

# Tracking shiny app usage

## A Reivew of tools

WMRUG

Aaron Clark  
Principal Data Scientist  
Biogen  
2022-07-12

# What is *user adoption*?

User adoption, sometimes just called onboarding, is the process by which new users become acclimated to a product or service and decide to keep using it. Users only adopt a product if it helps them achieve a goal of theirs.

— Jenny Booth

Source: [mixpanel.com](https://mixpanel.com)

# Why do I care?

- As a shiny developer, we often times spend vast amounts of time programming a user experience everyone can appreciate, so having people use and appreciate your hard work feels good, and it means you've created something truly valuable.

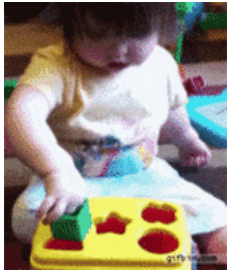


gif source

# If people aren't using your app

You probably...

- misunderstood your stakeholders goals/ needs
- made it too difficult to achieve their goals/ needs
- Any others?



Maybe it means going back to the drawing board to re-assess your approach.

gif source

# The strategy

■ "Help users fall into a pit of success!" - Hadley Wickham



How do we do this this?

**Measure, measure, measure**

[gif source](#)

# Measure, measure, measure

- Before writing code & at each iteration of the project, understand users:
  - motivations
  - needs
  - environments
  - beliefs
  - complaints



- After the project launch, continue assessing.

# What if...

You have limited communication with end users?

OR

You aren't getting the whole story via user feedback?

## Trust me, you aren't!

# Enter: Shiny App Usage Tracking

Every app should have it. Be able to answer questions like:

- Is my app even being used?
- How many users are there?
- Is a specific person using my app (like the CEO)?
- How often are they using it?
- How long are they using it?
- Which features or workflows are being used most/least?
  - Does that align with my expectations?
  - Do I need to enhance the user friendliness of certain widgets to increase the adoption of features?
- Do I need to spend extra time marketing my application to my target users, or perhaps market certain features?

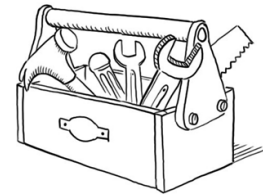


# Tools

There are many, and which one you choose will depend on your end goal

# Today's learning objectives:

- Gain a general working knowledge of
  - bit.ly (yep)
  - Google Analytics
  - Matomo Analytics
  - {shinylogs}
  - {shiny.stats}
  - RSCONnect instrumentation data
- Understand when & where to use each tool
- Encourage you: use them!



bit.ly

Don't overlook the power of monitoring clicks!

# bit.ly

What is it?

- Basically a free URL shortener that also counts URL clicks

Why use it?

- Know you if users are opening your app
- Also great for conferences when demo-ing an app to measure audience engagement
- Handy elsewhere - for example: on your app's documentation link(s)!

When to use?

- I'd recommend for public-facing URLs, since you don't want to "upload" a private URL to a public site like bit.ly.

# How do I create a bit.ly link?

Copy & paste a (long) link into their link shortener, and customize the back half to start collecting stats!

**<https://rinpharma.shinyapps.io/IDEAfilter/>**

[Edit](#)

Jul 5, 12:58 PM by [Aaron Clark](#)

[bit.ly/demo\\_IDEAfilter](#)

[Copy](#)

[Tweet](#)

[QR Code](#)

[Destination:](#) <https://rinpharma.shinyapps.io/IDEAfilter/>

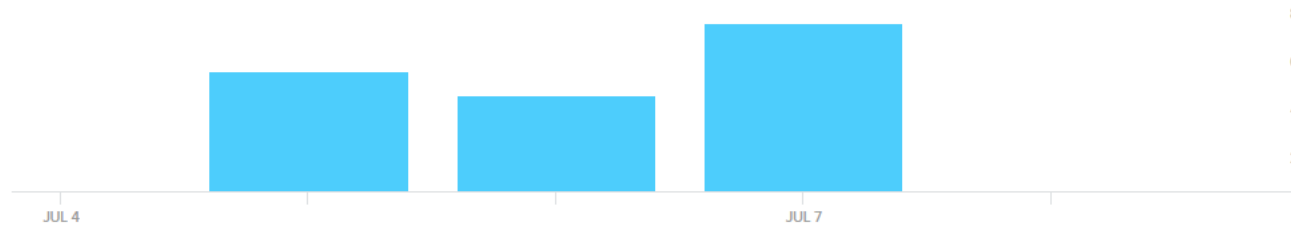
[Redirect](#)

[Add to link-in-bio page](#)

[Add tags](#)

**16**

TOTAL CLICKS



Easy peasy

# Google Analytics

Google Analytics is a free service that collects information about who visits your website and what they do while they're there.

Best used when you have a public facing app (i.e. [shinyapps.io](https://shinyapps.io))

# Google Analytics: Getting Started

- Create an account (analytics.google.com)
- Find your tracking code for your website (aka property)

```
/*<!-- Global site tag (gtag.js) - Google Analytics -->*/  
<script async src="https://www.googletagmanager.com/gtag/js?id=G-ThisIsNotReal">  
<script>  
  window.dataLayer = window.dataLayer || [];  
  function gtag(){dataLayer.push(arguments);}  
  gtag('js', new Date());  
  
  gtag('config', 'G-ThisIsNotReal');  
</script>
```

# Google Analytics: Getting Started (cont'd)

- Add it to the app

```
# somewhere inside the UI
shiny::tags$head(
  shiny::tags$script(
    src = "https://www.googletagmanager.com/gtag/js?id=G-ThisIsNotReal",
    async = ""
  ),
  shiny::tags$script(
    src = "www/gtag.js"
  )
)
```



# Google Analytics: Getting Started (cont'd)

- Redeploy & test

After deployment & visiting shiny app site, you may have to wait about 1 - 2 minutes (max) before data stream flowing into GA report.

