where
                                                         [?e :user/userId ?id]]}}]]
       :schemas [:example/user-schema :example/loan-schema]
       :forward-headers ["vaserequest-id"]}
  :v2 {:routes [["/hello" {:get #vase/respond {:name :example-v2/hello
                                               :enforce-format true
                                               :body "Another Hello World Route"}}]]}}
```

Service name

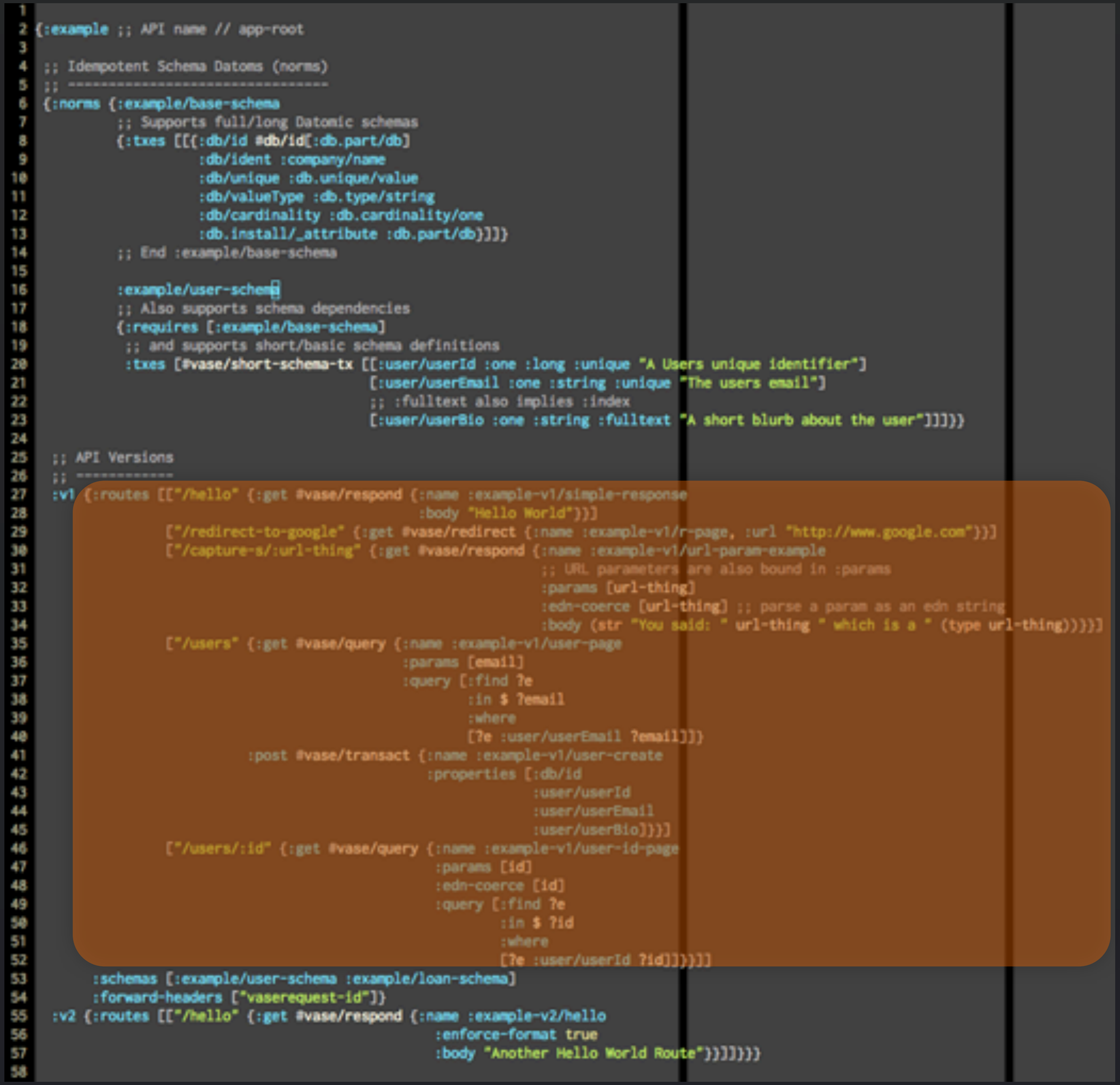
Container service



```
{:example ;; API name // app-root

 ;; Idempotent Schema Datoms (norms)
 ;; -------------------------------
 {:norms {:example/base-schema
          ;; Supports full/long Datomic schemas
          {:txes [[{:db/id #db/id[:db.part/db]
                    :db/ident :company/name
                    :db/unique :db.unique/value
                    :db/valueType :db.type/string
                    :db/cardinality :db.cardinality/one
                    :db.install/_attribute :db.part/db}]]}
          ;; End :example/base-schema

          :example/user-schema
          ;; Also supports schema dependencies
          {:requires [:example/base-schema]
           ;; and supports short/basic schema definitions
           :txes [#vase/short-schema-tx [[:user/userId :one :long :unique "A Users unique identifier"]
                                         [:user/userEmail :one :string :unique "The users email"]
                                         ;; :fulltext also implies :index
                                         [:user/userBio :one :string :fulltext "A short blurb about the user"]]]}}}

