



I made an entire e-commerce platform on Shiny

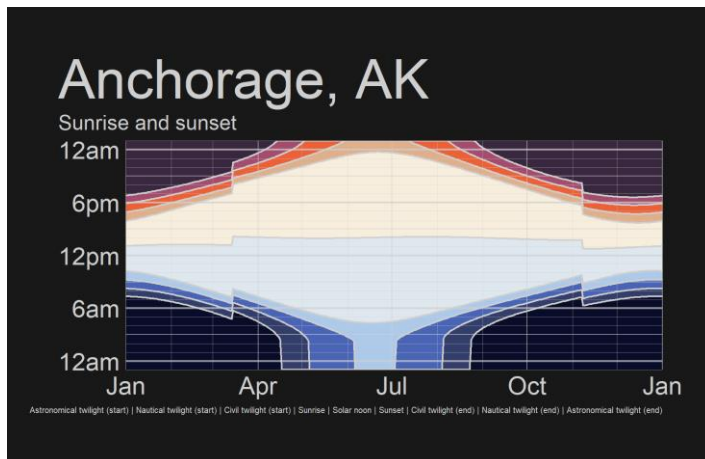
Jacqueline Nolis | @skyetetra | jnolis.com

What is {ggirl}?

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

```
remotes::install_github("j nolis/gggirl")
```

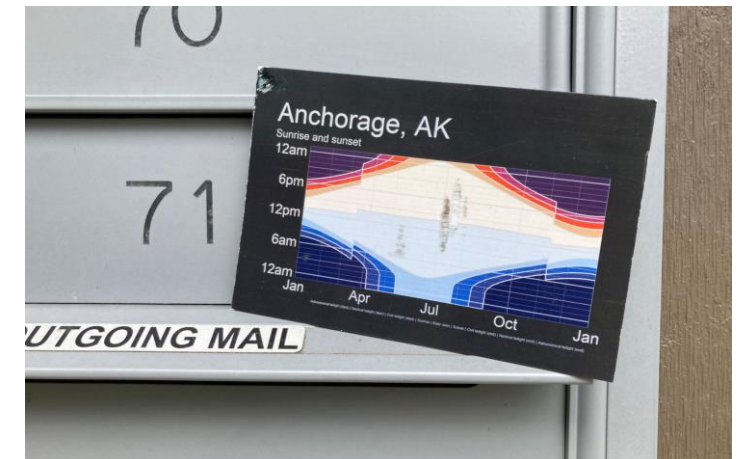
A ggplot2 plot



A modest fee



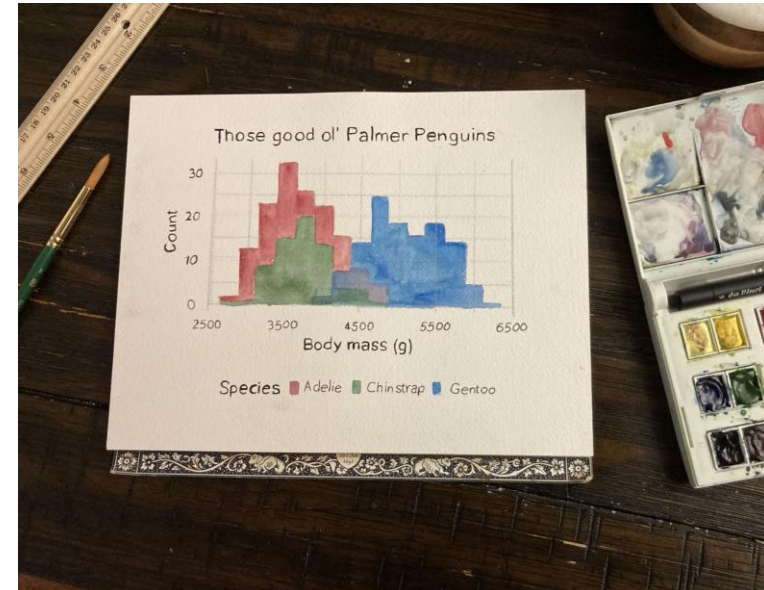
An actual postcard



ggartprint()



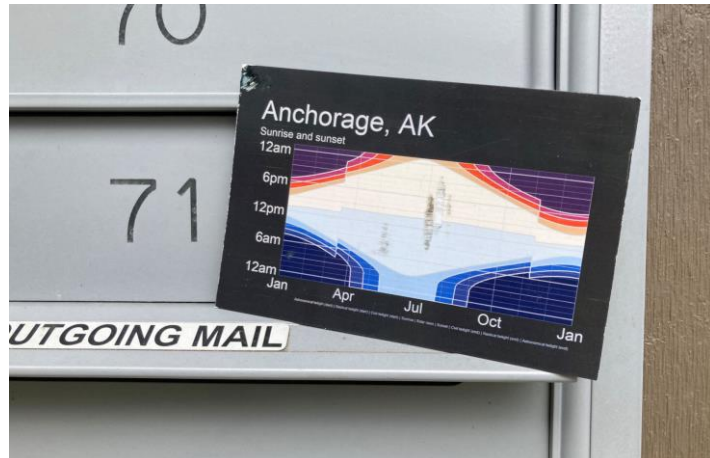
ggwatercolor()



Live demo time!

