

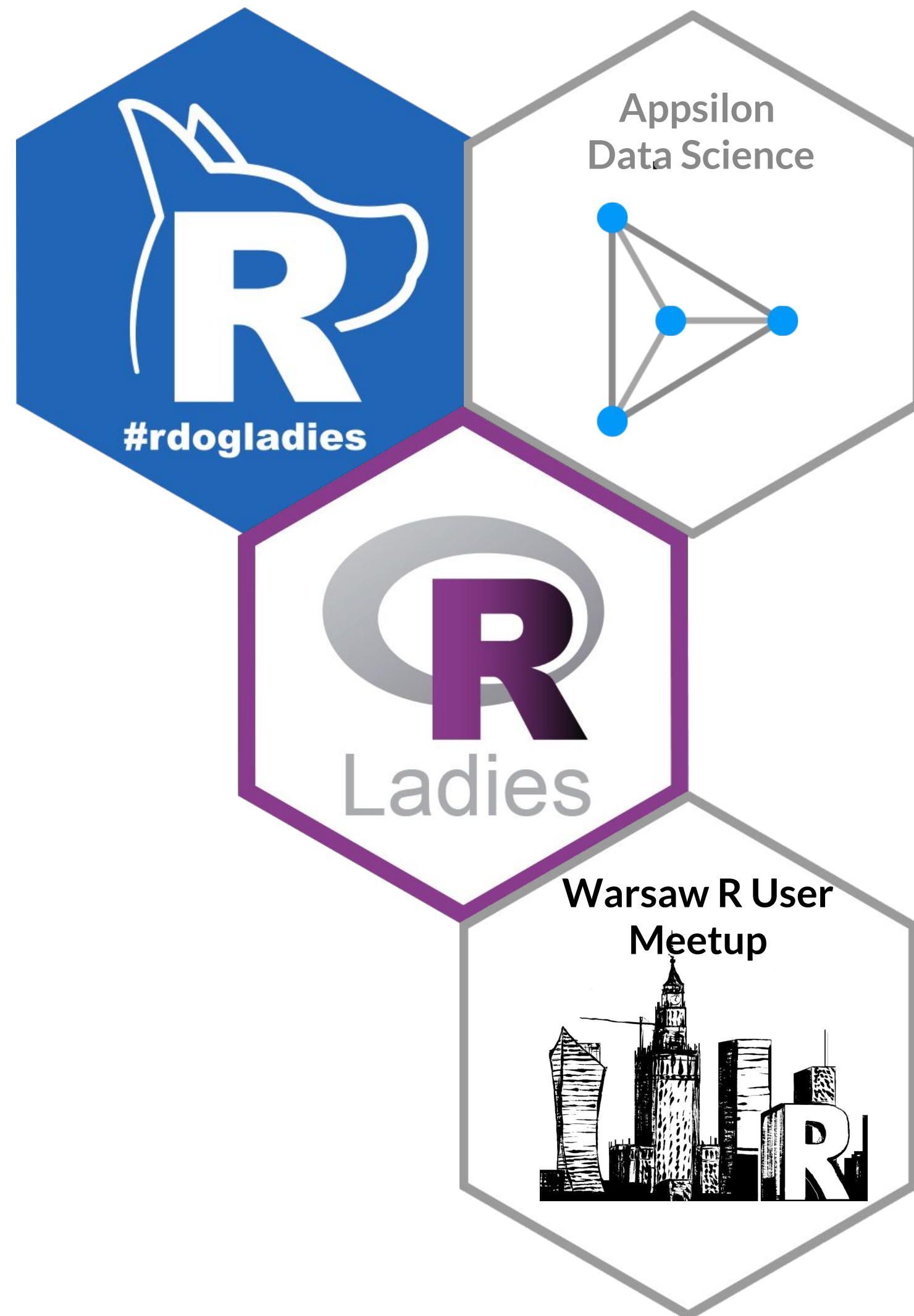


How we built a Shiny App used by 700 users?

Data Science team story

useR! Brussels 2017

@olga_mie



HELLO!

@olga_mie
@AppsilonDS

700 users Shiny App

PROJECT STORY



1

Shiny Application for 700 end users supporting their decision making

2

Team only Data Scientists

3

Prototype the next day, working demo after 2 weeks

4

Solution delivered for BCG



“...during the first week [of UAT],
we got **overwhelmingly positive**
feedback and **good results...**”.

