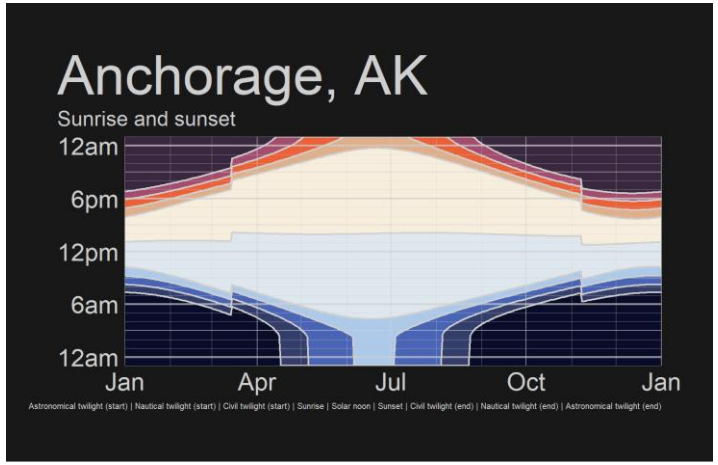
I made an entire e-commerce platform on Shiny

Jacqueline Nolis | @skyetetra | jnolis.com

{ggirl} – make your **GG**plots exist **I**n **R**eal **L**ife

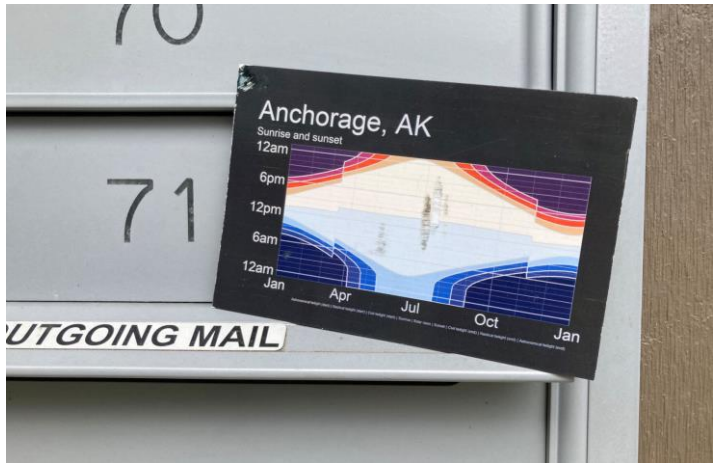
```
remotes::install_github("jnolis/ggirl")
```

A ggplot2 plot



Paying a modest fee

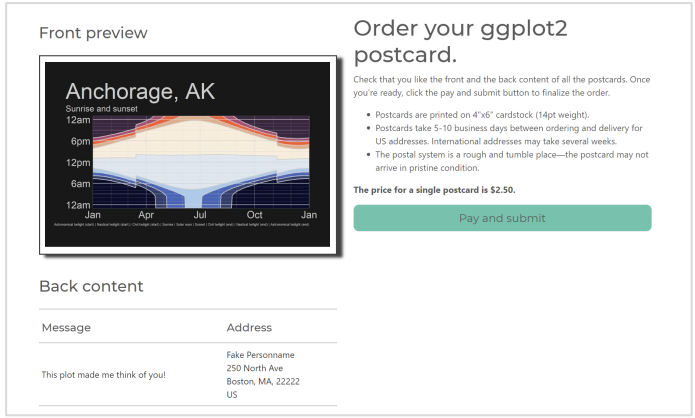
An actual postcard



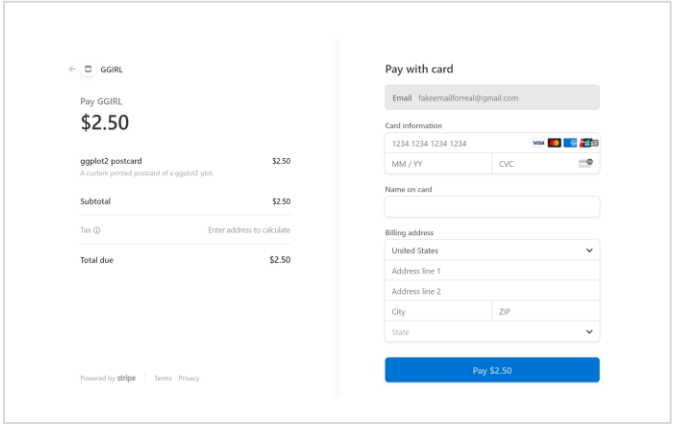
Live demo time!

```r
ggpostcard(plot = plot,
           contact_email = "ggirl@jnolis.com",
           messages = "Greetings from R!",
           send_addresses =
             address(name = "RStudio::conf",
               address_line_1 = "165 Waterfront ST",
               city = "National Harbor",
               state = "MD",
               postal_code = "20745",
               country = "US"))
```

