I made an entire e-commerce platform on Shiny

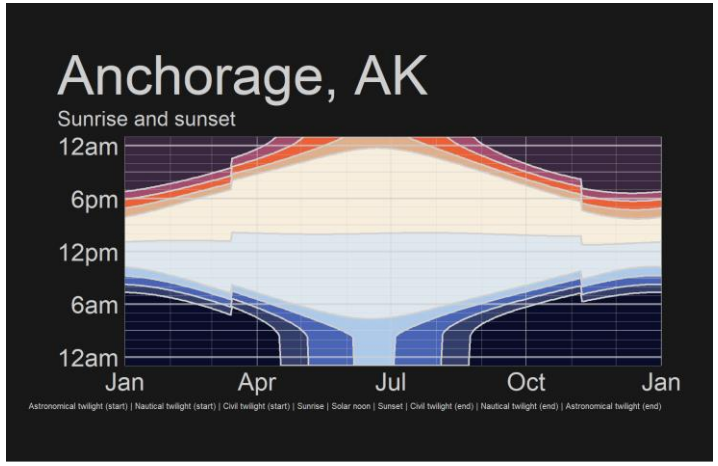
Jacqueline Nolis | @skyetetra | jnolis.com

What is {ggirl}?

So many cool ggplot2 plots out there—let them exist in the physical realm!

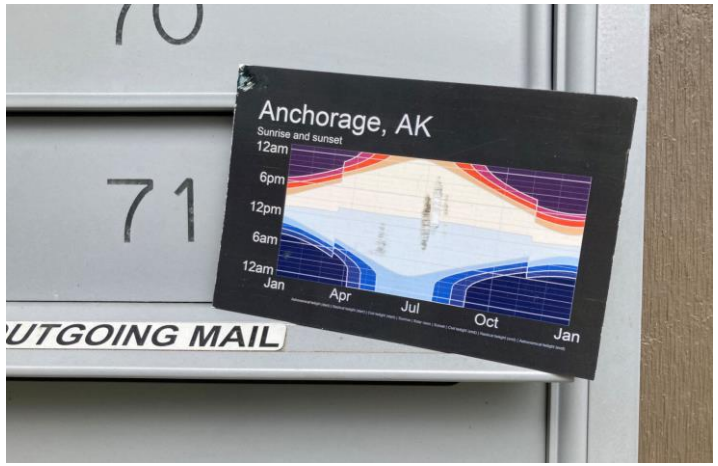
```
remotes::install_github("jnolis/ggirl")
```

A ggplot2 plot



A modest fee

An actual postcard



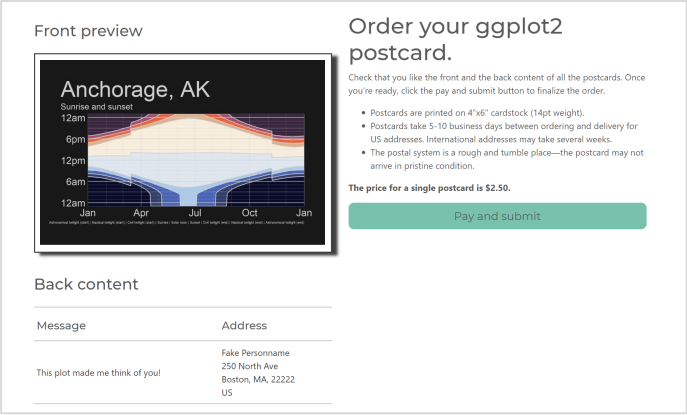
ggartprint()

ggwatercolor()

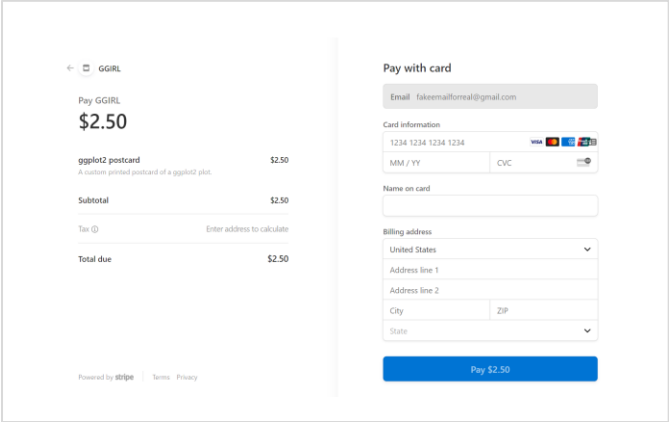
Live demo time!

```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"))
```

