



OS05-Mastering Production-Grade Shiny Apps with Rhino *Lessons and Insights*



12th November 2024

André Veríssimo
Vedha Viyash

A decorative pattern of hexagons in various shades of blue and teal, some with white outlines, located in the top right corner of the slide.

Avoiding Technical Pitfalls

Shiny Toolkit with Rhino

Case Studies

Key Takeaways



Avoiding Technical Pitfalls

Shiny Toolkit with Rhino

Case Studies

Key Takeaways





CHANGE? FORGET IT



CHANGE IS HARD



CHANGE IS EASY



start

time

end of project

technical
debt

CHANGE IS EASY



CHANGE IS HARD



CHANGE? FORGET IT



start

time

end of project

technical
debt

CHANGE IS EASY



CHANGE IS HARD



CHANGE? FORGET IT



Un-managed code

Best practices

start

time

end of project



technical
debt

I OWE YOU
multiple deployment
configs

I OWE YOU
difficult for devs to
debug

I OWE YOU
scattered code
R / JS / CSS

I OWE YOU
bugs that reduce
user confidence

I OWE YOU
unformatted code
hard to read

I OWE YOU
hardcoded to
environment

I OWE YOU
increasing
complexity

Good practices in R/Shiny development

11 years of projects

A decorative pattern of hexagons in various shades of blue and teal, some with white outlines, located in the top right corner of the slide.

Avoiding Technical Pitfalls

Shiny Toolkit with Rhino

Case Studies

Key Takeaways

Tools for Robust Shiny Apps



Code quality {lintr}, {styler}, {sass}

Environment & reproducibility {renv}

Testing {testthat}, {shinytest2}, cypress

Logging {logger}

Modularity {box}, Shiny modules

Configurable {config}



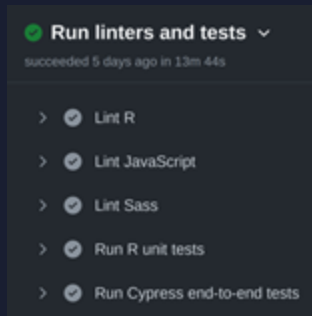
Rhino was built from our experience on best practices

“A framework for enterprise Shiny apps”

1. Project structure
2. Development toolbox
3. Good practices



Rhino gives all of this for free



No setup plus Continuous Integration workflows to automate tasks



File structure built to host Shiny Apps

- Separation between App: Logic / View / CSS / JS
- Tests



File structure built to host Shiny Apps

- rhino.yml
- config.yml
- .Rprofile

Config

- app.R
- app
 - main.R
 - logic
 - view
 - static
 - js
 - styles

Sources

- renv
- renv.lock
- dependencies.R

Deps

- tests
 - cypress
 - cypress.json
 - testthat

Tests

- .github
- .lintr
- example.Rproj

Tools



Modular approach

- Use and re-use building blocks
- Explicit about the dependencies between your files and packages



Code quality (R, CSS & JS)

- Keep good code readability
- Common conventions (*spacing, variable naming, ...*)
- Detect known pitfalls



Unit and end-to-end testing

- Early detection of features breaking
- Increase user confidence
- Profiling code across commits via Rhinoverse



Reproducibility

- Manage package and their exact versions across team / deployments
- ~~“But it works on my laptop”~~



Logging & Flexible configuration

- Easy setup of multiple deployments
- Help avoid security risks
- Lightweight logging

Setup



Setup Project Files

renv::init()

lintr::use_lintr()

install Node.js

Setup CSS infrastructure

Setup JS infrastructure

Setup testing infrastructure

VS.



install Node.js

rhino::init()

Usage



```
# Build SCSS to CSS
```

```
# Bundle JS code
```

```
lintr::lint_dir()
```

```
# lint CSS / SASS code via node.js
```

```
styler::style_dir()
```

```
testthat::test_dir()
```

```
# Run cypress tests
```

```
logger::log_*
```

VS.



```
rhino::build_sass()
```

```
rhino::build_js()
```

```
rhino::format_r()
```

```
rhino::lint_r()
```

```
rhino::lint_js()
```

```
rhino::lint_sass()
```

```
rhino::test_r()
```

```
rhino::test_e2e()
```

```
rhino::log$*
```

GitHub workflows that check linting and tests

A decorative pattern of hexagons in various shades of blue and teal, some with white outlines, located in the top right corner of the slide.