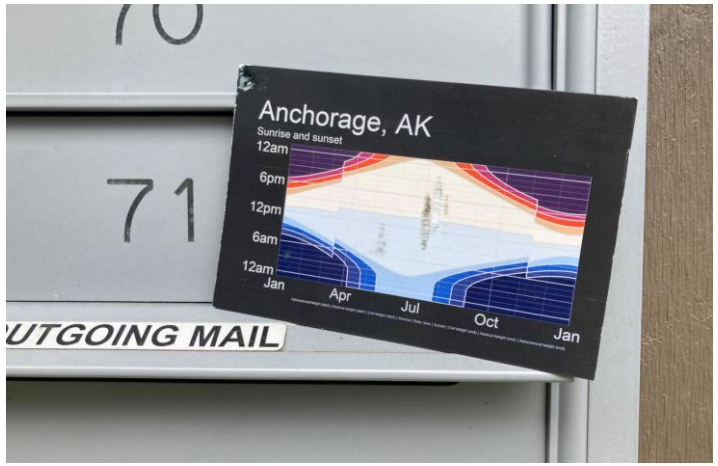
Call R code



Preview image on a Shiny webpage



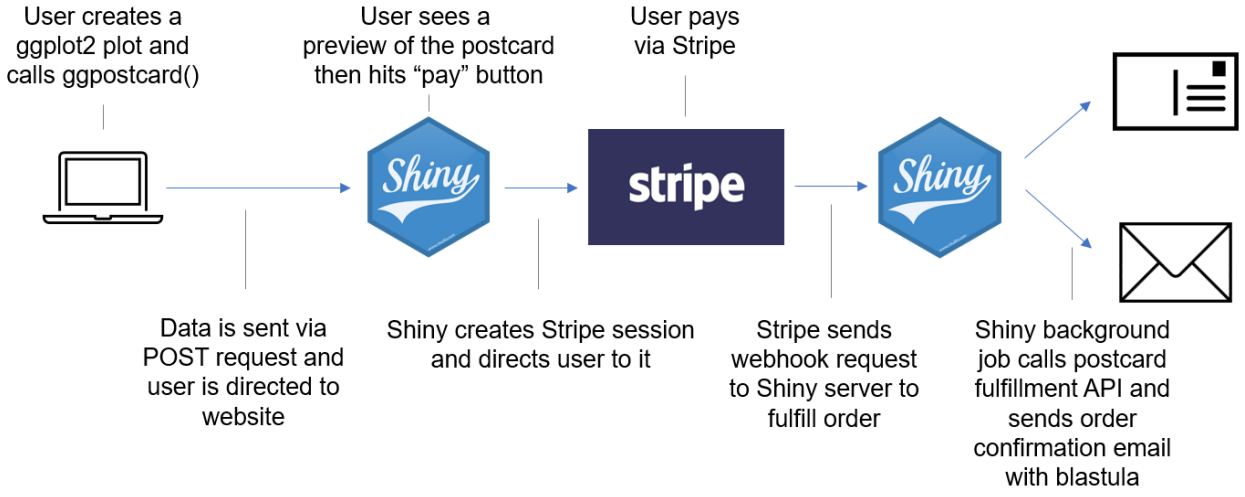
Make Stripe payment



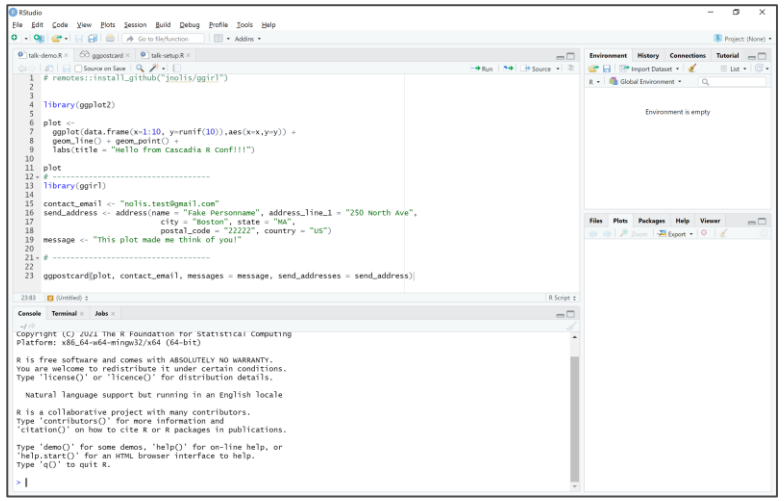
Get a postcard (& confirmation email)

