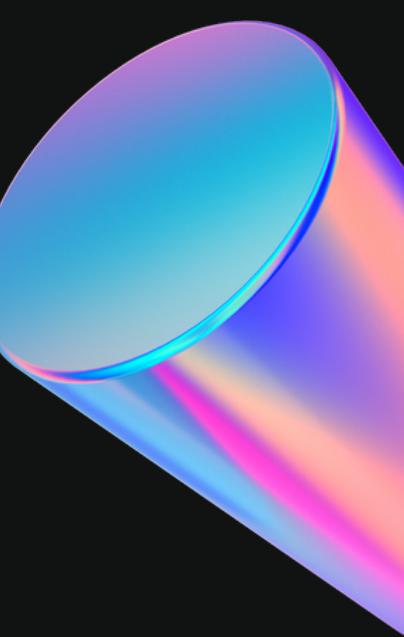


BRIGHT R SHINY OBJECTS

Achievable, shareable and understandable representations of data can help us communicate & collaborate more effectively. Shiny apps give us an easy way to do this.

UQ Advanced R Course - 2021
Dr Bill Venables & Rhett Chappell





Rhetta Chappell

Data Scientist – RIDL at Griffith University

BACKGROUND

- art, design
- anthropology
- research
- data science

PROFESSIONAL INTERESTS

- data visualisation & storytelling
- democratising data – data collaboratives and coops
- actionable insights for social good
- behavioural economics & psychology

Interact. Analyse. Communicate.

- Take a fresh, interactive approach to telling your data story with Shiny
- "Shiny combines the computational power of R with the interactivity of the modern web".
 - <https://shiny.rstudio.com/>

Top 8 reasons why I build Shiny apps:

1. Research translation (PhD students, sharing project outcomes)
2. Generate & share accessible & actionable insights (Community Orgs, Local Councils)
3. Stakeholder engagement
4. Policy advocacy
5. Civic engagement
6. Wireframing / prototyping more complex applications
7. Dashboarding (enterprise level data)
8. Data exploration & analysis

The ShinyR the better...

Shiny App examples:

PRACTICAL:

<https://connect.thinkr.fr/hexmake/>

DATA SCIENCE & OPEN DATA:

<https://nz-stefan.shinyapps.io/blog-explorer/>

- network graph +30,000 blog articles on R

LIFE SCIENCES/PUBLIC INTEREST:

[https://vac-lshtm.shinyapps.io/ncov_tracker/?
_ga=2.90997338.1162613968.1610319694-609160277.1538646407](https://vac-lshtm.shinyapps.io/ncov_tracker/?_ga=2.90997338.1162613968.1610319694-609160277.1538646407)

INDUSTRY:

[https://phillyo.shinyapps.io/intelligentsia/?
_ga=2.168124897.1162613968.1610319694-609160277.1538646407](https://phillyo.shinyapps.io/intelligentsia/?_ga=2.168124897.1162613968.1610319694-609160277.1538646407)

ENTERPRISE:

<https://demo.appsiion.ai/apps/shiny-enterprise-demo/>

POLITICS:

<https://regionalinnovationdatalab.shinyapps.io/qldvotes/>

Why shiny?

- You're already familiar with R
- No web development skills are required
 - Easy to understand interactivity through reactivity
- Quickly prototype and let users interact with your data and your analysis:
 - easily embed into .Rmd, blogdown and websites
 - highly customisable - basic to beautiful ui
 - free to host and share - shinyapps.io, RStudio Connect
 - lots of great packages
 - (shinydashboardPlus, shiny.demantic, shinydashboard, plumbR, leaflet, plotly, DT, data.table)
 - great for stakeholder engagement & collaboration

How does Shiny work?

- R Shiny is an R package which translates R code into html code to be understood by your web browser
 - R code = more concise and intelligible
 - Get started: `install.packages("shiny")`

```
library(shiny)
sliderInput("obs", "Number of observations:",
            min = 0, max = 1000, value = 500
)
```

```
<div class="form-group shiny-input-container">
  <label class="control-label" for="obs">Number of observations:</label>
  <input class="js-range-slider" id="obs" data-min="0" data-max="1000" data-from="500" data-step="1"
        data-grid="true" data-grid-num="10" data-grid-snap="false" data-prettify-separator=","
        data-prettify-enabled="true" data-keyboard="true" data-keyboard-step="0.1" data-type="number"/>
</div>
```

● **UI**

- front end interface
- simpler (in the beginning)
 - all users get same version of HTML
- defines how your app looks

● **SERVER**

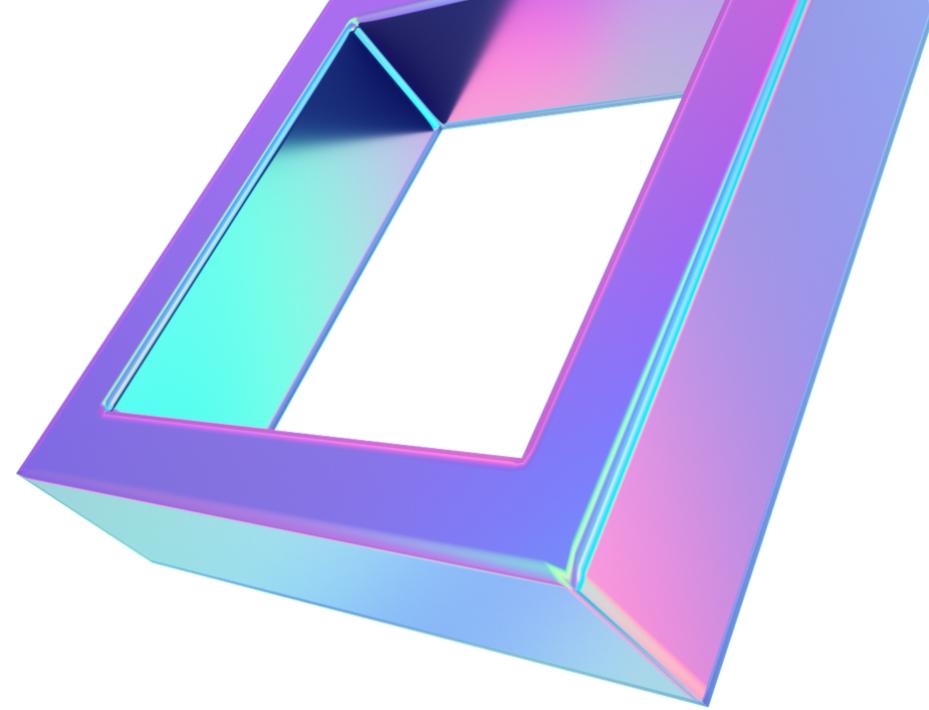
- back end logic
- more complex (in the beginning)
 - each user gets own version of app
- defines how your app works

