Jak pisać kod strony serwerowej Shiny?

Wojciech Leszkowicz, Małgorzata Stawińska, Tomasz Szmyd, Maciej Tadej

29.01.2021

1 Wstęp

Shiny to biblioteka w R pozwalająca na budowanie interaktywnych aplikacji w prosty i szybki sposób. Aplikacja Shiny składa się z dwóch części, opisywanych w dwóch osobnych plikach: interfejs użytkownika (UI), czyli jak aplikacja będzie wyglądać u użytkownika oraz sposób przetwarzania danych (serwer). W tej pracy zajmiemy się stroną serwerową Shiny.

2 Serwer Shiny

Aplikacje Shiny zazwyczaj budujemy w sytuacjach, w których mamy dane, chcemy obliczyć pewne rzeczy i narysować odpowiednie wykresy. Użytkownik widzi efekt końcowy, czyli to jak zaprogramowaliśmy gdzie ma się wyświetlać wynik, natomiast w części serwerowej opisujemy jak ten wynik ma być obliczony. Jest to więc część zależna od pliku UI.

Zobrazujemy to na prostym przykładzie danych opisujących popularność odcinków pewnego serialu. [1] Chcemy zaprezentować pewien wykres wraz z tekstem w panelu, co opisujemy w kodzie UI.

```
library(shiny)

shinyUI(fluidPage(
  titlePanel("Oceny kolejnych odcinków serialu 'Przyjaciele'"),
  sidebarLayout(
    sidebarPanel(
      p("Tutaj jest lewy panel")
    ),
    mainPanel(
      plotOutput("trend"),
      verbatimTextOutput("model")
    )
  )
))
```

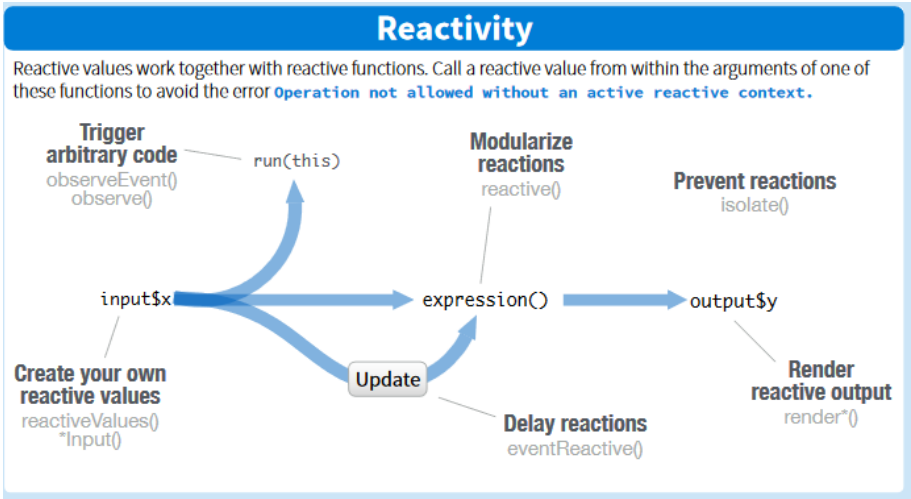
Rysunek 1: Kod UI

Musimy więc w kodzie serwera zamieścić oba obiekty opisane w UI: `trend` i `model`. Zauważmy, że tworzymy kod serwera jako funkcję od dwóch argumentów: `input, output`. W środku serwera definiujemy zależności pomiędzy `inputami` i `outputami`.

```
library(PogromcyDanych)
library(ggplot2)

shinyServer(function(input, output) {
  output$trend = renderPlot({
    przyjaciele <- serialeIMDB[serialeIMDB$serial == "Friends", ]

    ggplot(przyjaciele, aes(id, ocena, size=glosow, color=sezon)) +
      geom_point() + xlab("Numer odcinka")
  })

  output$model = renderPrint({
    przyjaciele <- serialeIMDB[serialeIMDB$serial == "Friends", ]

    summary(lm(ocena~id, przyjaciele))
  })
})
```

Rysunek 2: Kod strony serwerowej

Jedną z podstawowych zalet Shiny jest interaktywność, czyli użytkownik może na bieżąco zmieniać parametry i generować nowe wykresy. Jednak generowanie kodu na nowo przy każdej zmianie danych nie zawsze jest pożądane. Ważnym pojęciem przy pisaniu strony serwerowej jest reaktywność.



Rysunek 3: Reaktywność w Shiny. [2]

Jeśli zmienna jest reaktywna, to znaczy że jakakolwiek jej zmiana powoduje ponowne uruchomienie funkcji z nią powiązanych. Do budowania reaktywnych wyrażeń używamy

funkcji `reactive()`. Taka zmienna jest liczona tylko raz i funkcje z nią związane używają tej wartości aż do momentu zmiany jej wartości. Z pojęciem reaktywności wiąże się kila ważnych wyrażeń:

- `reactiveValues(...)`, które tworzy listę reaktywnych zmiennych
- `isolate(expr)` - zapobiega zależności od reaktywnych zmiennych
- `render*()` - funkcje tworzące obiekty do wyświetlenia, które zmieniają się wraz z reaktywnymi zmiennymi
- `observeEvent(...)` - gdy nie chcemy aby model od razu się zaktualizował przy zmianie danych, a przy jakiejś określonej akcji
- `reactive()` - tworzy reaktywne wyrażenia
- `eventReactive` - tworzy reaktywne wyrażenia, które nie zależą od wszystkich reaktywnych zmiennych, a zależą jedynie od akcji wymienionych w pierwszym argumencie

Tak jak na widocznym przykładzie poniżej[1], została użyta funkcja `reactive()` do wybrania konkretnego serialu i następnie użyto jej wewnątrz funkcji `renderPrint()`.

```
library(PogromcyDanych)
library(ggplot2)

shinyServer(function(input, output, session) {
  tylkoWybranySerial <- reactive({
    serialeIMDB[serialeIMDB$serial == input$wybranySerial, ]
  })

  output$listaOdcinkow <- renderUI({
    serial <- tylkoWybranySerial()

    selectInput("odcinki", "Odcinki w serialu", as.character(serial$nazwa) )
  })

  output$trend = renderPlot({
    serial <- tylkoWybranySerial()

    pl <- ggplot(serial, aes(id, ocena, size=glosow, color=sezon)) +
      geom_point() + xlab("Numer odcinka")
    if (input$liniaTrendu) {
      pl <- pl + geom_smooth(se=FALSE, method="lm", size=3)
    }
    pl
  })

  output$model = renderPrint({
    serial <- tylkoWybranySerial()

    summary(lm(ocena~id, serial))
  })
})
```

Rysunek 4: Użycie reaktywnej zmiennej.

Literatura

[1] Przemysław Biecek. Przewodnik po pakiecie R 4.0. `https://pbiecek.gitbooks.io/przewodnik/content/Programowanie/jak_tworzyc_aplikajce1.html`. [Online; dostęp 29-01-2021].

[2] RStudio. Shiny Cheat Sheet. `https://rstudio.com/wp-content/uploads/2016/01/shiny-cheatsheet.pdf`. [Online; dostęp 29-01-2021].