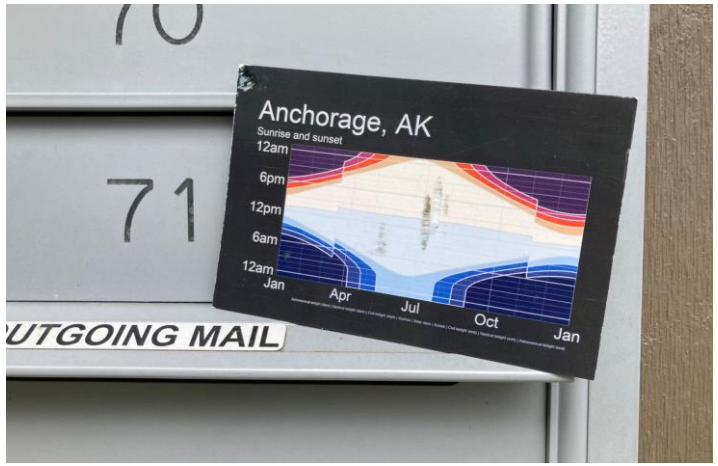
Call R code



Preview image on a webpage



Make Stripe payment

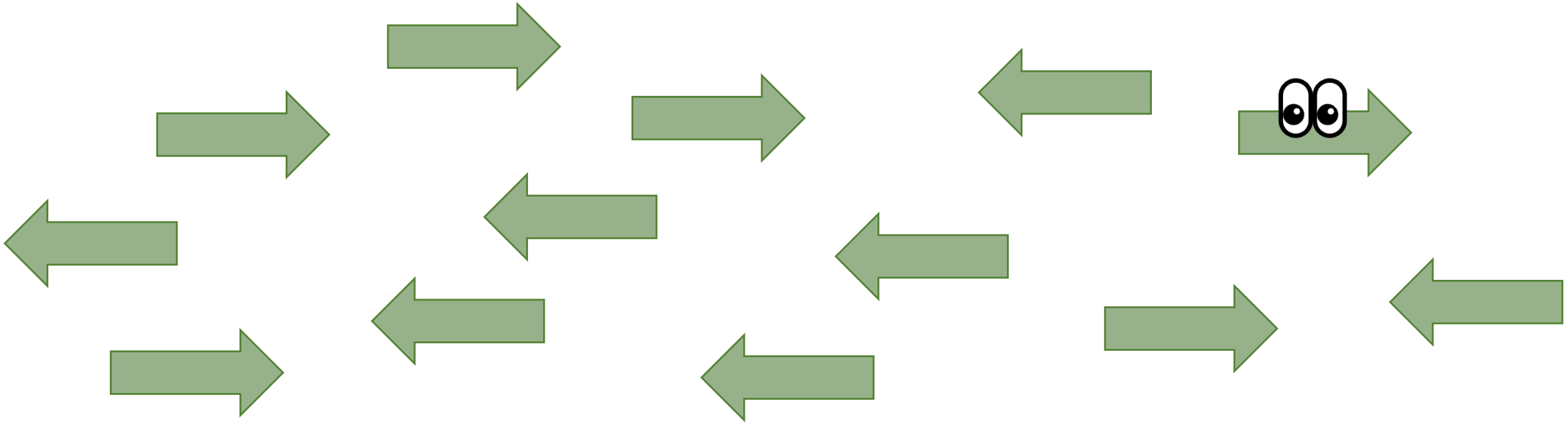


Get a postcard (& confirmation email)

The ggirl system

How does it work?

More HTTP POST requests than you could possibly image



An introduction to HTTP

- Websites and APIs use HTTP protocol
- You send a request to a URL, and get something back
- GET HTTP request – ask for something
  - Website: ask for a webpage, get HTML back
  - API: ask for a particular piece of data, get JSON (or whatever) back
- POST HTTP request – send some data
  - Website: submit a form
  - API: send data to the API
- You can *send* HTTP requests with the {httr} package

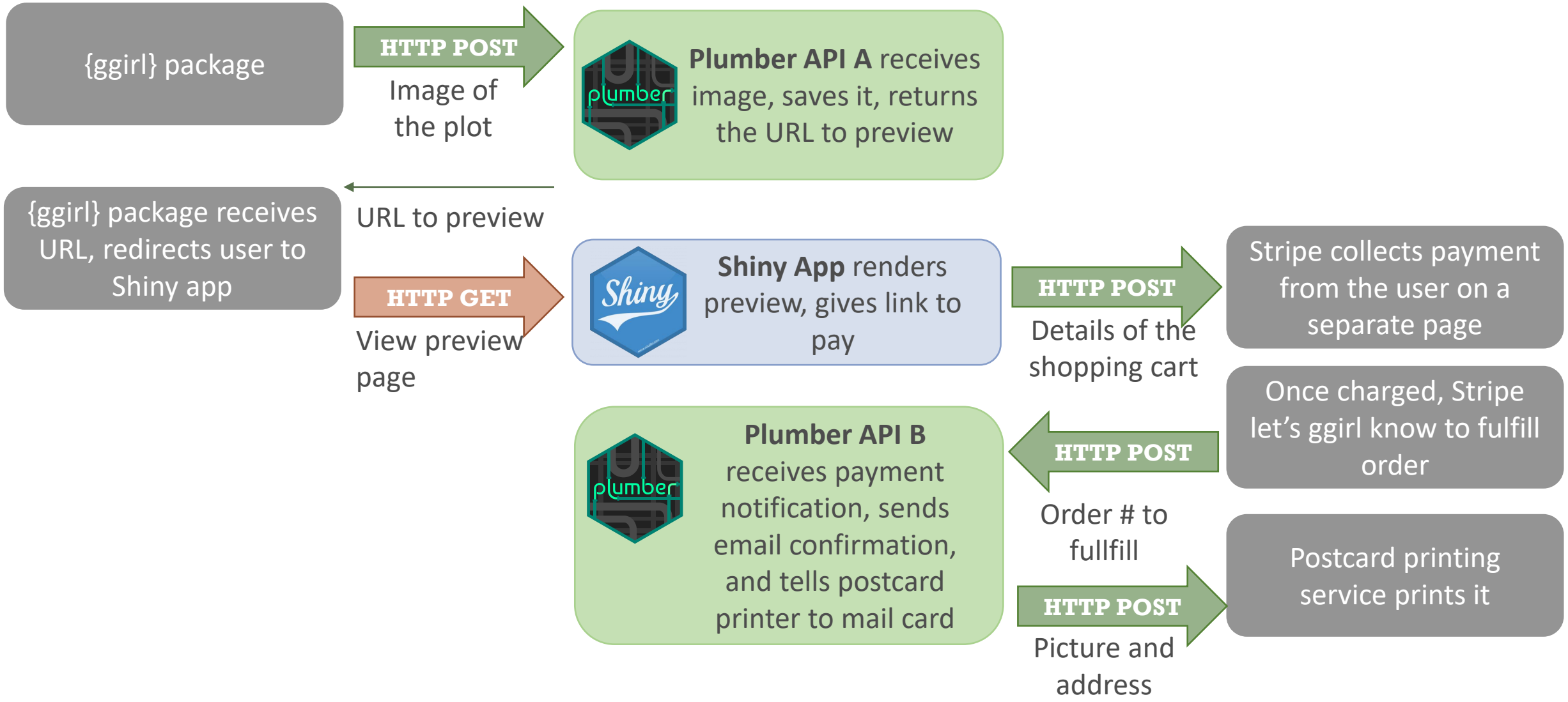
```
httr::GET("http://www.google.com") #returns HTML page you see when browsing
httr::POST("http://hypothetical-example-api.com",
           body = "A simple text string") #returns something depending on API
```

