Quantzig™

Powered by ART Accelerate Reinvent Transcend sanofi

Automated Testing in R Shiny

Using testthat and shinytest2

Presented by Sanjeev and Ramya

What is automated testing?

Automated testing is an approach to verifying code that execute tests automatically and then compare actual test results with expected results. It is essential for making sure that code works the way that you intend it to and keeps working even after you make changes to the code.

Why it is important?

- ✓ Automated testing improves the **coverage** of testing as automated execution of test cases is **faster** than manual execution.
- ✓ Automated testing reduces the **dependability** of testing on the availability of the test engineers.
- ✓ Automated testing takes far **less resources** in execution as compared to manual testing.
- ✓ It helps in testing which is not possible **without automation** such as reliability testing, stress testing, load and performance testing.
- ✓ It acts as test **data generator** and produces maximum test data to cover a large number of input and expected output for **result comparison.**
- ✓ Automated testing has **less chances of error** hence more reliable.

Automated testing in shiny R

There are following type of testing in shiny R:

- ✓ **Unit/Function tests (testthat)**
- ✓ **Module/Server function tests (testthat)**
- ✓ **Snapshot-based tests (shinytest2)**

For **unit tests and server function tests**, no web browser is involved, and the tests and test expectations are expressed in R code. This means that the tests run quickly, and that changing code in one part of an application will generally not affect tests of another part of the application. The **snapshot-based tests** require a headless web browser, and, as the name suggests, use snapshots.

testthat package is used to build the **unit and server function tests**, while **shinytest2** package is used to perform the **snapshot-based tests**.

Basic test structure

File

All test files live in ***tests/testthat folder***, and each test file should correspond to a code file in *R/*, e.g. the code in **R/module.R** should be tested by the code in **tests/testthat/test-module.R**.

Test

Each file is broken down into tests, i.e. a call to **test_that().** A test should generally check a Single property of a function. It's hard to describe exactly what this means, but a good heuristic is that you can easily describe the test in the first argument to test_that.

Expectation

Each test contains one or more expectations, with functions that start **with expect_.** These define exactly what you expect code to do, whether it's returning a specific value, throwing an error, or something else.

The art of testing is figuring out how to write tests that clearly define the expected behaviour of your function, without depending on incidental details that might change in the future.

testthat – create test

Step 1 : Before creating the test, **testthat** package need to be installed by using following command:

```
# Install the released version from CRAN
install.packages("testthat")
```

Step 2 : After installing the **testthat** package, test can be created in two ways :

- ✓ **Create test file manually:** All test files live in tests/testthat, and each test file should correspond to a code file in R/, e.g. the code in R/getdataset_module.R should be tested by the code in **tests/testthat/test-getdataset_module.R**.
- ✓ **Create test file using usethis::use_test function:** Fortunately, you don't have to remember that convention: just use usethis::use_test to automatically create or locate the test file corresponding to the currently open R file. If there is no "tests" directory in the project, then it will first create tests/testthat directory and then the respective test file in it.

Tree structure of **full-stack-r-shiny-testing** project directory:

```
full-stack-r-shiny-testing
├── R
│   ├── bin_calculation_function.R
│   ├── getdataforchart_module.R
│   └── getdataset_module.R
├── full-stack-r-shiny-testing.Rproj
├── global.R
├── renv
│   └── activate.R
├── renv.lock
├── server.R
├── tests
│   ├── testthat
│   │   ├── test-bin_calculation_function.R
│   │   ├── test-getdataforchart_module.R
│   │   ├── test-getdataset_module.R
│   │   └── test-server.R
│   └── testthat.R
└── ui.R
```

Corresponding test file
(tests/testthat/test-getdataset_module.R)

Reference link :
Testthat package installation

testthat – write test (for function) – step 1

Test for sum calculation function

```
# Test for bin calculation function
test_that("Test for sum calculation", {
  #Test to check the return value of the function if
  #any number out of two is non-numeric
  expect_equal(sum_of_two_numbers(num1 = "temp",num2 = 5),
               "sum can't be preformed")

  ...
  ...

})
```

1

Sum calculation function

```
#function to calculate the sum of two numbers

sum_of_two_numbers <- function(num1,num2){

    if((isFALSE(is.numeric(num1) & is.numeric(num2)))){
            #passing error message if any number is not numeric.
            result <- "sum can't be preformed"

    }else{
            #calculating sum of two numbers
            result <- as.numeric(num1 + num2)
    }
    #return the result after calculating the sum
    return(result)

}
```

testthat – write test (for function) – step 2

Test for sum calculation function

```
# Test for bin calculation function
test_that("Test for sum calculation", {

  #Test to check the return value of the function if
  #any number out of two is non-numeric
  expect_equal(sum_of_two_numbers(num1 = "temp",num2 = 5),
               "sum can't be preformed")

  #test to check the return value of the function if both
  #the numbers are numeric
  expect_equal(sum_of_two_numbers(num1 = 10,num2 = 5),
                as.numeric(15))
  #test to check the type of return value from the function
  #if both the numbers are numeric
  expect_type(sum_of_two_numbers(num1 = 10,num2 = 5),
                "double")

})
```

2

Sum calculation function

```
#function to calculate the sum of two numbers

sum_of_two_numbers <- function(num1,num2){

    if((isFALSE(is.numeric(num1) & is.numeric(num2)))){

            #passing error message if any number is not numeric.
            result <- "sum can't be preformed"

    }else{

            #calculating sum of two numbers
            result <- as.numeric(num1 + num2)
    }

    #return the result after calculating the sum
    return(result)

 }
```

testthat – write test (for function)

Test for sum calculation function

```r
# Test for bin calculation function
test_that("Test for sum calculation", {

  #Test to check the return value of the function if
  #any number out of two is non-numeric
  expect_equal(sum_of_two_numbers(num1 = "temp",num2 = 5),
               "sum can't be preformed")

  #test to check the return value of the function if both
  #the numbers are numeric
  expect_equal(sum_of_two_numbers(num1 = 10,num2 = 5),
               as.numeric(15))
  #test to check the type of return value from the function
  #if both the numbers are numeric
  expect_type(sum_of_two_numbers(num1 = 10,num2 = 5),
              "double")

})
```

Sum calculation function

```r
#function to calculate the sum of two numbers

sum_of_two_numbers <- function(num1,num2){

    if((isFALSE(is.numeric(num1) & is.numeric(num2)))){

        #passing error message if any number is not numeric.
        result <- "sum can't be preformed"

    }else{

        #calculating sum of two numbers
        result <- as.numeric(num1 + num2)
    }

    #return the result after calculating the sum
    return(result)

  }
```

1

2

testthat – write test (for module) - step 1

Test for get dataset module (server section) (test-getdataset_module.R)

```r
test_that("Test for getdataset module",{

      #testing the module server
      testServer(getdatasetServer,{
      #Setting the value of dataset input to "mtcars"
      session$setInputs(dataset = "mtcars")

      ...
      ...
      ...
      ...

      })
})
```

1

Server function of get dataset module (getdataset_module.R)

```r
getdatasetServer <- function(id) {
  moduleServer(id, function(input, output, session) {

    #defining the reactive values object
    rv_getdataset <- reactiveValues()
    #observe event to capture the change in the
    #dataset input
    observeEvent(input$dataset,ignoreNULL = TRUE,{
    #loading the selected dataset from datasets
    #package and storing it into rv_getdataset
    rv_getdataset$dataset <- as.data.frame(get(
                             input$dataset,
                             "package:datasets"))
    #storing the name of the slected dataset into
    #rv_getdataset
    rv_getdataset$name <- as.character(input$dataset)
    })
    #returning the name and actual dataset
    return(reactive(rv_getdataset))
  })
}
```

testthat – write test (for module) - step 2

Test for get dataset module (server section) (test-getdataset_module.R)

```r
test_that("Test for getdataset module",{

      #testing the module server
      testServer(getdatasetServer,{
      #Setting the value of dataset input to "mtcars"
      session$setInputs(dataset = "mtcars")
      #Storing the return value of module into the object
      rv_getdataset_return <- session$getReturned()

      ...
      ...
      ...

      })
})
```

2

Server function of get dataset module (getdataset_module.R)

```r
getdatasetServer <- function(id) {
  moduleServer(id, function(input, output, session) {

    #defining the reactive values object
    rv_getdataset <- reactiveValues()
    #observe event to capture the change in the
    #dataset input
    observeEvent(input$dataset,ignoreNULL = TRUE,{
    #loading the selected dataset from datasets
    #package and storing it into rv_getdataset
    rv_getdataset$dataset <- as.data.frame(get(
                             input$dataset,
                             "package:datasets"))
    #storing the name of the slected dataset into
    #rv_getdataset
    rv_getdataset$name <- as.character(input$dataset)
    })
    #returning the name and actual dataset
    return(reactive(rv_getdataset))
  })
}
```

testthat – write test (for module) - step 3

Test for get dataset module (server section) (test-getdataset_module.R)

```
test_that("Test for getdataset module",{

      #testing the module server
      testServer(getdatasetServer,{
      #Setting the value of dataset input to "mtcars"
      session$setInputs(dataset = "mtcars")
      #Storing the return value of module into the object
      rv_getdataset_return <- session$getReturned()
      #Test to check the class of return object
      expect_type(rv_getdataset_return(),"list")

      ...
      ...

      })
})
```

3

Server function of get dataset module (getdataset_module.R)

```
getdatasetServer <- function(id) {
  moduleServer(id, function(input, output, session) {

    #defining the reactive values object
    rv_getdataset <- reactiveValues()
    #observe event to capture the change in the
    #dataset input
    observeEvent(input$dataset,ignoreNULL = TRUE,{
    #loading the selected dataset from datasets
    #package and storing it into rv_getdataset
    rv_getdataset$dataset <- as.data.frame(get(
                             input$dataset,
                             "package:datasets"))
    #storing the name of the slected dataset into
    #rv_getdataset
    rv_getdataset$name <- as.character(input$dataset)
    })
    #returning the name and actual dataset
    return(reactive(rv_getdataset))
  })
}
```

testthat – write test (for module) - step 4

Test for get dataset module (server section) (test-getdataset_module.R)

```
test_that("Test for getdataset module",{

      #testing the module server
      testServer(getdatasetServer,{
      #Setting the value of dataset input to "mtcars"
      session$setInputs(dataset = "mtcars")
      #Storing the return value of module into the object
      rv_getdataset_return <- session$getReturned()
      #Test to check the class of return object
      expect_type(rv_getdataset_return(),"list")

      #Test to check the class of return dataset
      expect_s3_class(rv_getdataset_return()$dataset,"data.frame")
      #Test to check that the return dataframe is same
      #as per the set value of dataset input
      expect_equal(rv_getdataset_return()$dataset, mtcars)

      ...

      })
})
```

