Shinymeta app (…almost)

```r
downloads <- metaReactive2({
  req(input$package)
  metaExpr(cranlogs::cran_downloads(input$package,
    from = Sys.Date() - 365, to = Sys.Date()))
})

downloads_rolling <- metaReactive({
  downloads() %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
})

output$plot <- metaRender(renderPlot, {
  ggplot(downloads_rolling(), aes(date, count)) +
    geom_line() +
    ggtitle("Seven day rolling average”)
})
```

This syntax is weird, sorry

Shinymeta app (…almost)

```r
downloads <- metaReactive2({
  req(input$package)
  metaExpr(cranlogs::cran_downloads(input$package,
    from = Sys.Date() - 365, to = Sys.Date()))
})

downloads_rolling <- metaReactive({
  downloads() %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
})

output$plot <- metaRender(renderPlot, {
  ggplot(downloads_rolling(), aes(date, count)) +
    geom_line() +
    ggtitle("Seven day rolling average”)
})
```

This syntax is weird, sorry

Using shinymeta

1. You (the app author) **identify the domain logic in your app code** so we can separate it from the reactive structure

2. Within that domain logic, you **identify references to reactive values and reactive expressions** that need to be replaced with static values and static code, respectively

3. At runtime, **choose which pieces** of domain logic to export, and in what order

4. **Present the code** to the user (in a window, as a downloadable script or report, etc.)