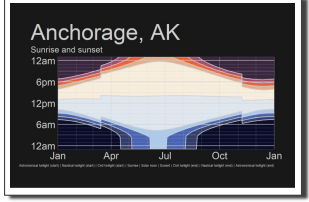
```
ggpostcard(plot = plot,  
  contact_email = "ggirl@jmolis.com",  
  messages = "Greetings from R!",  
  send_addresses =  
    address(name = "RStudio::conf",  
      address_line_1 = "165 Waterfront ST",  
      city = "National Harbor",  
      state = "MD",  
      postal_code = "20745",  
  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

Call R code



Get a postcard (& confirmation email)

Front preview



Order your ggplot2 postcard.

Check that you like the front and the back content of all the postcards. Once you're ready, click the pay and submit button to finalize the order.

- Postcards are printed on 4"x6" cardstock (14pt weight).
- Postcards take 5-10 business days between ordering and delivery for US addresses. International addresses may take several weeks.
- The postal system is a rough and tumble place—the postcard may not arrive in pristine condition.

The price for a single postcard is \$2.50.

Pay and submit

Back content

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

Preview image on a webpage

GGIRL

Pay GGIRL

\$2.50

ggplot2 postcard \$2.50

Subtotal \$2.50

Tax @ Enter address to calculate

Total due \$2.50

Pay with card

Email fakeemailforreal@gmail.com

Card information

1234 1234 1234 1234

MM / YY CVC

Name on card

Billing address

United States

Address line 1

Address line 2

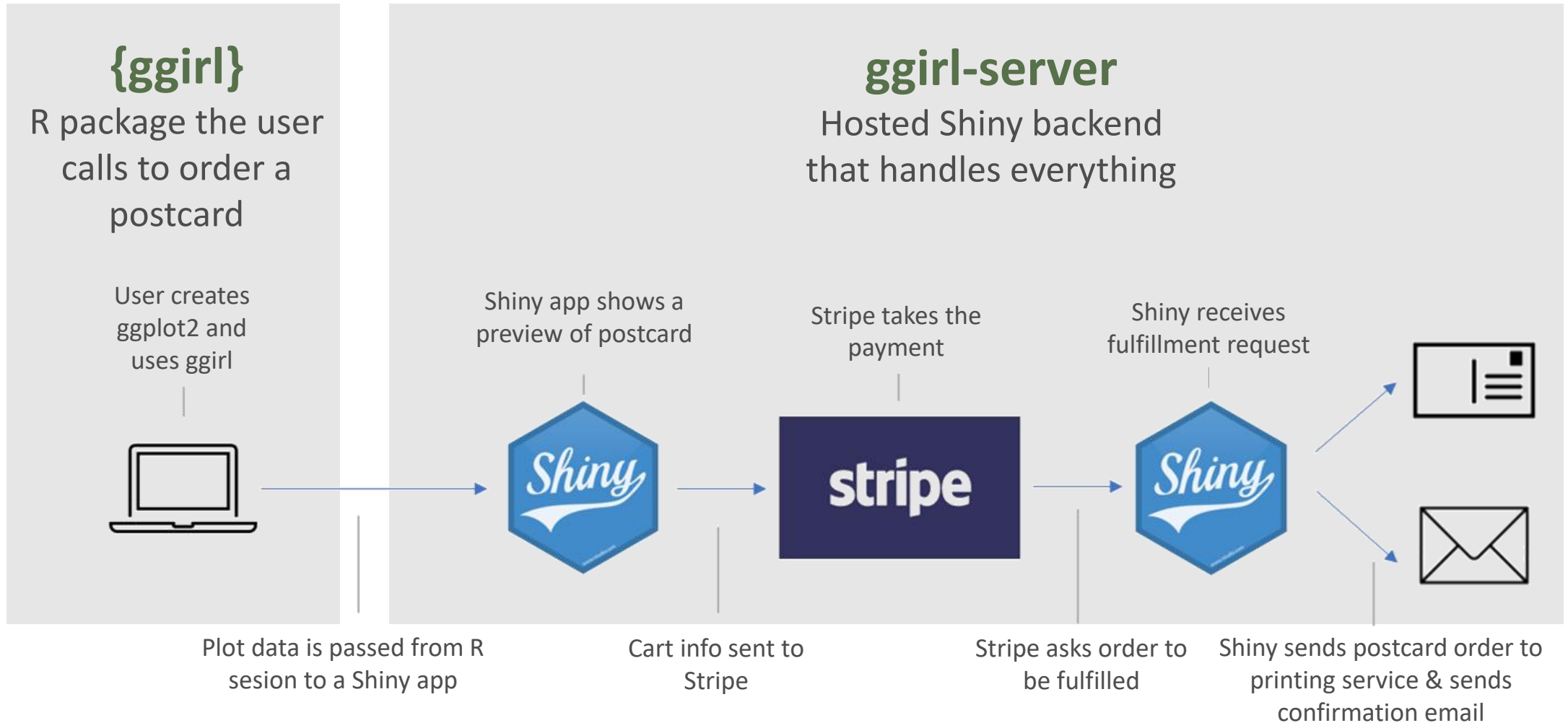
City ZIP

State

Pay \$2.50