  ;; API Versions
  ;; ------------
  :v1 {:routes [["/hello" {:get #vase/respond {:name :example-v1/simple-response
                                               :body "Hello World"}}]
                ["/redirect-to-google" {:get #vase/redirect {:name :example-v1/r-page, :url "http://www.google.com"}}]
                ["/capture-s/:url-thing" {:get #vase/respond {:name :example-v1/url-param-example
                                                              ;; URL parameters are also bound in :params
                                                              :params [url-thing]
                                                              :edn-coerce [url-thing] ;; parse a param as an edn string
                                                              :body (str "You said: " url-thing " which is a " (type url-thing))}}]
                ["/users" {:get #vase/query {:name :example-v1/user-page
                                            :params [email]
                                            :query [:find ?e
                                                    :in $ ?email
                                                    :where
                                                    [?e :user/userEmail ?email]]}
                           :post #vase/transact {:name :example-v1/user-create
                                                 :properties [:db/id
                                                              :user/userId
                                                              :user/userEmail
                                                              :user/userBio]}}]
                ["/users/:id" {:get #vase/query {:name :example-v1/user-id-page
                                                 :params [id]
                                                 :edn-coerce [id]
                                                 :query [:find ?e
                                                         :in $ ?id
                                                         :where
                                                         [?e :user/userId ?id]]}}]]
       :schemas [:example/user-schema :example/loan-schema]
       :forward-headers ["vaserequest-id"]}
  :v2 {:routes [["/hello" {:get #vase/respond {:name :example-v2/hello
                                                :enforce-format true
                                                :body "Another Hello World Route"}}]]}}}
```

Schema definition

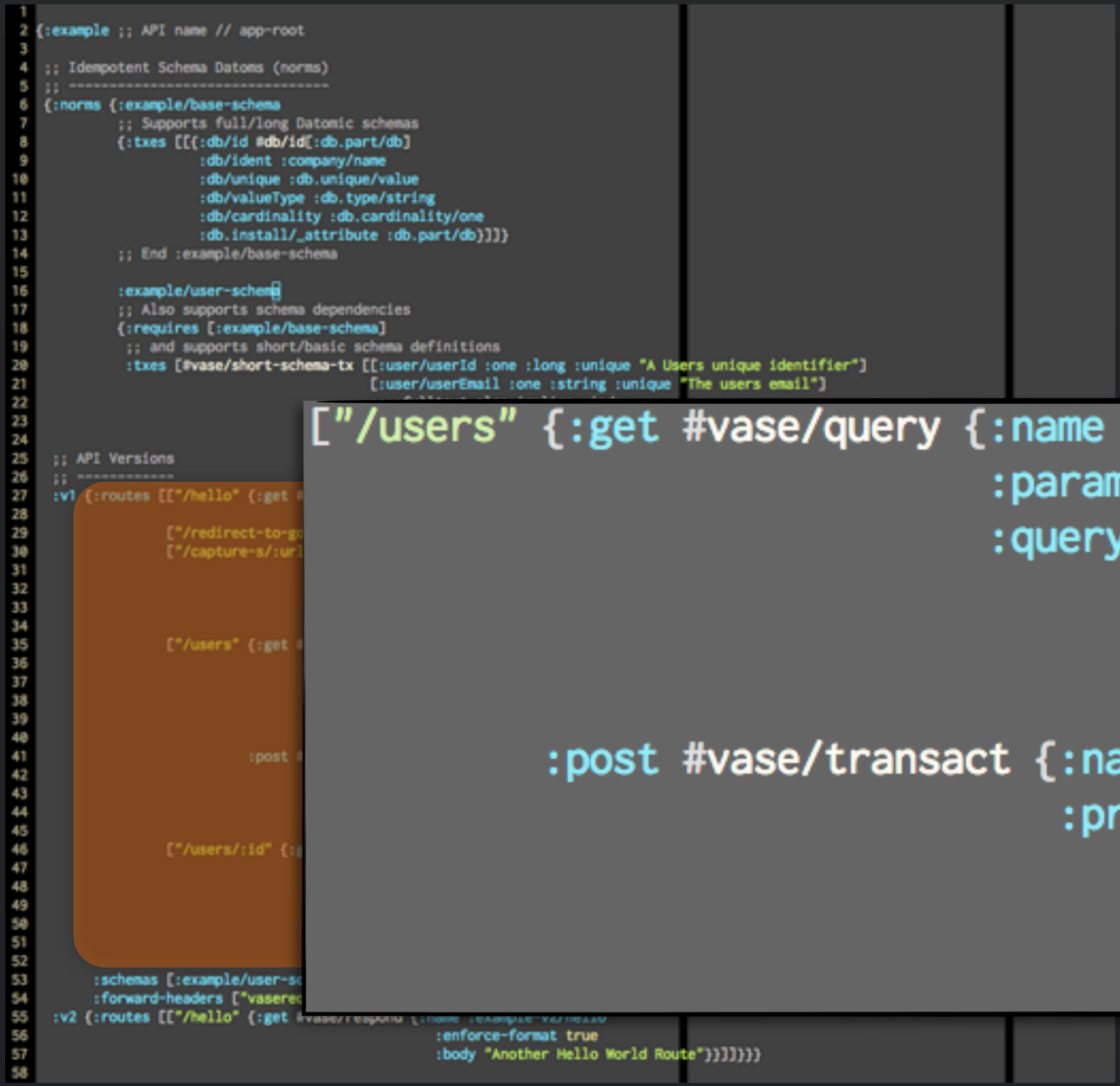
Container service



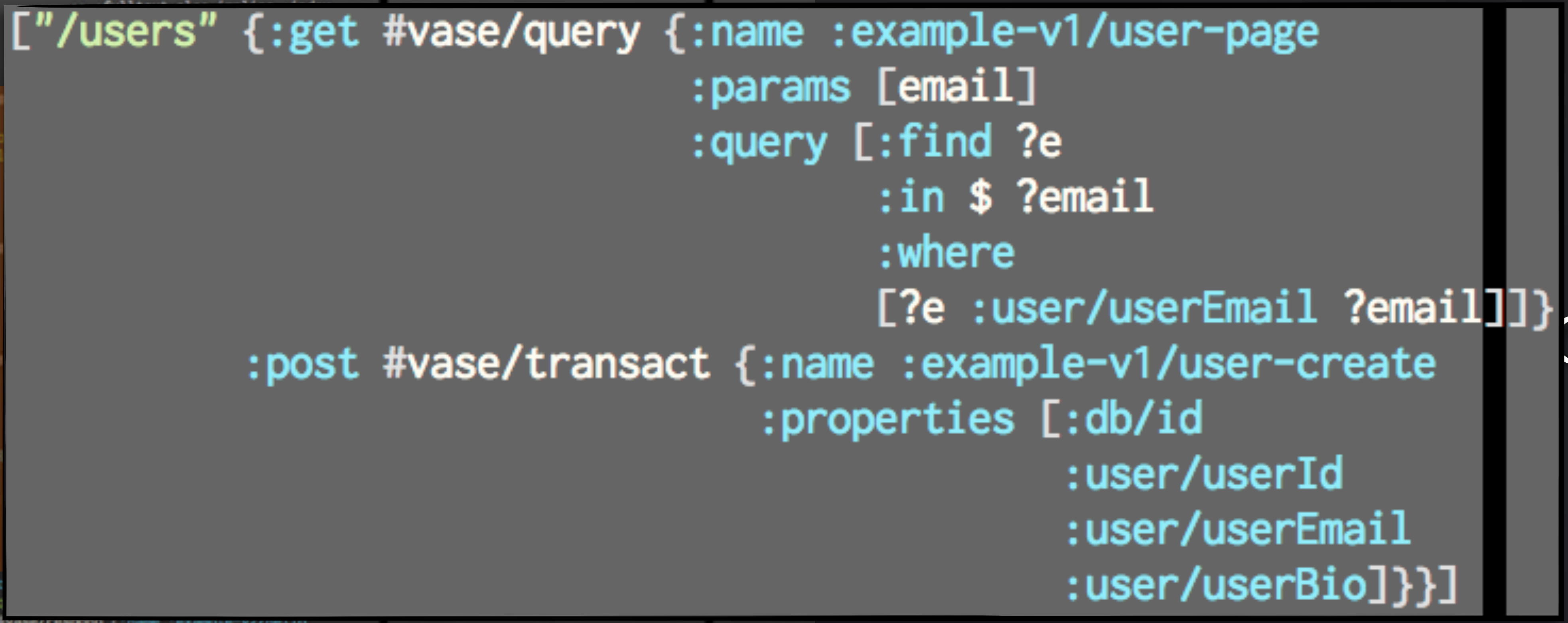
```
{:example ;; API name // app-root