2 key components of every Shiny app:

```
library(shiny)
#1
ui <- fluidPage(
)

#2
server <- function(input, output, session) {

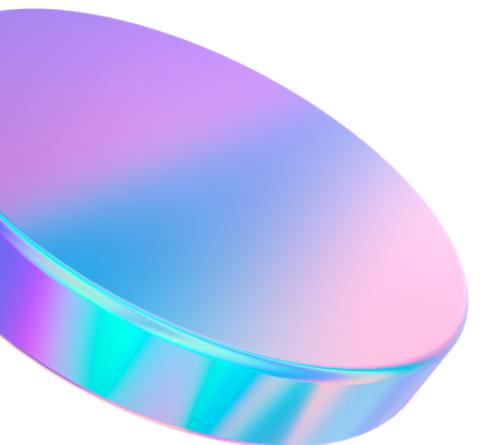
}
shinyApp(ui, server)
```



Server functions take three parameters:

1. input
2. output
3. session

```
server <- function(input, output, session) {  
  }
```





```
server <- function(input, output, session) { }
```

input parameter

- The input argument is a list-like object that contains all the input data sent from the browser, named according to the input ID.

```
ui <- fluidPage(  
  numericInput( id = "count",  
    label = "Number of values",  
    value = 100) #end numeric input  
)#end fluidPage
```

- you can access the value of that input with `input$count`
 - It will initially contain the value 100, and it will be automatically updated as the user changes the value in the browser.
- Unlike a typical list, input objects are read-only.
 - To read from an input, you must be in a reactive context - more on this later

```
server <- function(input, output, session) {}
```

output & input parameters

- Note that the ID is quoted in the UI, but not in the server.

```
ui <- fluidPage(  
  numericInput(id = "count", ↗  
    label = "Number of values",  
    value = 10), # end numeric input  
  
  textOutput(id = "message") # end text output  
  
) #end fluidPage
```

```
server <- function(input, output, session) {
```

```
  ↗  
  output$message <- renderText({  
    paste0("You selected ", input$count, "!") #end RenderText
```

```
}
```



```
server <- function(input, output, session) { }
```

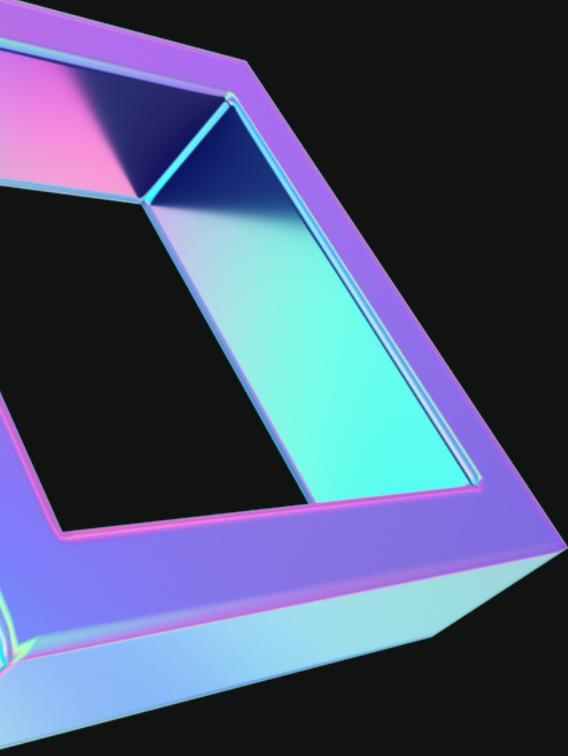
output parameter

- also a list-like object named according to the output ID
- used for sending output instead of receiving input
- always use the `output` object together with a `render` function

```
ui <- fluidPage(  
  textOutput(id = "message")  
)#end fluidPage
```

```
server <- function(input, output, session) {  
  output$message <- renderText("Advanced R 2021!")#end fluidPage  
}
```





```
server <- function(input, output, session) { }
```

output & input parameters are PICKY how you use them

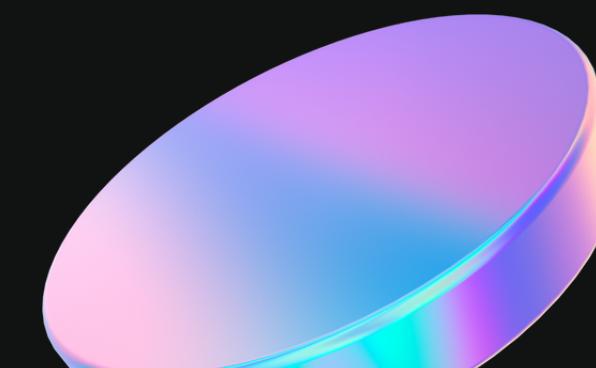
- If you forget the render function - ERROR

```
server <- function(input, output, session) {  
  output$greeting <- "Hello human"  
}  
shinyApp(ui, server)  
#> Error: Unexpected character output for greeting
```



- If you attempt to read from an output - ERROR

```
server <- function(input, output, session) {  
  message("The greeting is ", output$greeting)  
}  
shinyApp(ui, server)  
#> Error: Reading from shinyoutput object is not allowed.
```