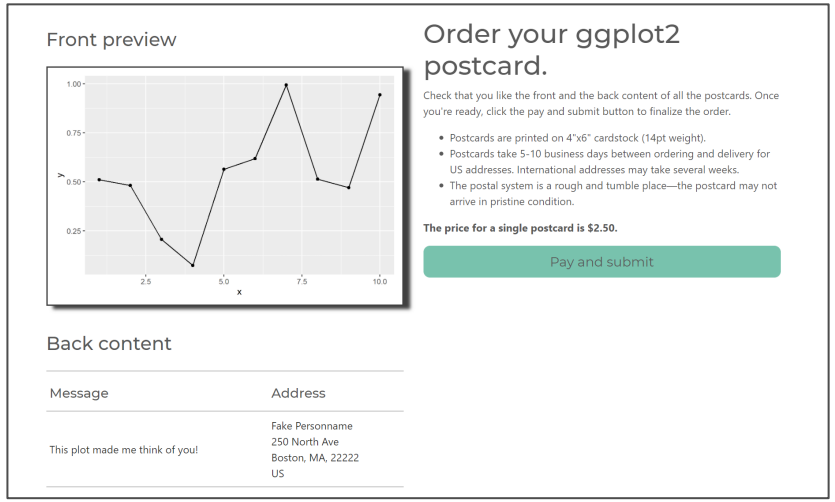
How does it work?

- {ggirl}: R package that takes plot and passes it to a Shiny server
- ggirl-server: Shiny app that does the work of
  - Showing the user a preview
  - Passing user to Stripe to pay
  - Noticing when user paid on Stripe
  - Fulfilling the postcard order via an API
  - Sending confirmation email (or error email)



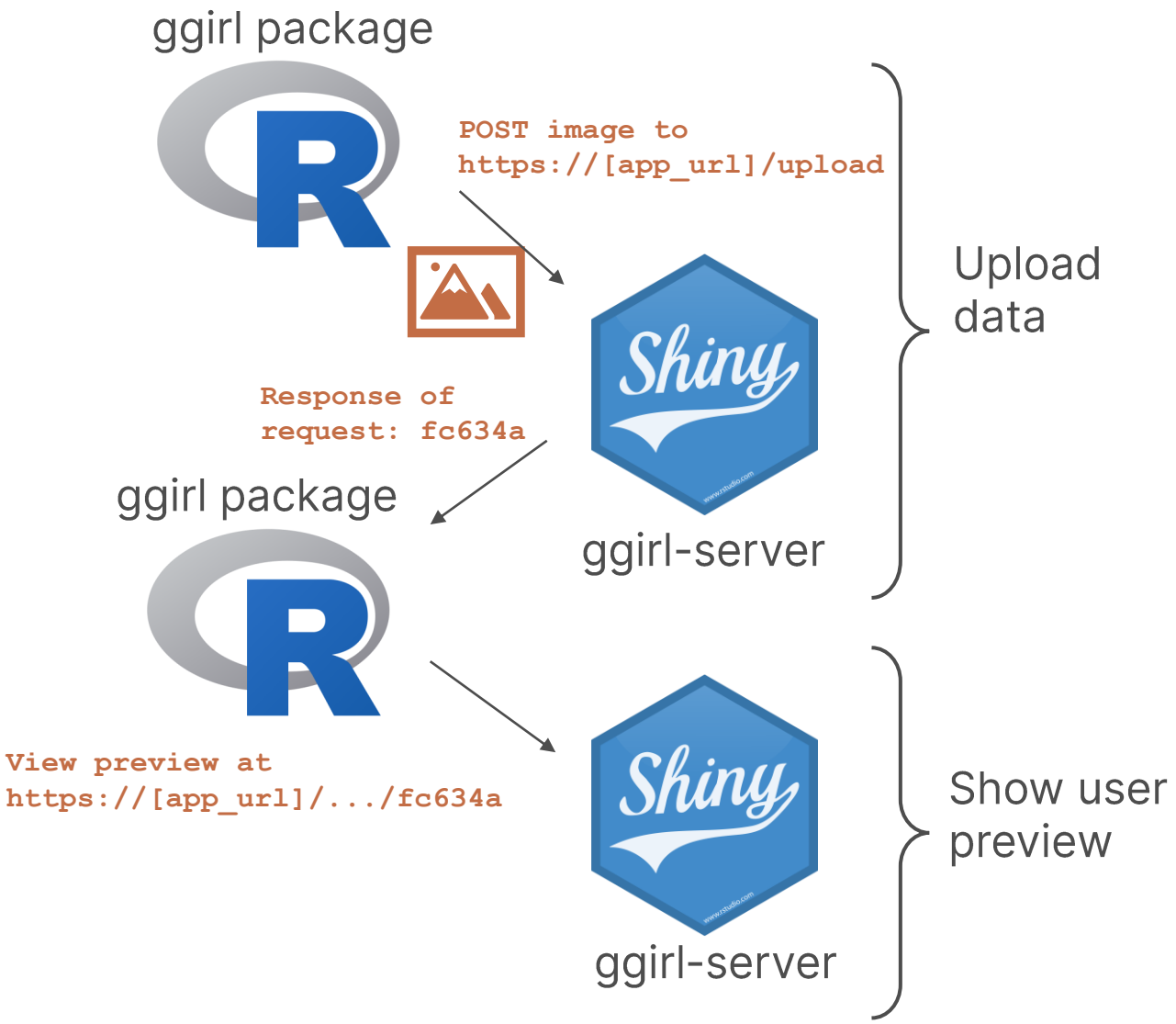
ggirl-server *should* be a collection of Shiny/plumber apps, but I wanted just a single Shiny app for ease of maintenance