Views by Page title and screen name

#1 tidyCDISC

2

100%



PAGE TITLE AND S...

VIEWS

tidyCDISC

2

# Track events with GA

Note: requires some javascript knowledge!

# Tracking events w/ GA

Let's say you have the following dropdown input, like in the [forest ranges shiny app](#).

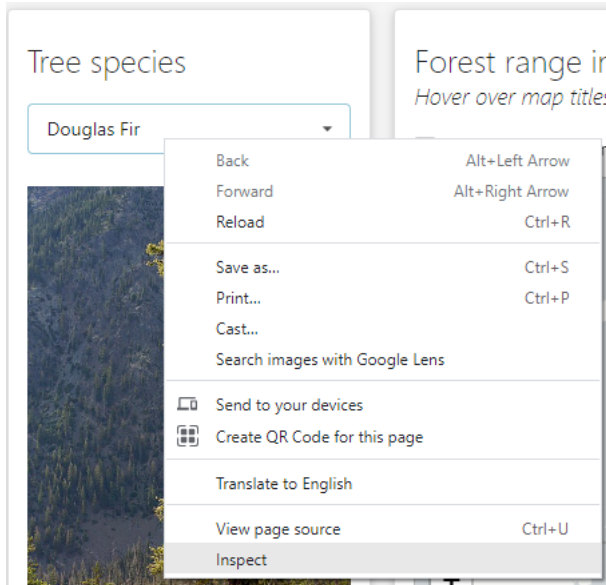
```
shiny.semantic::dropdown_input(  
  input_id = ns("tree_specie"),  
  choices = c("Baltic Pine", "European silver fir", "Douglas fir"),  
  choices_value = c("Pinus Sylvestris", "Abies Alba", "Albus Dumbledore"),  
  default_text = "Tree species",  
  value = "Douglas fir"  
)
```

## Finding an element

`dropdown_input` generates a dropdown menu where each element has a class `item`, so since this is our only dropdown in the app, we can find this element pretty easily using `$('.item')` jQuery Syntax.

# Finding an element can be tough

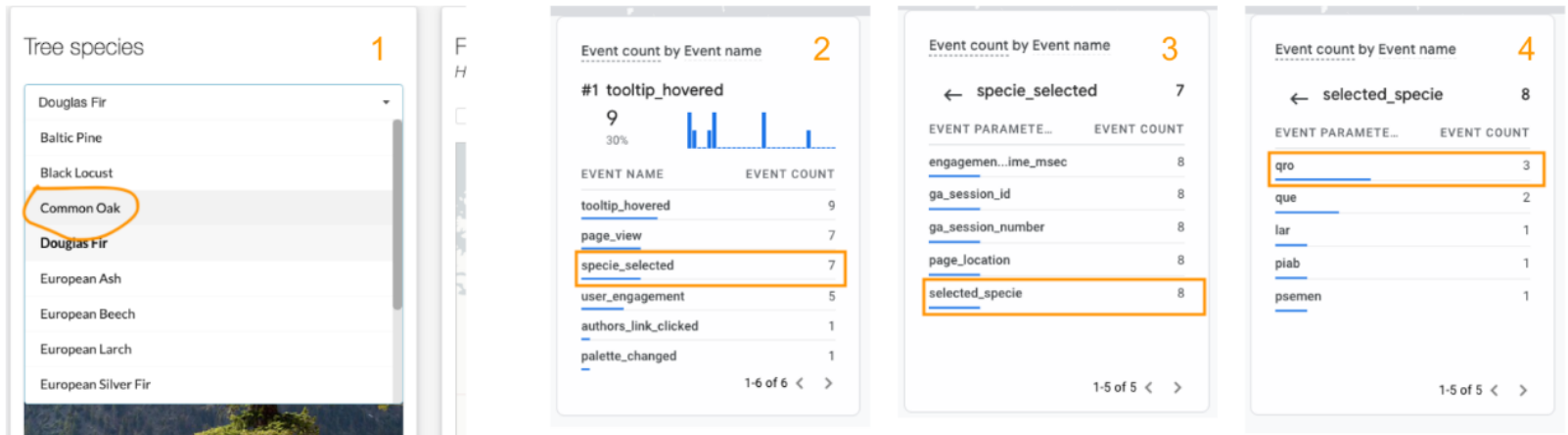
In chrome, right click on the element and select `inspect` to pull up the developer console. There, you can weed through the html looking for classes or `ids` specific to the element:



# Add an event listener

```
$('.item').on('click', (event) => {  
  const selectedSpecie = $(event.currentTarget).data().value;  
  gtag('event', 'specie_selected', {  
    selected_specie: selectedSpecie  
  });  
});
```

^^^ Sends the data to GA:



# Matomo Analytics

Google Analytics, but open source, offering an on-prem opportunity. I suggest using when you have a private, enterprise app (i.e. internal Shiny Server or RS Connect).