Receiving HTTP requests with R

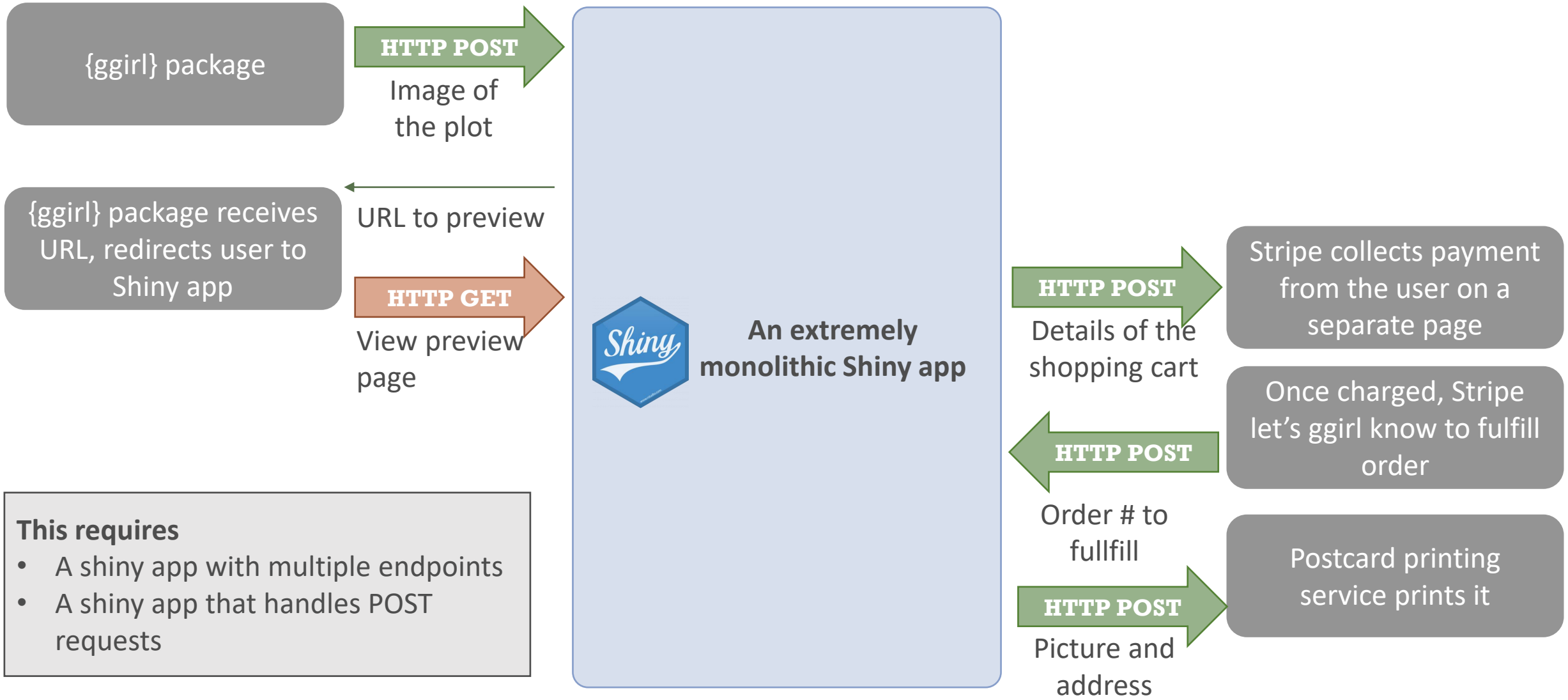
- Lots of tools to design HTML/JavaScript
- Great UI
- Only (on its face) takes GET requests
- Only (on its face) has one URL endpoint

- Handles POST/GET/whatever
- No HTML pretty stuff, just send/receive data
- Can have many endpoints (example: /photos, /videos, etc)

Here is the "correct" architecture



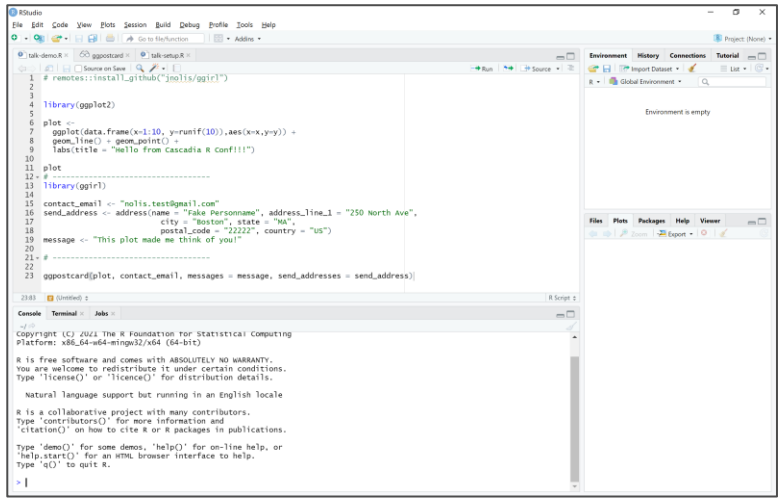
Here is my architecture (I am lazy)