1. How do you pass the plot from R to the Shiny app?

Get data to Shiny with HTTP POST



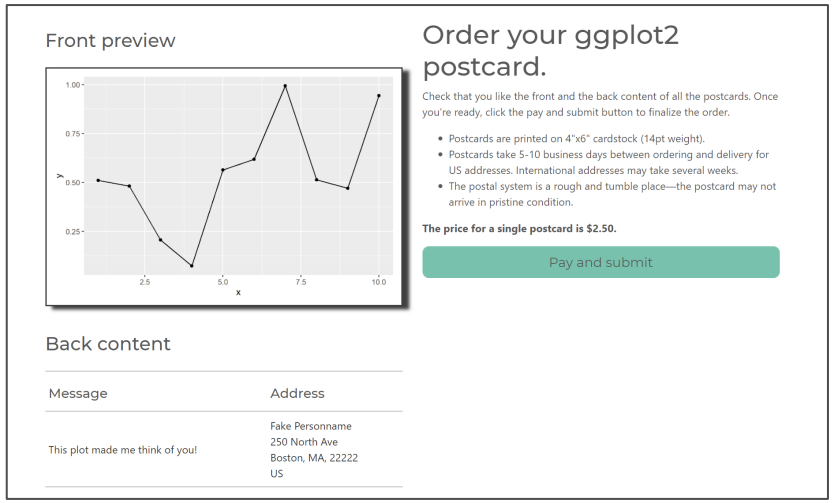
- Should be done with a HTTP POST request
  - Shiny does not have that documented functionality
    - It should!
- There are actually a number of undocumented ways to do this

The {brochure} package

- Experimental {brochure} package from Colin Fay lets you:
  - **Connect multiple Shiny apps** together in a single bigger app as different endpoints
  - **Listen for requests besides just GET**
- Solves two of my problems!
  - Removes need for multiple apps/APIs
  - Removes need for Plumber for POST requests

github.com/ColinFay/brochure

Code for brochure and two other ways to POST to Shiny available at github.com/jnolis/ggirl-examples



