

# Server-side web-development

with Duct & Integrant

<https://github.com/danielytics/duct-talk>

# Integrant What?

*"Integrant is a Clojure (and ClojureScript) micro-framework for building applications with data-driven architecture."*

- A way to define “lifecycles” for component
- “init” and “halt” functions
- Like Stuart Sierra’s Component, but:
  - Multimethods instead of Protocols
  - Leverages “derive” relationships
  - Built-in easy EDN config

# Duct What?

*"Duct is a highly modular framework for building server-side applications in Clojure using data-driven architecture."*

- A “framework” that gives you:
  - A project template
  - Common functionality
  - A small amount of structure
  - A set of premade modules
  - “Reloaded” workflow
- Based on Integrant

# What to Build?

- Server-side REST API
- Hacker News style site
- Using Duct's defaults

We will use HTTPie to interact our API

<https://httpie.org/>

# What to Build?

## Stories

- Get list of stories

GET /

- Submit new story

POST /stories

- Upvote a story

POST /stories/<id>/votes

## Comments

- Get list of comments for story

GET /stories/<id>/comments

- Add new comment to story

POST /stories/<id>/comments

- Upvote a comment

POST /stories/<id>/comments/<id>/votes

# Getting Started

```
lein new duct duct-talk +api +sqlite +ataraxy +example
```

- **+api** Add API middleware and handler
- **+sqlite** Add SQLite dependency and component
- **+ataraxy** Add Duct's Ataraxy request router
- **+example** Add an example handler

# Using the REPL

```
cd duct-talk
```

```
lein duct setup
```

```
lein repl
```

```
user=> (dev)
```

```
dev=> (go)
```

To reload after making changes:

```
dev=> (reset)
```

```
http localhost:3000/example
```

```
HTTP/1.1 200 OK
```

```
Content-Length: 18
```

```
Content-Type: application/json;  
charset=utf-8
```

```
Date: Tue, 03 Sep 2019 13:19:35 GMT
```

```
Server: Jetty(9.2.21.v20170120)
```

```
{  
  "example": "data"  
}
```

# Duct Configuration

Duct template sets up configuration with multiple profiles

- Base configuration: core config and modules  
`resources/duct_talk/config.edn`
- Dev configuration: development profile  
`dev/resources/dev.edn`
- Local configuration: local profile, not committed to git  
`dev/resources/local.edn`



# Base Configuration

```
{:duct.profile/base
  {:duct.core/project-ns duct-talk

   :duct.router/ataraxy
  {:routes {[:get "/example"] [:duct-talk.handler/example]}}}

 :duct-talk.handler/example
{:db #ig/ref :duct.database/sql}}

:duct.profile/dev    #duct/include "dev"
:duct.profile/local  #duct/include "local"
:duct.profile/prod   {}

:duct.module/logging {}
:duct.module.web/api {}
:duct.module/sql     {}}}
```

# Base Configuration

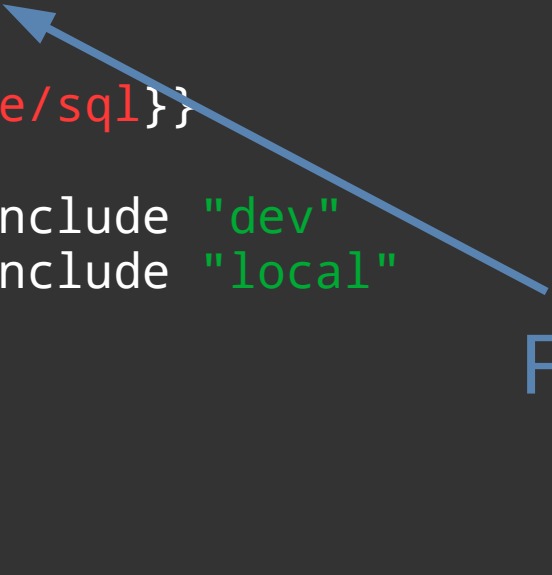
```
{:duct.profile/base
{:duct.core/project-ns duct-talk

:duct.router/ataraxy
{:routes {[:get "/example"] [:duct-talk.handler/example]}}

:duct-talk.handler/example
{:db #ig/ref :duct.database/sql}}

:duct.profile/dev    #duct/include "dev"
:duct.profile/local  #duct/include "local"
:duct.profile/prod    {}

:duct.module/logging {}
:duct.module.web/api  {}
:duct.module/sql      {}}
```