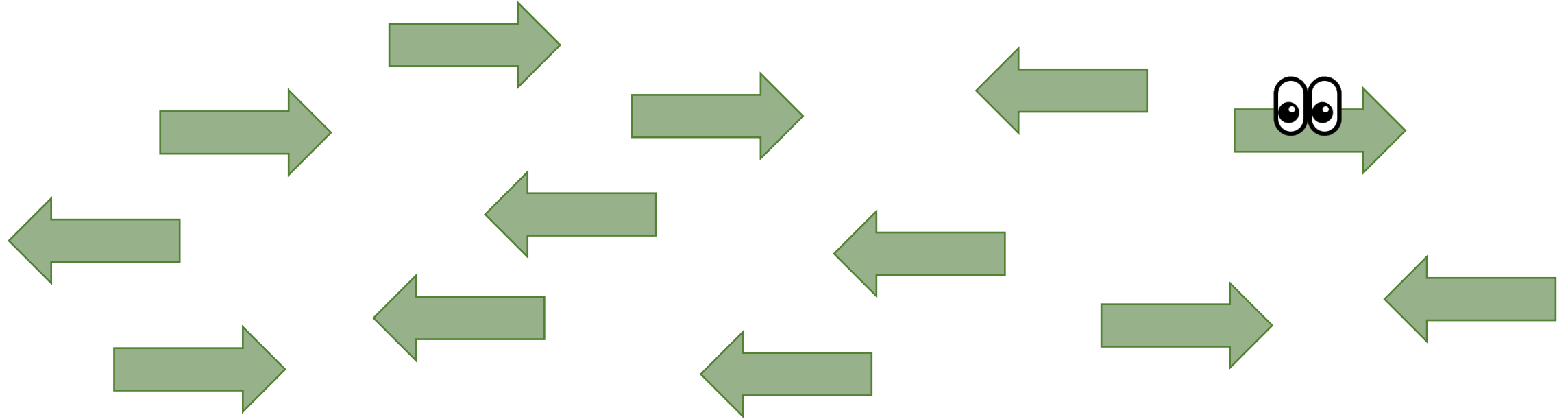
Make Stripe payment

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 {http} package

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

Receiving HTTP requests with R

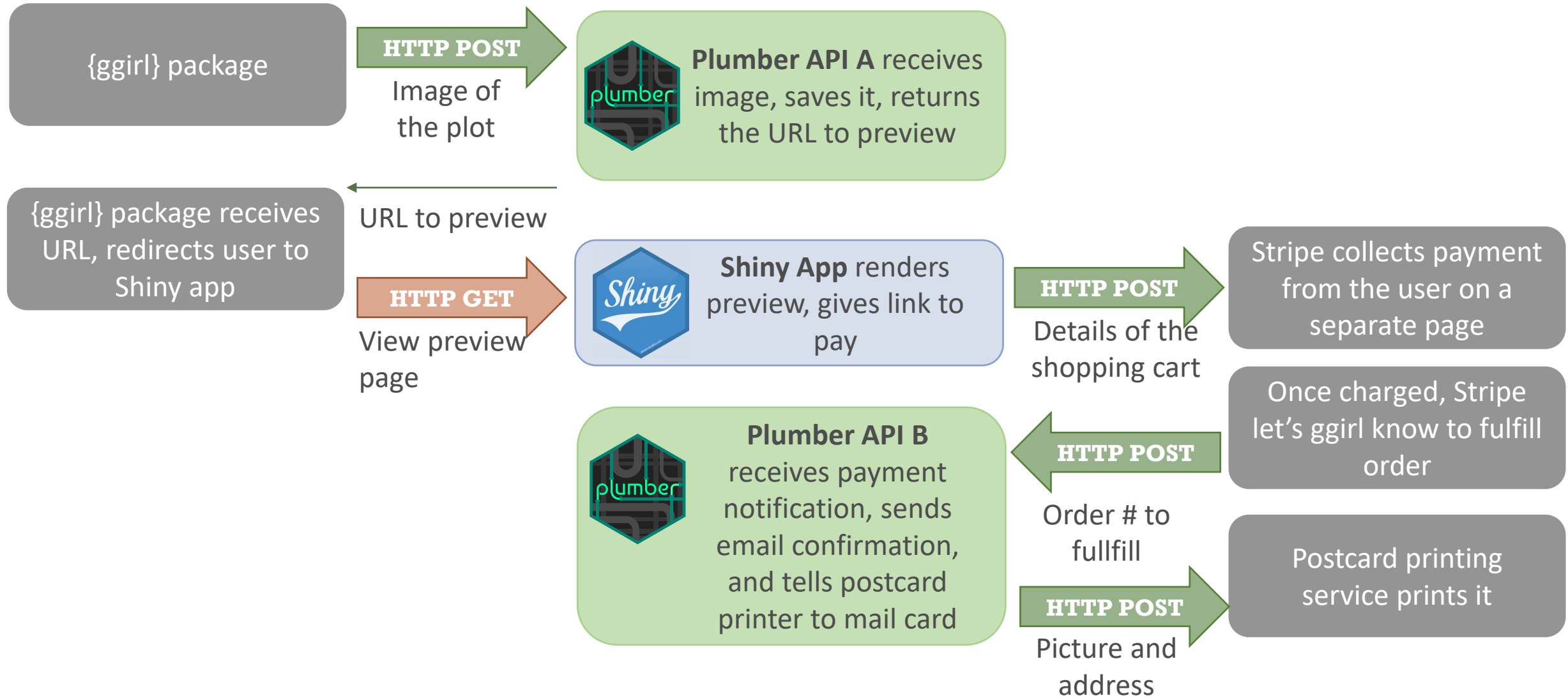


- Lots of tools to design HTML/JavaScript
- Great UI
- Only (on its face) takes GET requests
- Only (on its face) has one URL endpoint

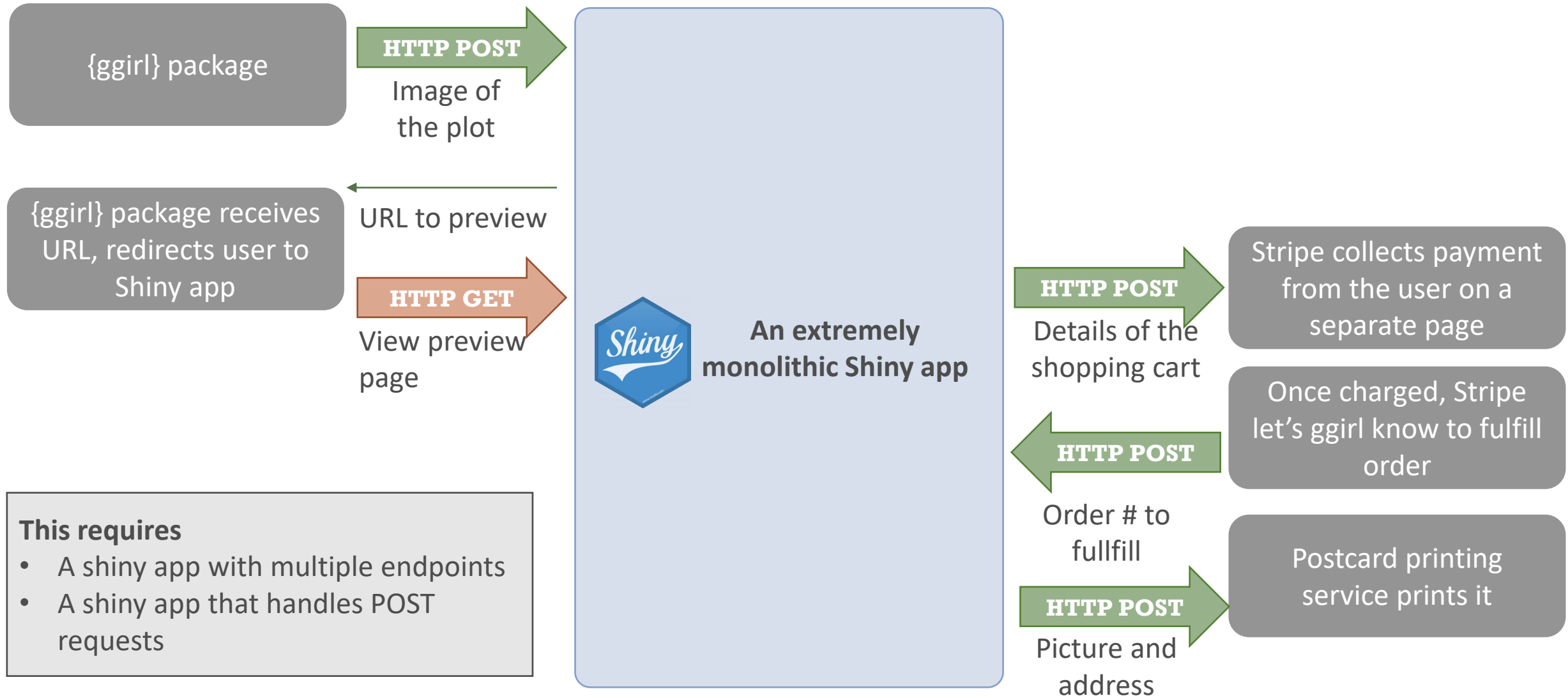


- Handles POST/GET/whatever
- No HTML pretty stuff, just send/receive data
- Can have many endpoints (example: /photos, /videos, etc)

Here is the “correct” architecture



Here is my architecture (I am lazy)

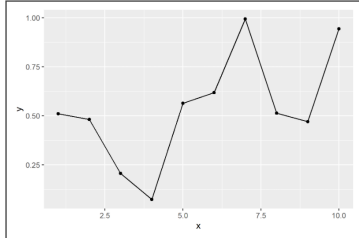


Tackling the architecture step by step

