Supplementing your R package with a Shiny app

Posted on April 21, 2015 | Last updated April 3, 2018

The R community is generally very fond of open-source-ness and the idea of releasing all code to the public. Writing packages has become such an easy experience now that Hadley's `devtools` is so powerful, and as a result there are new packages being released by useRs every single day.

A good package needs to have two things: useful functionality, and clear usage instructions. The former is a no-brainer, while the latter is what developers usually dread the most - the D-word (Documentation. Yikes.). Proper documentation is essential so that others will know what your package can do and how to do it. And with the use of Shiny, we now have another great tool we can use to showcase a package's capabilities.

Incorporating Shiny

In a nutshell, Shiny (https://shiny.rstudio.com/) is a package that lets you run your R code as an interactive webpage. What this means for package developers is that you can have an interactive webpage that lets users experiment with your package and see what it can do before having to read through the potentially lengthy function documentations/vignette.

As an example, I recently released a package for adding marginal plots to ggplot2. You be the judge: after that one-sentence description of some functionality, would you rather go straight to the README (https://github.com/daattali/ggExtra), or see it in action first in a Shiny app online (https://daattali.com/shiny/ggExtra-ggMarginal-demo/)? I might be wrong, but I think it's useful to interactively see what the package can do.

In fact, I'm such a strong believer in this idea, that almost every package I have released has a corresponding demo shiny app. The `shinyjs` demo app (https://daattali.com/shiny/shinyjs-demo/) and `colourpicker` demo app (https://daattali.com/shiny/colourInput/) are good examples of that.

Making a Shiny app doesn't necessarily always make sense for every package, but there are certainly many times when it can be a great addition to a package's "documentation". I think that if a new package has some functions that *can* be easily illustrated in a simple Shiny app, it's worth it to take the extra 1-2 hours to develop it. This way, a user who finds your package and isn't quite sure what to do with it can try the Shiny app to see whether or not this is the functionality they were looking for. You can have several Shiny apps, each showing the usage of a particular function, or one app that is representative of a whole package. Whatever makes the most sense. Of course, having a Shiny app is in no way a replacement to documentation, it's just a useful add-on.

There are two ways to complement a package with a Shiny app that shows its main usage. These two methods are NOT mutually exclusive; I personally do both of them together:

1. Host the Shiny app online

You can host your Shiny app somewhere that is publicly available, such as shinyapps.io (http://www.shinyapps.io/) or on your own Shiny Server (https://deanattali.com/2015/05/09/setup-rstudio-shiny-server-digital-ocean). Then you can include a link in the package's README or vignette or function documentation that points to the Shiny app.

As an example, I host my own Shiny Server (https://daattali.com/shiny) where I can host my Shiny apps, and whenever I release a new package, I include a link in the README to a demo app.

The advantage of doing this is that people can play around with your package before even downloading it.

2. Include the app in the package and add a function to launch it

I recommend including the source code of the Shiny app in your package, and having a function such as `runExample()` that will launch the app. Here are the steps to do this (I've learned a lot from looking at `shiny::runExample` source code - thanks RStudio):

First, add `shiny` as a dependency in your `DESCRIPTION` file.

Then place your Shiny app folder under `inst/shiny-examples/` and add an R file called `runExample.R`. The package's tree structure should look like this

```
- mypacakge
  |- inst
     |- shiny-examples
        |- myapp
           |- ui.R
           |- server.R
  |- R
     |- runExample.R
     |- ...
  |- DESCRIPTION
  |- ...
```

Your `runExample.R` will be simple - it will just look for the Shiny app and launch it.

```
#' @export
runExample <- function() {
  appDir <- system.file("shiny-examples", "myapp", package = "mypackage")
  if (appDir == "") {
    stop("Could not find example directory. Try re-installing `mypackage`.", call. = FALSE
  }

  shiny::runApp(appDir, display.mode = "normal")
}
```

Of course, don't forget to document this function! Now users can try out an app showcasing your package by running `mypackage::runExample()` .

This method can easily support more than one Shiny app as well, simply place each app under `inst/shiny-examples/` and change the runExample code to something like this

```
runExample <- function(example) {
  # locate all the shiny app examples that exist
  validExamples <- list.files(system.file("shiny-examples", package = "mypackage"))

  validExamplesMsg <-
    paste0(
      "Valid examples are: '",
      paste(validExamples, collapse = "', '"),
      "'")

  # if an invalid example is given, throw an error
  if (missing(example) || !nzchar(example) ||
      !example %in% validExamples) {
    stop(
      'Please run `runExample()` with a valid example app as an argument.\n',
      validExamplesMsg,
      call. = FALSE)
  }

  # find and launch the app
  appDir <- system.file("shiny-examples", example, package = "mypackage")
  shiny::runApp(appDir, display.mode = "normal")
}
```

Now running `runExample("myapp")` will launch the "myapp" app, and running `runExample()` will generate a message telling the user what examples are allowed.

3. The best of both worlds: include the app in a package AND host it on a shiny server

The solution that I end up using most often myself is a combination of both of the above. I like to supplement my pacakges with a shiny app (or sometimes create a package that is mostly a shiny app), and showcase it on my shiny server (https://daattali.com/shiny/).