Route

# Base Configuration

```
{:duct.profile/base  
{:duct.core/project-ns duct-talk
```

```
:duct.router/ataraxy  
{:routes {[:get "/example"] [:duct-talk.handler/example]}}
```

```
:duct-talk.handler/example  
{:db #ig/ref :duct.database/sql}}
```

```
:duct.profile/dev    #duct/include "dev"  
:duct.profile/local  #duct/include "local"  
:duct.profile/prod   {}
```

```
:duct.module/logging {}  
:duct.module.web/api  {}  
:duct.module/sql      {}}}
```



Handler Component

# Base Configuration

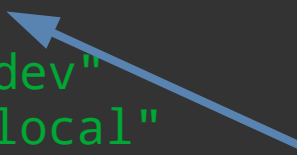
```
{:duct.profile/base
  {:duct.core/project-ns duct-talk

  :duct.router/ataraxy
  {:routes {[:get "/example"] [:duct-talk.handler/example]}}

  :duct-talk.handler/example
  {:db #ig/ref :duct.database/sql}}

:duct.profile/dev    #duct/include "dev"
:duct.profile/local  #duct/include "local"
:duct.profile/prod   {}

:duct.module/logging {}
:duct.module.web/api {}
:duct.module/sql     {}}}
```



Dependency on  
Database component

# Dev Configuration

```
{:duct.database/sql  
  {:connection-uri "jdbc:sqlite:db/dev.sqlite"}}}
```

# Handler Component

```
(ns duct-talk.handler.example
  (:require [ataraxy.core :as ataraxy]
            [ataraxy.response :as response]
            [integrant.core :as ig]))

(defmethod ig/init-key :duct-talk.handler/example
  [_ options]
  (fn [{[_] :ataraxy/result}]
    [::response/ok {:example "data"}]))
```

# Add Routes for Stories

```
{:duct.profile/base  
  {:duct.core/project-ns duct-talk}
```

```
:duct.profile/dev    #duct/include "dev"  
:duct.profile/local  #duct/include "local"  
:duct.profile/prod    {}
```

```
:duct.module/ataraxy  
{"/" {:get [:stories/index]  
      "stories" {:post [:stories/submit]  
                    [:post "/" ^int story-d "/votes"] [:stories/upvote story-id]]}}}
```

```
:duct.module/logging {}  
:duct.module.web/api {}  
:duct.module/sql {}}}
```

# Add Handler Components for Stories

```
{:duct.profile/base
{:duct.core/project-ns duct-talk
 :duct-talk.handler.stories/index
{:db #ig/ref :duct.database/sql}

:duct-talk.handler.stories/submit
{:db #ig/ref :duct.database/sql}

:duct-talk.handler.stories/upvote
{:db #ig/ref :duct.database/sql}}}
```

```
:duct.profile/dev    #duct/include "dev"
:duct.profile/local  #duct/include "local"
:duct.profile/prod    {}
```

...