Avoiding Technical Pitfalls

Shiny Toolkit with Rhino

Case Studies

Key Takeaways

Single file app that grew too big

Case study

Single file app

Case study 1

Single app.R file with 4.5k lines

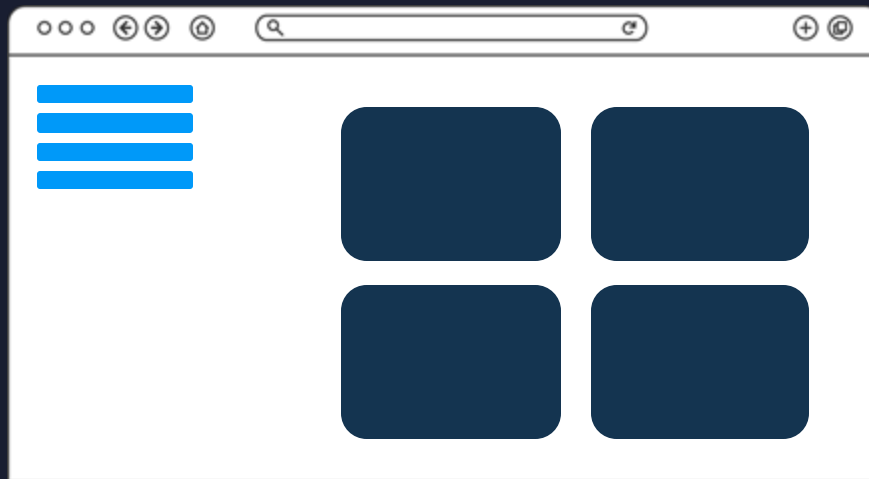
Serving hundreds of users

16 different plots with data transformations

Performed data retrieval (AWS) and processing inside Shiny App body

Organized by comments chunks

Very slow and reactive



Single file app

Case study 1



Separate views from data retrieval and processing logic



Break Shiny outputs into modules and views



Run code analysis and cleanup



Optimize and compare benchmark tests



Faster deployments,
environment setup
and remove security vulnerabilities

Complex & inefficient codebase

Case study

Complex & inefficient codebase

Case study 2

From

Shiny app with 13 plots and 5 tables

Complex file dependency

Repetitive code

Goal

Interactive plots

Create new apps with similar visualizations using different data sources

Periodic data update

Complex & inefficient codebase

Case study 2

Use of Shiny Modules
&
Managing R files using {modules}

Use of SASS to manage big CSS codebase

renv
&
1 big SQLite file as a data source

```
> .Rproj.user
  constants
  > .ipynb_checkpoints
    constants.R
  data
  db
  db_methods.R
  modules
  biomarker_consensus_boxplot_view.R
  biomarker_consensus_waterfall_view.R
  biomarker_multivariate_summary_view.R
  biomarker_multivariate_volcano_view.R
  biomarker_univariate_summary_view.R
  metrics_view.R
  target_consensus_waterfall_view.R
  target_dependency_summary_view.R
  target_depmap_crispr_distribution_view.R
  target_depmap_crispr_piechart_view.R
  target_depmap_crispr_waterfall_view.R
  target_depmap_distribution_view.R
  target_depmap_sirna_distribution_view.R
  target_depmap_sirna_piechart_view.R
  target_depmap_sirna_waterfall_view.R
  target_enrichment_summary_view.R
  target_k1_distribution_view.R
  target_k1_piechart_view.R
  target_k1_waterfall_view.R
  target_primary_disease_barplot_view.R
  > renv
  rsconnect
  styles
  > modules
  > partials
  main.scss
```



Honestly, better than most
projects I've seen.