```
1 # install.packages("ggplot2")
2
3 library(ggplot2)
4
5 plot <-
6   ggplot(data.frame(x=1:10, y=runif(10)), aes(x=x, y=y)) +
7   geom_line() + geom_point() +
8   labs(title = "Hello From Cascadia R Conf!!!")
9
10 plot
11
12 # -----
13 library(ggir)
14
15 contact_email <- "mollis.test@gmail.com"
16 send_address <- address(name = "Fake Personname", address_line_1 = "250 North Ave",
17   city = "Boston", state = "MA",
18   postal_code = "22222", country = "us")
19 message <- "This plot made me think of you!"
20
21 # -----
22
23 ggpostcard(plot, contact_email, messages = message, send_addresses = send_address)
```



Front preview



Back content

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

Order your ggplot2 postcard.

Check that you like the front and the back content of all the postcards. Once you're ready, click the pay and submit button to finalize the order.

- Postcards are printed on 4"x6" cardstock (14pt weight).
- Postcards take 5-10 business days between ordering and delivery for US addresses. International addresses may take several weeks.
- The postal system is a rough and tumble place—the postcard may not arrive in pristine condition.

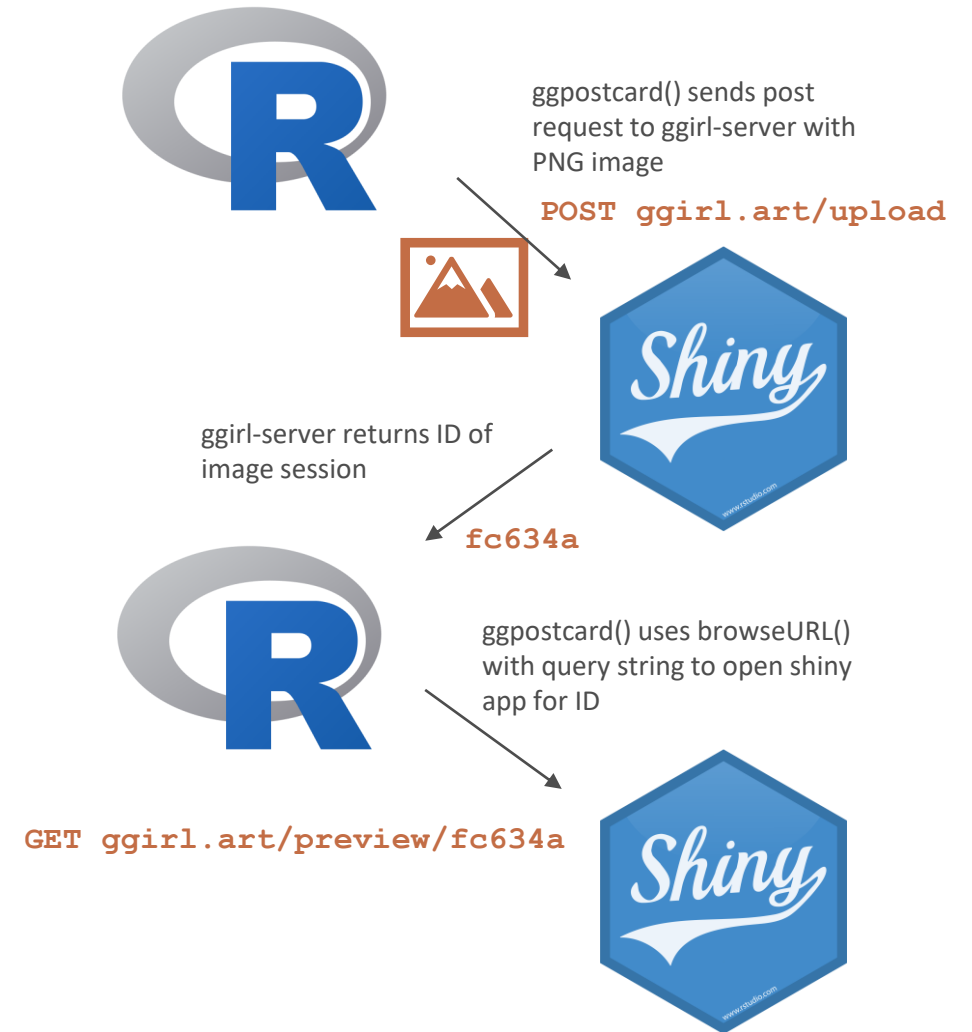
The price for a single postcard is \$2.50.

Pay and submit

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

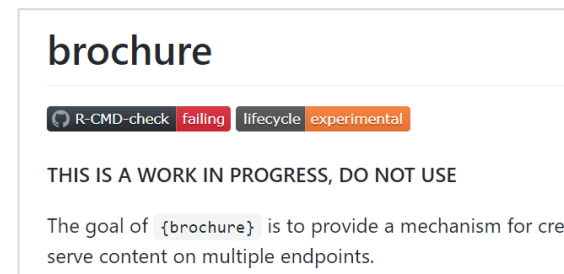
Get Shiny to handle POST requests

- 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

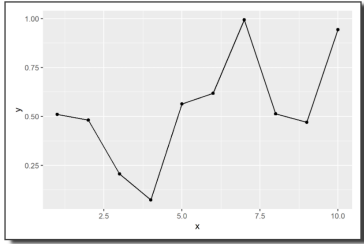


Final POST request

- {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]
```