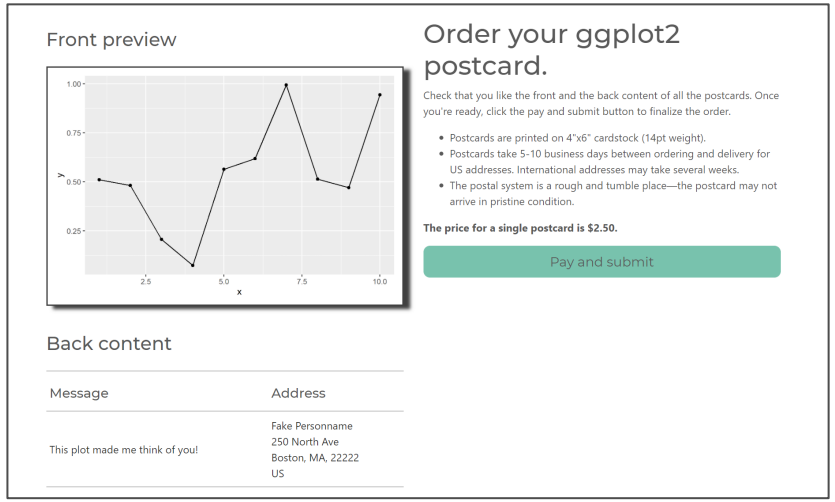


This requires

- A shiny app with multiple endpoints
- A shiny app that handles POST requests

Tackling the architecture step by step

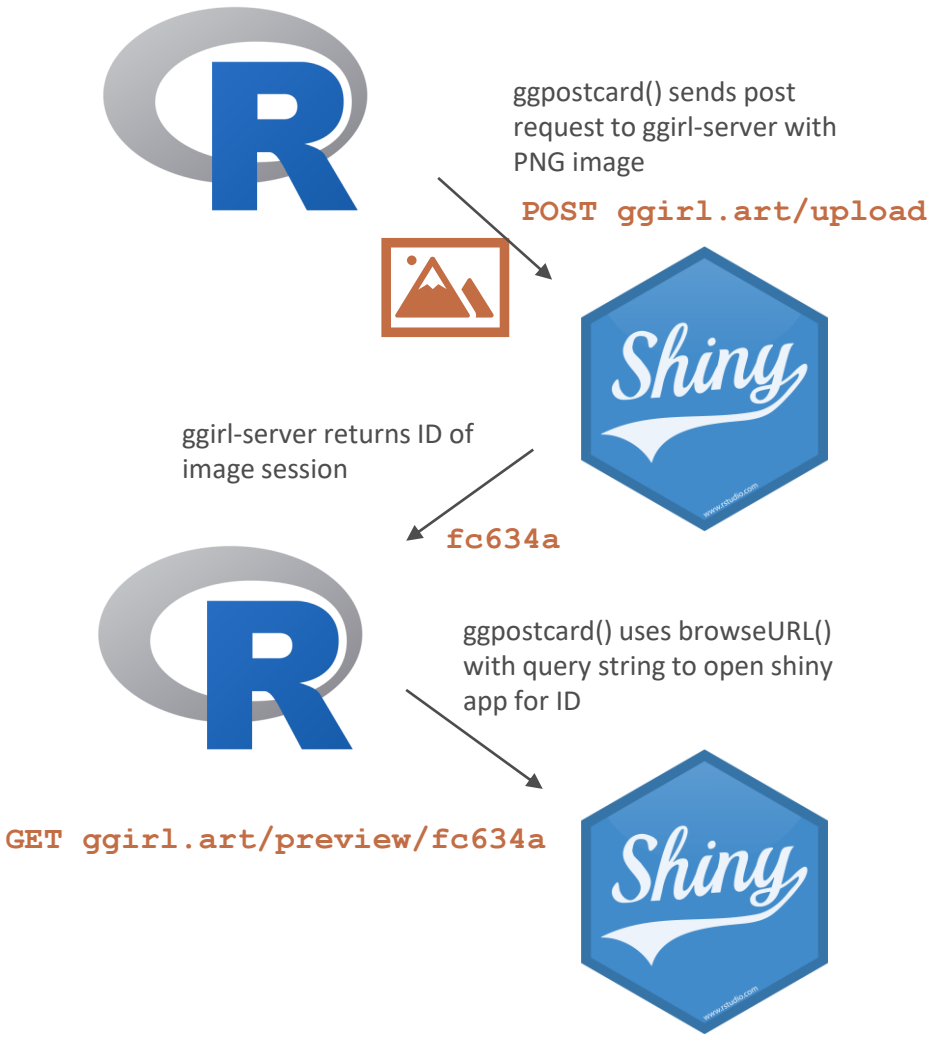




1. How do you pass the plot from R to the Shiny app?

Get Shiny to handle POST requests

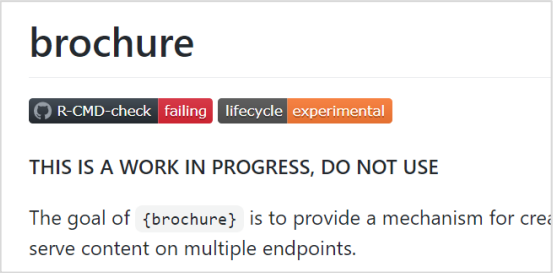
- Shiny does not have an intuitive way to pass it data
- Should be done with a HTTP POST request
  - Shiny does not have that documented functionality
    - It should!
- There are actually a number of undocumented ways to do this



The {brochure} package

- Experimental {brochure} package from Colin Fay lets you:
  - **Connect multiple Shiny apps** together in a single bigger app
  - **Listen for requests besides just GET**
- Solves both of my problems!
  - Removes need for multiple apps/APIs
  - Removes need for Plumber for POST requests
- github.com/ColinFay/brochure
- "Experimental"
  - I hit 1-2 bugs but nothing bad