4

Server function of get dataset module (getdataset_module.R)

```
getdatasetServer <- function(id) {
  moduleServer(id, function(input, output, session) {

    #defining the reactive values object
    rv_getdataset <- reactiveValues()
    #observe event to capture the change in the
    #dataset input
    observeEvent(input$dataset,ignoreNULL = TRUE,{
    #loading the selected dataset from datasets
    #package and storing it into rv_getdataset
    rv_getdataset$dataset <- as.data.frame(get(
                                input$dataset,
                                "package:datasets"))
    #storing the name of the slected dataset into
    #rv_getdataset
    rv_getdataset$name <- as.character(input$dataset)
    })
    #returning the name and actual dataset
    return(reactive(rv_getdataset))
  })
}
```

testthat – write test (for module) - step 5

Test for get dataset module (server section) (test-getdataset_module.R)

```
test_that("Test for getdataset module",{

      #testing the module server
      testServer(getdatasetServer,{
      #Setting the value of dataset input to "mtcars"
      session$setInputs(dataset = "mtcars")
      #Storing the return value of module into the object
      rv_getdataset_return <- session$getReturned()
      #Test to check the class of return object
      expect_type(rv_getdataset_return(),"list")
      #Test to check the class of return dataset
      expect_s3_class(rv_getdataset_return()$dataset,"data.frame")
      #Test to check that the return dataframe is same
      #as per the set value of dataset input
      expect_equal(rv_getdataset_return()$dataset, mtcars)

      #Test to check the class of return dataset name
      expect_type(rv_getdataset_return()$name,"character")
      #Test to check that the name of the return dataset #is same as
      per the selected "dataset" input
      expect_equal(rv_getdataset_return()$name, "mtcars")

      })
})
```

5

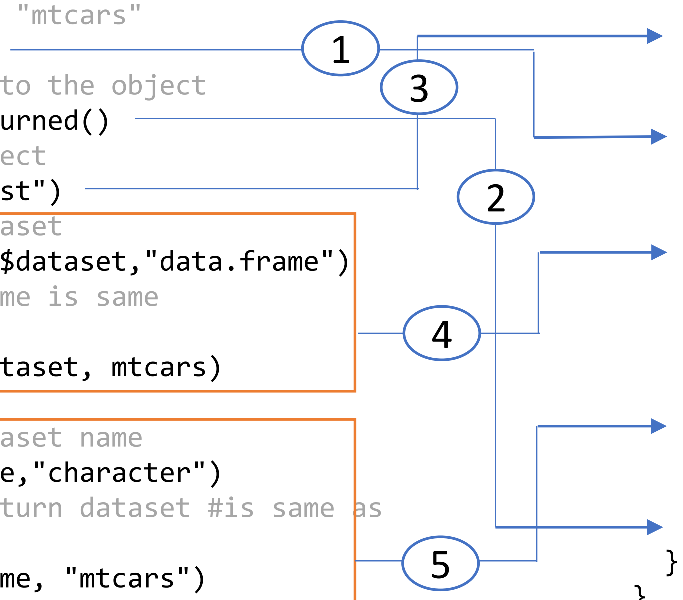
Server function of get dataset module (getdataset_module.R)

```
getdatasetServer <- function(id) {
  moduleServer(id, function(input, output, session) {

    #defining the reactive values object
    rv_getdataset <- reactiveValues()
    #observe event to capture the change in the
    #dataset input
    observeEvent(input$dataset,ignoreNULL = TRUE,{
    #loading the selected dataset from datasets
    #package and storing it into rv_getdataset
    rv_getdataset$dataset <- as.data.frame(get(
                             input$dataset,
                             "package:datasets"))
    #storing the name of the slected dataset into
    #rv_getdataset
    rv_getdataset$name <- as.character(input$dataset)
    })
    #returning the name and actual dataset
    return(reactive(rv_getdataset))
  })
}
```

testthat – write test (for module)

Test for get dataset module (server section) (test-getdataset_module.R)

```
test_that("Test for getdataset module",{

      #testing the module server
      testServer(getdatasetServer,{
      #Setting the value of dataset input to "mtcars"
      session$setInputs(dataset = "mtcars")
      #Storing the return value of module into the object
      rv_getdataset_return <- session$getReturned()
      #Test to check the class of return object
      expect_type(rv_getdataset_return(),"list")
      #Test to check the class of return dataset
      expect_s3_class(rv_getdataset_return()$dataset,"data.frame")
      #Test to check that the return dataframe is same
      #as per the set value of dataset input
      expect_equal(rv_getdataset_return()$dataset, mtcars)

      #Test to check the class of return dataset name
      expect_type(rv_getdataset_return()$name,"character")
      #Test to check that the name of the return dataset #is same as
      per the selected "dataset" input
      expect_equal(rv_getdataset_return()$name, "mtcars")

      })
})
```



Server function of get dataset module (getdataset_module.R)

```
getdatasetServer <- function(id) {
  moduleServer(id, function(input, output, session) {

    #defining the reactive values object
    rv_getdataset <- reactiveValues()
    #observe event to capture the change in the
    #dataset input
    observeEvent(input$dataset,ignoreNULL = TRUE,{
    #loading the selected dataset from datasets
    #package and storing it into rv_getdataset
    rv_getdataset$dataset <- as.data.frame(get(
                             input$dataset,
                             "package:datasets"))
    #storing the name of the slected dataset into
    #rv_getdataset
    rv_getdataset$name <- as.character(input$dataset)
    })
    #returning the name and actual dataset
    return(reactive(rv_getdataset))
  })
}
```

testthat – write test (app server) – step 1

Test for application server function

```
#Below is the test for the server function of the
application
test_that("Test for application server",{
 #testing the module server
 testServer(expr = {
  #Setting the value of emailid input
  session$setInputs(emailid = "temp@abc.com")
  #clicked on email_submit_btn button
  session$setInputs(email_submit_btn = 1)
  #Test to check the value of emailid_valid_flag. In this
  #case, it should be false.
  expect_false(rv_server$emailid_valid_flag)

  ...
  ...
  ...
  ...


 })
})
```

1

Server function of application

```
#server part of the application
app_server <- function(input, output, session) {
  #defining the reactive values object
  rv_server <- reactiveValues()
  ...
  ...
  ...

  #observe event to validate the email id and show the git
  #repository link
  observeEvent(input$email_submit_btn, {

  #assigning the value of emailid input to the reactive list
  rv_server$emailid <- input$emailid

  #validating the email id and assigning the logical flag to
  #the reactive list
  rv_server$emailid_valid_flag <- grepl("@sanofi.com",
   as.character(rv_server$emailid),ignore.case = T)

   ...
   ...

 })
}
```

testthat – write test (app server) – step 2

Test for application server function

```
#Below is the test for the server function of the
application
test_that("Test for application server",{
 #testing the module server
 testServer(expr = {
  #Setting the value of emailid input
  session$setInputs(emailid = "temp@abc.com")
  #clicked on email_submit_btn button
  session$setInputs(email_submit_btn = 1)
  #Test to check the value of emailid_valid_flag. In this
  #case, it should be false.
  expect_false(rv_server$emailid_valid_flag)
  #Setting the value of emailid input
  session$setInputs(emailid = "temp@sanofi.com")
  #clicked on email_submit_btn
  session$setInputs(email_submit_btn = 2)
  #Test to check the value of emailid_valid_flag. In this
  #case, it should be true.
  expect_true(rv_server$emailid_valid_flag)

  ...
  ...

 })
})
```

2

Server function of application

```
#server part of the application
app_server <- function(input, output, session) {
  #defining the reactive values object
  rv_server <- reactiveValues()
  ...
  ...
  ...

  #observe event to validate the email id and show the git
  #repository link
  observeEvent(input$email_submit_btn, {

  #assigning the value of emailid input to the reactive list
  rv_server$emailid <- input$emailid

  #validating the email id and assigning the logical flag to
  #the reactive list
  rv_server$emailid_valid_flag <- grepl("@sanofi.com",
   as.character(rv_server$emailid),ignore.case = T)

   ...
   ...

 })
}
```

testthat – write test (app server) – step 3

Test for application server function

```
#Below is the test for the server function of the
application
test_that("Test for application server",{
 #testing the module server
 testServer(expr = {
  #Setting the value of emailid input
  session$setInputs(emailid = "temp@abc.com")
  #clicked on email_submit_btn button
  session$setInputs(email_submit_btn = 1)
  #Test to check the value of emailid_valid_flag. In this
  #case, it should be false.
  expect_false(rv_server$emailid_valid_flag)
  #Setting the value of emailid input
  session$setInputs(emailid = "temp@sanofi.com")
  #clicked on email_submit_btn
  session$setInputs(email_submit_btn = 2)
  #Test to check the value of emailid_valid_flag. In this
  #case, it should be true.
  expect_true(rv_server$emailid_valid_flag)
  #Test to check the type of emailid_valid_flag. it should
  #always return the logical value.
  expect_type(rv_server$emailid_valid_flag,"logical")

 })
})
```

3

Server function of application

```
#server part of the application
app_server <- function(input, output, session) {
  #defining the reactive values object
  rv_server <- reactiveValues()
  ...
  ...
  ...

  #observe event to validate the email id and show the git
  #repository link
  observeEvent(input$email_submit_btn, {

  #assigning the value of emailid input to the reactive list
  rv_server$emailid <- input$emailid

  #validating the email id and assigning the logical flag to
  #the reactive list
  rv_server$emailid_valid_flag <- grepl("@sanofi.com",
   as.character(rv_server$emailid),ignore.case = T)

   ...
   ...

 })
}
```

testthat – write test (app server)

Test for application server function

```
#Below is the test for the server function of the
application
test_that("Test for application server",{
 #testing the module server
 testServer(expr = {
  #Setting the value of emailid input
  session$setInputs(emailid = "temp@abc.com")
  #clicked on email_submit_btn button
  session$setInputs(email_submit_btn = 1)
  #Test to check the value of emailid_valid_flag. In this
  #case, it should be false.
  expect_false(rv_server$emailid_valid_flag)
```

1

```
  #Setting the value of emailid input
  session$setInputs(emailid = "temp@sanofi.com")
  #clicked on email_submit_btn
  session$setInputs(email_submit_btn = 2)
  #Test to check the value of emailid_valid_flag. In this
  #case, it should be true.
  expect_true(rv_server$emailid_valid_flag)
```

2

```
  #Test to check the type of emailid_valid_flag.
  #it should always return the logical value.
  expect_type(rv_server$emailid_valid_flag,"logical")

 })
})
```