Front preview



x	y
0.0	0.60
1.0	0.58
2.0	0.20
3.0	0.05
4.0	0.65
5.0	0.70
6.5	1.00
7.5	0.60
8.5	0.58
10.0	0.95

Order your ggplot2 postcard.

Check that you like the front and the back content of all the postcards. Once you're ready, click the pay and submit button to finalize the order.

- Postcards are printed on 4"x6" cardstock (14pt weight).
- Postcards take 5-10 business days between ordering and delivery for US addresses. International addresses may take several weeks.
- The postal system is a rough and tumble place—the postcard may not arrive in pristine condition.

The price for a single postcard is \$2.50.

Pay and submit

Back content

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

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

Shadow and frame uses a few lines of custom css

Column layout via bootstrap grid

Order your ggplot2 postcard.

Check that you like the front and the back content of all the postcards. Once you're ready, click the pay and submit button to finalize the order.

- Postcards are printed on 4"x6" cardstock (14pt weight).
- Postcards take 5-10 business days between ordering and delivery for US addresses. International addresses may take several weeks.
- The postal system is a rough and tumble place—the postcard may not arrive in pristine condition.

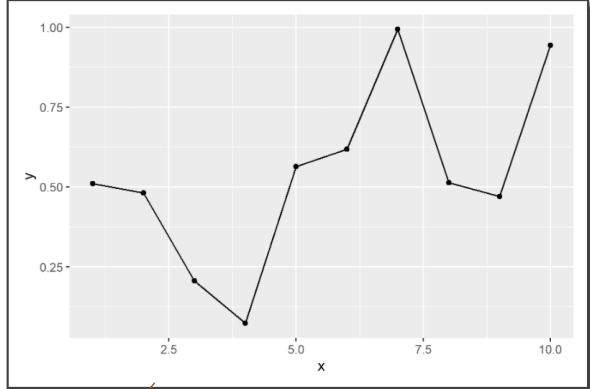
The price for a single postcard is \$2.50.

Pay and submit

The important button

{gt} table

Front preview



Back content

Message	Address
Thank you!	Fake Personname 250 North Ave Boston, MA, 22222 US

Margin added to image so printer doesn't cut off plot

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!!!



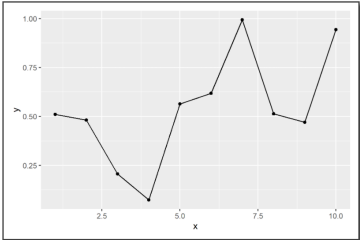
Early back preview

- Idea 1: render the back as a ggplots and use {ggfitttext} 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

Front preview



Order your ggplot2 postcard.

Check that you like the front and the back content of all the postcards. Once you're ready, click the pay and submit button to finalize the order.

- Postcards are printed on 4"x6" cardstock (14pt weight).
- Postcards take 5-10 business days between ordering and delivery for US addresses. International addresses may take several weeks.
- The postal system is a rough and tumble place—the postcard may not arrive in pristine condition.

The price for a single postcard is \$2.50.

Pay and submit

Back content

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



Pay ggplot2

\$2.50

ggplot2 postcard

A custom printed postcard of a ggplot2 plot.

Subtotal

\$2.50

Tax ID

Enter address to calculate

Total due

\$2.50

Pay with card

Email

nolis.test@gmail.com

Card information

1234 1234 1234 1234

MM / YY

CVC

Name on card

Billing address

United States

Address line 1

Address line 2

City

ZIP

State

☐ Save information to pay faster next time

Pay \$2.50

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/ggplot2-postcard

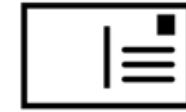
```
body_form <- list(  
  `line_items[0][price_data][unit_amount]`='250',  
  `line_items[0][price_data][currency]`='USD',  
  `line_items[0][price_data][product_data][name]`= 'ggplot2 postcard',  
  `line_items[0][quantity]`=as.character(quantity),  
)  
response <- httr::POST("https://api.stripe.com/v1/checkout/sessions",  
  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 = "}}"))
```

The screenshot shows a Stripe checkout page for a payment of \$2.50. The left sidebar lists the items: 'Pay ggirl' for \$2.50 and 'ggirl2 postcard' (a custom printed postcard of a ggirlx2 plot) for \$2.50. The subtotal and total due are both \$2.50. The right side, titled 'Pay with card', contains a form with fields for Email (nolixtest@gmail.com), Card information (1234 1234 1234 1234), Name on card, Billing address (United States), Address line 1, Address line 2, City, ZIP, and State. There is a checkbox for 'Save information to pay faster next time' and a blue 'Pay \$2.50' button at the bottom.