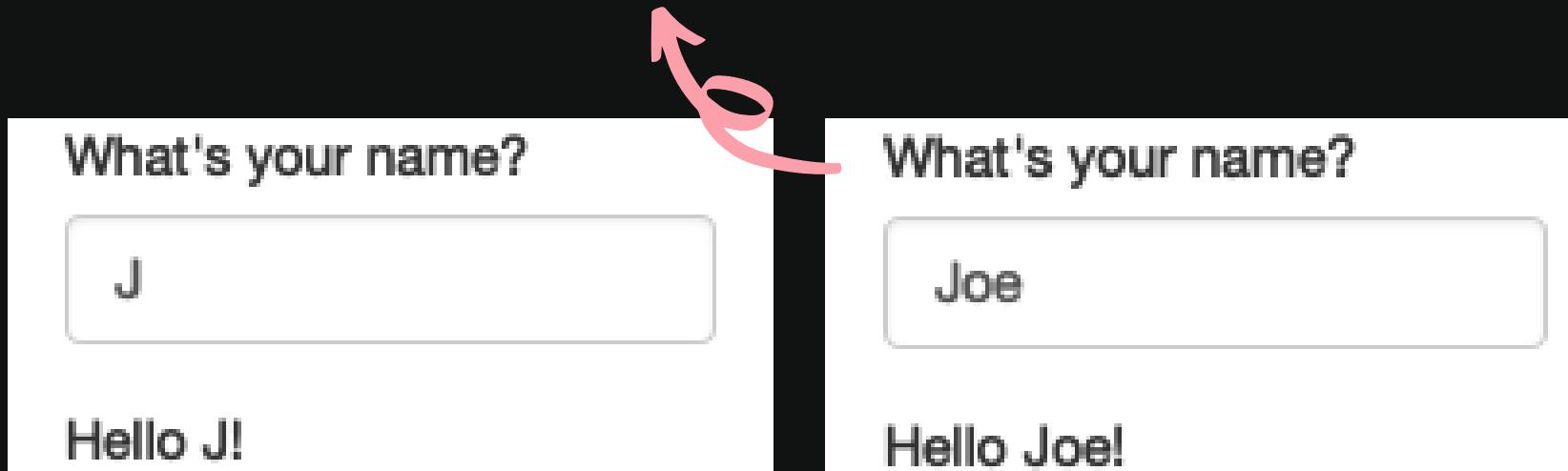


"RECIPES INSTEAD OF COMMANDS"

Copy, Paste & Run:

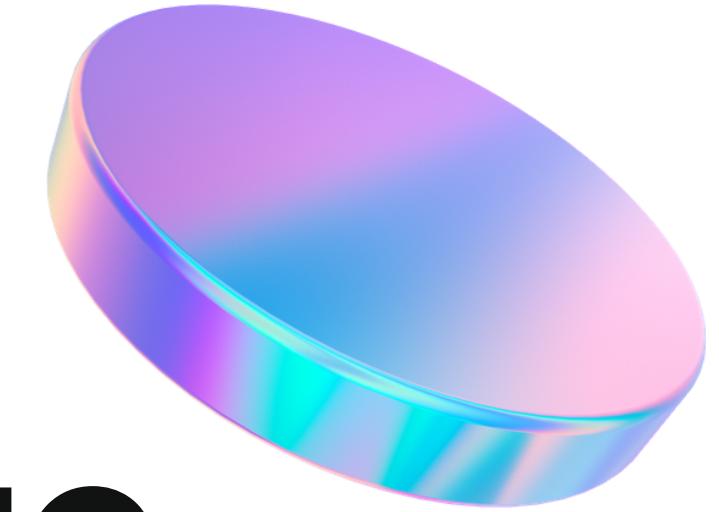
```
ui <- fluidPage(  
 textInput(id = "name",  
            label = "What's your name?"), #end textInput  
  textOutput("greeting")  
) # end fluidPage
```

```
server <- function(input, output, session) {  
  output$greeting <- renderText({  
    paste0("Hello ", input$name, "!") #end RenderText
```



11

IMPERITIVE & DECLARATIVE PROGRAMMING

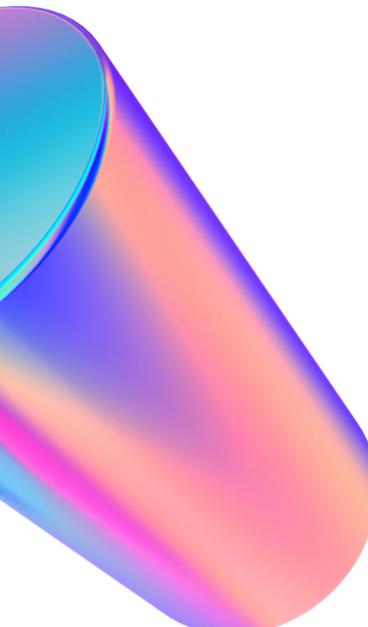


COMMANDS

IMPERITIVE - R SCRIPTS

In imperative programming, you issue a specific command and it's carried out immediately.

"Make me a Greek salad."