Complex & inefficient codebase

Case study 2

Old syntax 📌 Use of Shiny Modules
&
Less popular 📌 Managing R files using {modules}

Unwanted styles 📌 Use of SASS to manage big CSS codebase

Outdated deps 📌 renv
&
Not robust 📌 1 big SQLite file as a data source

```
> .Rproj.user
v constants
> .ipynb_checkpoints
  constants.R
> data
v db
  db_methods.R
v modules
  biomarker_consensus_boxplot_view.R
  biomarker_consensus_waterfall_view.R
  biomarker_multivariate_summary_view.R
  biomarker_multivariate_volcano_view.R
  biomarker_univariate_summary_view.R
  metrics_view.R
  target_consensus_waterfall_view.R
  target_dependency_summary_view.R
  target_depmap_crispr_distribution_view.R
  target_depmap_crispr_piechart_view.R
  target_depmap_crispr_waterfall_view.R
  target_depmap_distribution_view.R
  target_depmap_sirna_distribution_view.R
  target_depmap_sirna_piechart_view.R
  target_depmap_sirna_waterfall_view.R
  target_enrichment_summary_view.R
  target_k1_distribution_view.R
  target_k1_piechart_view.R
  target_k1_waterfall_view.R
  target_primary_disease_barplot_view.R
> renv
> rsconnect
v styles
  modules
  partials
  main.scss
```

📌 Honestly, better than most projects I've seen.

Complex & inefficient codebase

Case study 2



Use of Shiny Modules
&
Managing R files using {modules}



Use of SASS to manage big CSS codebase



renv
&
1 big SQLite file as a data source

```
> .Rproj.user
v constants
  > .ipynb_checkpoints
    constants.R
  > data
  v db
    db_methods.R
  v modules
    biomarker_consensus_boxplot_view.R
    biomarker_consensus_waterfall_view.R
    biomarker_multivariate_summary_view.R
    biomarker_multivariate_volcano_view.R
    biomarker_univariate_summary_view.R
    metrics_view.R
    target_consensus_waterfall_view.R
    target_dependency_summary_view.R
    target_depmap_crispr_distribution_view.R
    target_depmap_crispr_piechart_view.R
    target_depmap_crispr_waterfall_view.R
    target_depmap_distribution_view.R
    target_depmap_sirna_distribution_view.R
    target_depmap_sirna_piechart_view.R
    target_depmap_sirna_waterfall_view.R
    target_enrichment_summary_view.R
    target_k1_distribution_view.R
    target_k1_piechart_view.R
    target_k1_waterfall_view.R
    target_primary_disease_barplot_view.R
  > renv
  > rsconnect
  v styles
    > modules
    > partials
    main.scss
```



Honestly, better than most
projects I've seen.

Complex & inefficient codebase

Case study 2



Use of Shiny Modules
&
Managing R files using {modules}



Use of SASS to manage big CSS codebase



renv
&
1 big SQLite file as a data source

```
> .Rproj.user
v constants
  .jupyter_checkpoints
  constants.R
> data
v db
  db_methods.R
v modules
  biomarker_consensus_boxplot_view.R
  biomarker_consensus_waterfall_view.R
  biomarker_multivariate_summary_view.R
  biomarker_multivariate_volcano_view.R
  biomarker_univariate_summary_view.R
  metrics_view.R
  target_consensus_waterfall_view.R
  target_dependence_summary_view.R
  target_depmap_crispr_distribution_view.R
  target_depmap_crispr_piechart_view.R
  target_depmap_crispr_waterfall_view.R
  target_depmap_distribution_view.R
  target_depmap_sirna_distribution_view.R
  target_depmap_sirna_piechart_view.R
  target_depmap_sirna_waterfall_view.R
  target_enrichment_summary_view.R
  target_k1_distribution_view.R
  target_k1_piechart_view.R
  target_k1_waterfall_view.R
  target_primary_disease_barplot_view.R
> renv
> rsconnect
v styles
  modules
  partials
  main.scss
```