John Dannberg, The Boston Consulting Group

700 users Shiny App

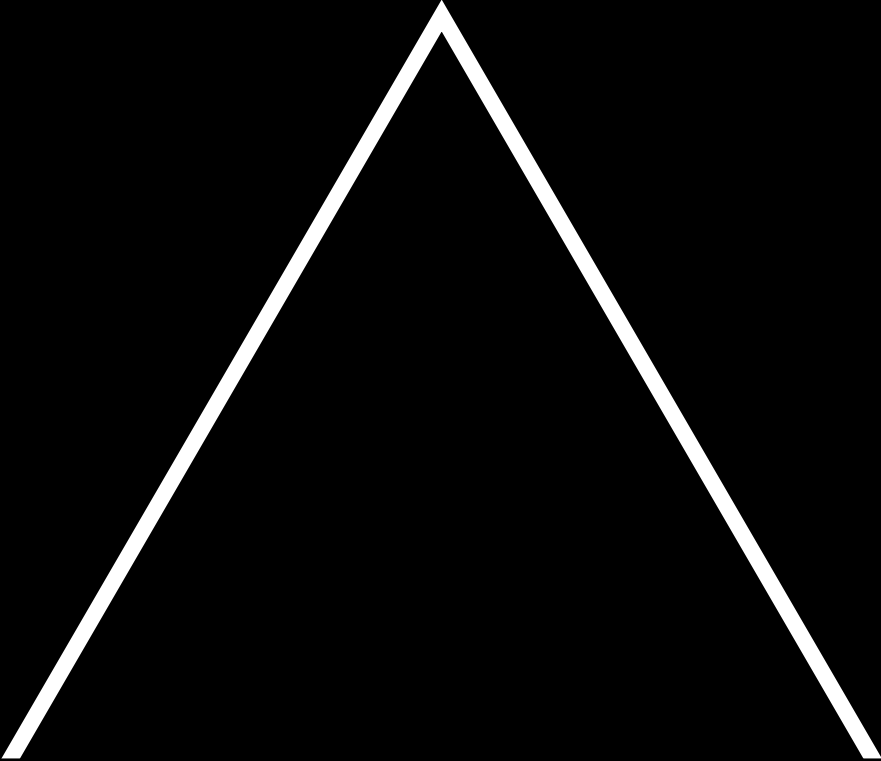
CHALLENGES



UI

UX

SCALE



BEAUTIFUL UI



Client's info

Name	John Smiths
City	Warsaw, Poland
Client since	07/2014
Our rating	★★★★★
Monthly spendings	\$2500 aprox.

Calls history

Show 10  entries

Search:

	Date	Topic	Consultant
1	2016-12-08	Connection problem	Kate Lees
2	2016-12-22	New plan	You
3	2016-12-23	Resume services	Mike Bradley
4	2017-01-01	Suspension of service	Kate Lees

Showing 1 to 4 of 4 entries

Previous

1

Next

Convert to 'Pay as you go' plan

Estimated reduction

33% cost reduction based on historical data

+ Proceed

Offer health insurance

Upselling opportunity

Clients of following profile are 2 times more likely to accept our health insurance plan

+ Proceed

Sell credit card for spouse

Upselling opportunity

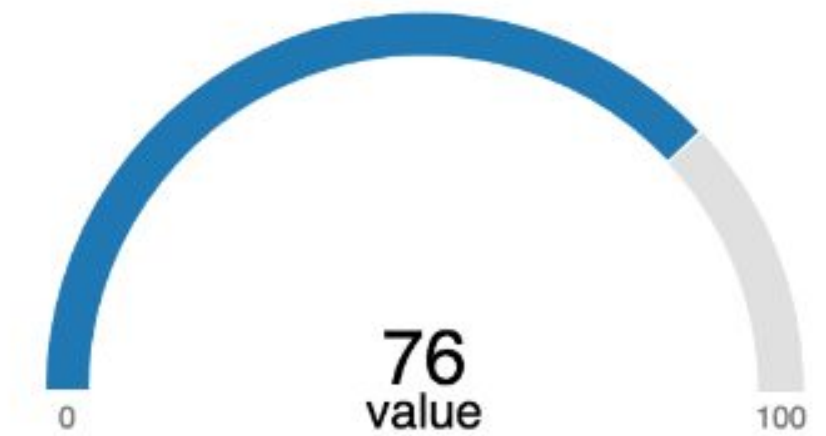
Accounts used by familiy members are likely to accept additional card and increase number of transactions.