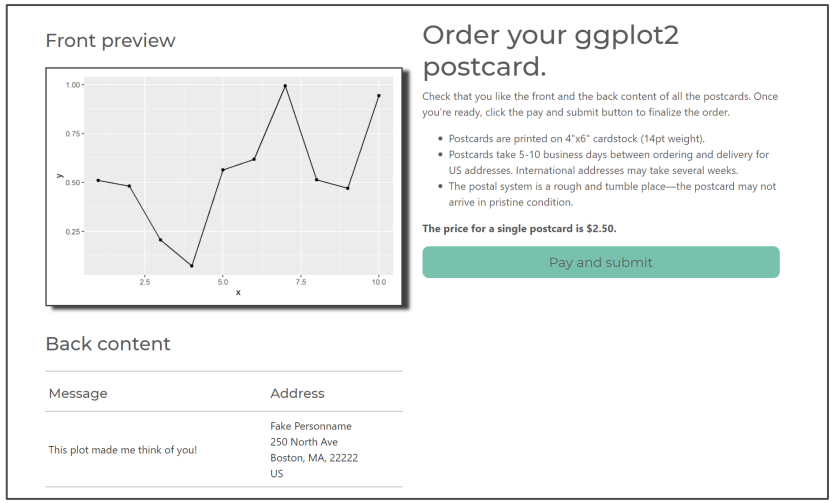
Brochure hex sticker TBD



Final POST request

- {ggirl} package sends the shiny app:
  - **Type** (postcard, artprint, watercolor)
  - **Message** (postcard only)
  - **Addresses** (email, mailing)
  - **PNG image of plot** (sending ggplots was complicated)
  - **Package version** (to ensure not outdated)
- Shiny app receives request, saves data to Google Cloud, and returns a unique token for the session
- Token is then used to route user's browser to the Shiny app

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



2. How to show the image preview page

Showing a user preview

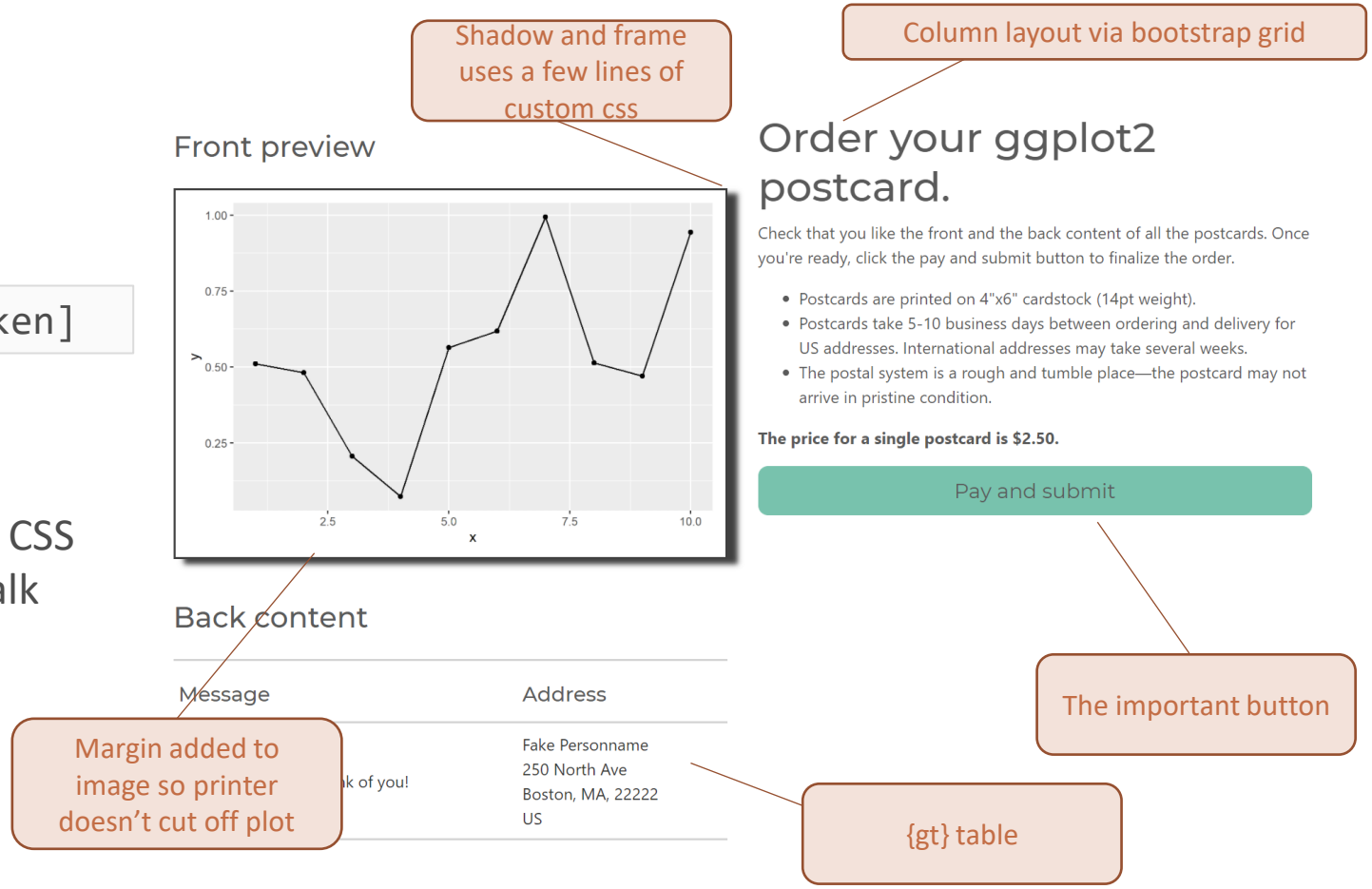
This is a Shiny app with one button.

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

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

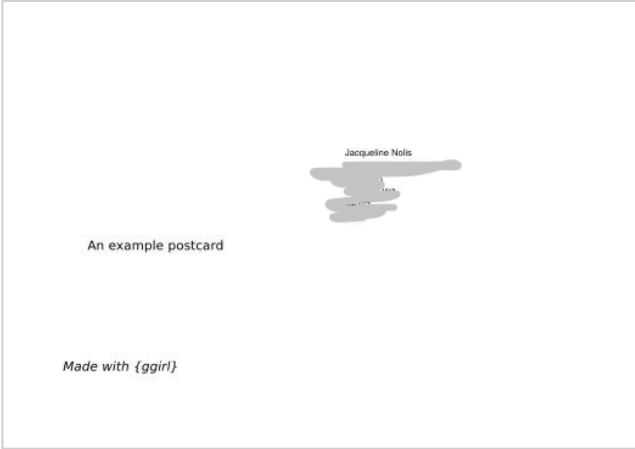
Extreme care needed for accurate previews!