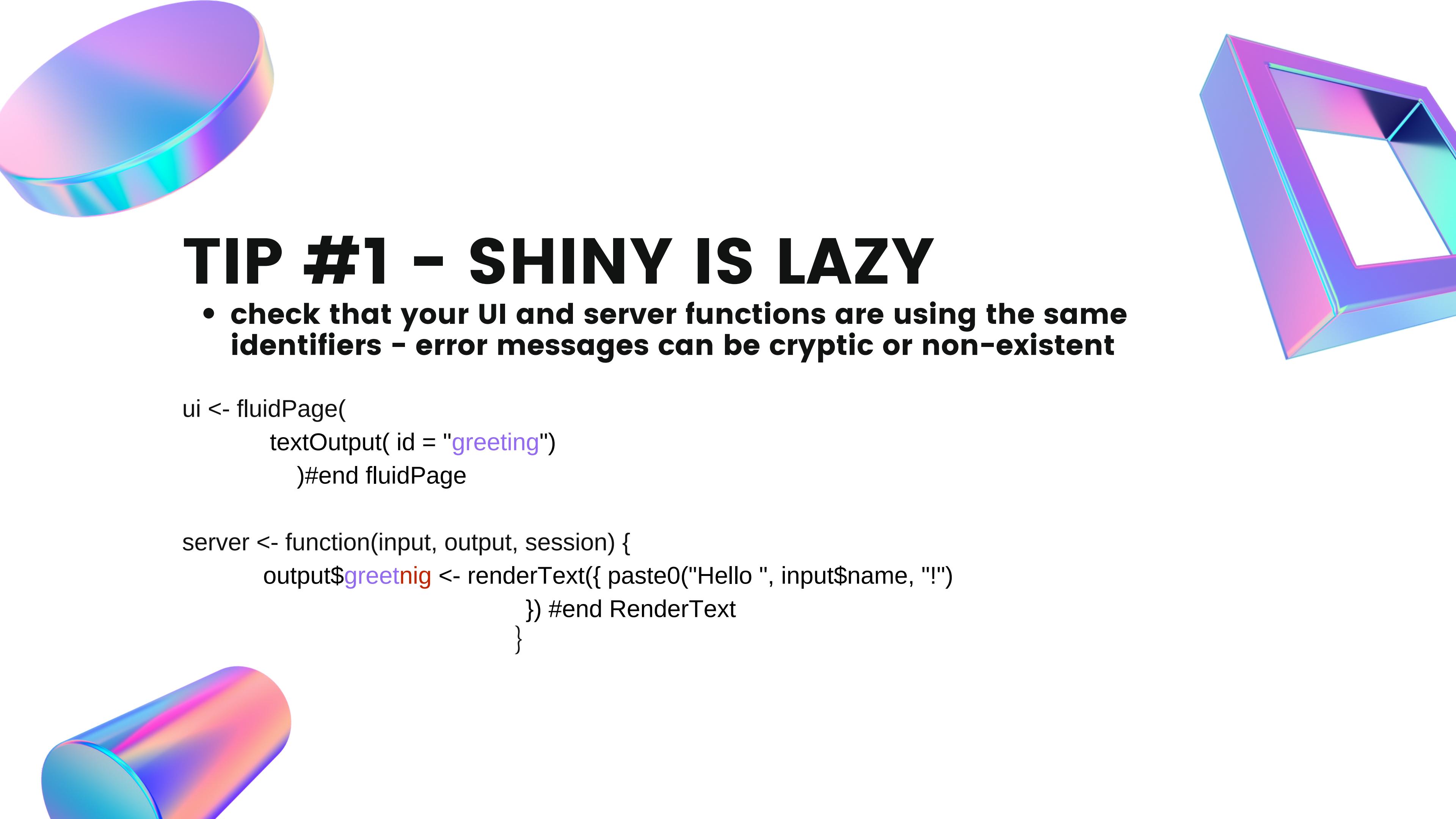
RECIPES

DECLARATIVE - R SHINY

In declarative programming, you express higher-level goals or describe important constraints, and rely on someone else to decide how and/or when to translate that into action.

- can sometimes be tricky to frame what you want this way

"Ensure there is a Greek salad in the fridge whenever I look inside it".



TIP #1 - SHINY IS LAZY

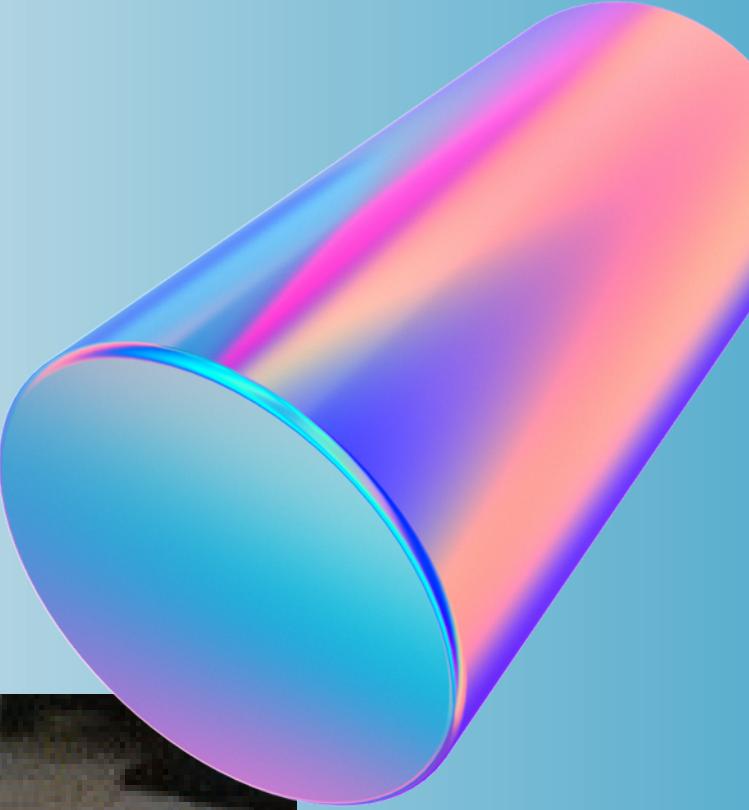
- **check that your UI and server functions are using the same identifiers – error messages can be cryptic or non-existent**

```
ui <- fluidPage(  
  textOutput( id = "greeting")  
 )#end fluidPage
```

```
server <- function(input, output, session) {  
  output$greetnig <- renderText({ paste0("Hello ", input$name, "!"})  
    }) #end RenderText  
}
```

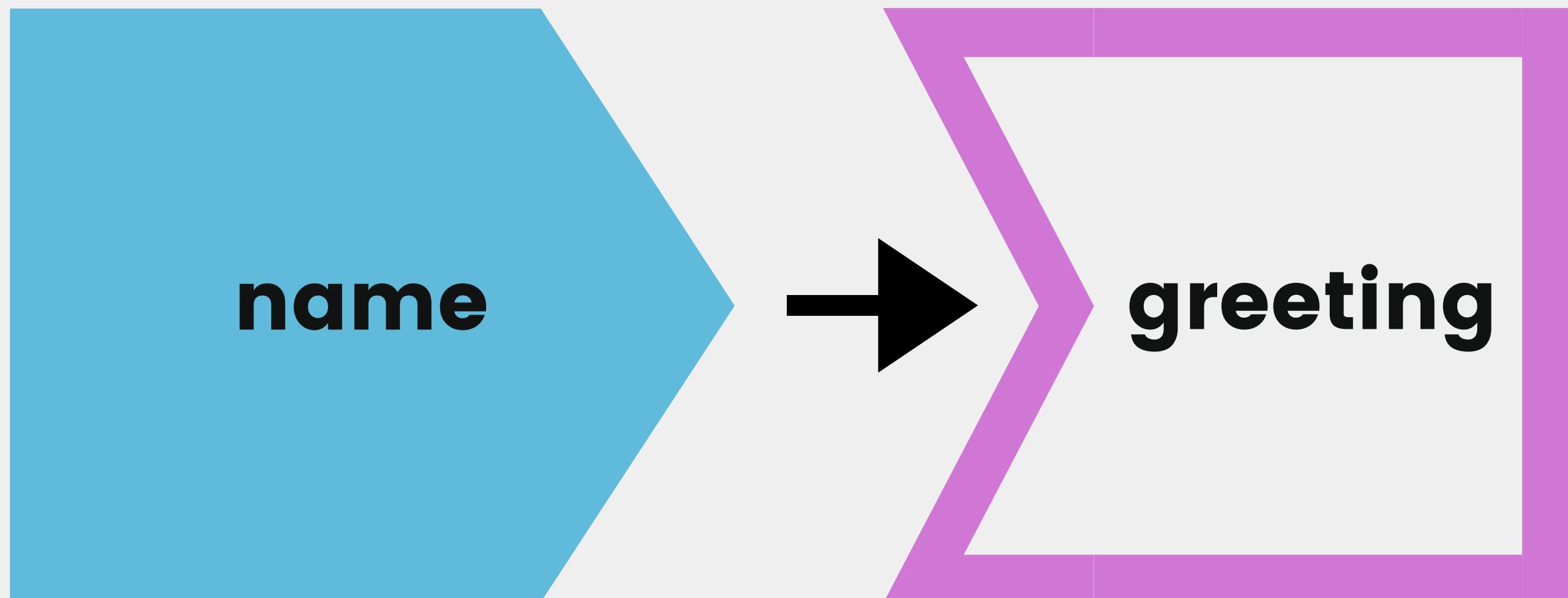
Order of execution

- Not top to bottom
- Only executes as needed
 - user interaction
 - unique sessions



Reactivity

Shiny uses a reactive programming model.