 ;; Idempotent Schema Datoms (norms)
 ;; --------------------------------
 {:norms {:example/base-schema
         ;; Supports full/long Datomic schemas
         {:txes [[{:db/id #db/id[:db.part/db]
                   :db/ident :company/name
                   :db/unique :db.unique/value
                   :db/valueType :db.type/string
                   :db/cardinality :db.cardinality/one
                   :db.install/_attribute :db.part/db}]]}
         ;; End :example/base-schema

         :example/user-schema
         ;; Also supports schema dependencies
         {:requires [:example/base-schema]
          ;; and supports short/basic schema definitions
          :txes [#vase/short-schema-tx [[:user/userId :one :long :unique "A Users unique identifier"]
                                        [:user/userEmail :one :string :unique "The users email"]
                                        ;; :fulltext also implies :index
                                        [:user/userBio :one :string :fulltext "A short blurb about the user"]]]}}

  ;; API Versions
  ;; ------------
  :v1 {:routes [["/hello" {:get #vase/respond {:name :example-v1/simple-response
                                                :body "Hello World"}}]
                ["/redirect-to-google" {:get #vase/redirect {:name :example-v1/r-page, :url "http://www.google.com"}}]
                ["/capture-s/:url-thing" {:get #vase/respond {:name :example-v1/url-param-example
                                                              ;; URL parameters are also bound in :params
                                                              :params [url-thing]
                                                              :edn-coerce [url-thing] ;; parse a param as an edn string
                                                              :body (str "You said: " url-thing " which is a " (type url-thing))}}]
                ["/users" {:get #vase/query {:name :example-v1/user-page
                                             :params [email]
                                             :query [:find ?e
                                                     :in $ ?email
                                                     :where
                                                     [?e :user/userEmail ?email]]}
                           :post #vase/transact {:name :example-v1/user-create
                                                 :properties [:db/id
                                                              :user/userId
                                                              :user/userEmail
                                                              :user/userBio]}}]
                ["/users/:id" {:get #vase/query {:name :example-v1/user-id-page
                                                 :params [id]
                                                 :edn-coerce [id]
                                                 :query [:find ?e
                                                         :in $ ?id
                                                         :where
                                                         [?e :user/userId ?id]]}}]]
       :schemas [:example/user-schema :example/loan-schema]
       :forward-headers ["vaserequest-id"]}
  :v2 {:routes [["/hello" {:get #vase/respond {:name :example-v2/hello
                                                :enforce-format true
                                                :body "Another Hello World Route"}}]]}}
```

Route definition

Container service



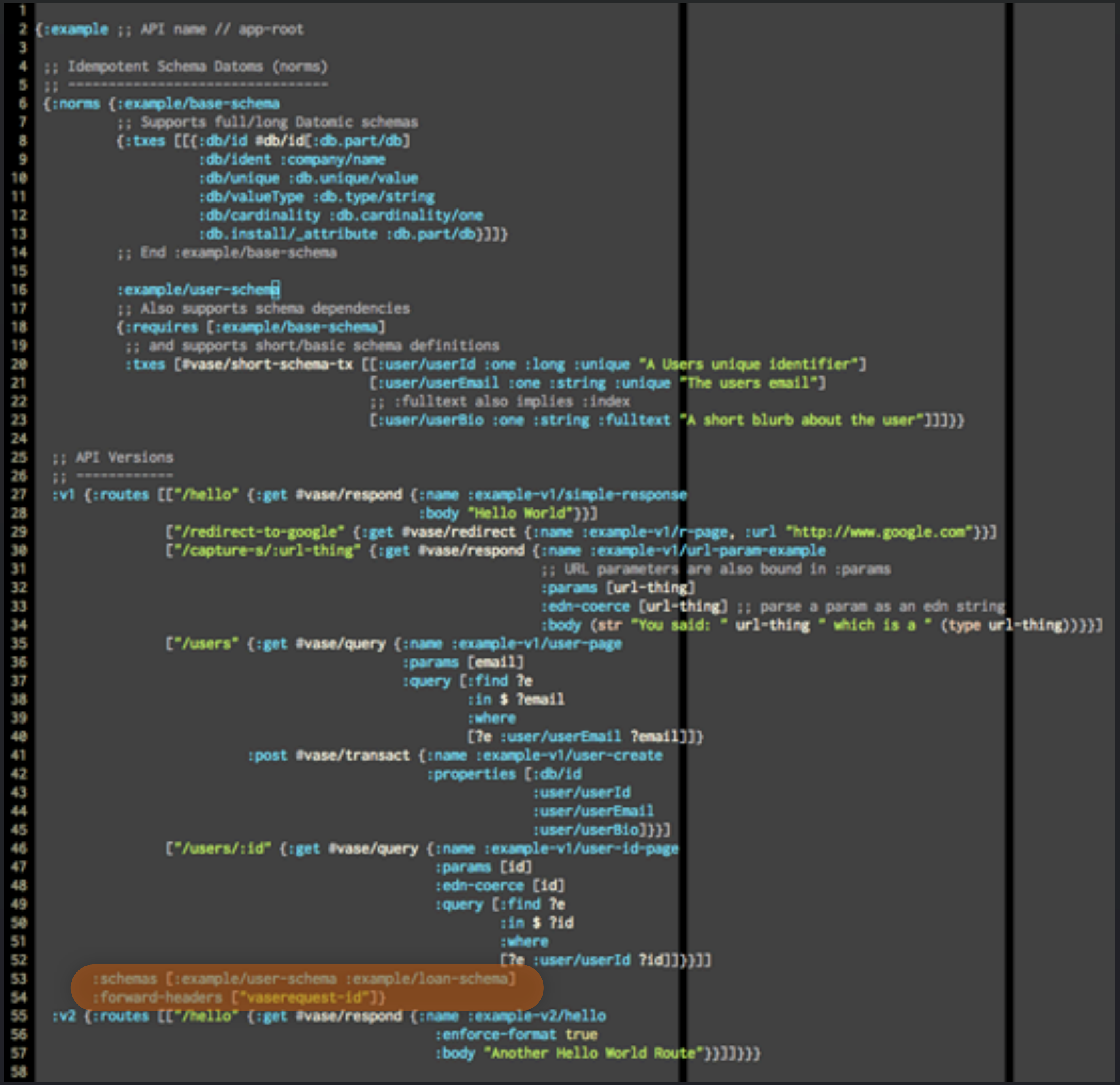
```
{:example ;; API name // app-root