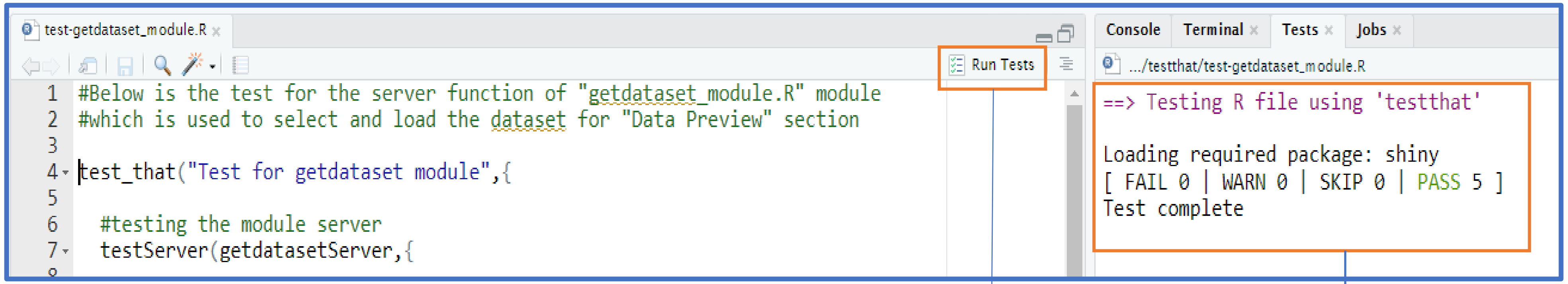
3

Server function of application

```
#server part of the application
app_server <- function(input, output, session) {
  #defining the reactive values object
  rv_server <- reactiveValues()
  ...
  ...
  ...

  #observe event to validate the email id and show the git
  #repository link
  observeEvent(input$email_submit_btn, {

  #assigning the value of emailid input to the reactive list
  rv_server$emailid <- input$emailid

  #validating the email id and assigning the logical flag to
  #the reactive list
  rv_server$emailid_valid_flag <- grepl("@sanofi.com",
   as.character(rv_server$emailid),ignore.case = T)

   ...
   ...

 })
}
```

testthat – run test(individual)

"Run Tests" button used to run the test individually. As soon as this button clicked, it run the current open test and show the test result under the Tests tab. This button will only appear for test file.



```
#Below is the test for the server function of "getdataset_module.R" module
#which is used to select and load the dataset for "Data Preview" section

test_that("Test for getdataset module",{

  #testing the module server
  testServer(getdatasetServer,{
```

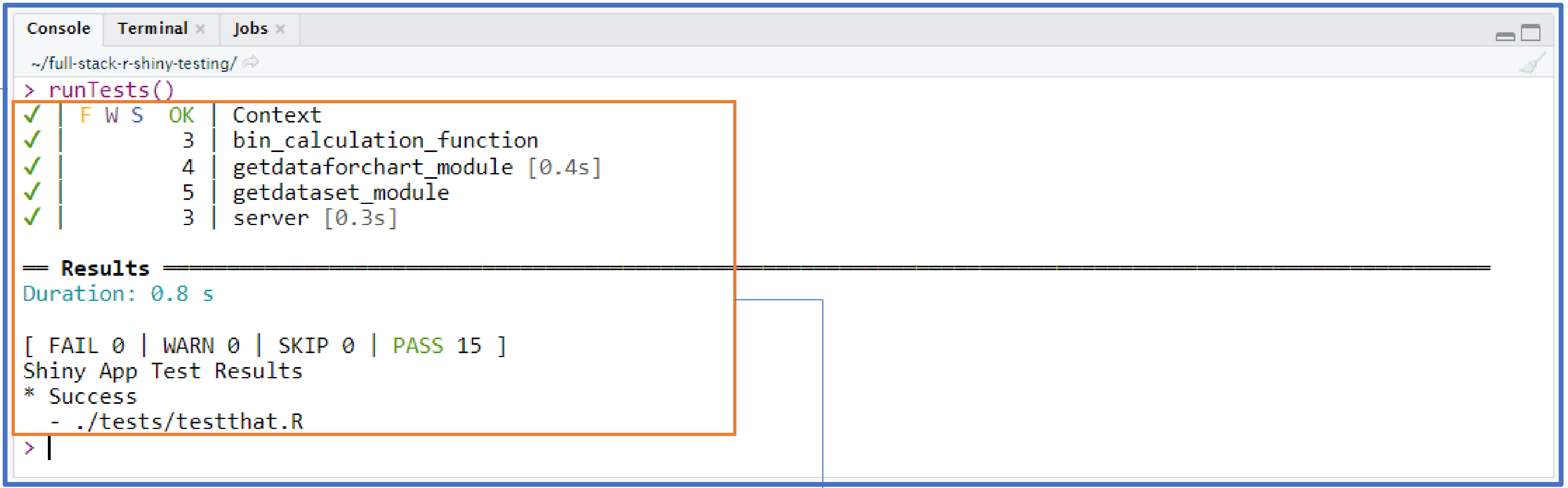
```
==> Testing R file using 'testthat'

Loading required package: shiny
[ FAIL 0 | WARN 0 | SKIP 0 | PASS 5 ]
Test complete
```

Step 1: Click on "**Run Tests**" button to run the current open test.

Step 2: After clicking on "**Run Tests**" button, Test result will be displayed in "**Tests**" tab.

testthat – run test(all)

runTests() function used to run all the tests which are exist in the tests/testthat directory. All tests run one by one automatically and show the cumulative result in the console.



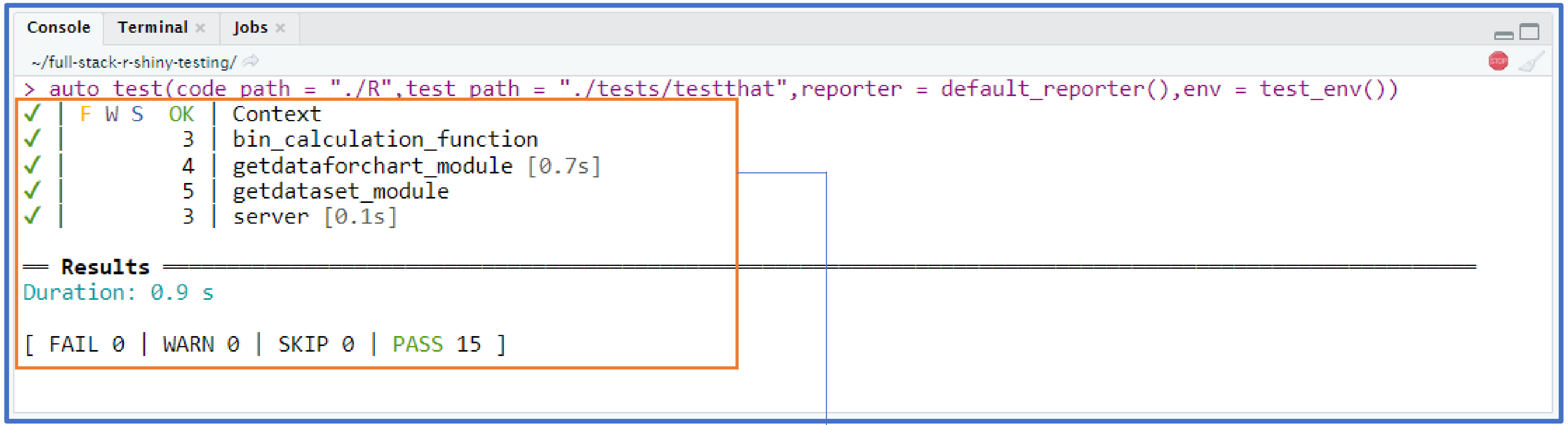
```
> runTests()
✓ |  F W S  OK | Context
✓ |          3 | bin_calculation_function
✓ |          4 | getdataforchart_module [0.4s]
✓ |          5 | getdataset_module
✓ |          3 | server [0.3s]

══ Results ═════════════════════════════════════════════════════════
Duration: 0.8 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 15 ]
Shiny App Test Results
* Success
  - ./tests/testthat.R
> 
```

Step 1: Run the **runTests()** function in the console. It will run all the tests which are exist in the *testthat* folder.

Step 2: After successfully passed all the tests, expected output will look like this.

testthat – run test(auto)

As soon as **auto_test()** function run in the console, it runs all the tests in the background which are exist in the tests/testthat directory and display the test result. It rerun automatically if any change saved in the corresponding code file. Just press "esc" button or click on terminate button in the console to stop the auto test mode.

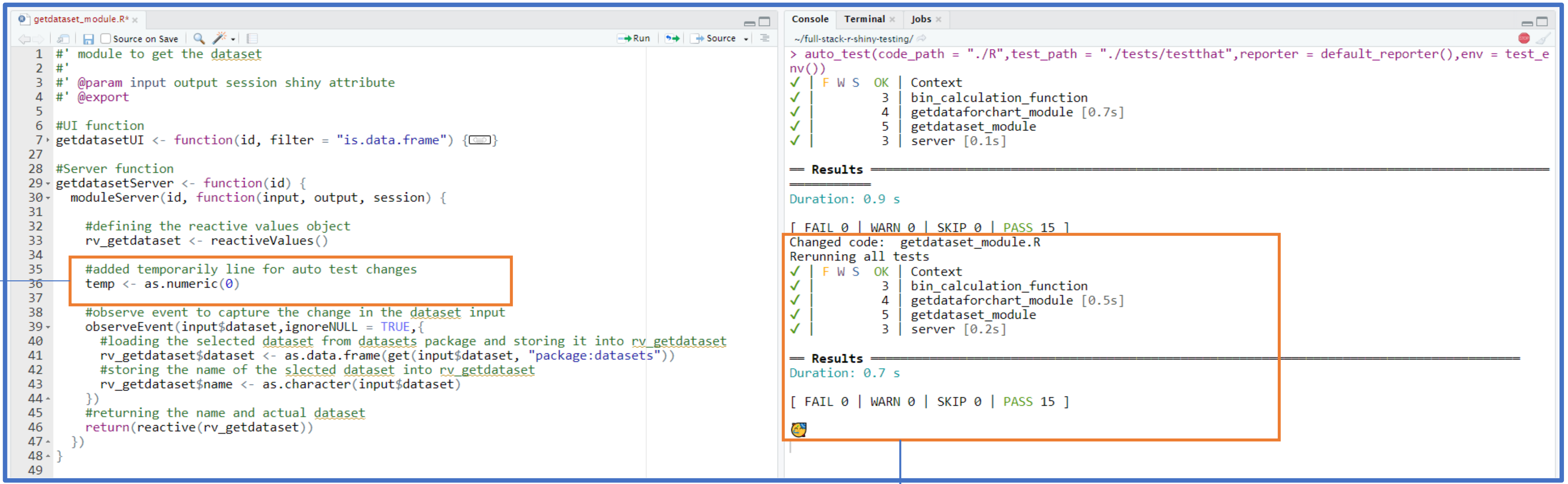


```
> auto_test(code_path = "./R",test_path = "./tests/testthat",reporter = default_reporter(),env = test_env())
✓ | F W S  OK | Context
✓ |         3 | bin_calculation_function
✓ |         4 | getdataforchart_module [0.7s]
✓ |         5 | getdataset_module
✓ |         3 | server [0.1s]

══ Results ═════════════════════════════════════════════════════════════════
Duration: 0.9 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 15 ]
```

Step 1: execute **auto_test()** command in the console which will run the tests in the background.