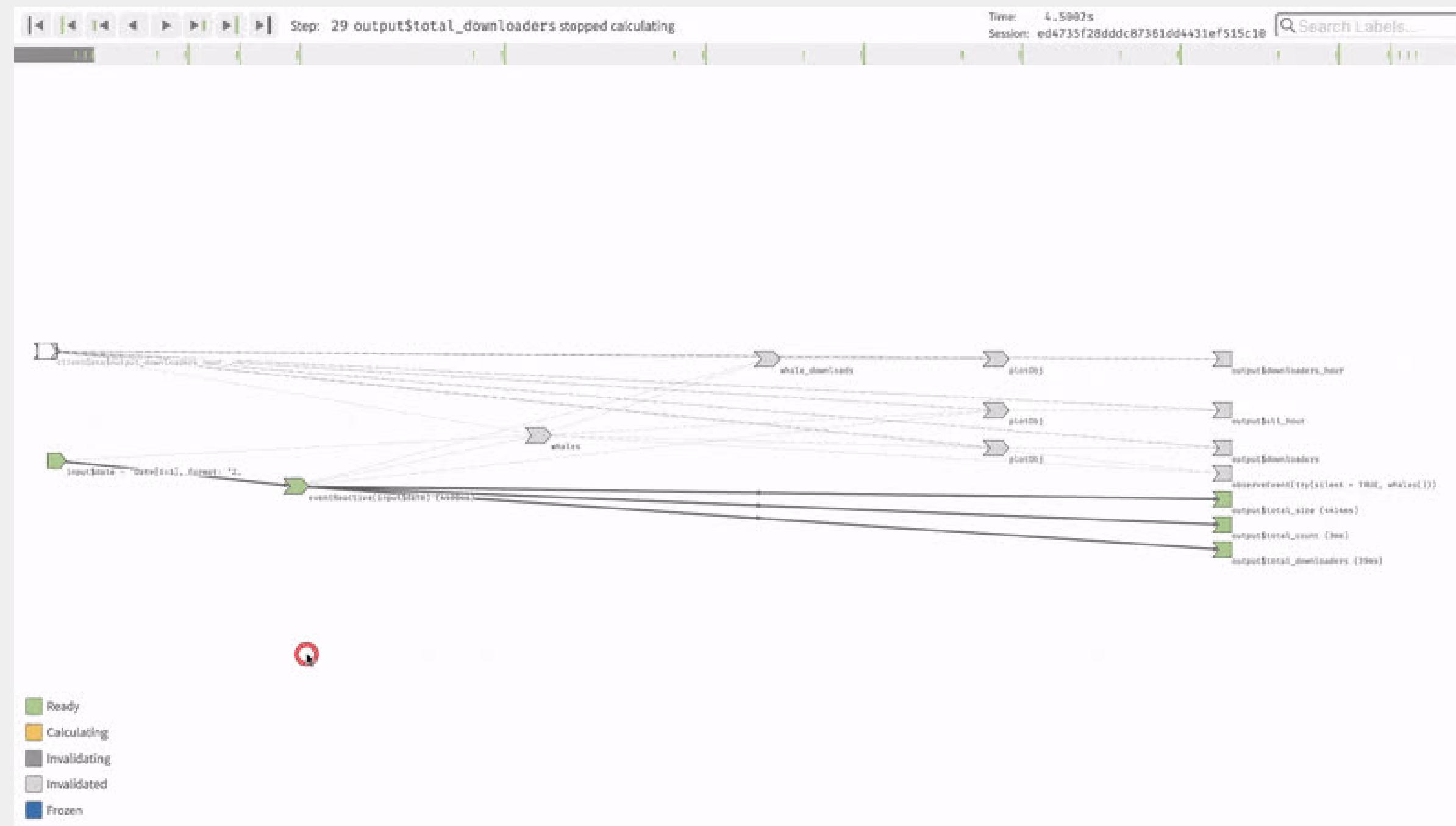
The reactive graph contains one symbol for every **input** and **output**, and we connect an **input** to an **output** whenever the **output** accesses the **input**.

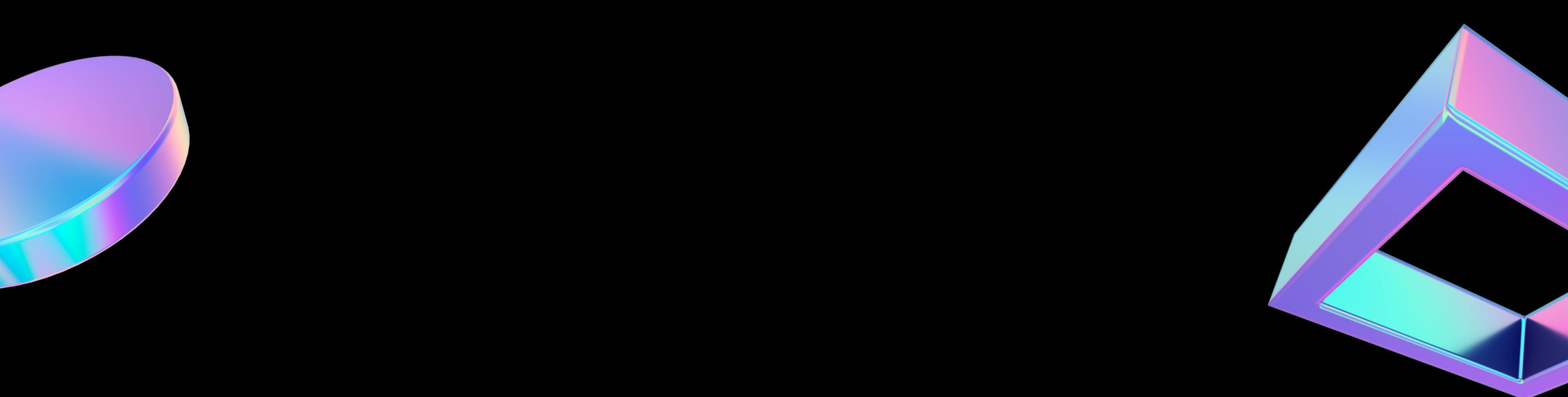
For more complex apps:

<https://cloud.r-project.org/package=reactlog>

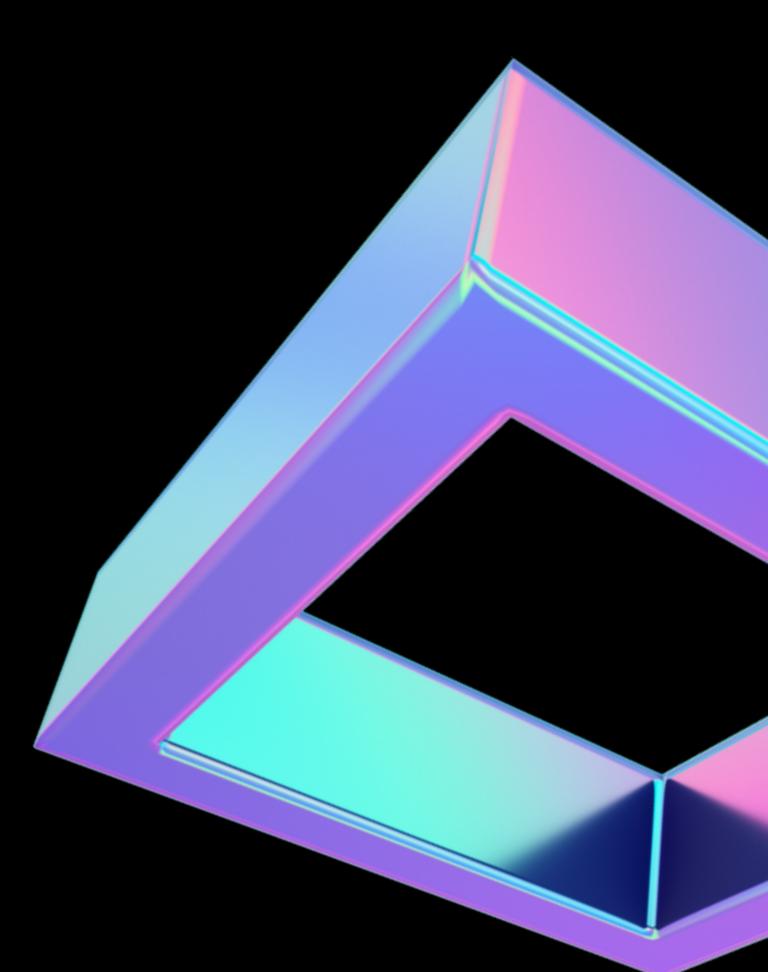
Display the reactivity dependency graph of your Shiny applications:

- `install.packages("reactlog")`





**BEAUTIFUL &
FUNCTIONAL UI MATTERS**



UI PRINCIPLES

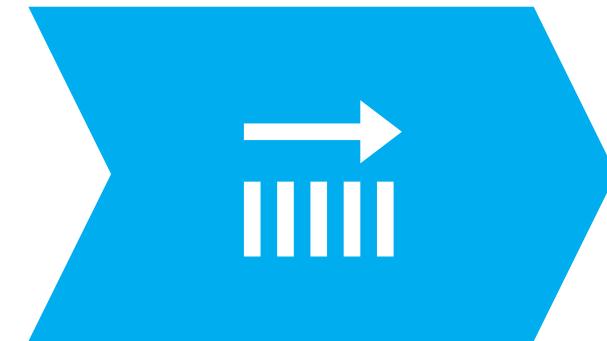
consider your audience's accessibility needs

KISS



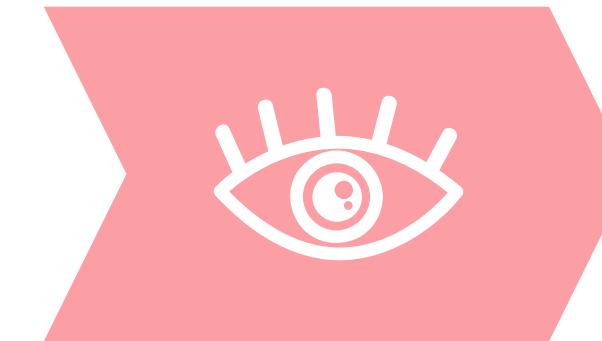
Keep it simple, stupid

**Be
consistent**



Colour, buttons,
widgets, flows, layout,
arrangement

**Readability &
scanability**



Highlight key
elements (info,
navigation) using
layout, contrast, light,
space, font and size

**Reduce
frustration**



Provide users with
feedback, don't let
them feel lost (status
bar, loaders,
messages, updates)

Iterate



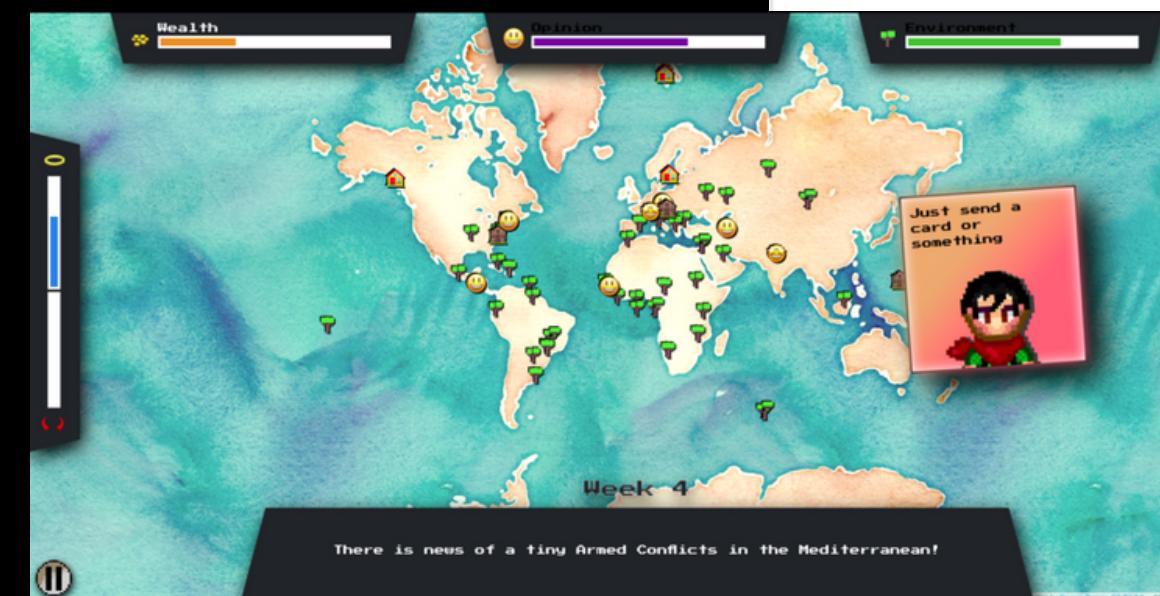
Be ready and willing
to adjust your UI to
suit your users, SEEK
OUT feedback, stay
up to date on best
practices and new
packages