To do that, follow the instructions above to create a package. Then follow my instructions to create a shiny server (https://deanattali.com/2015/05/09/setup-rstudio-shiny-server-digital-ocean/) and make sure to install your new package. The last step is to make your new package's shiny app runnable from the shiny server; simply create a file `/srv/shiny-server/myapp/app.R` with the following code:

```
dir <- system.file("shiny-examples", "myapp", package = "mypackage")
setwd(dir)
shiny::shinyAppDir(".")
```

And now your app will be available at `http://<your_shiny_server_url>/myapp`.

Tags: professional (/tags#professional) rstats (/tags#rstats) r-bloggers (/tags#r-bloggers) shiny (/tags#shiny) packages (/tags#packages) tutorial (/tags#tutorial)

Sign up to my Shiny newsletter

Email

Subscribe

← PREVIOUS POST (/2015/04/02/FLAKE-IT-TILL-YOU-MAKE-IT/)

NEXT POST → (/2015/04/23/SHINYJS-R-PACKAGE/)

16 Comments **Dean Attali** 1 **Login**

Recommend 3 Tweet Share **Sort by Newest**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Tanushree pareek • a year ago

I followed the instruction as you mentioned but when i hit the command mypackage::launch_application()

I am getting error saying :Error: Could not find example directory. Try re-installing mypackage

Is there any step I am missing after creating the function?

Thanks

• Reply • Share ›

Sungpil Han • 2 years ago

Great post. Helped me a lot. Thanks.

• Reply • Share ›

Antonio Jesús Pérez Luque • 2 years ago

Thanks for the great post.
A simple question: How to include a pkg function in the server.R? Do I need to put the function in the subdirectory of the shiny app?
Thanks

1 • Reply • Share ›

Need Shiny help? Hire me! (/shiny)

Sign up to my Shiny newsletter

Email

Subscribe

Clayton Phillips-Dorsett ➦ Antonio Jesús Pérez Luque • a year ago

Also curious. Just call 'source()' in global.R the path to the function in the R directory?

• Reply • Share ›

dougmet • 3 years ago

I've referenced this page on a GitHub issue for the runExample function: https://github.com/rstudio/.... It'd be cool if we could write shiny::runExample("name", package="packageName"). This is a really nice workaround in the meantime.

• Reply • Share ›

Dean Attali Mod ➦ dougmet • 3 years ago

Wow, what an old issue, this is before I even knew what R is :) I'm sure they'd appreciate a PR. Maybe try tagging Joe Cheng (jcheng5 on github) to get his attention, to confirm that he'd be interested in a PR

• Reply • Share ›

dougmet ➦ Dean Attali • 3 years ago

Yeah good plan. I only found it because I was about to submit a similar issue and thought I'd better check if anyone else had had the same idea!

• Reply • Share ›

jovial • 3 years ago

Awesome post.. Thanks a lot.

• Reply • Share ›

Gustavo Paterno • 4 years ago

Very nice post! I am writing an R package that is a wraper of shiny apps for basic statistic learning. For one example: https://paternogbc.shinyapp.... The students can just load different shiny apps from R studio (even offline). Your solution just made it possible and simple to organize everything in one package. Thanks a lot!

PS: I agree that including the ui and server code inside the function it is problematic, specialy if you have extra files in www folder. ;)

• Reply • Share ›

Dean Attali Mod ➦ Gustavo Paterno • 4 years ago

Awesome, good to know this actually helps

• Reply • Share ›

Need Shiny help? **Hire me! (/shiny)**

Sign up to my Shiny newsletter

Email

Subscribe

veliko • 4 years ago

There's another very easy tool to turn your models into web apps - http://www.intuitics.com , definitely worth the try

• Reply • Share ›

Dean Attali Mod ➦ veliko • 4 years ago

I'm embarrassed to say I've never heard of it, I couldn't quite understand how it differs from Shiny - do you mind sharing a two-sentence comparison?

Reply • Share ›

veliko Dean Attali • 4 years ago

Hey, sorry for the late reply - it is a Drag'n'Drop visual interface builder. It is in fact not as flexible as Shiny but it makes app development much faster, easier and lean.

The best part for you as a programmer is that you don't need to adjust your code other than to break it into functions - this makes it easy to test outside of the system unlike Shiny. Shiny is still great, this package is just more business and enterprise ready.

Reply • Share ›

Laurent Gatto • 4 years ago

Nice post, thanks!

I personally prefer to write the ui and server code directly in the functions body

```
shinyfun <- function() {
   app <- list(ui = ...
                    server = ...)
   runApp(app)
}
```

as it allows to keep the code in the ./R package dir (and hence benefit of the package checker) and document it as a any function.

Reply • Share ›

Dean Attali Mod Laurent Gatto • 4 years ago

You're right that's definitely another viable option. For more complex apps I like to separate the view from the logic and but it does have the benefit you mention

1 • Reply • Share ›

Tanushree pareek Dean Attali • a year ago

It may be very obvious question OR Am i just got confused. Sorry for that
The function to launch the shiny application, Where i have to written? I have UI.R and server.R file.....Do i need to make other R script and there I have to launch the app?

thanks

Reply • Share ›

Need Shiny help? **Hire me! (/shiny)**

Sign up to my Shiny newsletter

Email

Subscribe

● (mailto:dean@attalitech.com?subject=Hello from deanattali.com)

● (https://calendly.com/attalitech/meeting) ● (https://twitter.com/daattali)

● (https://github.com/daattali) ● (https://www.facebook.com/daattali)

● (https://stackoverflow.com/users/3943160/daattali)

Email

Subscribe

Need Shiny help? **Hire me! (/shiny)**

Sign up to my Shiny newsletter

Email

Subscribe