4. How to fulfill the order?

Have Stripe send a POST telling Shiny to order postcards

Steps:

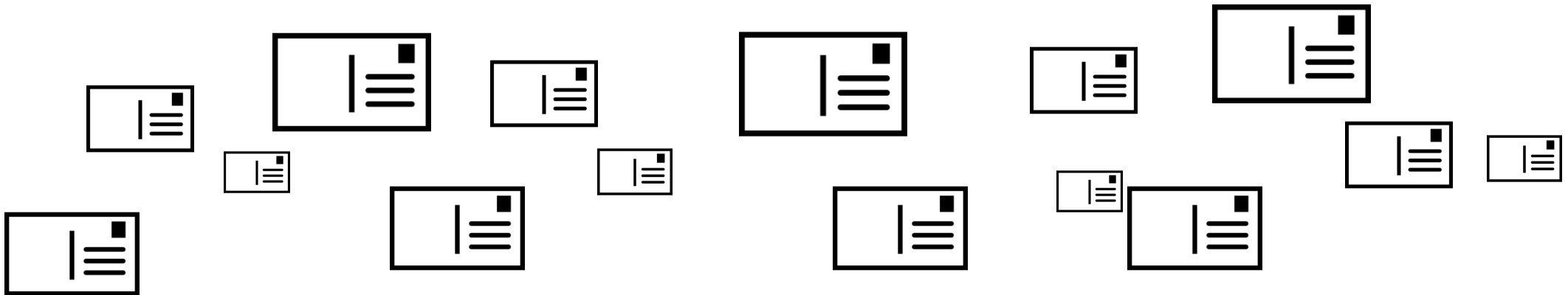
1. Stripe sends a POST request (aka webhook) saying “Order xyz has been paid for” (+ keys to ensure authentic)
2. Shiny receives the request and sends a POST to the postcard printing company
3. Shiny send an email to the user saying the order is confirmed