waterfall

x 5

distribution

x 2

volcano

x 1

summary

x 4

boxplot

x 1

piechart

x 3

barplot

x 1

Complex & inefficient codebase

Case study 2

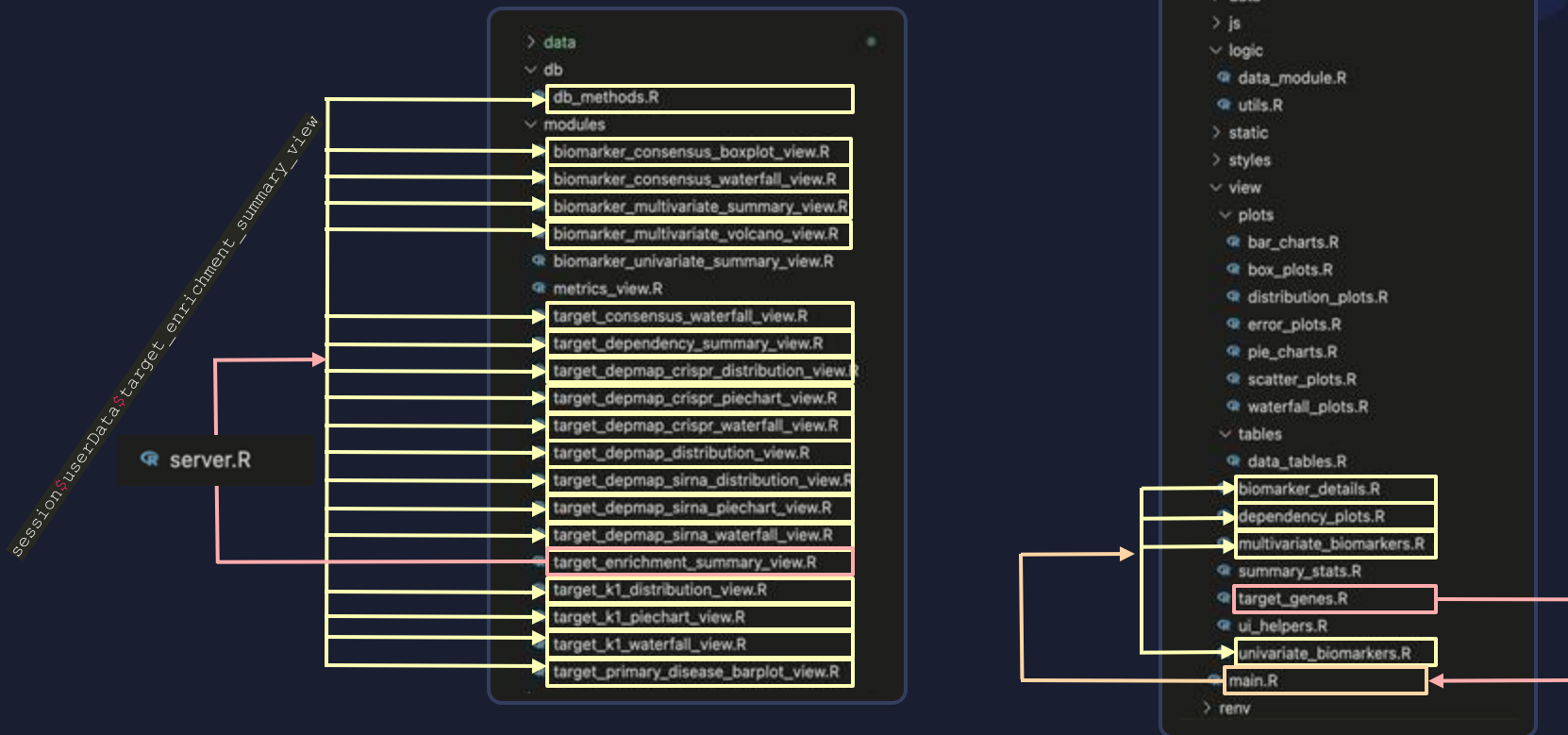
```
> data
  > db
    @ db_methods.R
  > modules
    @ biomarker_consensus_boxplot_view.R
    @ biomarker_consensus_waterfall_view.R
    @ biomarker_multivariate_summary_view.R
    @ biomarker_multivariate_volcano_view.R
    @ biomarker_univariate_summary_view.R
    @ metrics_view.R
    @ target_consensus_waterfall_view.R
    @ target_dependency_summary_view.R
    @ target_depmap_crispr_distribution_view.R
    @ target_depmap_crispr_piechart_view.R
    @ target_depmap_crispr_waterfall_view.R
    @ target_depmap_distribution_view.R
    @ target_depmap_sirna_distribution_view.R
    @ target_depmap_sirna_piechart_view.R
    @ target_depmap_sirna_waterfall_view.R
    @ target_enrichment_summary_view.R
    @ target_k1_distribution_view.R
    @ target_k1_piechart_view.R
    @ target_k1_waterfall_view.R
    @ target_primary_disease_barplot_view.R
```



```
  > app
    > data
    > js
  > logic
    @ data_module.R
    @ utils.R
  > static
  > styles
  > view
    > plots
      @ bar_charts.R
      @ box_plots.R
      @ distribution_plots.R
      @ error_plots.R
      @ pie_charts.R
      @ scatter_plots.R
      @ waterfall_plots.R
    > tables
      @ data_tables.R
      @ biomarker_details.R
      @ dependency_plots.R
      @ multivariate_biomarkers.R
      @ summary_stats.R
      @ target_genes.R
      @ ui_helpers.R
      @ univariate_biomarkers.R
    @ main.R
  > renv
```