Step 2: After running the **auto_test()** command, It will show the above result if all the tests run successfully.

testthat – run test(auto)



```r
#' module to get the dataset
#'
#' @param input output session shiny attribute
#' @export

#UI function
getdatasetUI <- function(id, filter = "is.data.frame") {...}

#Server function
getdatasetServer <- function(id) {
  moduleServer(id, function(input, output, session) {

    #defining the reactive values object
    rv_getdataset <- reactiveValues()

    #added temporarily line for auto test changes
    temp <- as.numeric(0)

    #observe event to capture the change in the dataset input
    observeEvent(input$dataset,ignoreNULL = TRUE,{
      #loading the selected dataset from datasets package and storing it into rv_getdataset
      rv_getdataset$dataset <- as.data.frame(get(input$dataset, "package:datasets"))
      #storing the name of the slected dataset into rv_getdataset
      rv_getdataset$name <- as.character(input$dataset)
    })
    #returning the name and actual dataset
    return(reactive(rv_getdataset))
  })
}
```

```
> auto_test(code_path = "./R",test_path = "./tests/testthat",reporter = default_reporter(),env = test_e
nv())
✓ | F W S  OK | Context
✓ |          3 | bin_calculation_function
✓ |          4 | getdataforchart_module [0.7s]
✓ |          5 | getdataset_module
✓ |          3 | server [0.1s]

══ Results ═════════════════════════════════════════════════════════════
═══════════
Duration: 0.9 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 15 ]
Changed code:  getdataset_module.R
Rerunning all tests
✓ | F W S  OK | Context
✓ |          3 | bin_calculation_function
✓ |          4 | getdataforchart_module [0.5s]
✓ |          5 | getdataset_module
✓ |          3 | server [0.2s]

══ Results ═════════════════════════════════════════════════════════════
Duration: 0.7 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 15 ]

🥳
```

Step 3: Made some changes in the code and save it.

Step 4: As soon as code is saved with the changes, it **rerun** all the tests and show the above highlighted result.

shinytest2 – create test

Step 1 : Before creating the test, **shinytest2** package need to be installed by using following command:

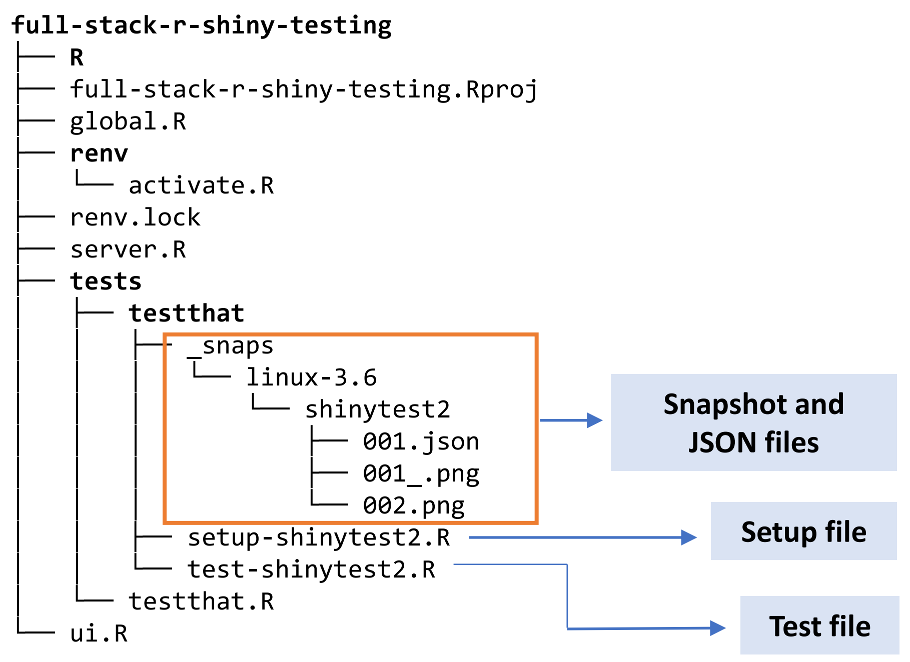
```
# Install the released version from CRAN
install.packages("shinytest2")
```

Step 2 : After installing the package, test can be created in two ways:

✓ **Create test file using shinytest2::record_test():**
To record the test, run shinytest2::record_test() in app directory. record_test() will auto-generate a test file which contains all the app interactions as code output. The snapshots are saved as .png file in .tests/testthat/_snaps folder.

✓ **Create test file using shinytest2::use_shinytest2_test():**
use_shinytest2_test() function is used to create tests manually, by setting up initial values/inputs in shinytest2. It creates a test file in .tests/testthat/test-shinytest2.R.

Tree structure of **full-stack-r-shiny-testing** project directory:



```
full-stack-r-shiny-testing
├── R
├── full-stack-r-shiny-testing.Rproj
├── global.R
├── renv
│   └── activate.R
├── renv.lock
├── server.R
├── tests
│   ├── testthat
│   │   ├── _snaps
│   │   │   └── linux-3.6
│   │   │       └── shinytest2
│   │   │           ├── 001.json
│   │   │           ├── 001_.png
│   │   │           └── 002.png
│   │   ├── setup-shinytest2.R
│   │   └── test-shinytest2.R
│   └── testthat.R
└── ui.R
```

Reference link :
shinytest2 package installation

shinytest2– Create test(Multiple Test scripts)

Test code(test-shinytest2.R)

```
#Below are the recorded shinytests
library(shinytest2)
```

```
test_that("{shinytest2} recording: full-stack-r-
shiny-testing", {
#adjusts screen height and Width automatically
app <- AppDriver$new(variant
= platform_variant(), name = "full-stack-r-shiny-
testing", height = 617, width = 1065)
#Sets dataset input id to "cars"
app$set_inputs(`getdataset-dataset` = "cars")
...
...
})
```

Test-1

```
test_that("{shinytest2} recording:
"Testing_scenario_1", {
#adjusts screen height and Width automatically
app <- AppDriver$new(variant
= platform_variant(), name
= "Testing_scenario_1", height = 617, width =
1065)
#Sets dataset input id to "cars"
app$set_inputs(`getdataset-dataset` = "CO2")
...
})
```

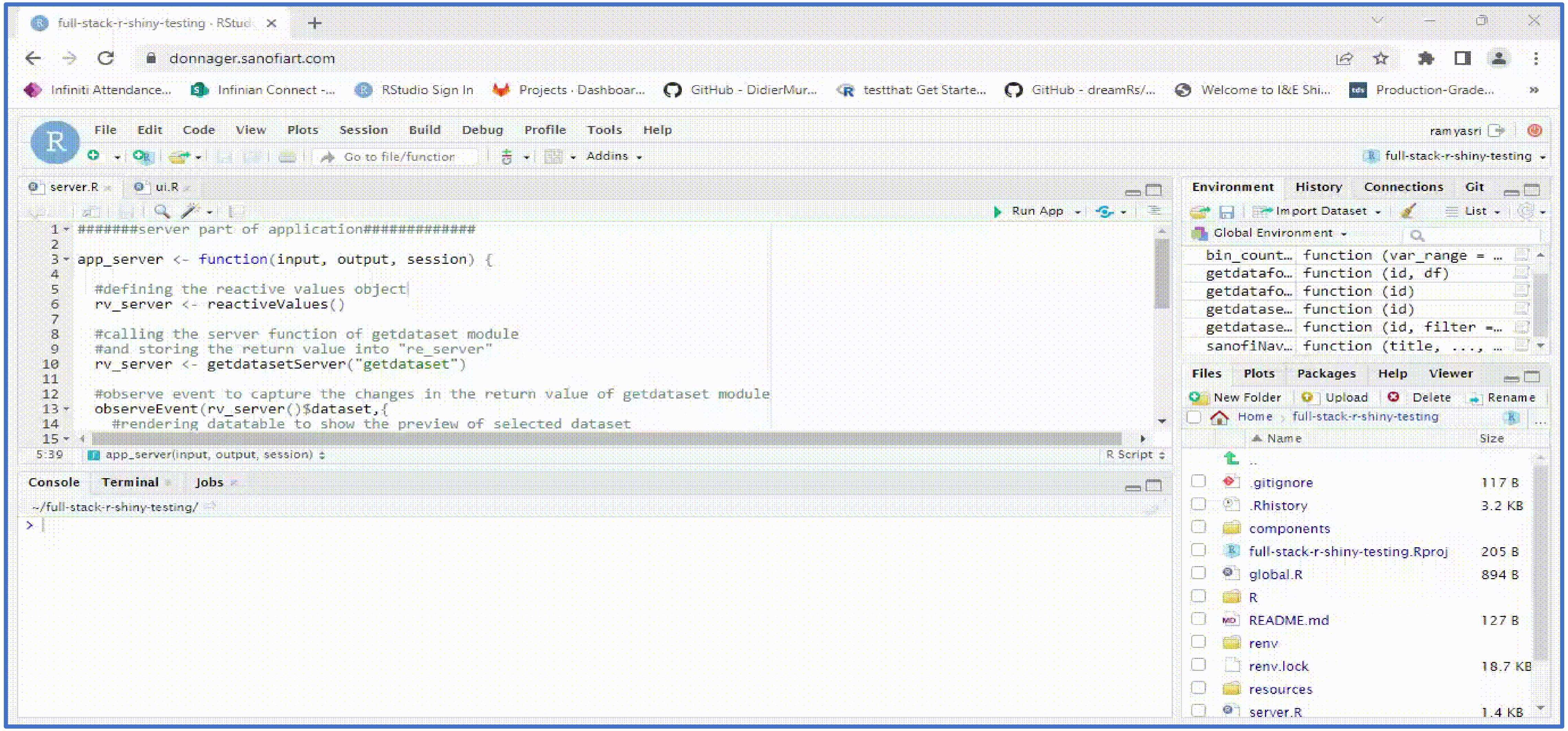
Test-2

- ✓ By default, all the recorded tests will be saved in test-shinytest2.R file.
- ✓ User can rename the default filename shinytest2.R and run the tests with user given name.
- ✓ Multiple tests will be recorded for a shiny application. Each test should be saved with a unique name.
- ✓ For each test user can create separate test file under "tests/testthat/" folder with a naming convention of "test-" as a prefix before test name.