Basically the same approach as GA, so we'll skip the "how to".

{shinylogs}

# {shinylogs}

Logging tool for Shiny applications: record inputs or outputs changes, and info about user's session. All recording is done client-side to not slow down the application and occupy the server.

## Install

```
# From CRAN
install.packages("shinylogs")

# From Github
remotes::install_github("dreamRs/shinylogs")
```



# {shinylogs} Usage

Call the main function `track_usage()` in server part of application, and specify where to write logs:

```
# UI
use_tracking()

# server
track_usage(storage_mode = store_json(path = "logs/"))
```

The function will record :

- **inputs:** each time an input changes, the name, time stamp and value will be saved
- **errors:** errors propagated through outputs
- **outputs:** each time an output is re-generated
- **session:** information about user's browser and the application

# {shiny} logs } Storage modes

Six modes are available:

- **store\_json()**: store logs has separate JSON files (one by session).
- **store\_rds()**: store logs has individually RDS files (one by session).
- **store\_sqlite()**: store logs in a SQLite database.
- **store\_null()**: don't write logs on disk, print them in R console.
- **store\_custom()**: use a custom function to save logs wherever you want.
- **storegoogledrive()**: store logs(as JSON files) in Googledrive.

# Deployment with {shinylogs}

Each storage mode comes in handy for different deployment environment.

Deployment with {shinylogs}

# Shiny server, RStudio Connect

On a server, if you want to save logs on disk, don't forget to set write permission on the folder you want to save logs.

On RStudio Connect, you need to use an absolute path to specify the directory where to save logs. You can find more information here: [Persistent Storage on RStudio Connect](#).

# shinyapps.io

There's no persistent data storage! So you can't save logs as JSON or RDS files, you have to use a **remote storage method**. For example, you can send logs to Google Drive with `store_gogledrive` or use `store_custom` to send logs wherever you want (dropbox, a remote database, etc...).

To use Google Drive, you'll need to work with Google's API and set auth to your account, see [{gargle} documentation](#) for examples and how to.

# ShinyProxy

With **ShinyProxy**, you can use a Docker volume to write logs outside of the application container. In `application.yml`, you use can something like this in the specs describing the application:

```
container-volumes: [ "/var/log/shinylogs:/root/logs" ]
```

`/var/log/shinylogs` is a directory on the server where you deploy your applications with ShinyProxy. `/root/logs` is a directory inside your Docker image, it's the path you can use in `track_usage()`, e.g. :

```
track_usage(  
  storage_mode = store_json(path = "/root/logs")  
)
```

Information recorded by {shinylogs}

# Session Info

Metadata about the application and the user's browser:

- **app**: name of the application
- **user**: name of the user (if using Shiny-server pro for example)
- **server\_connected**: when application has been launched (server time)
- **sessionid**: a session ID to match the session with other recorded tables
- **server\_disconnected**: when the application was disconnected (server time)
- **user\_agent**: browser user-agent
- **screen\_res**: resolution of the user screen (width x height)
- **browser\_res**: resolution of the user browser (width x height)
- **pixel\_ratio**: pixel ratio of the browser
- **browser\_connected**: when application has been launched. Uses browser time, which depends on user timezone.



## Session: example

```
#>      app      user      server_connected      sessionid
#> 1 iris-cluster dreamRs 2019-06-19 14:07:09 9815194cfaa6317fbea68ae9537d63d1
#> 2 iris-cluster dreamRs 2019-06-19 14:56:32 0feeacf3201f5cca088059cec2b0a710
#> 3 iris-cluster dreamRs 2019-06-19 14:57:45 8b12c138f159cc5805e136338ddac59
#> 4 iris-cluster dreamRs 2019-06-19 15:03:00 f3950a1588b78d45c53c756a4136df67
#> 5 iris-cluster dreamRs 2019-06-19 15:08:53 254cafb3d5866384f13022d066546cae
#> 6 iris-cluster dreamRs 2019-08-04 10:58:47 8a431f4b90b3b1926047dd2539be0793
#>      server_disconnected
#> 1 2019-06-19 14:07:17
#> 2 2019-06-19 14:57:39
#> 3 2019-06-19 15:01:36
#> 4 2019-06-19 15:08:13
#> 5 2019-06-19 15:09:38
#> 6 2019-08-04 10:58:49
#>
#>      user_agent
#> 1 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.169 Safari/537.36
#> 2 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.169 Safari/537.36
#> 3 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.169 Safari/537.36
#> 4 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.169 Safari/537.36
#> 5 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.169 Safari/537.36
#> 6
<NA>
#>      screen_res browser_res pixel_ratio browser_connected
#> 1 1920x1080 1574x724 1 2019-06-19 14:07:09
#> 2 1920x1080 1920x937 1 2019-06-19 14:56:32
#> 3 1920x1080 1920x937 1 2019-06-19 14:57:45
#> 4 1920x1080 1920x937 1 2019-06-19 15:03:00
#> 5 1920x1080 1920x937 1 2019-06-19 15:08:53
#> 6 <NA> <NA> NA <NA>
```

# Inputs

Data about inputs, by default all inputs are recorded (even those not define by developer, like with {htmlwidgets} : {DT}, {leaflet}, ...)

