1. Master reactivity

You've for sure heard of Shiny being lazy. Use a reactlog to visualize the dependency structure, which can quickly get complicated if you use many reactive expressions.

2. Structure your app

Create your app within an R package. As complexity increases, extend the structure: split the logic & big calculations in reusable modules and functions.

3. Test

Automate unit testing to ensure that single functions keep working as expected. It's a time investment up front, but it will save you hours of debugging and struggling. Add integration tests to ensure the interactions between reactives remain smooth. Test performance too.

4. Manage the dependencies

Define the dependencies within the package. Apply renv to ensure using the packages with specific versions.

5. Use modules

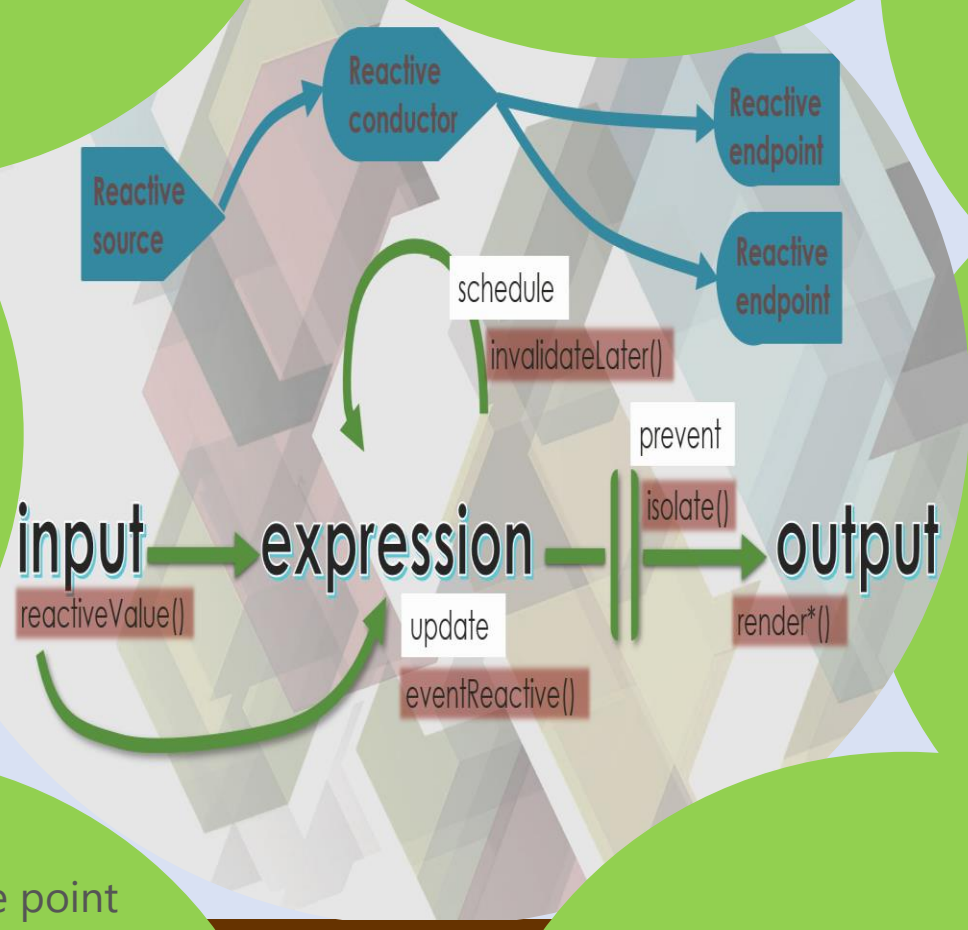
If you are going to copy-paste a functionality, create a module instead. Shiny modules are isolated and re-usable pieces of code, acting as a coordinator between the UI and the back-end.

6. Debug

Unfortunately at some point something will go wrong: unexpected errors, weird results or values not getting re-evaluated. Read the tracebacks, use the interactive debugger, figure out what is causing the issue with the help of `browser()`. If a reactive does not what it is supposed to, then use the basic technique of message() to print the calculated value.

7. Ease the maintenance

Document the code, make sure to explain complex chunks of code. Select meaningful variables, functions and files names, adopt a naming convention. Get inspired by R best practice syntax.



Visit our workshop
'Advanced shiny app'

Register Now!

- Package your Shiny project with 'golem'
- Structure your app in modules
- Explore dynamic UI
- Get an overview of different testing approaches for a shiny app
- Learn about best practices like dependencies control and automation

24.03.2021
14:00-17:00

CHF 149

Online

MiraiSolutions

www.mirai-solutions.com | training@mirai-solutions.com | Check our workshop offer