shinytest2 – record test

- ✓ Run **shinytest2::record_test()**, it launches shiny app with an iframe on left side know as Target app and have some controls on the right side know as Recorder app.
- ✓ When user interacts with target app those actions will be recorded as code output and saved in **tests/testthat/test-shinytest2.R**.
- ✓ It performs some scripted interactions with the app and takes one or more snapshots of the application's state. These snapshots are saved to disk which will used as **reference** images for future runs of the tests to compare with this reference image.
- ✓ When you are done recording the sequence of events, save the test with a unique test name and click on "**Save and Exit**" button.
- ✓ Once you quit the Recorder app, it will automatically run the test scripts in the console and saves the snapshots and json file under **snaps** folder and creates the setup file and updates the shinytest2.R file.

shinytest2 – record test



```r
######server part of application############

app_server <- function(input, output, session) {

  #defining the reactive values object
  rv_server <- reactiveValues()

  #calling the server function of getdataset module
  #and storing the return value into "re_server"
  rv_server <- getdatasetServer("getdataset")

  #observe event to capture the changes in the return value of getdataset module
  observeEvent(rv_server()$dataset,{
    #rendering datatable to show the preview of selected dataset
```

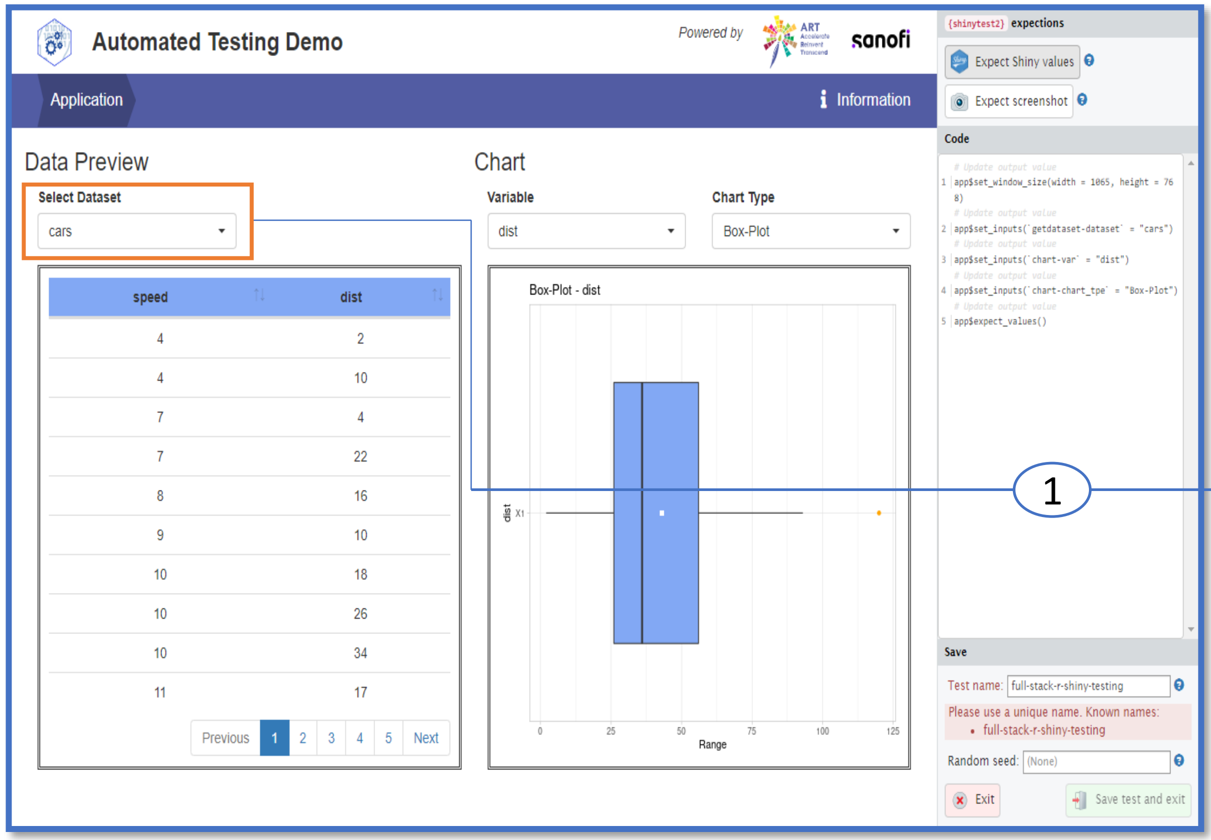
```
bin_count…    function (var_range = …
getdatafo…    function (id, df)
getdatafo…    function (id)
getdatase…    function (id)
getdatase…    function (id, filter =…
sanofiNav…    function (title, ..., …
```

```
..
.gitignore                      117 B
.Rhistory                       3.2 KB
components
full-stack-r-shiny-testing.Rproj 205 B
global.R                        894 B
R
README.md                       127 B
renv
renv.lock                       18.7 KB
resources
server.R                        1.4 KB
```

full-stack-r-shiny-testing - RStud
donnager.sanofiart.com
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
ramyasri
server.R
ui.R
Run App
5:39 app_server(input, output, session)
R Script
Console Terminal Jobs
~/full-stack-r-shiny-testing/
Environment History Connections Git
Import Dataset
List
Global Environment
Files Plots Packages Help Viewer
New Folder Upload Delete Rename
Home › full-stack-r-shiny-testing
Name Size

shinytest2– write test (step1)

Below is the test recorder screen where inputs and outputs are interactive for recording the user interface activities:



Select Dataset: cars

Variable: dist

Chart Type: Box-Plot

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10
10	18
10	26
10	34
11	17

```
# Update output value
app$set_window_size(width = 1065, height = 768)
# Update output value
app$set_inputs(`getdataset-dataset` = "cars")
# Update output value
app$set_inputs(`chart-var` = "dist")
# Update output value
app$set_inputs(`chart-chart_tpe` = "Box-Plot")
# Update output value
app$expect_values()
```

Test name: full-stack-r-shiny-testing

Please use a unique name. Known names:
- full-stack-r-shiny-testing

Random seed: (None)

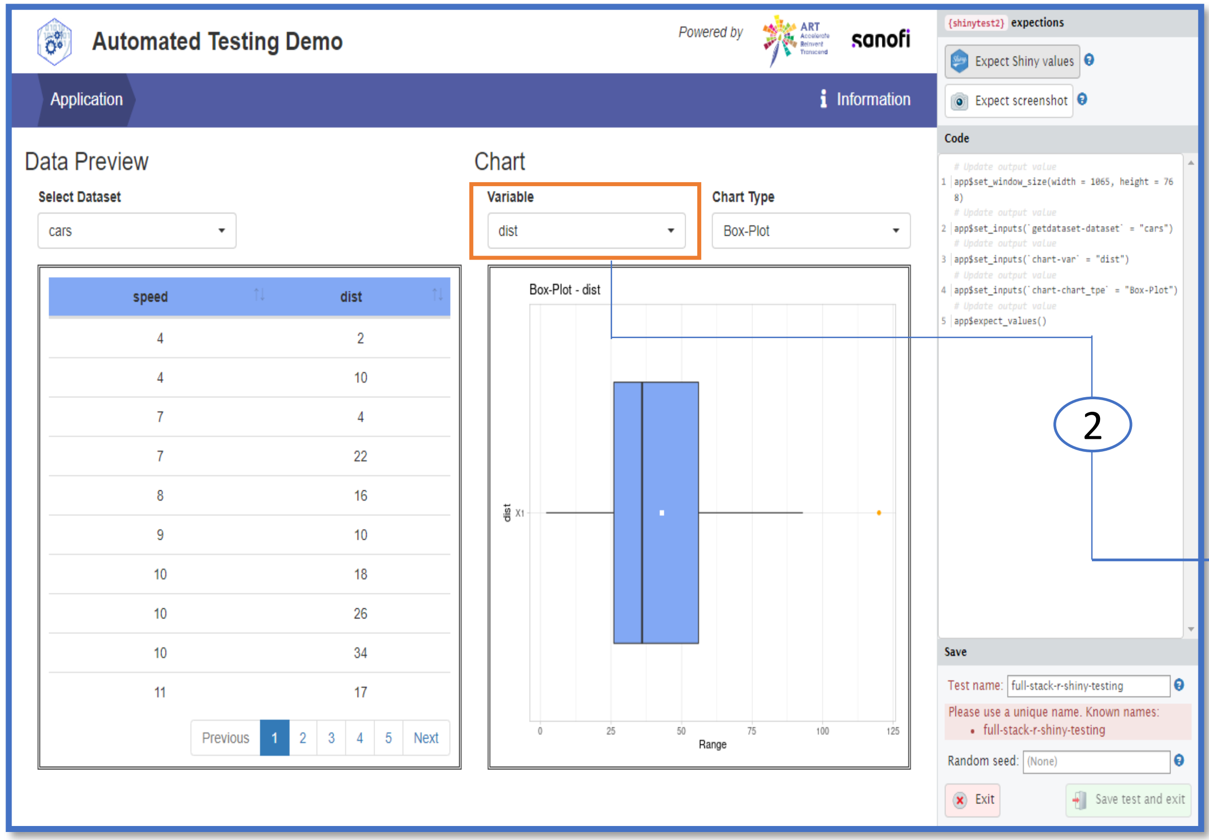
1

Test code(test-shinytest2.R)

```
#Below is the recorded shinytest2 code which sets
the input and captures the output values according
to the interactions with the app
library(shinytest2)

test_that("{shinytest2} recording: full-stack-r-
shiny-testing", {
#when app loads, it adjusts screen height and
Width automatically
app <- AppDriver$new(variant
= platform_variant(), name = "full-stack-r-shiny-
testing", height = 617, width = 1065)
#Sets dataset input id to "cars"
app$set_inputs(`getdataset-dataset` = "cars")

...
...
...
...

})
```

shinytest2– write test (step2)