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



Rendering postcard back was profoundly annoying

- Like **really** annoying
- What if you have a long word? Too many characters?? What if it cuts off the stamp??? WHAT ABOUT NEW LINES! Ahhh!!!

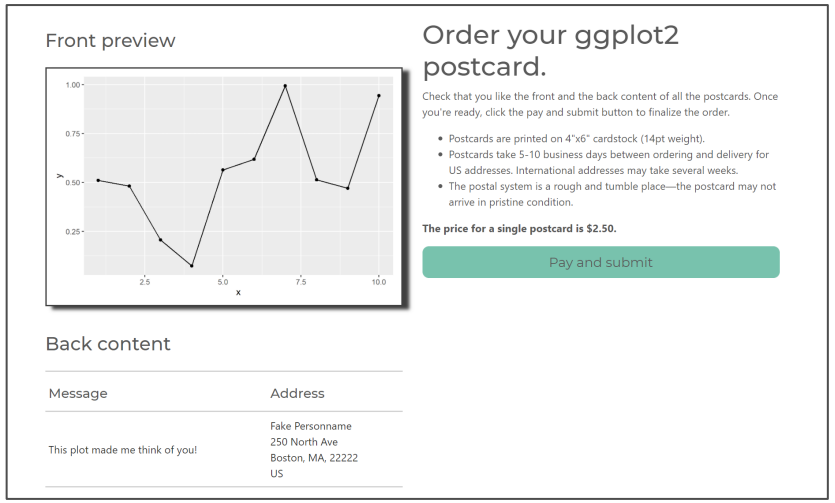


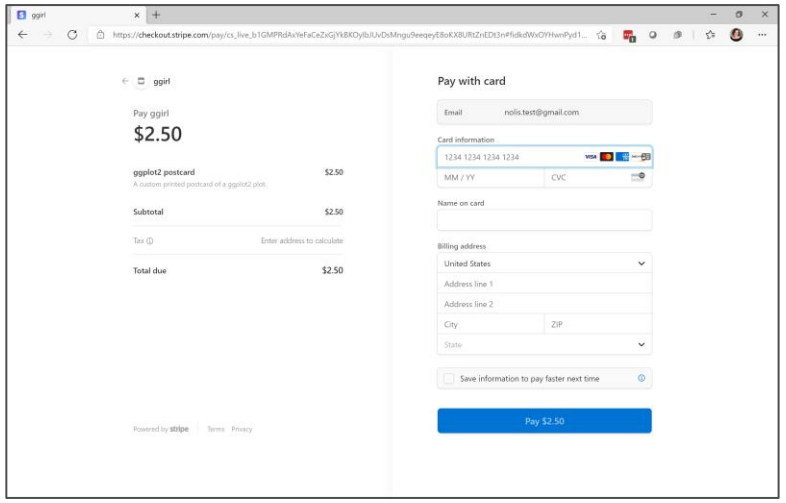
Early back preview

- Idea 1: render the back as a ggplots and use {ggfittext} to align data
- Idea 2: use a printer who took a message printed the back for me
- Idea 3: use a printer who accepted an HTML file

Back content

Message	Address
This plot made me think of you!	Fake Personname 250 North Ave Boston, MA, 22222 US





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

Having users pay with Stripe

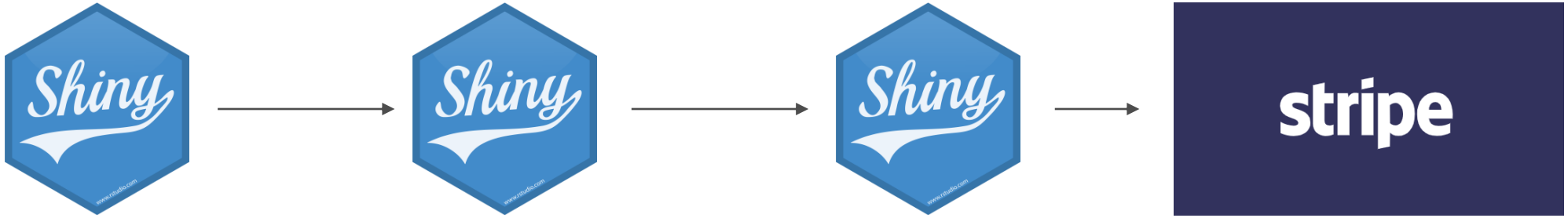
Surprisingly easy! Just two steps.

User presses button

1. Shiny creates session

2. Shiny triggers Stripe JavaScript

User is redirected to stripe



Step 1: UI creates Stripe session

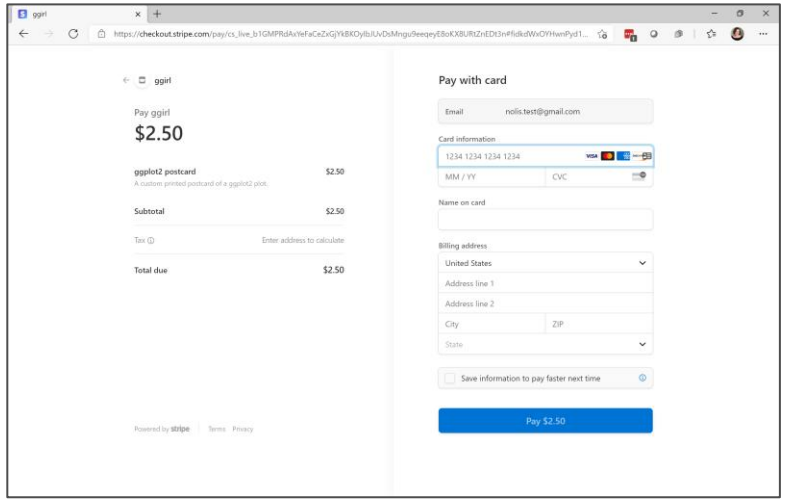
- Send a POST request to Stripe with what's in the user's cart
- It returns an ID for a session, which gets used in the next step
- You can see a full working example at github.com/jnolis/ggirl-examples

```
body_form <- list(
  `line_items[0][price_data][unit_amount]`='250',
  `line_items[0][price_data][currency]`='USD',
  `line_items[0][price_data][product_data][name]`= 'ggplot2 postcard',
  `line_items[0][quantity]`=as.character(quantity),
)
response <- httr::POST("https://api.stripe.com/v1/checkout/sessions",
                       httr::authenticate(user = stripe_creds$secret,
                                          password=""),
                       body = body_form,
                       encode = "form"
)
httr::content(response)$id
```