2. How to show the image preview page

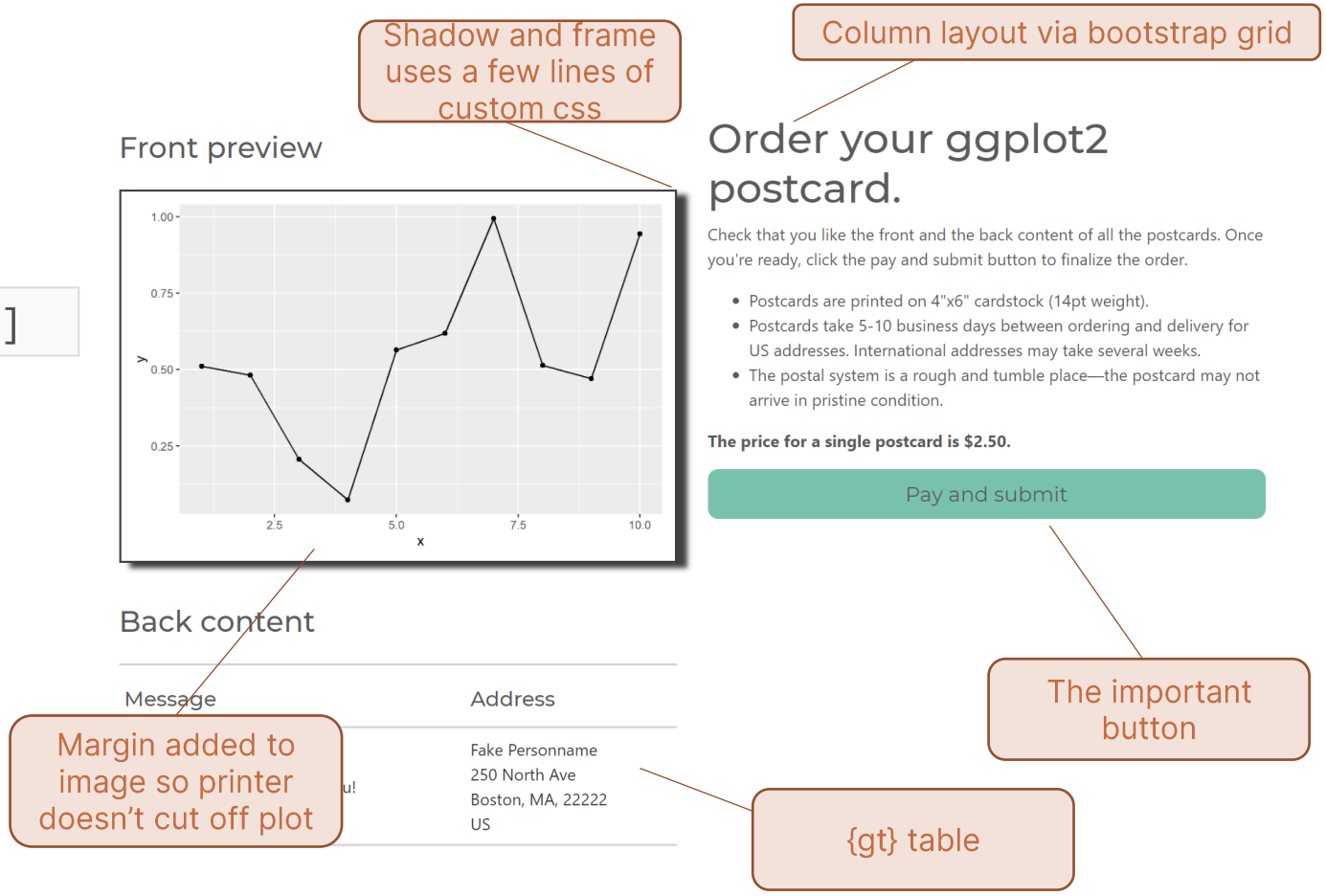
Showing a user preview

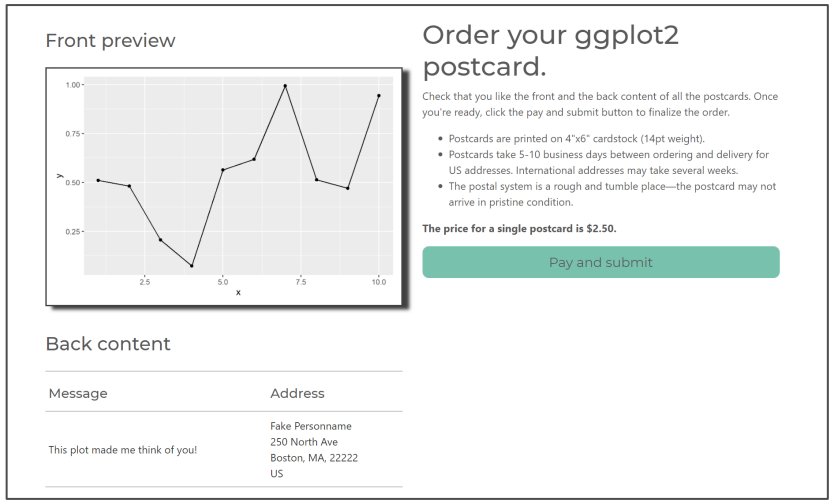
This is a Shiny app with one button.