 ;; Idempotent Schema Datoms (norms)
 ;; ------------------------------
 {:norms {:example/base-schema
          ;; Supports full/long Datomic schemas
          {:txes [[{:db/id #db/id[:db.part/db]
                    :db/ident :company/name
                    :db/unique :db.unique/value
                    :db/valueType :db.type/string
                    :db/cardinality :db.cardinality/one
                    :db.install/_attribute :db.part/db}]]}
          ;; End :example/base-schema

          :example/user-schema
          ;; Also supports schema dependencies
          {:requires [:example/base-schema]
           ;; and supports short/basic schema definitions
           :txes [#vase/short-schema-tx [[:user/userId :one :long :unique "A Users unique identifier"]
                                         [:user/userEmail :one :string :unique "The users email"]



  ;; API Versions
  ;; -----------
  :v1 {:routes [["/hello" {:get

                 ["/redirect-to-go
                 ["/capture-s/:url




                 ["/users" {:get






                          :post




                 ["/users/:id" {:g






      :schemas [:example/user-sc
      :forward-headers ["vasereq
  :v2 {:routes [["/hello" {:get
                                            :enforce-format true
                                            :body "Another Hello World Route"}]]]}}}
```



```
["/users" {:get #vase/query {:name :example-v1/user-page
                             :params [email]
                             :query [:find ?e
                                     :in $ ?email
                                     :where
                                     [?e :user/userEmail ?email]]}
           :post #vase/transact {:name :example-v1/user-create
                                 :properties [:db/id
                                              :user/userId
                                              :user/userEmail
                                              :user/userBio]}}]
```

on

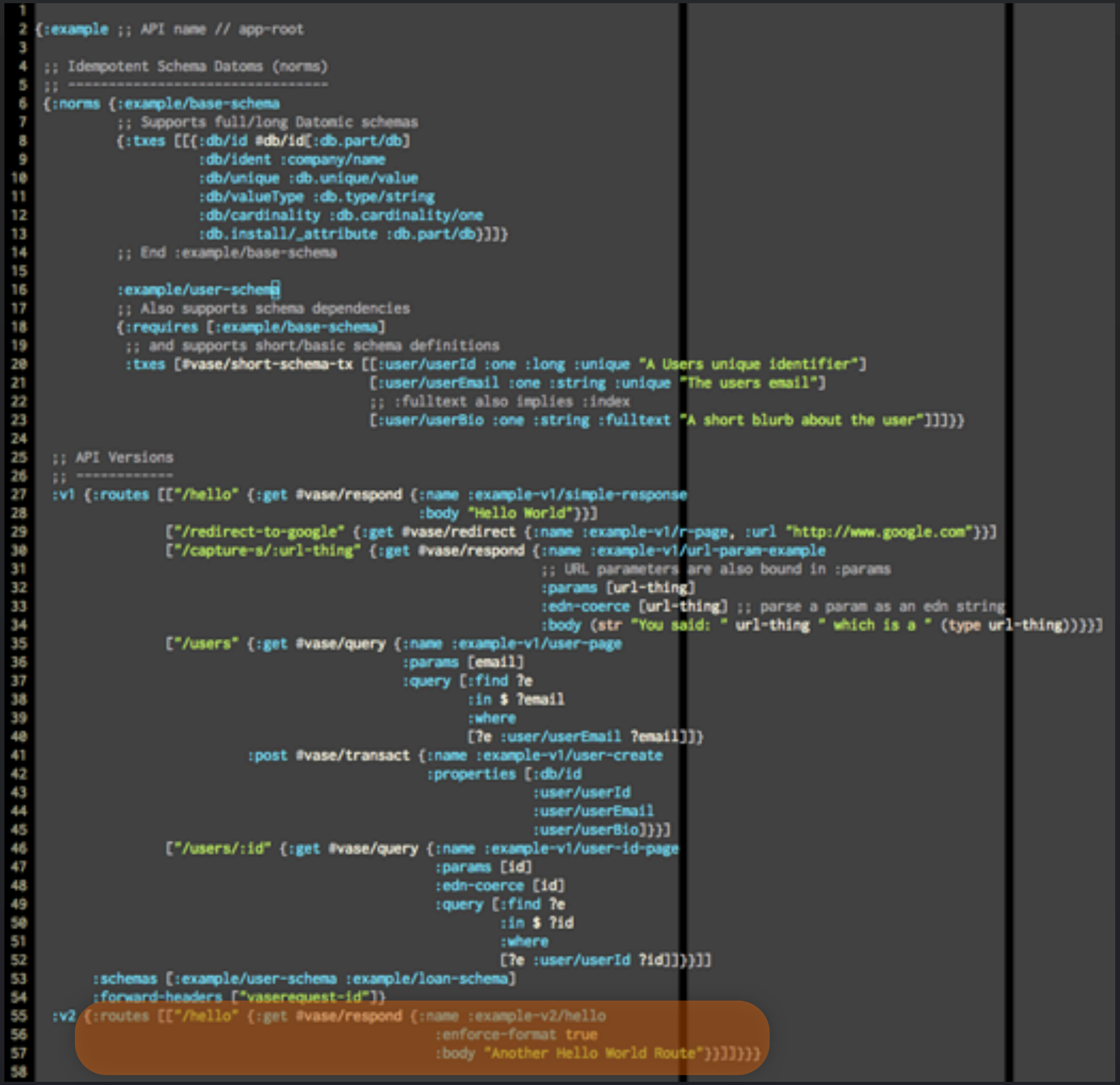
Container service