+ Proceed

Current call

05:37

EMOTIONS



Voice analysis: Interested

FEEDBACK

Score client:

★★★★★

Churn risk - your feelings:

★★★★★

using shiny.semantic

USER INTERFACE



Package: shiny.semantic - <http://appsilon.github.io/shiny.semantic>

What is it?

- R package available on CRAN and GH

```
install.packages('shiny.semantic')  
devtools::install_github('Appsilon/shiny.semantic')
```

- Package for Semantic UI components
- **alternative to currently available Bootstrap**
- **domain specific language** wrapping HTML tags

using shiny.semantic

USER INTERFACE



Package: shiny.semantic - <http://appsilon.github.io/shiny.semantic>

What it does?

- downloads and imports **Semantic UI CSS** classes
- creates the abstraction enabling the user to

define Shiny (text) inputs

- contains ready to use **pre-defined more complex elements** as R functions



USER EXPERIENCE

Filtering in the backend

FAST DATA LOOKUPS



Fast Lookups: Indexing using *data.table* 25x faster than *dplyr::filter*

```
library(dplyr)

benchmark({
  key_to_lookup <- select_random_key()
  time(data %>% filter(col1 == key_to_lookup))
})

##      min      max    mean
## 0.0960 0.2440 0.1124
```

```
library(data.table)

time(setkey(data_table, col1)) ## 5.645

benchmark({
  key_to_lookup <- select_random_key()
  time(data_table[.(key_to_lookup), nomatch = 0L])
})

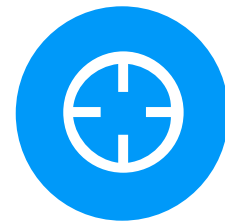
##      min      max    mean
## 0.00200 0.01200 0.00455
```

*data has 10 M rows.

Filtering in the frontend

OWN API

...



Fast custom search with server side API

- Server-side solution sends only matching results

Why not use shiny selectize?

- Better UI element
- Flexible functionality of searching
- Custom search algorithm

Appsilon
Data Science

Client

Clie

CLIENT NAME1 - 21052
ID: 21052

CLIENT NAME2 - 73296
ID: 73296

CLIENT NAME3 - 85214
ID: 85214

CLIENT NAME4 - 78840
ID: 78840

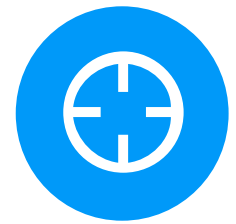
CLIENT NAME5 - 33197
ID: 33197

search more than 1 criteria

Filtering in the frontend

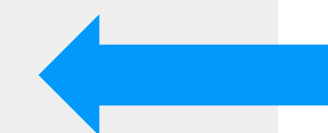
OWN API

...



Use `shiny::registerDataObj` to create API from R

```
register_search <- function(session, data, search_query) {  
  session$registerDataObj("search_api", data,  
    function(data, request) {  
      response <- jsonlite::toJSON(list(  
        success = TRUE,  
        results = search_query(data, request)  
      ))  
      shiny:::httpResponse(200, 'application/json',  
        enc2utf8(response))  
    }  
  )  
}
```