- **sessionid**: the same ID as in session object
- **name**: the inputs inputId
- **timestamp**: timestamp when the input has changed
- **value**: the value taken by the input (can be a list in case of complex input)
- **type**: type of input (if defined)
- **binding**: binding for the input (if defined)

Information recorded by {shinylogs}

## Inputs: example

```
#>           sessionid      name      timestamp      value
#> 1 9815194cfaa6317fbea68ae9537d63d1      xcol 2019-06-19 14:07:11 Sepal.Width
#> 2 9815194cfaa6317fbea68ae9537d63d1 clusters 2019-06-19 14:07:15           6
#> 3 9815194cfaa6317fbea68ae9537d63d1 clusters 2019-06-19 14:07:15           4
#> 4 9815194cfaa6317fbea68ae9537d63d1 clusters 2019-06-19 14:07:15           5
#> 5 9815194cfaa6317fbea68ae9537d63d1 clusters 2019-06-19 14:07:14           7
#> 6 9815194cfaa6317fbea68ae9537d63d1 clusters 2019-06-19 14:07:13           4

#>           type      binding
#> 1      <NA> shiny.selectInput
#> 2 shiny.number shiny.numberInput
#> 3 shiny.number shiny.numberInput
#> 4 shiny.number shiny.numberInput
#> 5 shiny.number shiny.numberInput
#> 6 shiny.number shiny.numberInput
```

# Outputs

Data recorded each time an output is refreshed:

- **sessionid**: the same ID as in session object
- **name**: outputId of the output
- **timestamp**: timestamp when the output has been updated
- **type**: type of output (if defined)
- **binding**: binding for the output (if defined)

Information recorded by {shinylogs}

## Outputs: example

```
#>
#> 1 9815194cfaa6317fbae68ae9537d63d1 plot1 2019-06-19 14:07:15 shiny.imageOutput
#> 2 9815194cfaa6317fbae68ae9537d63d1 plot1 2019-06-19 14:07:15 shiny.imageOutput
#> 3 9815194cfaa6317fbae68ae9537d63d1 plot1 2019-06-19 14:07:14 shiny.imageOutput
#> 4 9815194cfaa6317fbae68ae9537d63d1 plot1 2019-06-19 14:07:15 shiny.imageOutput
#> 5 9815194cfaa6317fbae68ae9537d63d1 plot1 2019-06-19 14:07:11 shiny.imageOutput
#> 6 9815194cfaa6317fbae68ae9537d63d1 plot1 2019-06-19 14:07:10 shiny.imageOutput
```

## Errors

Errors are recorded only when propagated through an output, this is the red message users see in application, info saved are:

- **sessionid**: the same ID as in session object
- **name**: outputId of the output where an error happened
- **timestamp**: timestamp of the error
- **error**: error message (if any)
- **value**: additional data for the error (generally NULL) (if defined)

Information recorded by {shinylogs}

## Errors: example

```
#>           sessionid  name      timestamp
#> 1 f8f50a3743023aae7d0d6350a2fd6841 plot1 2019-06-19 14:07:18
#>                                     error
#> 1 NA/NaN/Inf in foreign function call (arg 1)
```

# {shiny.stats}

Yet another way to log user data



# {shiny.stats}

Easy way for logging users' activity **and** creating a dashboard to ingest the data. The only requirement is that you have an accessible database, which can be as simple as a local PostgreSQL database or a sqlite file.

First step is installation:

```
devtools::install_github("Appsiilon/shiny.stats")
```

# {shiny.stats} Set up

The next step is to initialize the database you want to use with shiny.stats. You should only do this once. The snippet below will connect to a sqlite database named user\_stats:

```
connection <- DBI::dbConnect(RSQLite::SQLite(), dbname = "user_stats.sqlite")  
DBI::dbDisconnect(connection)
```

# {shiny.stats} Set up (cont'd)

Now comes the potentially tricky part (depending on how your app's authentication is set up): you'll need to define a function to extract the username. The below example will extract it from the URL, from a dedicated username parameter.

```
get_user <- function(session) {  
  parseQueryString(isolate(session$clientData$url_search))$username  
}
```