```
{:example ;; API name // app-root

 ;; Idempotent Schema Datoms (norms)
 ;; ---------------------------------
 {:norms {:example/base-schema
          ;; Supports full/long Datomic schemas
          {:txes [[{:db/id #db/id[:db.part/db]
                    :db/ident :company/name
                    :db/unique :db.unique/value
                    :db/valueType :db.type/string
                    :db/cardinality :db.cardinality/one
                    :db.install/_attribute :db.part/db}]]}
          ;; End :example/base-schema

          :example/user-schema
          ;; Also supports schema dependencies
          {:requires [:example/base-schema]
           ;; and supports short/basic schema definitions
           :txes [#vase/short-schema-tx [[:user/userId :one :long :unique "A Users unique identifier"]
                                         [:user/userEmail :one :string :unique "The users email"]
                                         ;; :fulltext also implies :index
                                         [:user/userBio :one :string :fulltext "A short blurb about the user"]]]}}

  ;; API Versions
  ;; ------------
  :v1 {:routes [["/hello" {:get #vase/respond {:name :example-v1/simple-response
                                               :body "Hello World"}}]
                ["/redirect-to-google" {:get #vase/redirect {:name :example-v1/r-page, :url "http://www.google.com"}}]
                ["/capture-s/:url-thing" {:get #vase/respond {:name :example-v1/url-param-example
                                                              ;; URL parameters are also bound in :params
                                                              :params [url-thing]
                                                              :edn-coerce [url-thing] ;; parse a param as an edn string
                                                              :body (str "You said: " url-thing " which is a " (type url-thing))}}]
                ["/users" {:get #vase/query {:name :example-v1/user-page
                                             :params [email]
                                             :query [:find ?e
                                                     :in $ ?email
                                                     :where
                                                     [?e :user/userEmail ?email]]}
                           :post #vase/transact {:name :example-v1/user-create
                                                 :properties [:db/id
                                                              :user/userId
                                                              :user/userEmail
                                                              :user/userBio]}}]
                ["/users/:id" {:get #vase/query {:name :example-v1/user-id-page
                                                 :params [id]
                                                 :edn-coerce [id]
                                                 :query [:find ?e
                                                         :in $ ?id
                                                         :where
                                                         [?e :user/userId ?id]]}}]]
       :schemas [:example/user-schema :example/loan-schema]
       :forward-headers ["vaserequest-id"]}
  :v2 {:routes [["/hello" {:get #vase/respond {:name :example-v2/hello
                                               :enforce-format true
                                               :body "Another Hello World Route"}}]]}}}
```

Service properties

Container service



```
{:example ;; API name // app-root

 ;; Idempotent Schema Datoms (norms)
 ;; --------------------------------
 {:norms {:example/base-schema
          ;; Supports full/long Datomic schemas
          {:txes [[{:db/id #db/id[:db.part/db]
                    :db/ident :company/name
                    :db/unique :db.unique/value
                    :db/valueType :db.type/string
                    :db/cardinality :db.cardinality/one
                    :db.install/_attribute :db.part/db}]]}
          ;; End :example/base-schema

          :example/user-schema
          ;; Also supports schema dependencies
          {:requires [:example/base-schema]
           ;; and supports short/basic schema definitions
           :txes [#vase/short-schema-tx [[:user/userId :one :long :unique "A Users unique identifier"]
                                         [:user/userEmail :one :string :unique "The users email"]
                                         ;; :fulltext also implies :index
                                         [:user/userBio :one :string :fulltext "A short blurb about the user"]]]}}