Step 2: JavaScript sends user to Stripe

- A tiny JavaScript script is added to the top of the Shiny app
- It gets fired after the button is finished being pressed using the Shiny JavaScript function `Shiny.addCustomMessageHandler`
- All the function does is call the stripe JavaScript api to handle the redirect

```
tags$head(
  tags$script(src="https://js.stripe.com/v3/"),
  tags$script(glue::glue('
        Shiny.addCustomMessageHandler("session_id", function(session_id) {
        Stripe("{{stripe_creds$public}}").redirectToCheckout({ sessionId: session_id});
      });', .open = "{{", .close = "}}"))
```



4. How to fulfill the order?

Have Stripe send a POST telling Shiny to order postcards

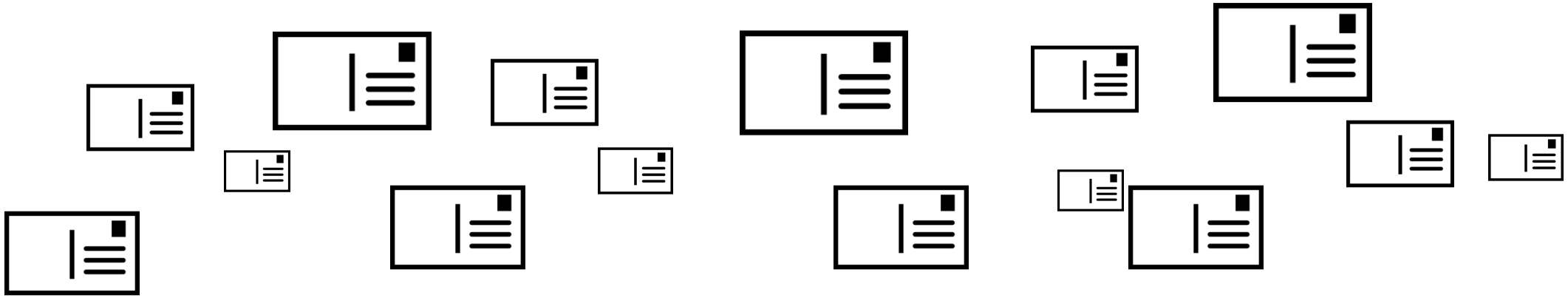
Steps:

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…

Problem: Stripe needs a fast response to it's POSTs

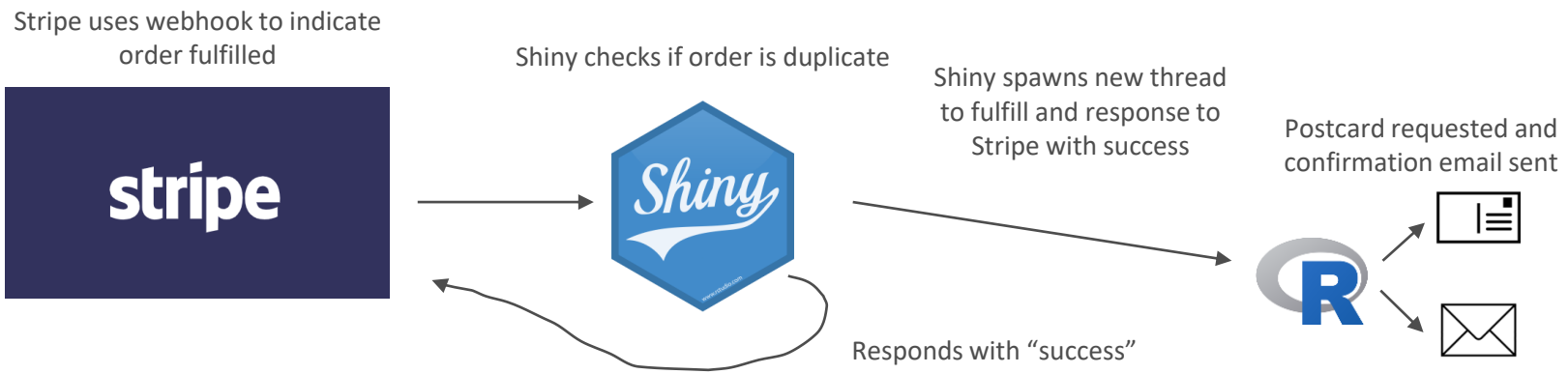
- Stripe needs a response to know the message was received
  - If the response doesn't come REALLY QUICKLY, Stripe assumes the request failed & retry
- Shiny won't respond to a request until it's work is done
- Result: Stripe will order over and over and over…



Fulfill orders in background process

- Use {callr} package to spawn a separate parallel workflow to do the fulfillment
- Respond to the POST request while it's still being fulfilled
- Based on a solution to a GitHub issue in plumber presented by Barret Schloerke
- I love this so much I wrote a blog post about it jnolis.com/blog/shiny_background_processes

```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))
            }
        }
    })
}
```



Actually ordering the postcards

- There are many companies that have APIs to mail postcards for you
- Basic idea: send a POST request with an image and address and they change your credit card and mail it
- Many of these are bad!!!
- Time between sending the order and getting the postcard was generally 2 weeks
- This was awful