# {shiny.stats} Server Side Logic

The magic happens in the server function where the database connection is established and user behavior is monitored:

```
# creates user connection list and ensures required tables exist in DB
user_connection <- initialize_connection(connection,
                                         username = get_user(session))

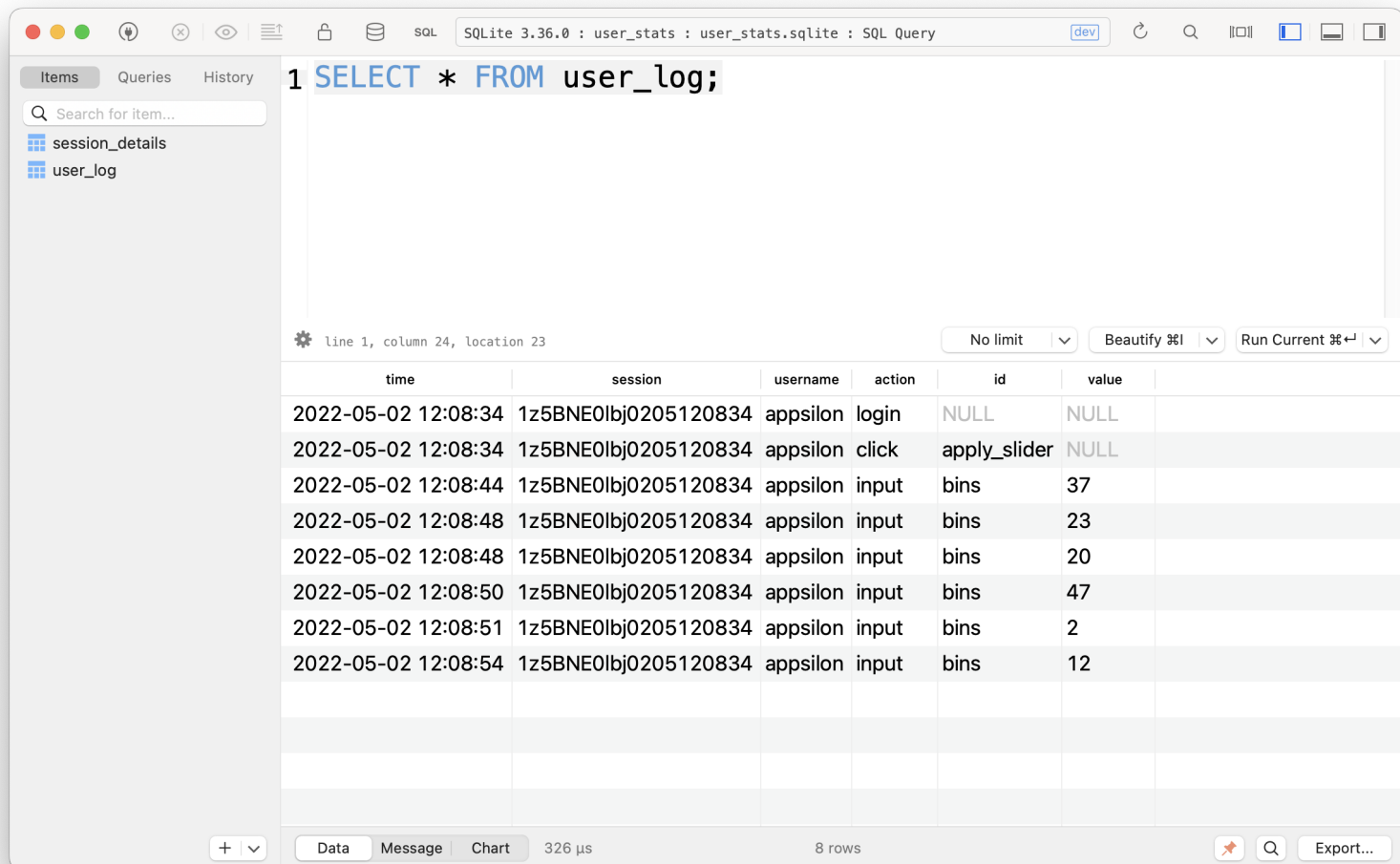
# register login
log_login(user_connection)

# register actions to watch...
log_click(user_connection, id = "apply_slider") # button
log_input(user_connection, input, input_id = "bins") # slider

# registering logout
log_logout(user_connection)
```

# View {shiny.stats} database

Open the sqlite file in your favorite database management tool



The screenshot shows a SQLite database management tool interface. The title bar indicates the database is 'SQLite 3.36.0 : user\_stats : user\_stats.sqlite : SQL Query'. The left sidebar shows a tree view with 'session\_details' and 'user\_log'. The main query editor contains the SQL query: `1 SELECT * FROM user_log;`. Below the query editor, there are controls for 'No limit', 'Beautify %I', and 'Run Current %I'. The query result is displayed in a table with 7 columns: 'time', 'session', 'username', 'action', 'id', 'value', and an empty column. The table contains 8 rows of data, all from the 'user\_log' table, showing user actions like 'login', 'click', and 'input' with associated session IDs and values.

| time                | session              | username | action | id           | value |  |
|---------------------|----------------------|----------|--------|--------------|-------|--|
| 2022-05-02 12:08:34 | 1z5BNE0Ibj0205120834 | appsilon | login  | NULL         | NULL  |  |
| 2022-05-02 12:08:34 | 1z5BNE0Ibj0205120834 | appsilon | click  | apply_slider | NULL  |  |
| 2022-05-02 12:08:44 | 1z5BNE0Ibj0205120834 | appsilon | input  | bins         | 37    |  |
| 2022-05-02 12:08:48 | 1z5BNE0Ibj0205120834 | appsilon | input  | bins         | 23    |  |
| 2022-05-02 12:08:48 | 1z5BNE0Ibj0205120834 | appsilon | input  | bins         | 20    |  |
| 2022-05-02 12:08:50 | 1z5BNE0Ibj0205120834 | appsilon | input  | bins         | 47    |  |
| 2022-05-02 12:08:51 | 1z5BNE0Ibj0205120834 | appsilon | input  | bins         | 2     |  |
| 2022-05-02 12:08:54 | 1z5BNE0Ibj0205120834 | appsilon | input  | bins         | 12    |  |
|                     |                      |          |        |              |       |  |
|                     |                      |          |        |              |       |  |
|                     |                      |          |        |              |       |  |

At the bottom of the interface, there are tabs for 'Data', 'Message', and 'Chart', along with a status bar showing '326 μs', '8 rows', and an 'Export...' button.