  ;; API Versions
  ;; ------------
  :v1 {:routes [["/hello" {:get #vase/respond {:name :example-v1/simple-response
                                               :body "Hello World"}}]
                ["/redirect-to-google" {:get #vase/redirect {:name :example-v1/r-page, :url "http://www.google.com"}}]
                ["/capture-s/:url-thing" {:get #vase/respond {:name :example-v1/url-param-example
                                                              ;; URL parameters are also bound in :params
                                                              :params [url-thing]
                                                              :edn-coerce [url-thing] ;; parse a param as an edn string
                                                              :body (str "You said: " url-thing " which is a " (type url-thing))}}]
                ["/users" {:get #vase/query {:name :example-v1/user-page
                                             :params [email]
                                             :query [:find ?e
                                                     :in $ ?email
                                                     :where
                                                     [?e :user/userEmail ?email]]}
                          :post #vase/transact {:name :example-v1/user-create
                                                :properties [:db/id
                                                             :user/userId
                                                             :user/userEmail
                                                             :user/userBio]}}]
                ["/users/:id" {:get #vase/query {:name :example-v1/user-id-page
                                                 :params [id]
                                                 :edn-coerce [id]
                                                 :query [:find ?e
                                                         :in $ ?id
                                                         :where
                                                         [?e :user/userId ?id]]}}]]
       :schemas [:example/user-schema :example/loan-schema]
       :forward-headers ["vaserequest-id"]}
  :v2 {:routes [["/hello" {:get #vase/respond {:name :example-v2/hello
                                               :enforce-format true
                                               :body "Another Hello World Route"}}]]}}}
```

Another version

Container service

- Programming with values
- Open for extension
- Code and data rollback
- Datalog enforced service-wide properties

“This is AWESOME!”

“This is AWESOME!”
“But what about our data?”

Datomic import

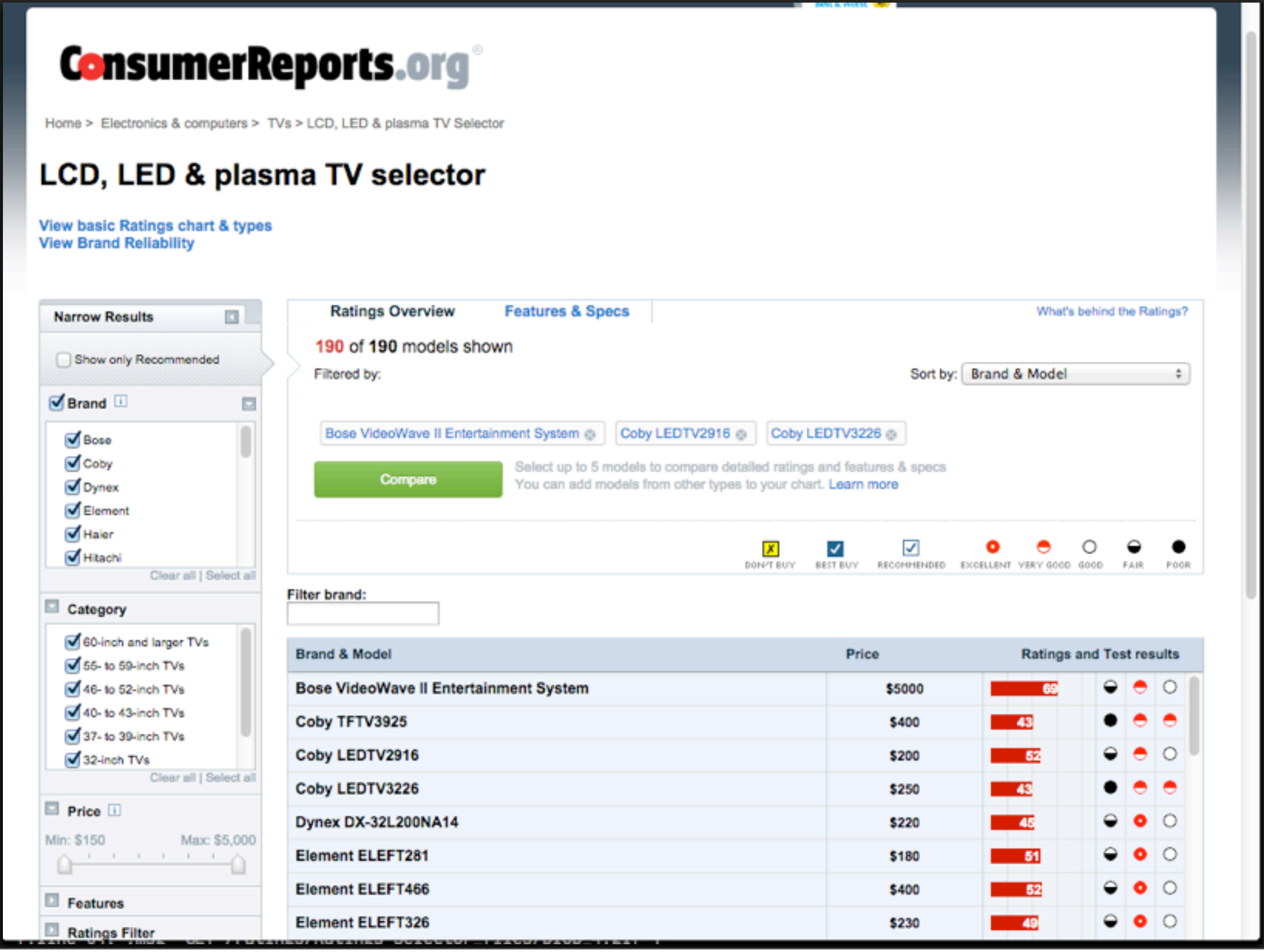
- Data source agnostic
- Multi-pass design
  - Hint and produce a schema
  - Use a schema to import data
- All “knobs” are data-driven

“*This* is AWESOME!”

“*This* is AWESOME!”

“But what about rich-client apps?”

Data-described clients



ConsumerReports.org

Home > Electronics & computers > TVs > LCD, LED & plasma TV Selector

LCD, LED & plasma TV selector

View basic Ratings chart & types
View Brand Reliability

Narrow Results

☐ Show only Recommended

✓ Brand

✓ Bose
✓ Coby
✓ Dynex
✓ Element
✓ Haier
✓ Hitachi

Clear all | Select all

Category

✓ 60-inch and larger TVs
✓ 55- to 59-inch TVs
✓ 46- to 52-inch TVs
✓ 40- to 43-inch TVs
✓ 37- to 39-inch TVs
✓ 32-inch TVs

Clear all | Select all

Price

Min: $150 Max: $5,000

Features

Ratings Filter

Ratings Overview | Features & Specs | What's behind the Ratings?

190 of 190 models shown

Filtered by:

Sort by: Brand & Model

Bose VideoWave II Entertainment System | Coby LEDTV2916 | Coby LEDTV3226

Compare

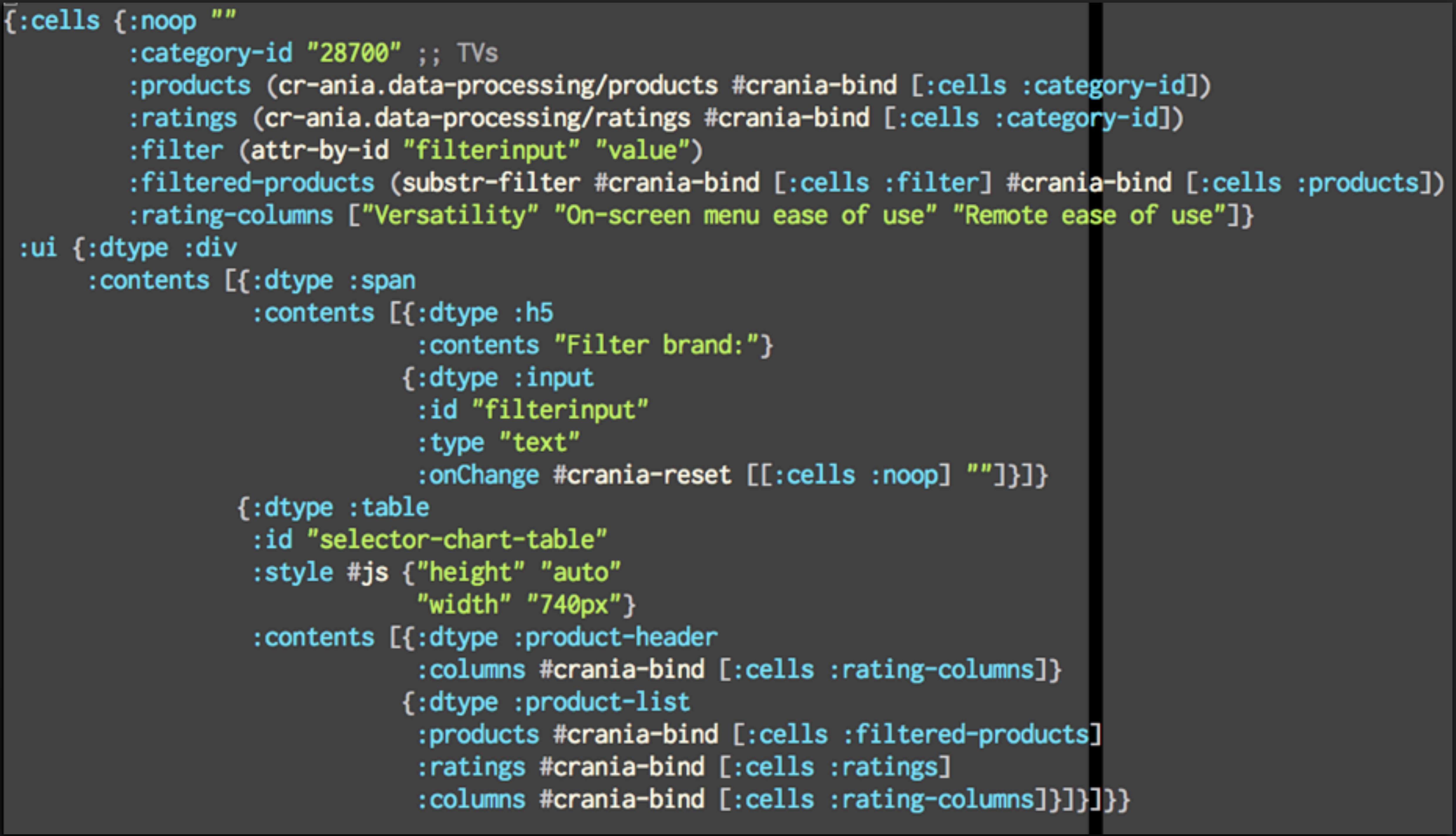
Select up to 5 models to compare detailed ratings and features & specs
You can add models from other types to your chart. Learn more

✗ DON'T BUY	✓ BEST BUY	✓ RECOMMENDED	⊙ EXCELLENT	◓ VERY GOOD	○ GOOD	◒ FAIR	● POOR

Filter brand:

Brand & Model	Price	Ratings and Test results			
Bose VideoWave II Entertainment System	$5000	69	◒	◓	○
Coby TFTV3925	$400	43	●	◓	◓
Coby LEDTV2916	$200	52	◒	◓	○
Coby LEDTV3226	$250	43	●	◓	◓
Dynex DX-32L200NA14	$220	45	◒	⊙	○
Element ELEFT281	$180	51	◒	⊙	○
Element ELEFT466	$400	52	◒	⊙	○
Element ELEFT326	$230	49	◒	⊙	○

Data-described clients

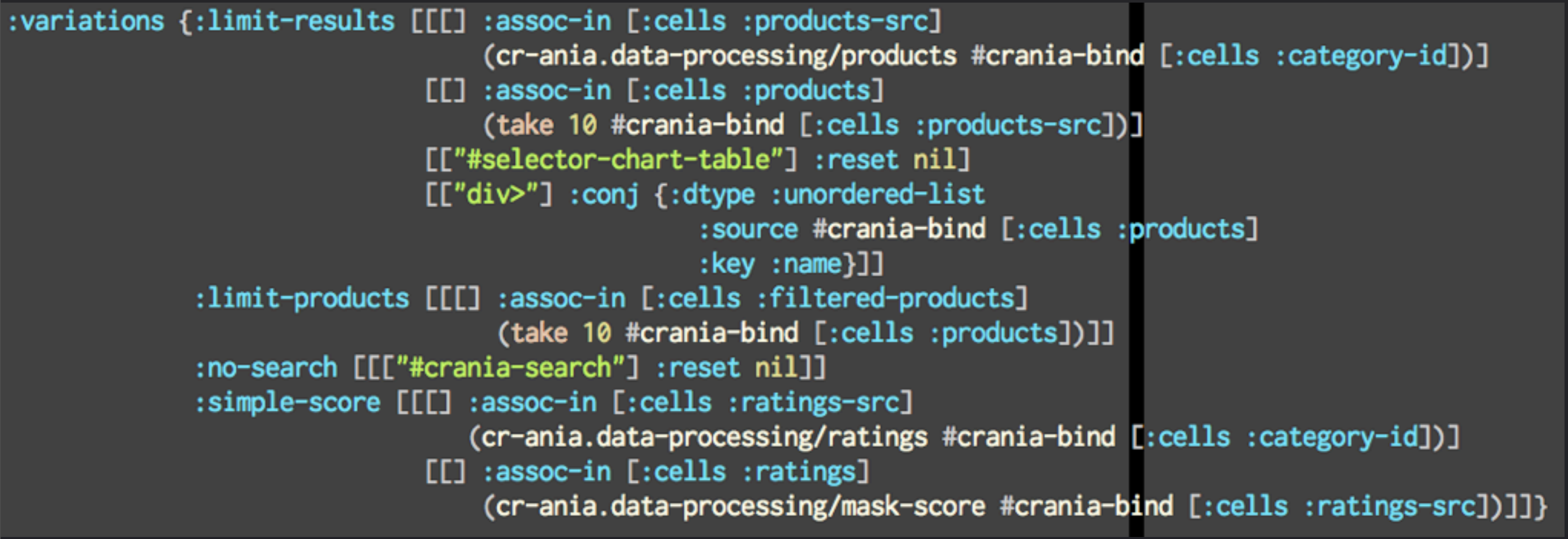