# Add Handler Logic

```
(ns duct-talk.handler.stories
  (:require [ataraxy.core :as ataraxy]
            [ataraxy.response :as response]
            [integrant.core :as ig]))

(defmethod ig/init-key ::index
  [_ options]
  (fn [{[_] :ataraxy/result}]
    [::response/ok []]))

(defmethod ig/init-key ::submit
  [_ options]
  (fn [{[_] :ataraxy/result}]
    [::response/ok "submitted"]))

(defmethod ig/init-key ::upvote
  [_ options]
  (fn [{[_] :ataraxy/result}]
    [::response/ok "voted"]))
```

# Demo

Reload changes:

dev=> (reset)

http GET localhost:3000/

```
HTTP/1.1 200 OK
Content-Length: 2
Content-Type: application/json;
charset=utf-8
Date: Tue, 03 Sep 2019 15:57:12 GMT
Server: Jetty(9.2.21.v20170120)
```

[]

http GET localhost:3000/stories

```
{
  "error": "method-not-allowed"
}
```

http POST localhost:3000/stories

submitted

http POST localhost:3000/stories/1

voted

# Add Database Migrations

```
{:duct.profile/base
{:duct.core/project-ns duct-talk
 :duct-talk.handler/stories/index
{:db #ig/ref :duct.database/sql}

:duct-talk.handler/stories/submit
{:db #ig/ref :duct.database/sql}

:duct-talk.handler/stories/upvote
{:db #ig/ref :duct.database/sql}
```

```
[{:duct.migrator.ragtime/sql :migrations/init-stories]
{:up    [#duct/resource "migrations/init-stories.up.sql"]
 :down  [#duct/resource "migrations/init-stories.down.sql"]}

:duct.migrator/ragtime
{:migrations [#ig/ref :migrations/init-stories]}}
```

...

# Migration SQL

resources/migrations/init-stories.up.sql

```
CREATE TABLE stories (  
  id INTEGER PRIMARY KEY,  
  title TEXT NOT NULL,  
  url TEXT NOT NULL,  
  votes INTEGER DEFAULT 0  
);
```

resources/migrations/init-stories.down.sql

```
DROP TABLE stories;
```

# Boundaries

*"Boundaries not only allow you to control how data enters and exits your application; boundaries are also useful for testing."*

- Protocol
- Interface to an external service
- Should be split by purpose
- Should hide implementation details

# Stories Boundary

```
(ns duct-talk.boundaries.stories
  (:require [duct.database.sql]
            [hugsql.core :as hugsql]))

(hugsql/def-db-fns "duct_talk/boundaries/stories.sql")

(defprotocol Stories
  (index [db])
  (create! [db title url])
  (upvote! [db id]))

(extend-protocol Stories
  duct.database.sql.Boundary
  (index [{db :spec}]
    (list-stories db))

  (create! [{db :spec} title link]
    (create-story! db {:title title
                       :link link}))

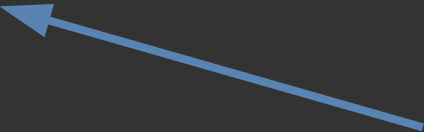
  (upvote! [{db :spec} id]
    (upvote-story! db {:id id})))
```

# Stories Boundary

```
(ns duct-talk.boundaries.stories
  (:require [duct.database.sql]
            [hugsql.core :as hugsql]))

(hugsql/def-db-fns "duct_talk/boundaries/stories.sql")
```

```
(defprotocol Stories
  (index [db])
  (create! [db title url])
  (upvote! [db id]))
```



```
(extend-protocol Stories
  duct.database.sql.Boundary
  (index [{db :spec}]
    (list-stories db))

  (create! [{db :spec} title link]
    (create-story! db {:title title
                       :link link}))

  (upvote! [{db :spec} id]
    (upvote-story! db {:id id})))
```

Boundary Protocol

# Stories Boundary

```
(ns duct-talk.boundaries.stories
  (:require [duct.database.sql]
            [hugsql.core :as hugsql]))

(hugsql/def-db-fns "duct_talk/boundaries/stories.sql")

(defprotocol Stories
  (index [db])
  (create! [db title url])
  (upvote! [db id]))
```

```
(extend-protocol Stories
  duct.database.sql.Boundary
  (index [{db :spec}]
    (list-stories db))

  (create! [{db :spec} title link]
    (create-story! db {:title title
                       :link link}))

  (upvote! [{db :spec} id]
    (upvote-story! db {:id id})))
```