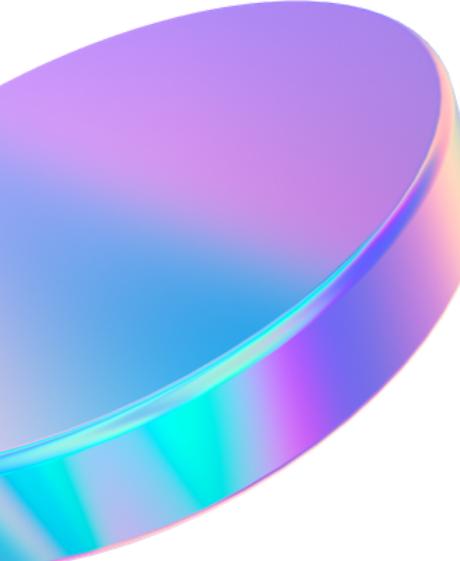
Don't make boring apps!

- Bootstrap open source, responsive grid system and library of shiny components
 - <https://shiny.rstudio.com/articles/layout-guide.html>
- Many people don't bother to customise their UI
 - HTML, CSS style sheets, JavaScript, SASS

UI packages I'm loving at the moment:

- Layout/Widgets/Themes/UX:
 - shinydashboard
 - shinydashboardPlus
 - shiny.semantic
- Maps:
 - leaflet
- Data Vis:
 - plotly, ggplot (ggplotly)
 - r2d3 - D3 visualizations (advanced)
- Displaying tables, upload / download data:
 - data.table
 - DT





TIP #2 - BE KIND TO YOUR FUTURE SELF

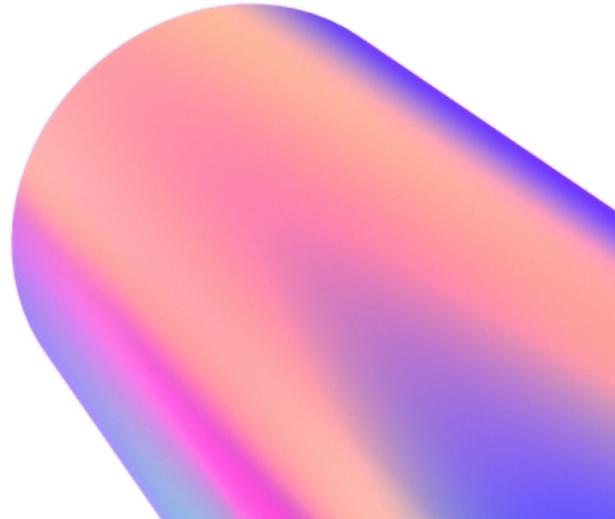
COMMENT LIKE CRAZY IN YOUR UI

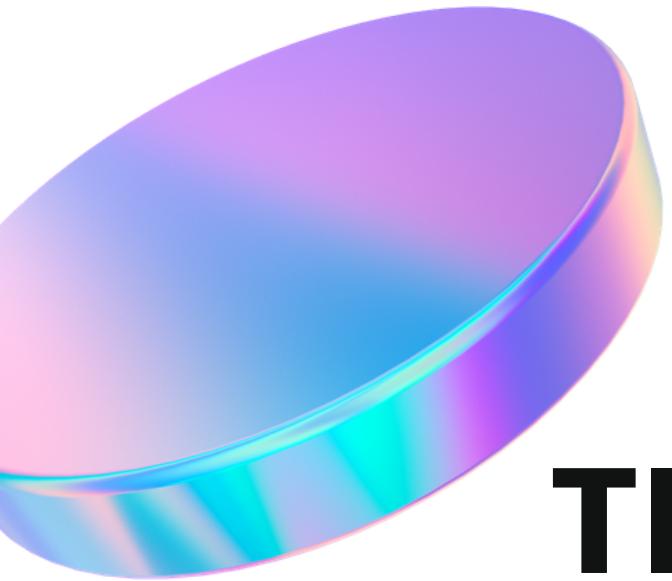
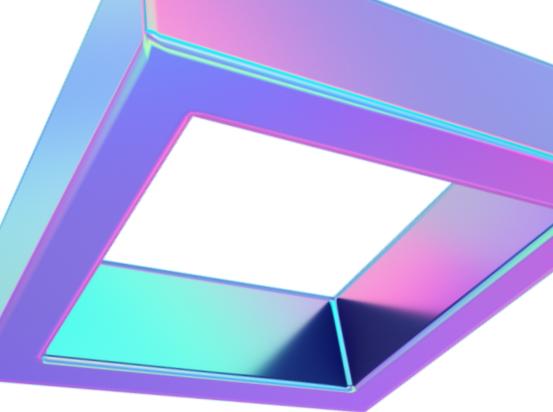
```
library(shiny)
library(shinydashboardPlus)

ui <- fluidPage(
  #drop down menu - user selects child safety service centre (CSSC)
  shinydashboardPlus::pickerInput(inputId = 'cssc_input1',
    label = 'Select a CSSC to update the graphs & map:',
    width = "100%",
    choices = unique(df_cssc$CSSC), # end choices
    options = list(`style` = 'btn-primary') #end options
  ) #end pickerInput
) #end fluidPage

server <- function(input, output, session) {

} #end server
```