```
{:cells {:noop ""
         :category-id "28700" ;; TVs
         :products (cr-ania.data-processing/products #crania-bind [:cells :category-id])
         :ratings (cr-ania.data-processing/ratings #crania-bind [:cells :category-id])
         :filter (attr-by-id "filterinput" "value")
         :filtered-products (substr-filter #crania-bind [:cells :filter] #crania-bind [:cells :products])
         :rating-columns ["Versatility" "On-screen menu ease of use" "Remote ease of use"]}
 :ui {:dtype :div
      :contents [{:dtype :span
                  :contents [{:dtype :h5
                              :contents "Filter brand:"}
                             {:dtype :input
                              :id "filterinput"
                              :type "text"
                              :onChange #crania-reset [[:cells :noop] ""]}]}
                 {:dtype :table
                  :id "selector-chart-table"
                  :style #js {"height" "auto"
                              "width" "740px"}
                  :contents [{:dtype :product-header
                              :columns #crania-bind [:cells :rating-columns]}
                             {:dtype :product-list
                              :products #crania-bind [:cells :filtered-products]
                              :ratings #crania-bind [:cells :ratings]
                              :columns #crania-bind [:cells :rating-columns]}]}]}}
```

Apps like TV channels

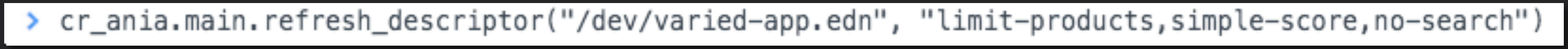
- Loaded/modified live
- Versioned, rolled back
- Queried, analyzed

Apps like TV channels

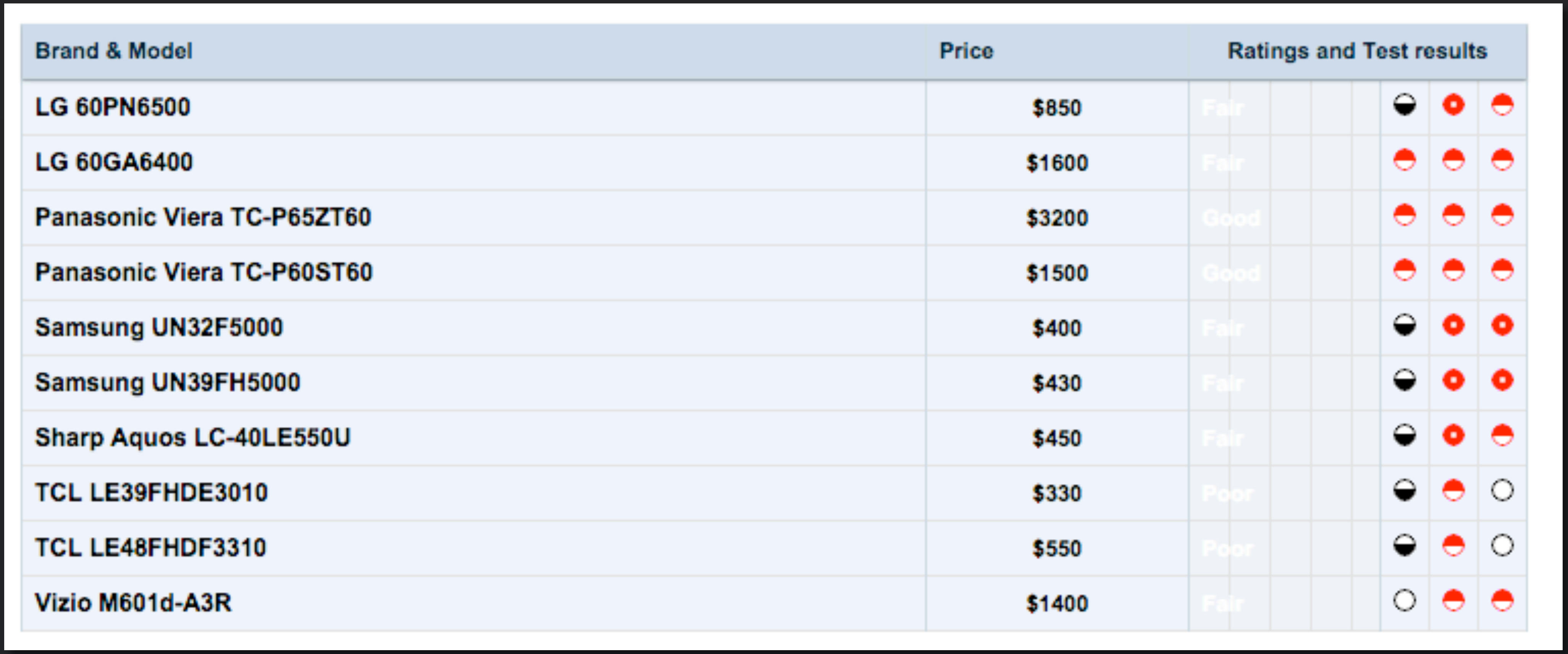