# Built-in {shiny.stats} dashboard

The most handy feature of {shiny.stats}? You can easily display users' stats in an R Shiny dashboard. The following example code snippet creates a dashboard that allows you to monitor user adoption:

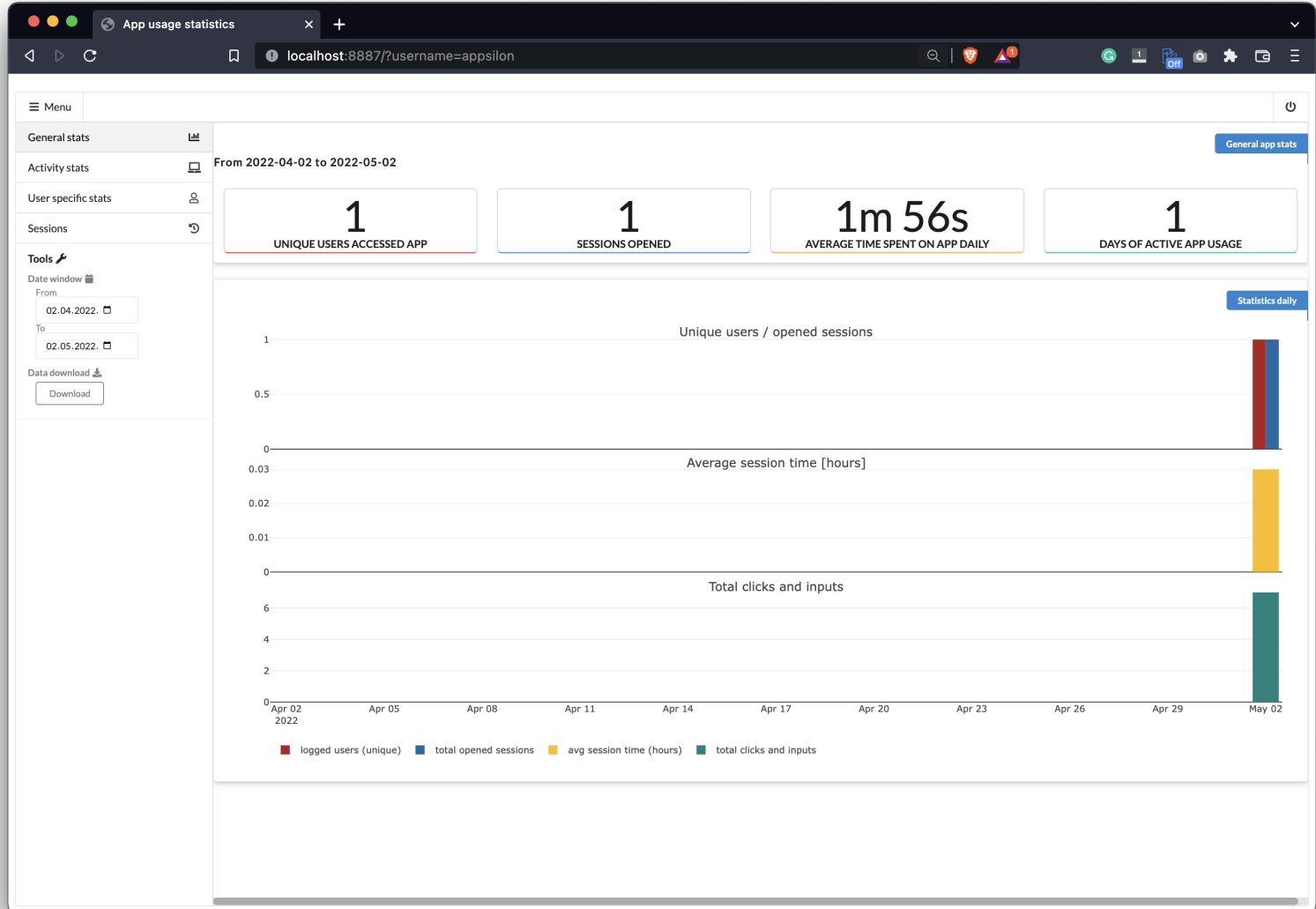
```
library(shiny);library(RSQLite);library(shiny.stats);

# prepare credentials list to access logs:
db_credentials <- list(DB_NAME = "user_stats.sqlite",
                      DB_DRIVER = "SQLite")

# define function for extracting username
get_user <- function(session) {
  username <- isolate(parseQueryString(session$clientData$url_search)$username)
  req(username)
  return(username)
}

# define ui and server
ui <- shiny_stats_ui()
server <- shiny_stats_server(get_user, db_credentials = db_credentials)
shinyApp(ui = ui, server = server,
         options = list(port = 8887, launch.browser = FALSE))
```

# Example {shiny.stats} dashboard



# RSConnect Instrumentation Data



# Extending Connect with the Server API

The **Server API** comes with the RStudio Connect product (automatically). I.E. this is NOT an API you develop with plumber, flask, etc.

Resources:

- [Documentation](#)
- [Cookbook](#)
- **R Client** - {connectapi} not on CRAN (yet)

```
remotes::install_github('rstudio/connectapi')
```

## Teaser

Future goal of {connectapi} - anything clickable in the UI can be mimicked programmatically via the API.

# The API's instrumentation data can answer...

- How many reports on the server?
- Which users have joined?
- Which users have access to which reports?
- Who looked at this report recently?

And the Rstudio team is working on many experimental feature requests as we speak.