TIP #2 - BE KIND TO YOUR FUTURE SELF COMMENT AT THE END OF FUNCTIONS IN YOUR UI CODE

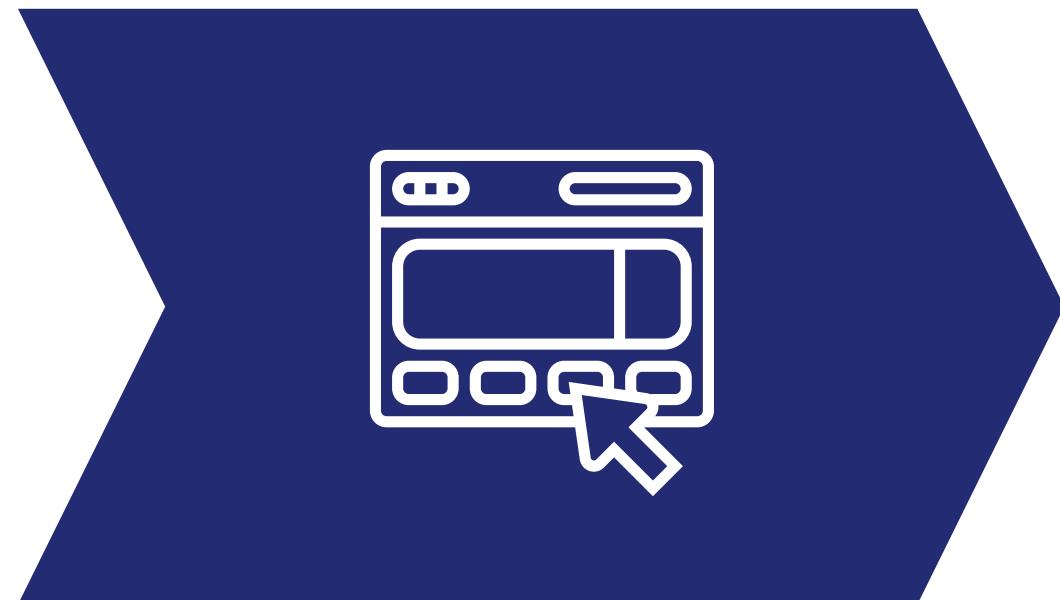
```
481 column(6,  
482   withSpinner(  
483     leafletOutput("map4",height = "340px", width="100%")  
484   )#end spinner  
485 )#end column  
486 )# end fluidrow  
487 ) # end FluidPage  
488
```

```
column(6,  
  withSpinner(  
    leafletOutput("map4",height = "340px", width="100%")  
  )  
)
```

SCALING SHINY APPS

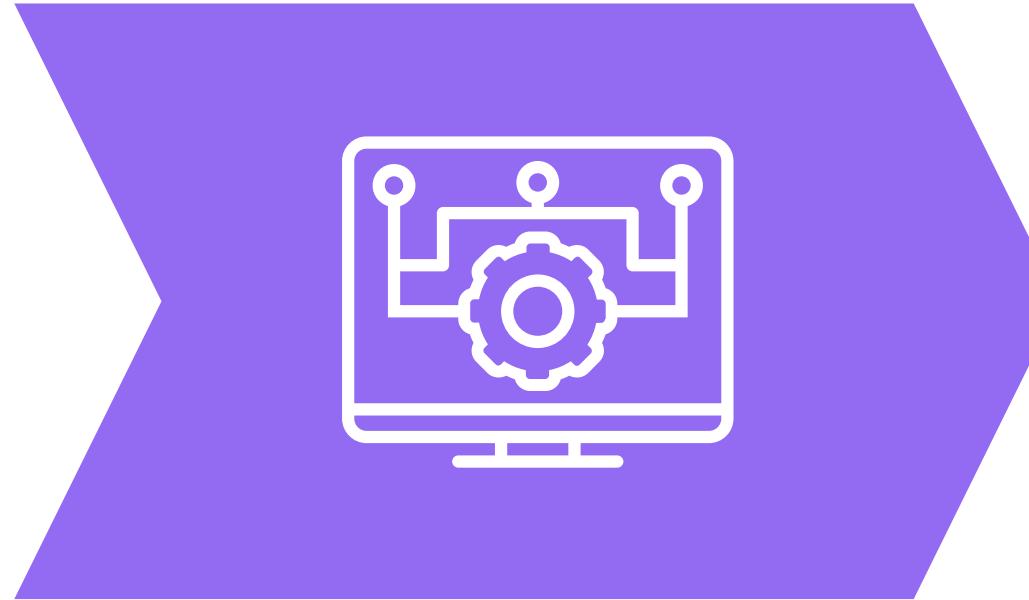
scale vertically first (more users 1 machine), then horizontally (across multiple machines)

Leverage the front end (ui)



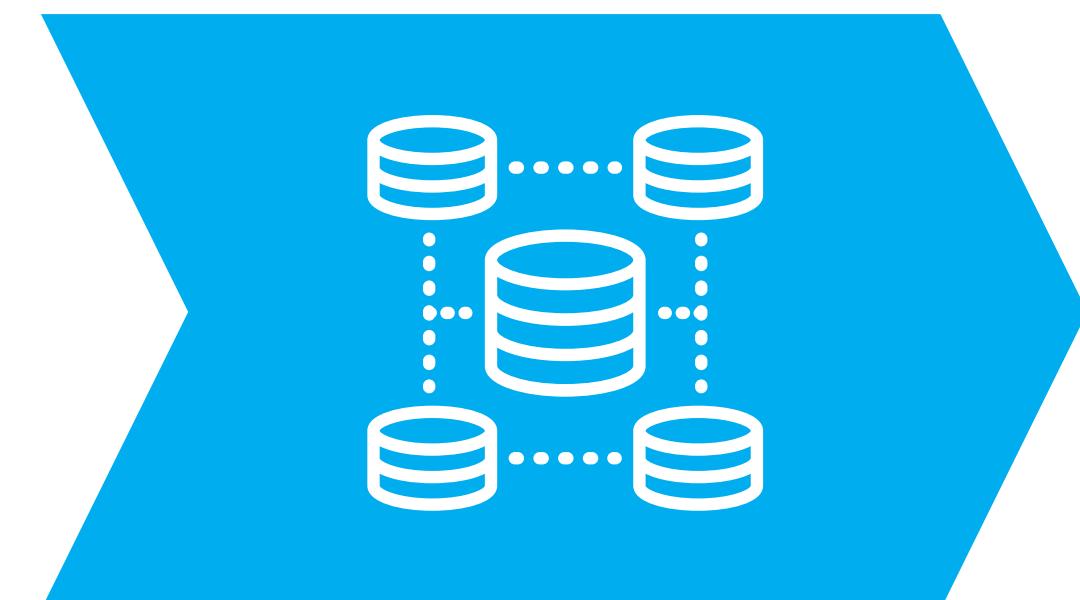
Render outputs in ui not server, JavaScript (package by Dean Atali [ShinyJS](#))

Extract computations



Remote API - load only what's needed (package called [Plumbr](#) - generate rest API)

Use a database & organise architecture



**Leaflet - adding markers colourised based on some logic
deploying shiny app**