```
:variations {:limit-results [[[] :assoc-in [:cells :products-src]
                                 (cr-ania.data-processing/products #crania-bind [:cells :category-id])]
                             [[] :assoc-in [:cells :products]
                                 (take 10 #crania-bind [:cells :products-src])]
                             [["#selector-chart-table"] :reset nil]
                             [["div>"] :conj {:dtype :unordered-list
                                              :source #crania-bind [:cells :products]
                                              :key :name}]]
             :limit-products [[[] :assoc-in [:cells :filtered-products]
                                  (take 10 #crania-bind [:cells :products])]]
             :no-search [[["#crania-search"] :reset nil]]
             :simple-score [[[] :assoc-in [:cells :ratings-src]
                                (cr-ania.data-processing/ratings #crania-bind [:cells :category-id])]
                            [[] :assoc-in [:cells :ratings]
                                (cr-ania.data-processing/mask-score #crania-bind [:cells :ratings-src])]]}
```

Apps like TV channels



```
> cr_ania.main.refresh_descriptor("/dev/varied-app.edn", "limit-products,simple-score,no-search")
```



Brand & Model	Price	Ratings and Test results			
LG 60PN6500	$850	Fair	◒	◉	◓
LG 60GA6400	$1600	Fair	◓	◓	◓
Panasonic Viera TC-P65ZT60	$3200	Good	◓	◓	◓
Panasonic Viera TC-P60ST60	$1500	Good	◓	◓	◓
Samsung UN32F5000	$400	Fair	◒	◉	◉
Samsung UN39FH5000	$430	Fair	◒	◉	◉
Sharp Aquos LC-40LE550U	$450	Fair	◒	◉	◓
TCL LE39FHDE3010	$330	Poor	◒	◓	○
TCL LE48FHDF3310	$550	Poor	◒	◓	○
Vizio M601d-A3R	$1400	Fair	○	◓	◓

Project metrics

- Two developers
- Each project took 16-24 days
  - Kickoff to final delivery
- Functionally complete in 12-16 days
- Design time for 4-8 days (1-2 weeks)

Super powers

- Clojure, ClojureScript, Datomic, core.logic, and more
- Power in constraints
- Combinations multiply impact
- Holistic designs cause exponential impact

Data driven

- Be exploratory; Think holistically
- Write it down
- Constrain the design space
- Think critically, think slowly
- Envision the outcomes and possibilities

Unlocking Data-driven Systems

Comments, Questions, Concerns

Paul deGrandis
@OhPauleez