**search_query our own search algorithm
function**

RENDERING HIDDEN OUTPUT

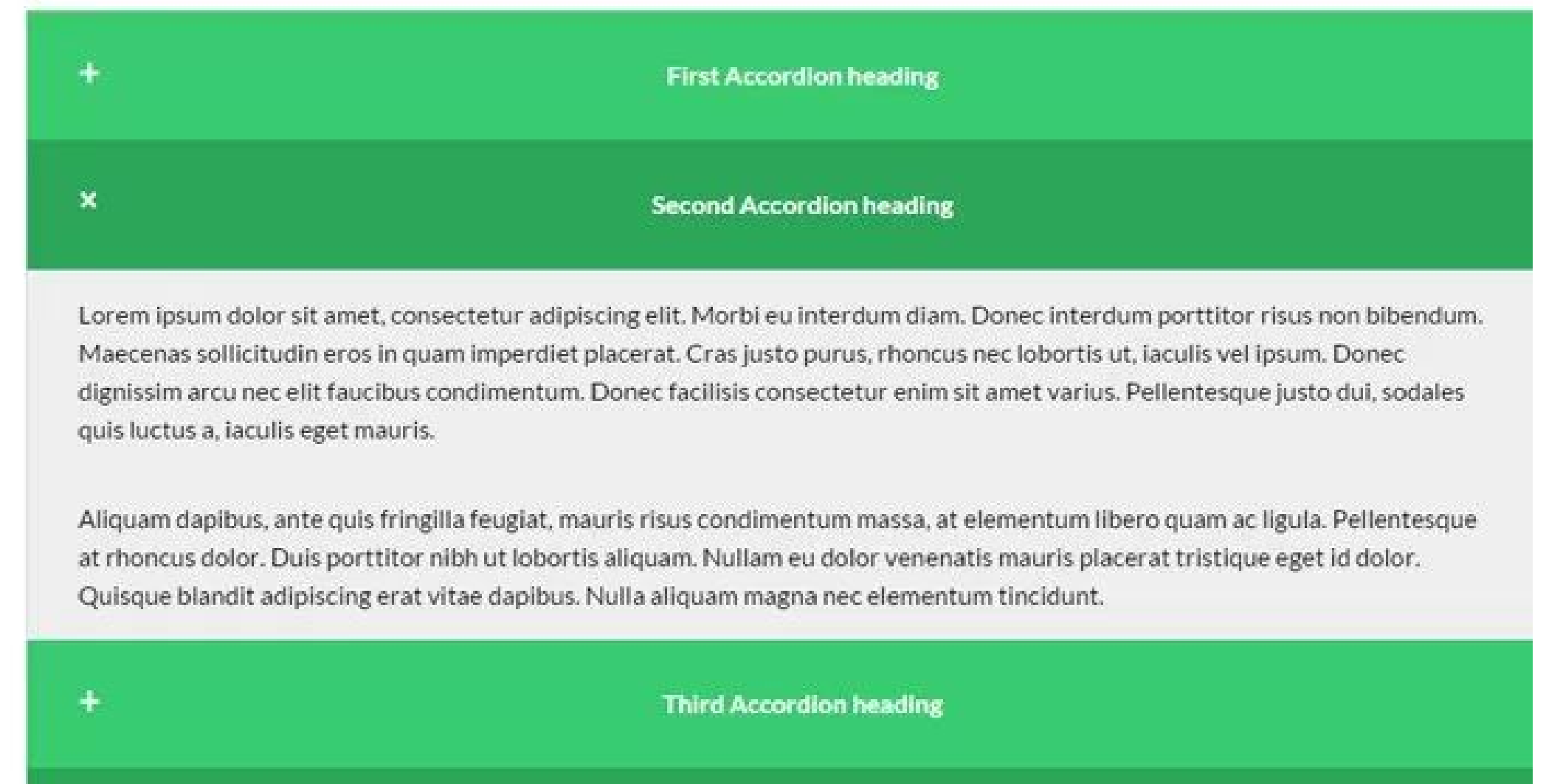
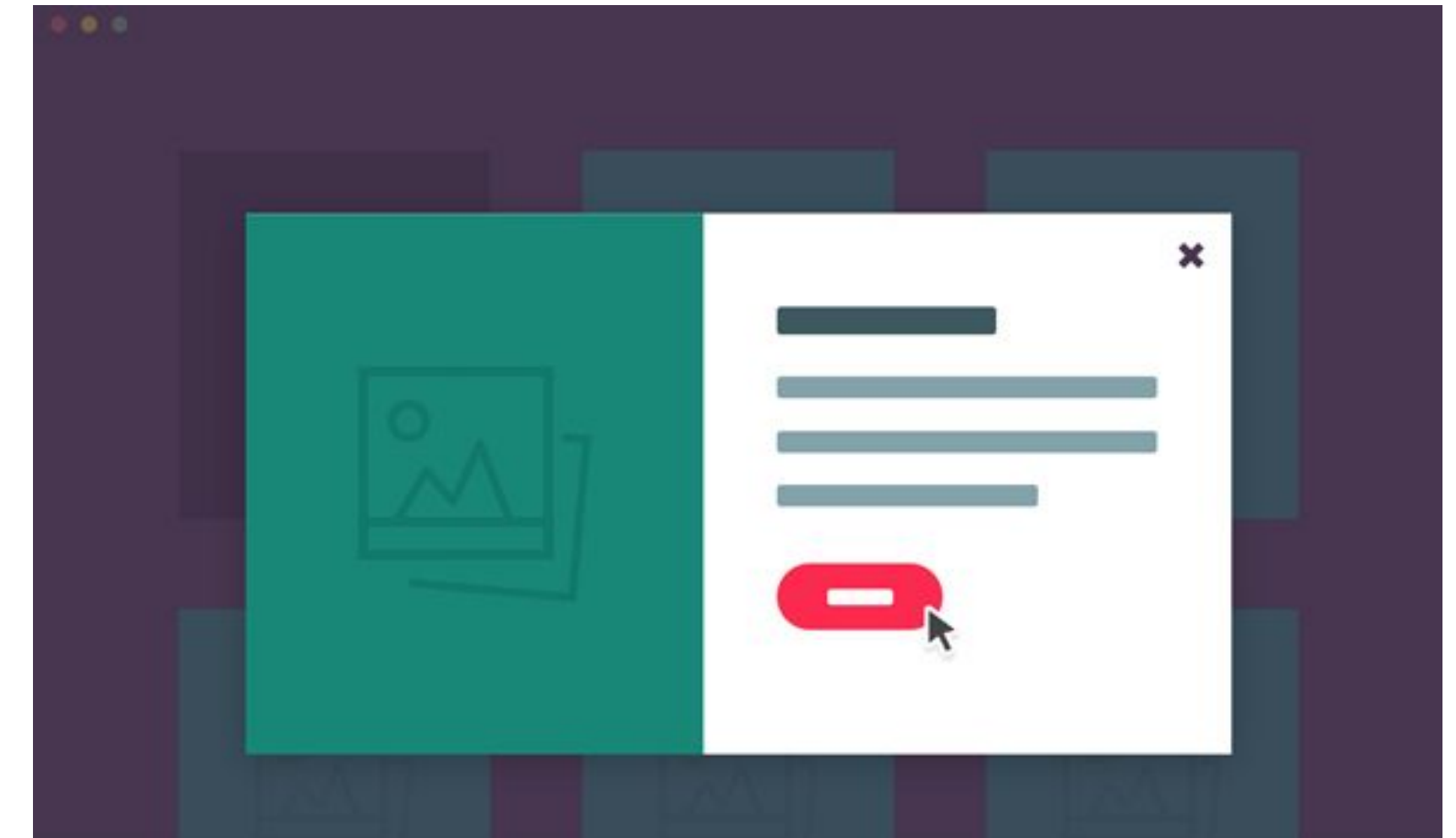