Below is the test recorder screen where inputs and outputs are interactive for recording the user interface activities:



speed	dist
4	2
4	10
7	4
7	22
8	16
9	10
10	18
10	26
10	34
11	17

```
# Update output value
app$set_window_size(width = 1065, height = 76
8)
# Update output value
app$set_inputs(`getdataset-dataset` = "cars")
# Update output value
app$set_inputs(`chart-var` = "dist")
# Update output value
app$set_inputs(`chart-chart_tpe` = "Box-Plot")
# Update output value
app$expect_values()
```

Test name: full-stack-r-shiny-testing

Please use a unique name. Known names:
- full-stack-r-shiny-testing

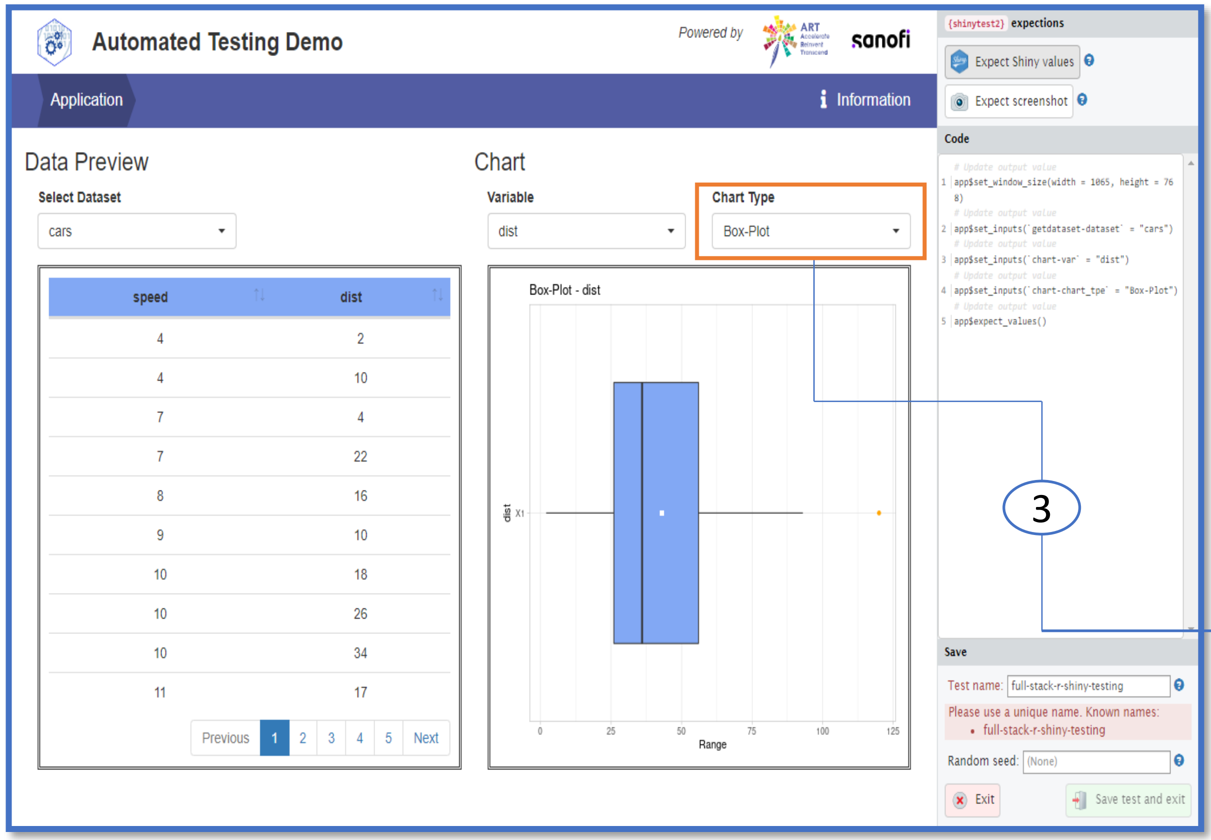
Random seed: (None)

Test code(test-shinytest2.R)

```
#Below is the recorded shinytest2 code which sets
the input and captures the output values according
to the interactions with the app
library(shinytest2)

test_that("{shinytest2} recording: full-stack-r-
shiny-testing", {
#when app loads, it adjusts screen height and
Width automatically
AppDriver$new(variant = platform_variant(), name =
"full-stack-r-     shiny-testing", height = 617,
width = 1065)
#Sets dataset input id to "cars"
app$set_inputs(`getdataset-dataset` = "cars")
#Sets input variable to "dist"
app$set_inputs(`chart-var` = "dist")

...
...
...

})
```

shinytest2– write test (step3)

Below is the test recorder screen where inputs and outputs are interactive for recording the user interface activities:



Select Dataset: cars

Variable: dist

Chart Type: Box-Plot

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10
10	18
10	26
10	34
11	17

Previous 1 2 3 4 5 Next

{shinytest2} expections

Expect Shiny values

Expect screenshot

Code

```
# Update output value
app$set_window_size(width = 1065, height = 76
8)
# Update output value
app$set_inputs(`getdataset-dataset` = "cars")
# Update output value
app$set_inputs(`chart-var` = "dist")
# Update output value
app$set_inputs(`chart-chart_tpe` = "Box-Plot")
# Update output value
app$expect_values()
```

Save

Test name: full-stack-r-shiny-testing

Please use a unique name. Known names:
- full-stack-r-shiny-testing

Random seed: (None)

Exit

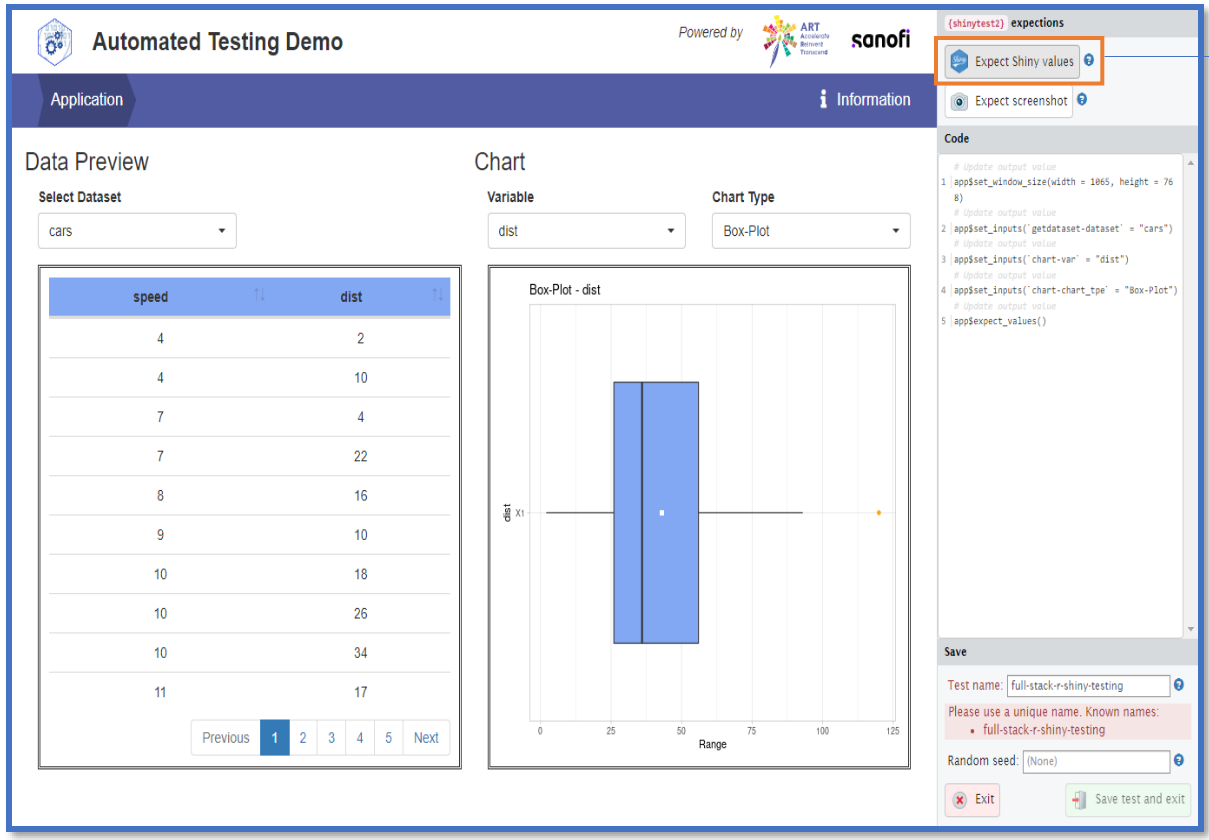
Save test and exit

Test code(test-shinytest2.R)

```
#Below is the recorded shinytest2 code which sets
the input and captures the output values according
to the interactions with the app
library(shinytest2)

test_that("{shinytest2} recording: full-stack-r-
shiny-testing", {
#when app loads, it adjusts screen height and
Width automatically
AppDriver$new(variant = platform_variant(), name =
"full-stack-r-     shiny-testing", height = 617,
width = 1065)
#Sets dataset input id to "cars"
app$set_inputs(`getdataset-dataset` = "cars")
#Sets input variable to "dist"
app$set_inputs(`chart-var` = "dist")
#Sets chart type to "Box-Plot"
app$set_inputs(`chart-chart_tpe` = "Box-Plot")

...
...

})
```

shinytest2– write test (step4)

Below is the test recorder screen where inputs and outputs are interactive for recording the user interface activities:



Select Dataset: cars

Variable: dist

Chart Type: Box-Plot

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10
10	18
10	26
10	34
11	17

{shinytest2} expections

Code

```
# Update output value
app$set_window_size(width = 1065, height = 76
8)
# Update output value
app$set_inputs(`getdataset-dataset` = "cars")
# Update output value
app$set_inputs(`chart-var` = "dist")
# Update output value
app$set_inputs(`chart-chart_tpe` = "Box-Plot")
# Update output value
app$expect_values()
```

Save

Test name: full-stack-r-shiny-testing

Please use a unique name. Known names:
- full-stack-r-shiny-testing

Random seed: (None)