Boundary  
Implementation for SQL





# Stories Boundary

```
(ns duct-talk.boundaries.stories
  (:require [duct.database.sql]
            [hugsql.core :as hugsql]))
```

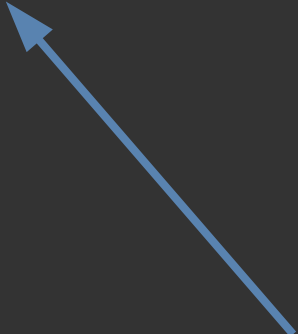
```
(hugsql/def-db-fns "duct_talk/boundaries/stories.sql")
```

```
(defprotocol Stories
  (index [db])
  (create! [db title url])
  (upvote! [db id]))
```

```
(extend-protocol Stories
  duct.database.sql.Boundary
  (index [{db :spec}]
    (list-stories db))
```

```
  (create! [{db :spec} title link]
    (create-story! db {:title title
                       :link link})))
```

```
  (upvote! [{db :spec} id]
    (upvote-story! db {:id id})))
```



SQL file of queries, using  
HugSQL

# SQL Queries for Stories

```
duct_talk/boundaries/stories.sql
```

```
-- :name list-stories :?
```

```
-- :doc Retrieve list of all stories
```

```
select id, title, url, votes from stories;
```

```
-- :name create-story! :!
```

```
-- :doc Create a new story submission
```

```
insert into stories (title, url) values (:title, :link);
```

```
-- :name upvote-story! :!
```

```
-- :doc Increment the vote count for a story
```

```
update stories set votes = votes + 1 where id = :id;
```

# Update Handler Logic

```
(ns duct-talk.handler.stories
  (:require [ataraxy.core :as ataraxy]
            [ataraxy.response :as response]
            [integrant.core :as ig]
            [duct-talk.boundaries.stories :as stories]))

(defmethod ig/init-key ::index
  [_ {:keys [db]}]
  (fn [_]
    [::response/ok (stories/index db)]))

(defmethod ig/init-key ::submit
  [_ {:keys [db]}]
  (fn [{{:keys [title link]} :body-params}]
    (stories/create! db title link)
    [::response/ok "submitted"]))

(defmethod ig/init-key ::upvote
  [_ {:keys [db]}]
  (fn [{{:id :story-id} :route-params}]
    (stories/upvote! db id)
    [::response/ok "voted"])))
```

# Demo

```
http GET localhost:3000/
```

```
[]
```

```
http POST localhost:3000/stories "title=New Story" link=https://google.com
```

```
submitted
```

```
http GET localhost:3000/
```

```
[  
  {  
    "id": 1,  
    "title": "New Story",  
    "url": "https://google.com",  
    "votes": 0  
  }  
]
```

# Add Database Migrations

...

```
:duct-talk.handler.stories/upvote  
{:db #ig/ref :duct.database/sql}
```

```
[ :duct.migrator.ragtime/sql :migrations/init-stories]  
{:up  [#duct/resource "migrations/init-stories.up.sql"]  
 :down [#duct/resource "migrations/init-stories.down.sql"]}
```

```
[ :duct.migrator.ragtime/sql :migrations/init-comments]  
{:up  [#duct/resource "migrations/init-comments.up.sql"]  
 :down [#duct/resource "migrations/init-comments.down.sql"]}
```

```
:duct.migrator/ragtime  
{:migrations [#ig/ref :migrations/init-stories  
                #ig/ref :migrations/init-comments]]}}
```

...