Rendering hidden output

- Output hidden in modal windows or accordions
- Default:

```
outputOptions(x, name, suspendWhenHidden = FALSE)
```

- Solution: *suspendWhenHidden* = **TRUE**



COMPLEX REACTIVITY



Render on demand

- **reactiveValue** breaks the reactivity chain if a value doesn't update
- force the rendering of the component using **reactive trigger**
- concept introduced by Joe Cheng, the author of Shiny
- button that triggers reactivity, but programmatically

```
make_trigger <- function() {  
  simple_value <- shiny::reactiveValues(a = FALSE)  
  list(  
    depend = function() {  
      simple_value$a  
      invisible()  
    },  
    trigger = function() {  
      simple_value$a <-  
        shiny::isolate(ifelse(simple_value$a, FALSE, TRUE))  
    }  
  )  
}
```



SCALE UP



Scale

SCALE-UP

...

LOAD BALANCER

Improves the distribution of
workloads across multiple
servers

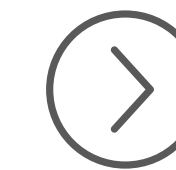
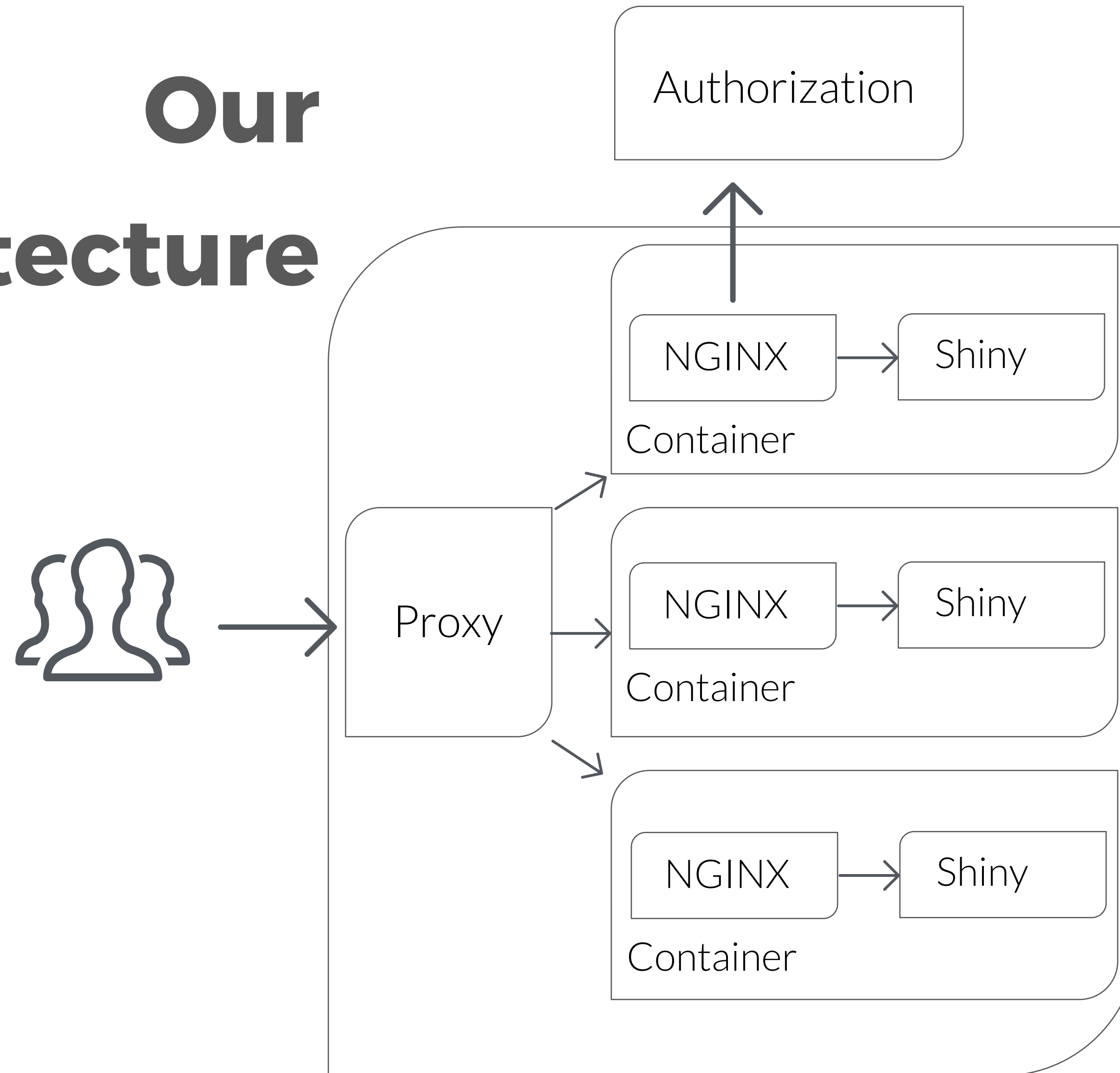
CONTAINERS

Each Shiny Server has the same
configuration, and we run the
same app in each container

AUTH

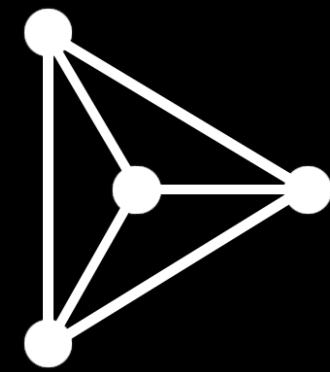
We move authorization to
a separate layer of our
stack

Our Architecture



COMPATIBLE WITH:





Appsilon
Data Science

**Data Science teams can efficiently scale
production ready Shiny Apps with a UI/UX
focus, quickly and aesthetically**

QUESTIONS?

olga_mie



appsilondatascience.com



olga@appsilondatascience.com