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

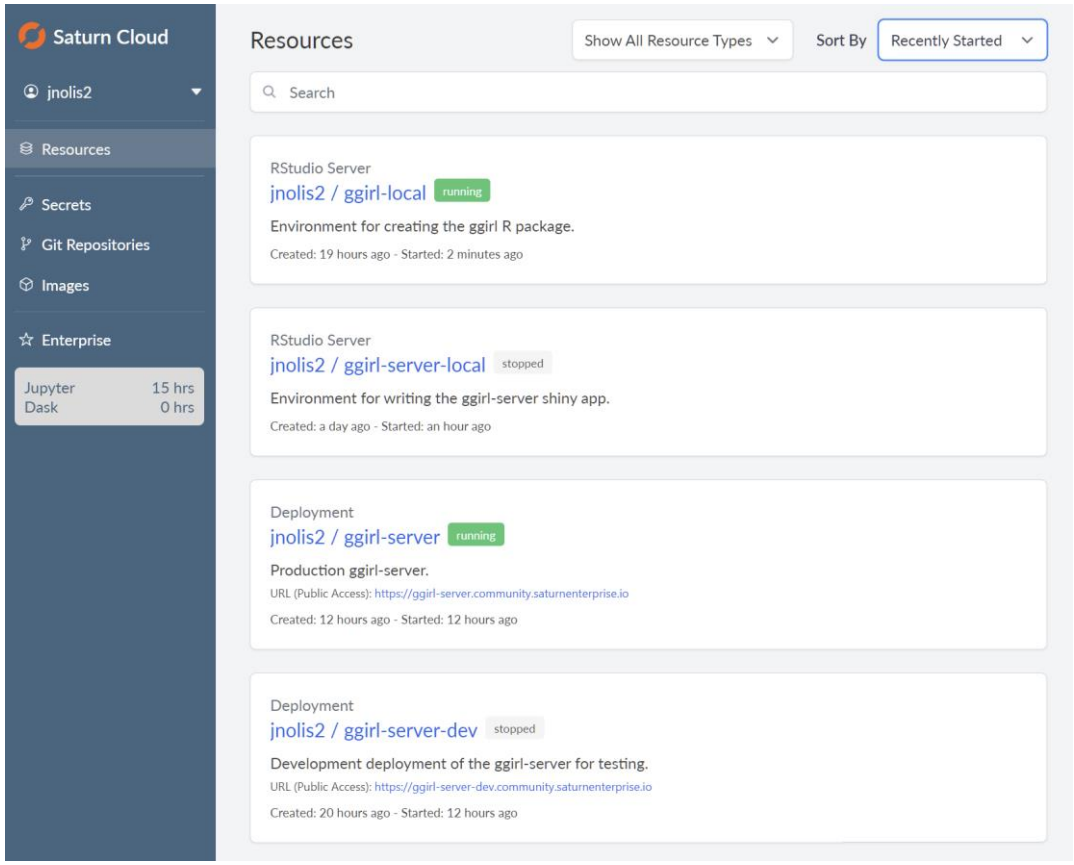
Send confirmation emails

- Use blastula to send email (easy!)
  - Include a thumbnail of postcard with {imagemagick}
- If there is an error in fulfillment function, have it email me the error message
- …if there is an error in sending an email function, send an email
- All of this is done in the background

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

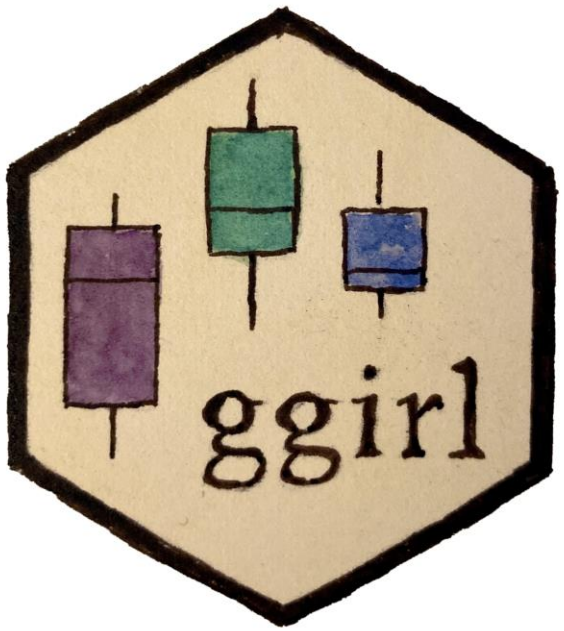
Prod deployment

Dev deployment

SaturnCloud

Wrapping it up

- {ggirl} is a cool package for ordering cool stuff
- There were a fair number of fickle parts but overall straightforward (and mostly POST requests):
  - {brochure} to handle all the requests
  - Bounce POST requests between the R client, Shiny, and Stripe
  - Use {callr} for processing in the background
  - Deploy on Saturn Cloud with dev and prod versions
- *I think I have broken even on this project!*



Thank you!

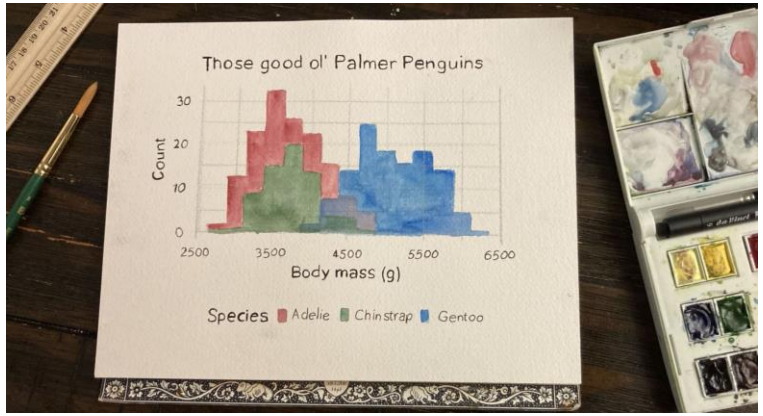
Jacqueline Nolis | @skyetetra | jnolis.com

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

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

ggpostcard()

ggartprint()



ggwatercolor()