# Migration SQL

resources/migrations/init-comments.up.sql

```
CREATE TABLE comments (  
  id INTEGER PRIMARY KEY,  
  story_id INTEGER NOT NULL,  
  comment TEXT NOT NULL,  
  votes INTEGER DEFAULT 0,  
  FOREIGN KEY (story_id) REFERENCES stories(id)  
);
```

resources/migrations/init-comments.down.sql

```
DROP TABLE comments;
```

# Add Components for Comments

...

```
:duct-talk.handler.stories/index  
{:db #ig/ref :duct.database/sql}
```

```
:duct-talk.handler.stories/submit  
{:db #ig/ref :duct.database/sql}
```

```
:duct-talk.handler.stories/upvote  
{:db #ig/ref :duct.database/sql}}
```

```
:duct-talk.handler.comments/list  
{:db #ig/ref :duct.database/sql}
```

```
:duct-talk.handler.comments/submit  
{:db #ig/ref :duct.database/sql}
```

```
:duct-talk.handler.comments/upvote  
{:db #ig/ref :duct.database/sql}
```

...

# Handler Logic for Comments

```
(ns duct-talk.handler.comments
  (:require [ataraxy.core :as ataraxy]
            [ataraxy.response :as response]
            [integrant.core :as ig]
            [duct-talk.boundaries.comments :as comments]))
```

```
(defmethod ig/init-key ::list
  [_ {:keys [db]}]
  (fn [{:id :story-id} :route-params]
    [::response/ok (comments/get-list db id)]))
```

```
(defmethod ig/init-key ::submit
  [_ {:keys [db]}]
  (fn [{:id :story-id} :route-params
        {text :comment} :body-params]
    (comments/create! db id text)
    [::response/ok "submitted"])))
```

```
(defmethod ig/init-key ::upvote
  [_ {:keys [db]}]
  (fn [{:id :story-id} :route-params]
    (comments/upvote! db id)
    [::response/ok "voted"])))
```



# Add Routes for Comments

...

```
:duct.module/ataraxy
{"/" {:get [:stories/index]
  "stories" {:post [:stories/submit]
    ["/" ^int story-id] [{[:post "/votes"] [:stories/upvote story-id]
      "/comments" {:get [:comments/list story-id]
        :post [:comments/submit story-id]
        [:post "/" ^int comment-id "/votes"]
        [:comments/upvote comment-id]]}}}}}
```

```
:duct.module/logging {}
:duct.module.web/api {}
:duct.module/sql {}}
```

# Comments Boundary

```
(ns duct-talk.boundaries.comments
  (:require [duct.database.sql]
            [hugsql.core :as hugsql]))

(hugsql/def-db-fns "duct_talk/boundaries/comments.sql")

(defprotocol Comments
  (get-list [db story-id])
  (create! [db story-id text])
  (upvote! [db id]))

(extend-protocol Comments
  duct.database.sql.Boundary
  (get-list [{db :spec} story-id]
    (list-comments db {:story-id story-id}))

  (create! [{db :spec} story-id text]
    (add-comment! db {:story-id story-id
                      :comment text}))

  (upvote! [{db :spec} id]
    (upvote-comment! db {:id id})))
```

# SQL Queries for Comments

duct\_talk/boundaries/comments.sql

-- :name list-comments :?

-- :doc Retrieve list of comments for a story

select id, comment, votes from comments where story\_id = :story-id;

-- :name add-comment! :! :1

-- :doc Add a new comment to a story

insert into comments (story\_id, comment) values (:story-id, :comment);

-- :name upvote-comment! :!

-- :doc Increment the vote count for a comment

update comments set votes = votes + 1 where id = :id;

# Demo

http POST localhost:3000/stories/1/comments "comment: This is a comment"

submitted

http POST localhost:3000/stories/1/comments/1/vote

voted

http GET localhost:3000/stories/1/comments

```
[
  {
    "comment": "this is a comment",
    "id": 1,
    "votes": 1
  }
]
```

# Questions?

## Any Questions?

<https://github.com/danielytics/duct-talk>