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

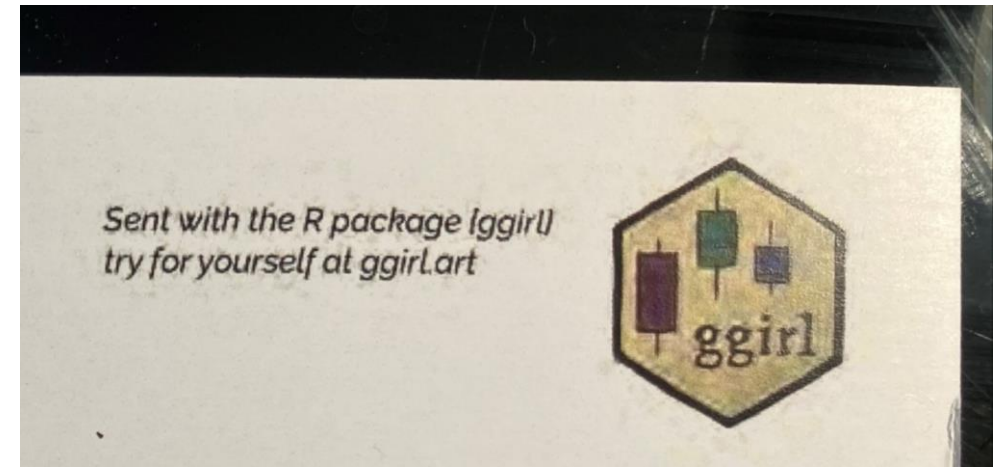
- Use {callr} package to spawn a separate parallel workflow to do the fulfillment
- Respond to the POST request while it's still being fulfilled
- Based on a solution to a GitHub issue in plumber presented by Barret Schloerke
- I love this so much I wrote a blog post about it j nol is .com/blog/shiny_background_processes

```
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

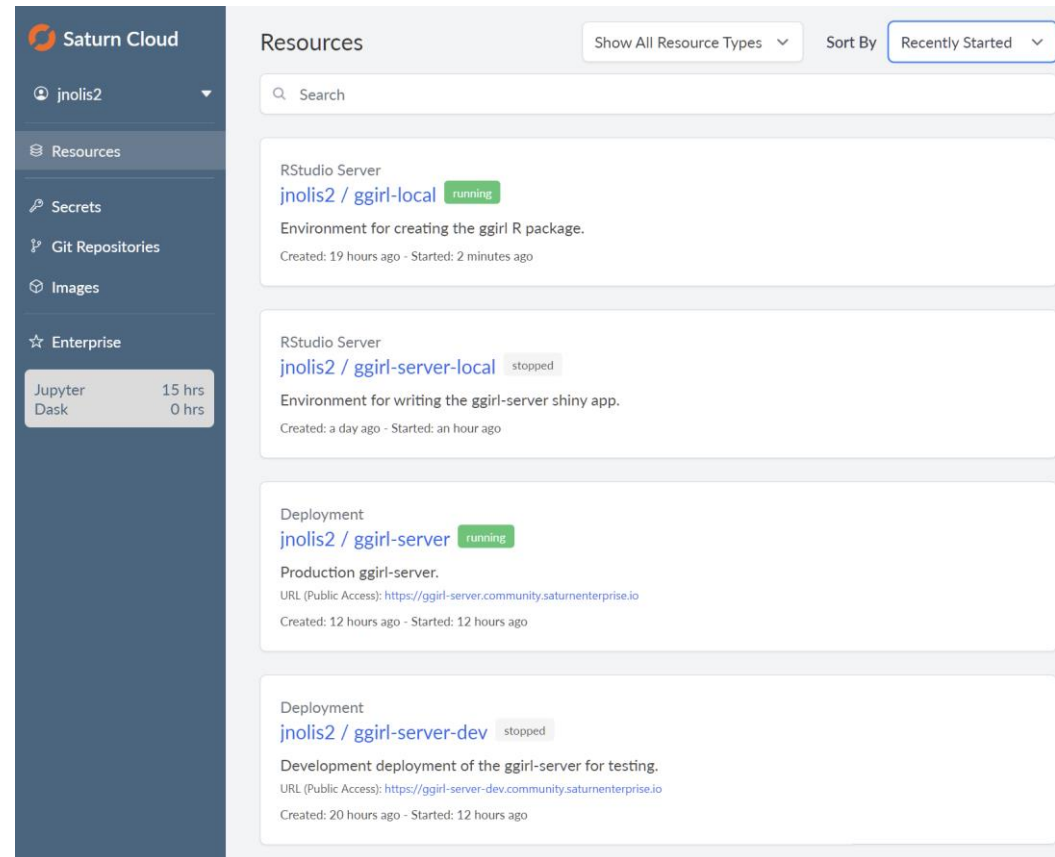
- Use blastula to send email (easy!)
 - Include a thumbnail of postcard with {imagemagick}
- If there is an error in fulfillment function, have it email me the error message
- ...if there is an error in sending an email function, send an email
- All of this is done in the background



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 ggir (R package)
 - Writing ggir-server (Shiny app)
 - Deploying dev ggir-server
 - Deploying prod ggir-server
- Ran multiple beta tests with real users to iron out the issues



The screenshot displays the Saturn Cloud web interface. On the left is a dark blue sidebar with navigation links: 'jnlis2' (selected), 'Resources', 'Secrets', 'Git Repositories', 'Images', and 'Enterprise'. The 'Resources' section is active, showing a table of resources. The table has columns for resource type, name, status, and description. The resources listed are:

Resource Type	Name	Status	Description
RStudio Server	jnlis2 / ggir-local	running	Environment for creating the ggir R package.
RStudio Server	jnlis2 / ggir-server-local	stopped	Environment for writing the ggir-server shiny app.
Deployment	jnlis2 / ggir-server	running	Production ggir-server.
Deployment	jnlis2 / ggir-server-dev	stopped	Development deployment of the ggir-server for testing.

At the top right of the resources list, there are filters: 'Show All Resource Types' and 'Sort By Recently Started'. The 'Enterprise' section in the sidebar shows 'Jupyter Dask' with '15 hrs' and '0 hrs' remaining.

R package development

Shiny app development

Prod deployment

Dev deployment

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")`

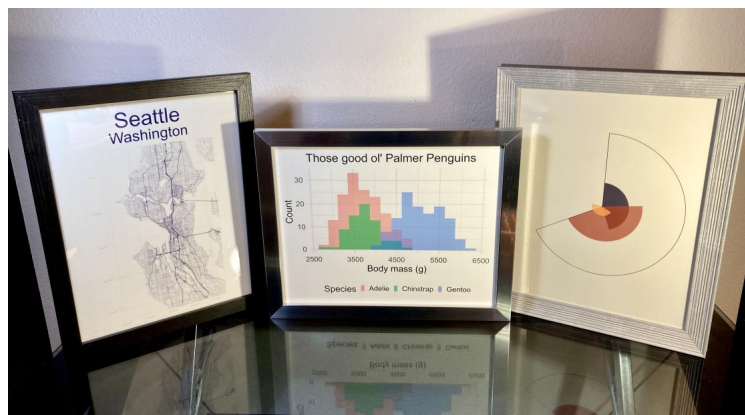
Code for the rstudio::conf(2022) postcard link.jnolis.com/rstudio22-code

Slides from this talk link.jnolis.com/rstudio22-slides

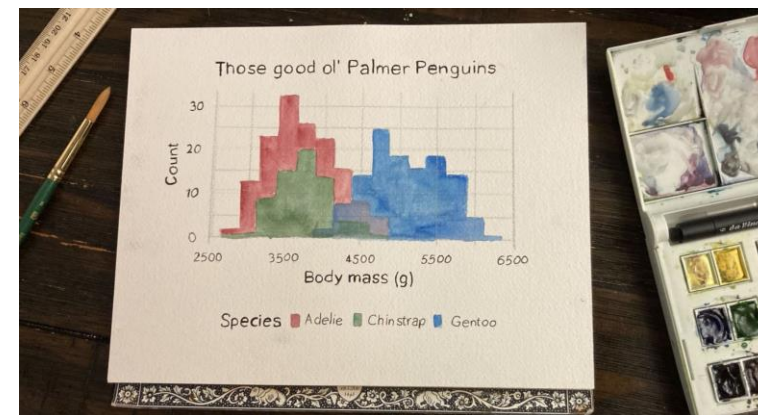
Come talk to me in person for a 80% off coupon!



ggpostcard()



ggartprint()



ggwatercolor()