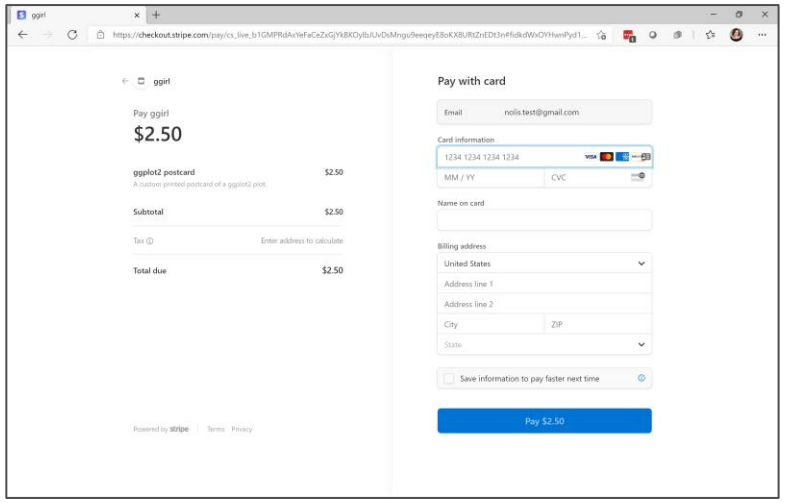
The ID of the particular ggplot is a query parameter in the URL.

```
https://[app_url]/postcard?token=[token]
```

Making Shiny apps look good with HTML and CSS is cool and fun and I gave a whole different talk about it at Shiny Conf 2022: link.jnolis.com/shiny-conf-2022







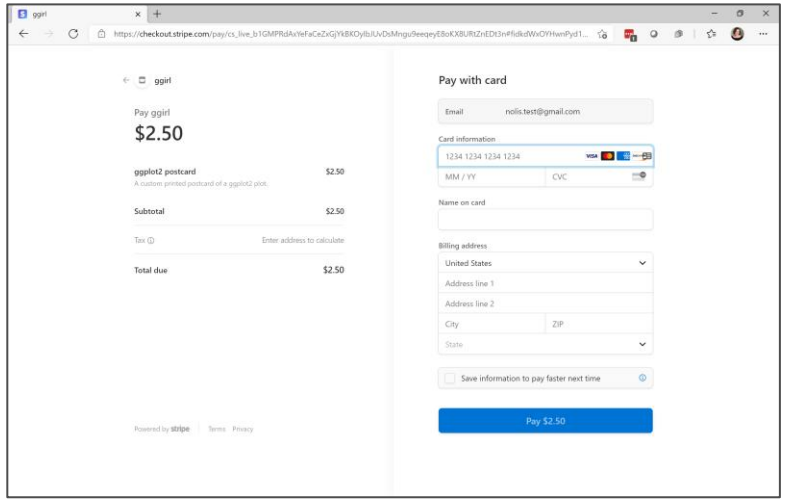
3. How to pass the user from Shiny to Stripe?

Having users pay with Stripe

- Surprisingly easy
- Steps in payment flow:
  - Shiny UI makes POST request for a new Stripe session which has the list of items to buy and prices
  - Stripe JavaScript library makes button send user to Stripe
    - Add library in Shiny with tags$script and Shiny.addCustomMessageHandler

User presses button → **1. Shiny send POST of cart info** → **2. Shiny triggers Stripe JavaScript** → User is redirected to stripe

Code for this is available at github.com/jnolis/ggirl-examples



4. How to fulfill the order?

Have Stripe send a POST telling Shiny to order postcards

Steps:

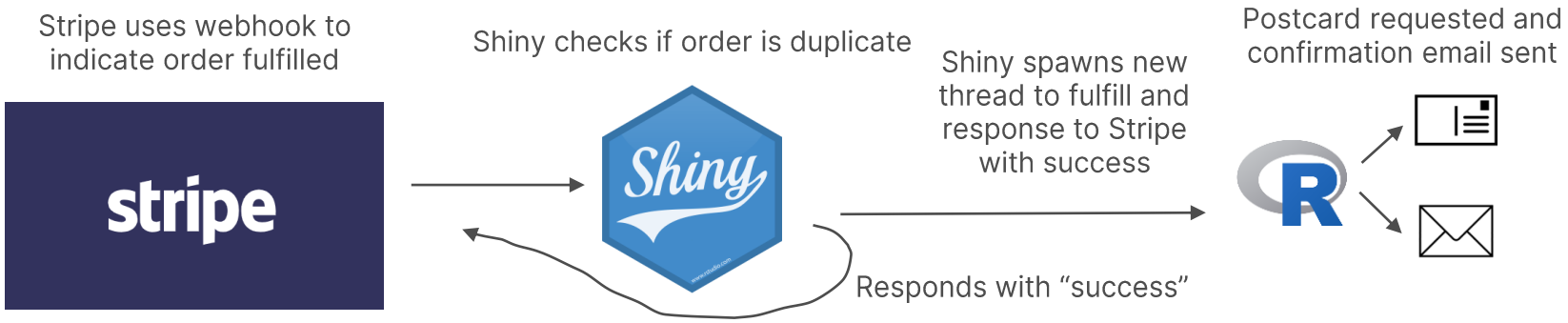
1. Stripe sends a POST request (aka webhook) saying “Order xyz has been paid for” (+ keys to ensure authentic)
2. Shiny receives the request and sends a POST to the postcard printing company
3. Shiny send an email to the user saying the order is confirmed

Feels like it should be straightforward…

Fulfill orders in background process

- Stripe needs a response to know the message was received
  - If the response doesn't come REALLY QUICKLY, Stripe retries
  - Shiny won't respond to a request until it's work is done
- Use {callr} package to spawn a separate parallel workflow to do the fulfillment
- I love this so much I gave a whole talk about it: link.jnolis.com/r-parallel-talk

```r
jobs <- list()
run_token <- function(token){
    Sys.sleep(10)
    write.csv(data.frame(),paste0(token,".csv"))
}
server <- function(input, output, session) {
    observe({
        if(input$startjob == 1){
            token <- UUIDgenerate()
            message(paste0("running task for token: ", token))
            if(is.null(jobs[[token]])){
              jobs[[token]] <<-
                callr::r_bg(run_token, args = list(token = token))
            }
        }
    })
}
```



Code for this is available at github.com/jnolis/ggirl-examples

Actually ordering the postcards

- There are many companies that have APIs to mail postcards for you via POST request
- Time between sending the order and getting the postcard was generally 2 weeks
- This took forever to find a good one

It's not your monitor: this actual real test postcard looked like a bad Facebook meme IRL. I did not go with them.