# Connect to the Connect Server API

```
library(connectapi)

client <- connect(
  server = 'https://connect.example.com',
  api_key = '<SUPER SECRET API KEY>'
)

# OR, if your server is defined by your environment variables, run:
client <- connect()
```

If your role is "Publisher", you only get access to your content hosted on RSConnect! The "Admin" role gives you access to all content.

```
users <- get_users(client) # email, username, first & last name, role, guid
groups <- get_groups(client) # group info - guid & group name
usage_shiny <- get_usage_shiny(client) # content & user guids, start and stop times
usage_static <- get_usage_static(client) # content & user guids, timestamp visited
some_content <- get_content(client) # guid, content's name, title, description, etc

# get all content
all_content <- get_content(client, limit = Inf)
```

# Example Instrumentation Data

For a shiny app:

```
usage_shiny <- get_usage_shiny(client) # content & user guids, start and stop times
```

|    | content_guid                         | user_guid                            | started             | ended               | data_version |
|----|--------------------------------------|--------------------------------------|---------------------|---------------------|--------------|
| 1  | 23d70dc2-d00c-4bc7-8d5c-a503670613f5 | NA                                   | 2021-12-02 02:39:45 | 2021-12-02 02:40:03 | 1            |
| 2  | d0a3b89f-bf0e-4c8d-ac2e-caad38848dd7 | NA                                   | 2021-12-02 03:29:48 | 2021-12-02 03:30:08 | 1            |
| 3  | c4f627fc-9e46-4b28-9b62-f6c692a6c694 | 92cb1a74-9bd8-47e2-9755-28afa5b40c56 | 2021-12-02 09:49:24 | 2021-12-02 10:08:27 | 1            |
| 4  | 5957cb9d-4f28-44fc-8b6b-86772a620c90 | NA                                   | 2021-12-02 10:06:33 | 2021-12-02 10:07:11 | 1            |
| 5  | 4ecb5fc7-333d-489d-a863-bc333504a531 | 193be3a3-f984-4944-a97e-6dbdea2d95ca | 2021-12-02 13:19:31 | 2021-12-02 13:19:54 | 1            |
| 6  | 4ecb5fc7-333d-489d-a863-bc333504a531 | 193be3a3-f984-4944-a97e-6dbdea2d95ca | 2021-12-02 13:19:49 | 2021-12-02 13:20:32 | 1            |
| 7  | 4ecb5fc7-333d-489d-a863-bc333504a531 | 193be3a3-f984-4944-a97e-6dbdea2d95ca | 2021-12-02 13:20:17 | 2021-12-02 13:20:44 | 1            |
| 8  | 4ecb5fc7-333d-489d-a863-bc333504a531 | 193be3a3-f984-4944-a97e-6dbdea2d95ca | 2021-12-02 13:20:30 | 2021-12-02 13:21:29 | 1            |
| 9  | 4ecb5fc7-333d-489d-a863-bc333504a531 | 193be3a3-f984-4944-a97e-6dbdea2d95ca | 2021-12-02 13:20:40 | 2021-12-02 13:21:11 | 1            |
| 10 | 4ecb5fc7-333d-489d-a863-bc333504a531 | 193be3a3-f984-4944-a97e-6dbdea2d95ca | 2021-12-02 13:36:15 | 2021-12-02 13:36:18 | 1            |

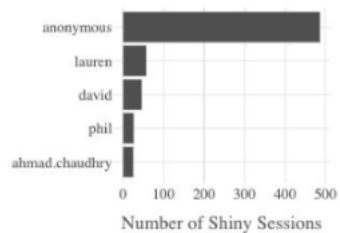
# Example Instrumentation Dashboard

## RStudio Connect Usage - Last 30 Days

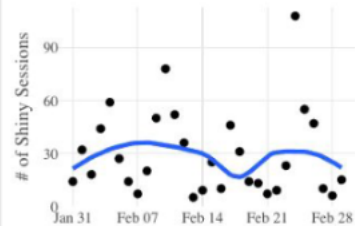
This content summary may contain privileged information. The report is generated using the [RStudio Connect Server API](#) and the source code is [available online](#) if you'd like to customize your analysis. Data is limited to the last 30 days.

The report uses the environment variables `RSTUDIO_CONNECT_SERVER` and `RSTUDIO_CONNECT_API_KEY` to collect the data. To limit the results to a single publisher, use a publisher API key.

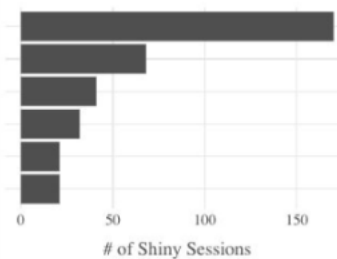
Shiny Sessions by User (Top 5)



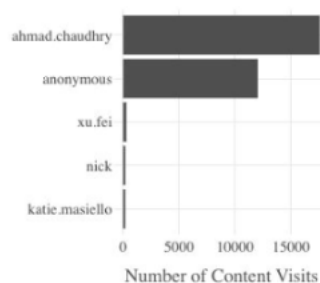
Shiny Sessions Over Time



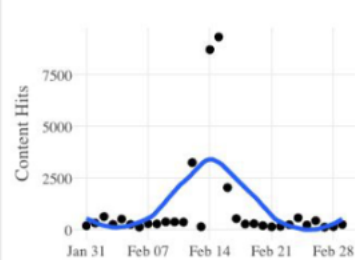
Top Applications



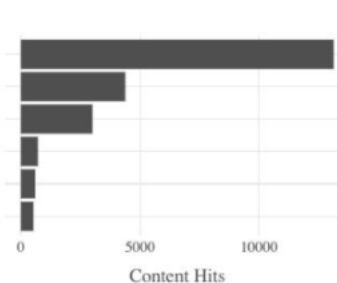
Static Content Hits by User (Top 5)