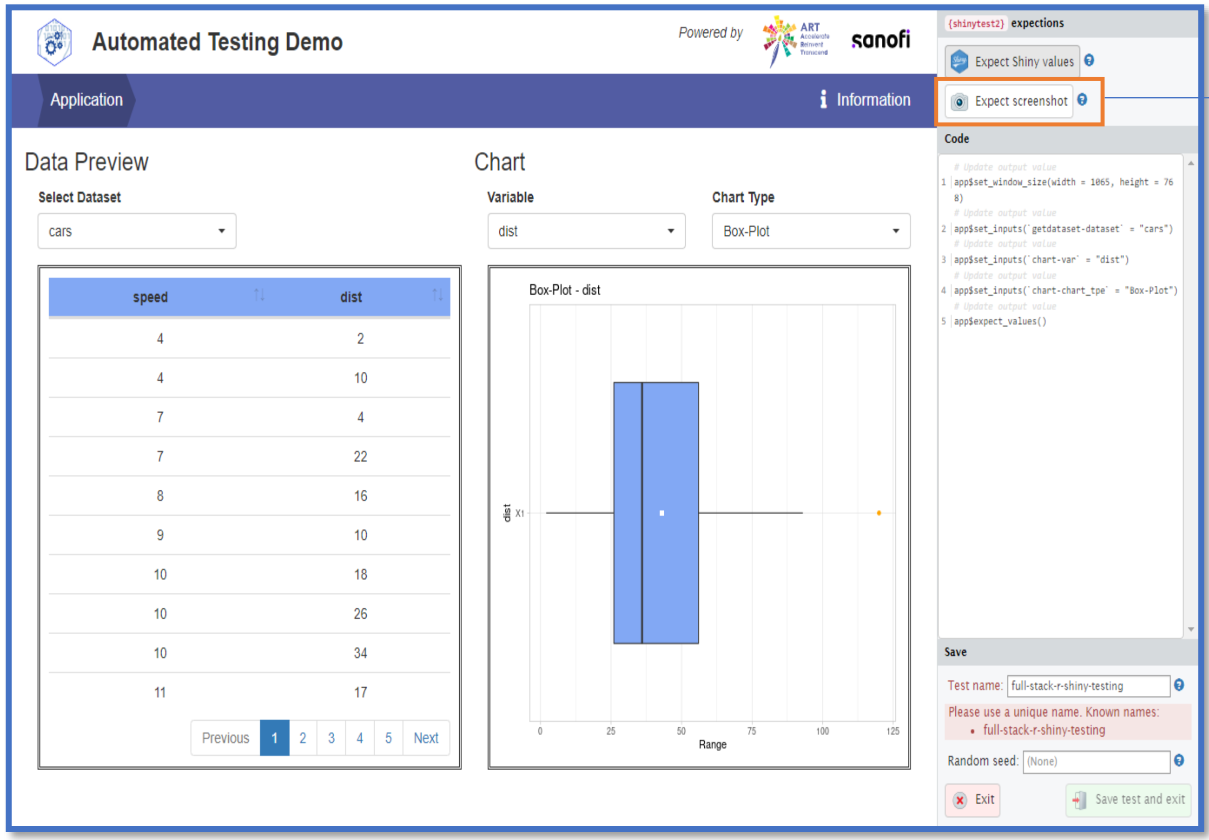
4

Test code(test-shinytest2.R)

```
#Below is the recorded shinytest2 code which sets
the input and captures the output values according
to the interactions with the app
library(shinytest2)

test_that("{shinytest2} recording: full-stack-r-
shiny-testing", {
#when app loads, it adjusts screen height and
Width automatically
AppDriver$new(variant = platform_variant(), name =
"full-stack-r-     shiny-testing", height = 617,
width = 1065)
#Sets dataset input id to "cars"
app$set_inputs(`getdataset-dataset` = "cars")
#Sets input variable to "dist"
app$set_inputs(`chart-var` = "dist")
#Sets chart type to "Box-Plot"
app$set_inputs(`chart-chart_tpe` = "Box-Plot")
#Captures shiny input and output values
app$expect_values()
...

})
```

shinytest2– write test (step5)

Below is the test recorder screen where inputs and outputs are interactive for recording the user interface activities:



Select Dataset: cars

Variable: dist

Chart Type: Box-Plot

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10
10	18
10	26
10	34
11	17

```
# Update output value
app$set_window_size(width = 1065, height = 768)
# Update output value
app$set_inputs(`getdataset-dataset` = "cars")
# Update output value
app$set_inputs(`chart-var` = "dist")
# Update output value
app$set_inputs(`chart-chart_tpe` = "Box-Plot")
# Update output value
app$expect_values()
```

Test name: full-stack-r-shiny-testing

Please use a unique name. Known names:
- full-stack-r-shiny-testing

Random seed: (None)

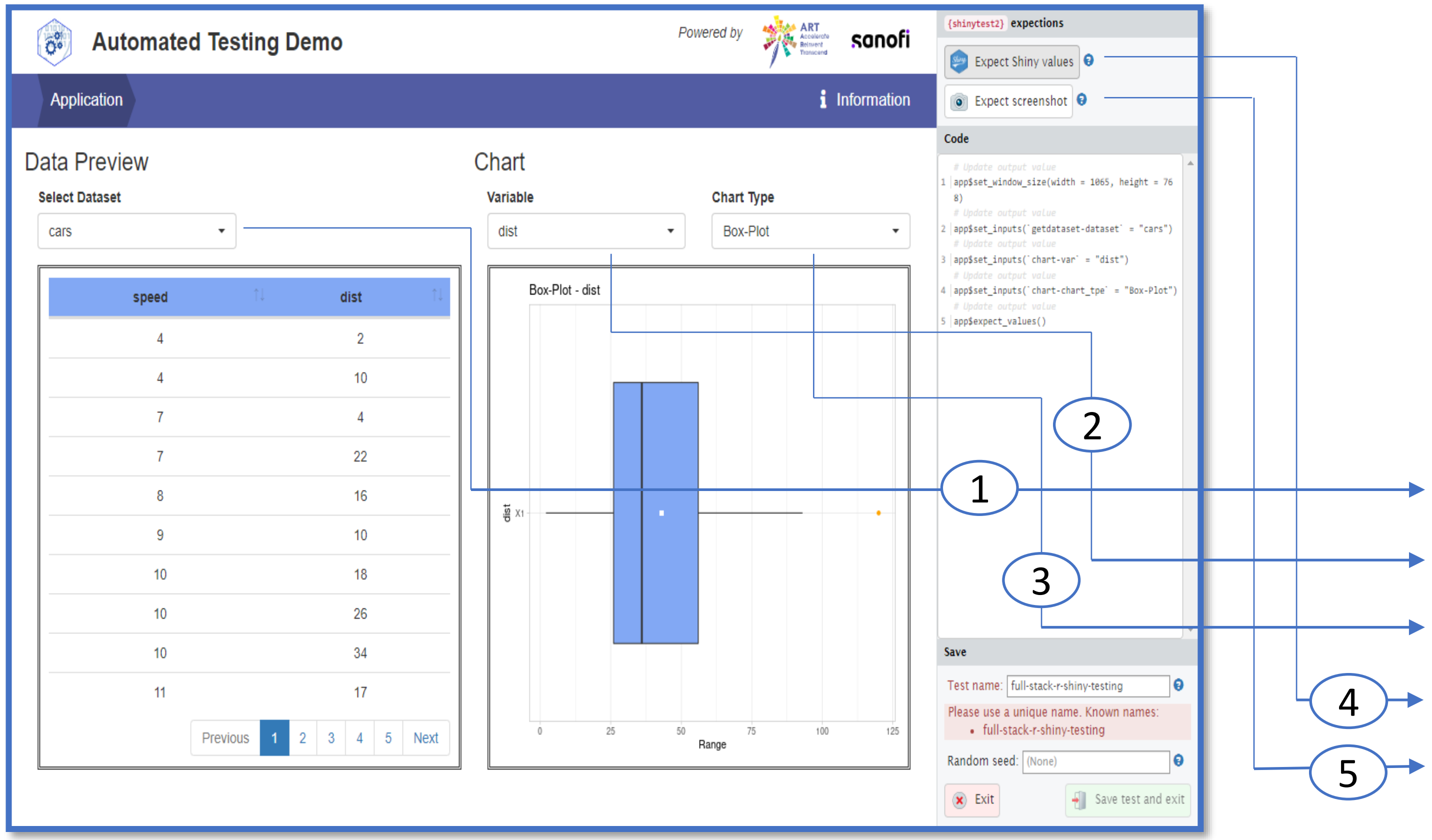
5

Test code(test-shinytest2.R)

```
#Below is the recorded shinytest2 code which sets
the input and captures the output values according
to the interactions with the app
library(shinytest2)

test_that("{shinytest2} recording: full-stack-r-
shiny-testing", {
#when app loads, it adjusts screen height and
Width automatically
AppDriver$new(variant = platform_variant(), name =
"full-stack-r-     shiny-testing", height = 617,
width = 1065)
#Sets dataset input id to "cars"
app$set_inputs(`getdataset-dataset` = "cars")
#Sets input variable to "dist"
app$set_inputs(`chart-var` = "dist")
#Sets chart type to "Box-Plot"
app$set_inputs(`chart-chart_tpe` = "Box-Plot")
#Captures shiny input and output values
app$expect_values()
#Captures screenshot with all the assigned inputs
app$expect_screenshot()
})
```

shinytest2– write test

Below is the test recorder screen where inputs and outputs are interactive for recording the user interface activities:



speed	dist
4	2
4	10
7	4
7	22
8	16
9	10
10	18
10	26
10	34
11	17

```
# Update output value
app$set_window_size(width = 1065, height = 768)
# Update output value
app$set_inputs(`getdataset-dataset` = "cars")
# Update output value
app$set_inputs(`chart-var` = "dist")
# Update output value
app$set_inputs(`chart-chart_tpe` = "Box-Plot")
# Update output value
app$expect_values()
```

Test name: full-stack-r-shiny-testing

Please use a unique name. Known names:
- full-stack-r-shiny-testing

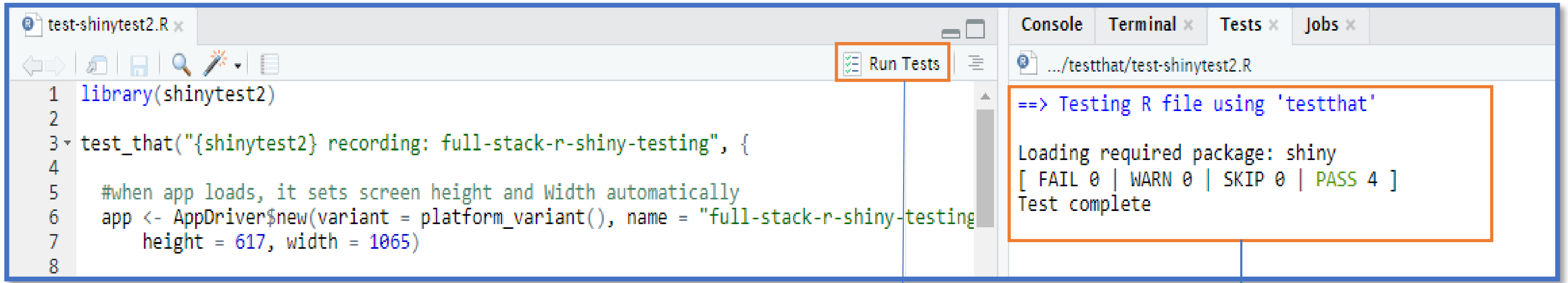
Random seed: (None)