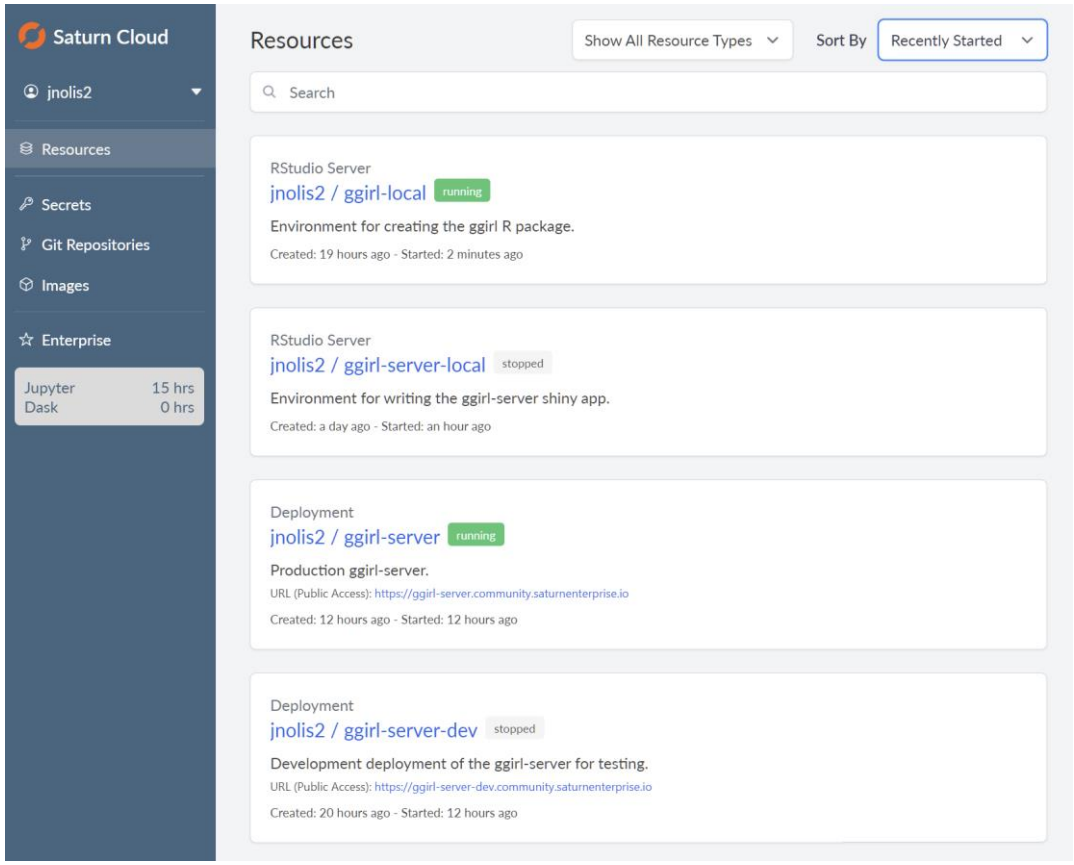
Send confirmation emails

- Use blastula to send email (easy!)
- Thoughtfully resize image with {imagemagick}
- If there is an error in fulfillment function, have it email me the error message

How to deploy it?

DevOps is easy with Saturn Cloud

- Saturn Cloud is a great data science platform
  - Write code in RStudio Server
  - Deploy Shiny apps to Saturn Cloud
  - Free to use for 30 hours a month
- I had environments for
  - Writing ggirl (R package)
  - Writing ggirl-server (Shiny app)
  - Deploying dev ggirl-server
  - Deploying prod ggirl-server
- Ran multiple beta tests with real users to iron out the issues



R package development

Shiny app development

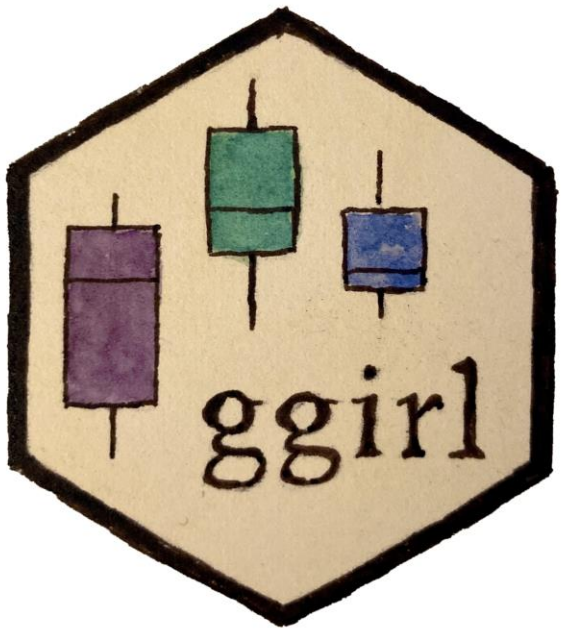
Prod deployment

Dev deployment

SaturnCloud

Wrapping it up

- {ggirl} is a cool package for ordering cool stuff
- Wild things I didn’t know if they could be done in Shiny but I did anyway:
  - Receiving data from an R package
  - Handling POST requests in Shiny
  - Sending/receiving Stripe API calls
  - Spawning background processes



Thank you!

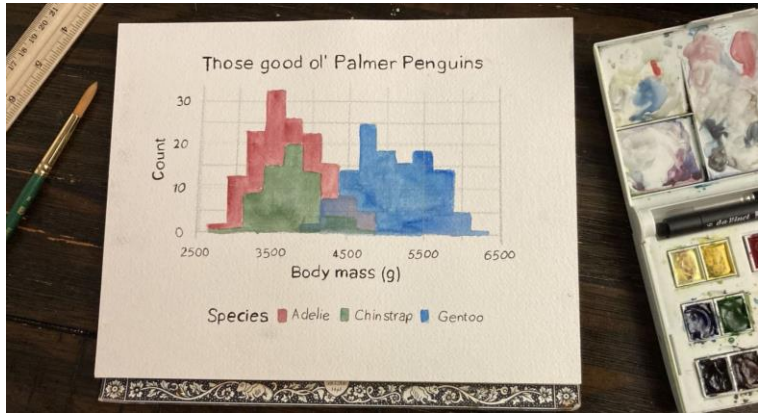
Jacqueline Nolis | @skyetetra | jnolis.com

Install the package	`remotes::install_github("jnolis/ggirl")`
Code for the rstudio::conf(2022) postcard	**link.jnolis.com/rstudio22-code**
Code examples from ggirl-server	**github.com/jnolis/ggirl-examples**
Slides from this talk	**link.jnolis.com/rstudio22-slides**

Come talk to me in person for a 80% off a postcard coupon!

ggpostcard()

ggartprint()



ggwatercolor()