Static Content Visits Over Time

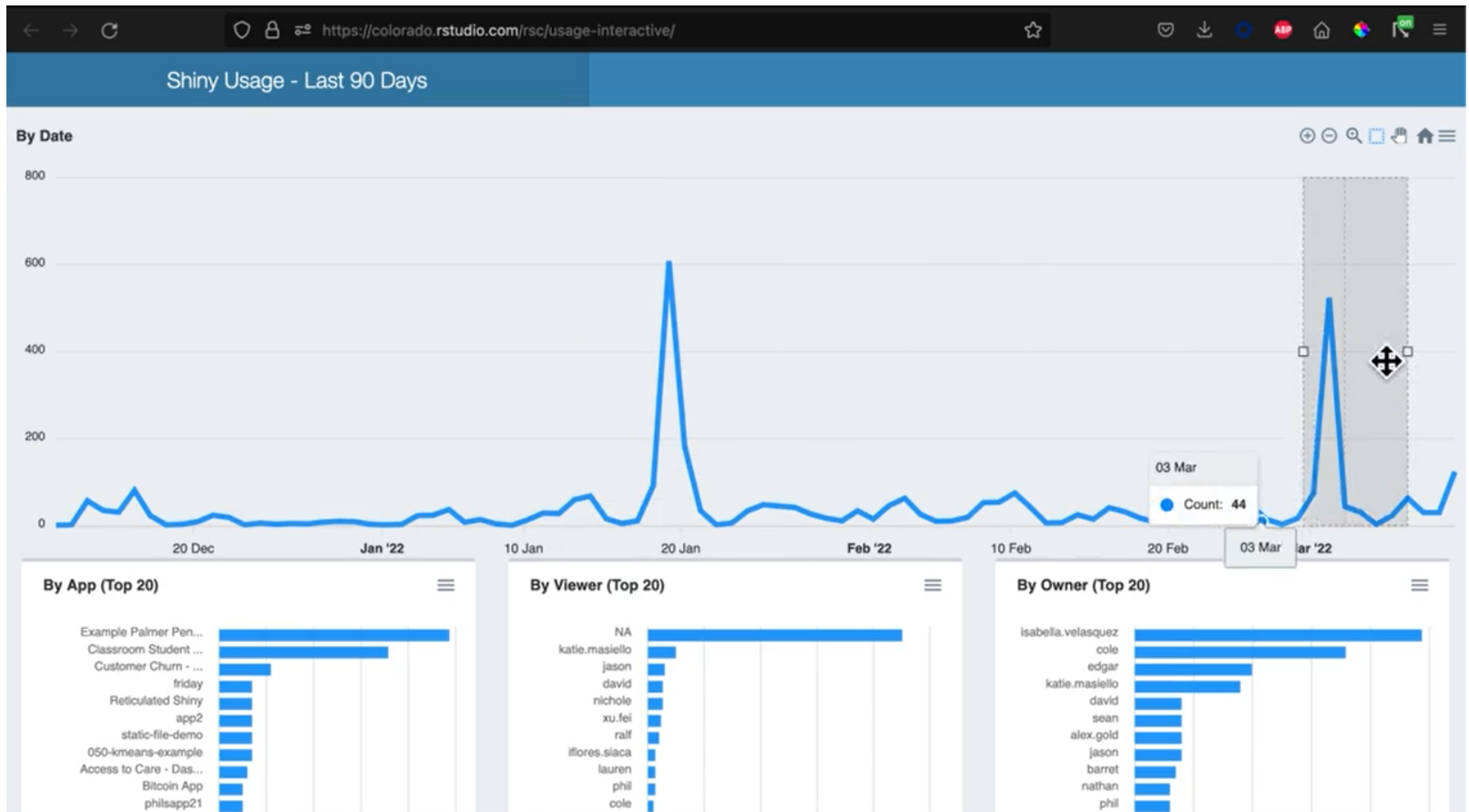


Top Static Content



[GitHub Repo](#)

# Example Instrumentation Shiny App



App: <https://colorado.rstudio.com/rsc/usage-interactive/>

[GitHub Repo](#)

Phew! Tool Review complete.

# In Review

- Did you gain a general working knowledge of
  - bit.ly (yep)
  - Google Analytics
  - Matomo Analytics
  - {shinylogs}
  - {shiny.stats}
  - RSConnect instrumentation data
- I hope you feel encouraged!
- **Measuring** your app's usage make you a well informed developer!



# My Resources

- [R Shiny Google Analytics: How to add GA to shiny Apps](#) By Appsilon
- [Top 3 Tools to Monitor User Adoption in R Shiny](#) By Appsilon
- [{shinylogs} Documentation](#)
- [{connectapi} Documentation](#)
- [Youtube: Shiny Usage Tracking in RStudio Connect](#) By Cole Arendt @ Rstudio

Thank you!

Questions?