Test code(test-shinytest2.R)

```
#Below is the recorded shinytest2 code which sets
the input and captures the output values according
to the interactions with the app
library(shinytest2)

test_that("{shinytest2} recording: full-stack-r-
shiny-testing", {
#when app loads, it adjusts screen height and
Width automatically
AppDriver$new(variant = platform_variant(), name =
"full-stack-r-     shiny-testing", height = 617,
width = 1065)
#Sets dataset input id to "cars"
app$set_inputs(`getdataset-dataset` = "cars")
#Sets input variable to "dist"
app$set_inputs(`chart-var` = "dist")
#Sets chart type to "Box-Plot"
app$set_inputs(`chart-chart_tpe` = "Box-Plot")
#Captures shiny input and output values
app$expect_values()
#Captures screenshot with all the assigned inputs
app$expect_screenshot()
})
```

shinytest2– run test (method 1)

“Run Tests” button will run the all the tests which are written in the shinytest2 test file (*test-shinytest2.R*). As soon as this button clicked, it runs the tests of current open test file and show the test result under the Tests tab.



```r
library(shinytest2)

test_that("{shinytest2} recording: full-stack-r-shiny-testing", {

  #when app loads, it sets screen height and Width automatically
  app <- AppDriver$new(variant = platform_variant(), name = "full-stack-r-shiny-testing
      height = 617, width = 1065)

```

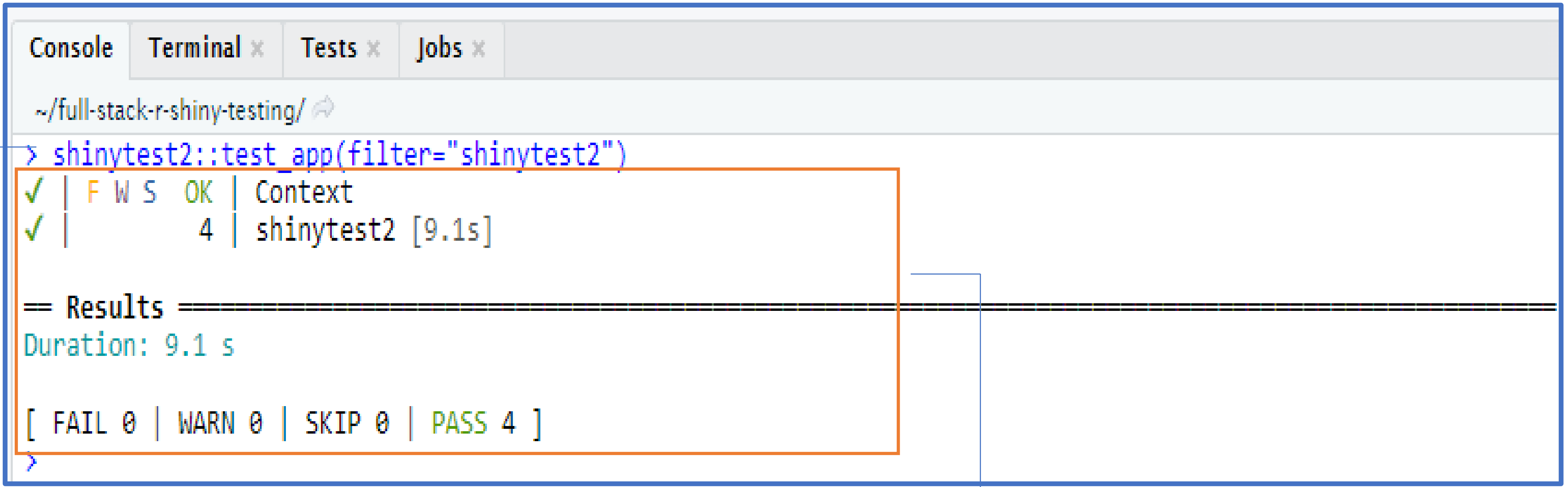
```
==> Testing R file using 'testthat'

Loading required package: shiny
[ FAIL 0 | WARN 0 | SKIP 0 | PASS 4 ]
Test complete
```

Step 1: Click on “**Run Tests**” button to run all the recorded shinytest2 tests.

Step 2: After clicking on “**Run Tests**” button, Test results will be displayed in “**Tests**” tab.

shinytest2– run test (method 2)

test_app() function used to run all the tests which are exist in the *tests/testthat* directory. All tests run one by one automatically and show the cumulative result in the console. It can also run the test of specific file by using **filter** parameter as shown in the below snap shot.

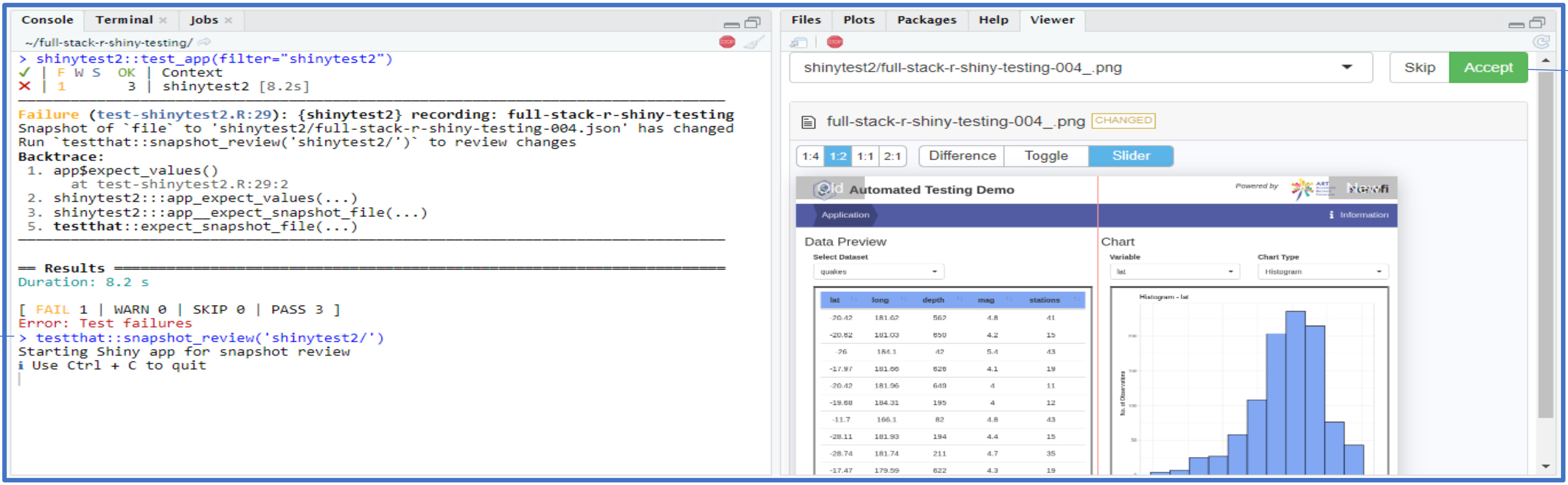


```
> shinytest2::test_app(filter="shinytest2")
✓ | F W S  OK | Context
✓ |         4 | shinytest2 [9.1s]

══ Results ═══════════════════════════════════════════════════════════
Duration: 9.1 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 4 ]
>
```

Step 1: Run **test_app(filter="shinytest2")** function in the console. It will only run shinytest2 file in the *testthat* folder.

Step 2: After test is successfully executed, results will be displayed with tests run duration.

shinytest2– Review failed Tests



```
> shinytest2::test_app(filter="shinytest2")
✓ | F W S  OK | Context
✗ | 1       3 | shinytest2 [8.2s]
──────────────────────────────────────────────────────────────────────────
Failure (test-shinytest2.R:29): {shinytest2} recording: full-stack-r-shiny-testing
Snapshot of `file` to 'shinytest2/full-stack-r-shiny-testing-004.json' has changed
Run `testthat::snapshot_review('shinytest2/')` to review changes
Backtrace:
 1. app$expect_values()
      at test-shinytest2.R:29:2
 2. shinytest2:::app_expect_values(...)
 3. shinytest2:::app__expect_snapshot_file(...)
 5. testthat::expect_snapshot_file(...)
──────────────────────────────────────────────────────────────────────────

══ Results ═══════════════════════════════════════════════════════════════
Duration: 8.2 s

[ FAIL 1 | WARN 0 | SKIP 0 | PASS 3 ]
Error: Test failures
> testthat::snapshot_review('shinytest2/')
Starting Shiny app for snapshot review
i Use Ctrl + C to quit
```

lat	long	depth	mag	stations
-20.42	181.62	562	4.8	41
-20.62	181.03	650	4.2	15
-26	184.1	42	5.4	43
-17.97	181.66	626	4.1	19
-20.42	181.96	649	4	11
-19.68	184.31	195	4	12
-11.7	166.1	82	4.8	43
-28.11	181.93	194	4.4	15
-28.74	181.74	211	4.7	35
-17.47	179.59	622	4.3	19

Run **testthat::snapshot_review("shinytest2/")** in the console to view the difference in the snapshots.

In the **viewer** tab, compare old and new changes in the screenshot/json file and update the results by clicking on **Accept** button else skip it.

Reference links

testhat

- ✓ testthat: **Overview** and **installation** of testthat package.
- ✓ Git repository: **Git repository** for testthat package.
- ✓ Function reference: List of **expect** and **run** test functions with description and example.
- ✓ Mastering Shiny book - testing: **Step by step** explanation of automated testing using testthat package.

shinytest2

- ✓ shinytest2: **Overview** and **installation** of Shinytest2 package.
- ✓ Get started: **Step by step** explanation of automated testing using shinytest2 package.
- ✓ Git repository: **Git repository** for shinytest2 package.
- ✓ Function reference: List of **making**, **running**, **setting up** and **migrating** test functions with description and example.

full-stack-r-shiny-testing

- ✓ Git repository: **Git repository** for **full-stack-r-shiny-testing** demo application. This application will help to build the end to end automated testing in shiny R.

Acknowledgement

Project Lead

Mukul Mittal

R Shiny Technical Team Lead

Mayank Agarwal

R Shiny Developer

Babit Ranjan Pradhan

R Shiny Developer

Pricy Jain

R Shiny Developer

Ramya Sri Pichikala

R Shiny Developer

Sanjeev Kumar Rathore

R Shiny Developer

Sujeet Kumar Das

Questions
&
Answers

Thank you