Integration Testing: a new way to test reactive behavior in {shiny} applications and modules

Jeff Allen, Alan Dipert, and the RStudio Shiny Team

Motivating Problems

- Reactive expressions ("reactives") are hard to test.
- Application and module server functions are hard to test as a result.

Potential Mitigations and their Tradeoffs

Factor reactives into functions with unit tests.

- Existing code must be changed in order to test it.
- Behavior of reactives over time remains untested.

Use {shinytest} for functional testing.

- Tests *everything*, not just reactive behavior.
- Snapshots not transferrable to other apps/packages.

Example: Testing a Module

```
library(shiny)

module <- function(input, output, session) {
  doubled <- reactive({ input$x * 2 })
  output$txt <- renderText({
    paste0("I am ", doubled())
  })
}

testModule(module, {
  stopifnot(is.null(input$x))
  session$setInputs(x = 2)
  stopifnot(doubled() == 4)
  stopifnot(output$txt == "I am 4")
})
```

`doubled` is a reactive internal to the module.

Text output that depends on `doubled`.

The expression depends on `input$x`.

`testModule()` establishes a context in which to test a module.

`input$x` is a "mock" input with an initial value of `NULL`.

`session$setInputs()` performs assignment.

Proposed Solution: "Integration Testing"

- Feature in forthcoming `{shiny}` 1.4
- Introduces `testModule()` and `testServer()`
- Adds `session$setInputs()` for reactive assignment.
- *Not* a new test methodology or framework.
- Usable with `{testthat}`, `{RUnit}`, or from arbitrary R code.

Implementation Notes

- Does not involve a web browser, headless or otherwise.
- Does not test UIs.
- Adds a `MockSession` class that represents an ongoing interaction with a mock user.

Ongoing Work

- Leveraging as part of Shiny release testing or "dogfooding".
- Incorporating into wider set of recommendations for Shiny app testing.
- Finalizing documentation and contributing more sophisticated examples.