Complex & inefficient codebase

Case study 2



A decorative pattern of hexagons in various shades of blue and teal, some with white outlines, located in the top right corner of the slide.

Avoiding Technical Pitfalls

Shiny Toolkit with Rhino

Case Studies

Key Takeaways

Be curious and look for new _____

Build upon experience



It's not the wand

It's the wizard





Getting Started with {Rhino} to Build Enterprise-Grade Shiny Apps



<https://appsilon.github.io/rhino>



[rhino](#) 1.10 • [Tutorials](#) • [Explanation](#) • [How-to Guides](#) • [Reference](#) • [Changelog](#) • [FAQ](#)

Rhino



Build high quality, enterprise-grade [Shiny](#) apps at speed.

Events

Upcoming

Would you like to learn about Rhino hands-on? Join our events! Upcoming events will be added to this section - stay tuned.

► **Past** (click to expand)

Why Rhino?

Rhino allows you to create Shiny apps **The Appsilon Way** - like a fullstack software engineer. Apply best software engineering practices, modularize your code, test it well, make UI beautiful, and think about user adoption from the very beginning. Rhino is an opinionated framework with a focus on software engineering practices and development tools.

Rhino supports your work in 3 main areas:

1. **Clear code:** scalable app architecture, modularization based on Box and Shiny modules.
2. **Quality:** unit tests, E2E tests with Cypress, logging and monitoring, linting.
3. **Automation:** project startup, CI with GitHub Actions, dependency management with renv, configuration management with config, Sass and JavaScript bundling with ES6 support via Node.js.

Links

- [View on CRAN](#)
- [Browse source code](#)
- [Report a bug](#)

License

- [LGPL-3](#)

Community

- [Contributing guide](#)

Citation

- [Citing rhino](#)

Developers

- Kamil Zyla
Author, maintainer
- Jakub Nowicki
Author
- Leszek Siemiński
Author
- Marek Rogala
Author
- Recle Vibal
Author
- Tymoteusz Makowski



André Veríssimo
andre.verissimo@appsilon.com



Vedha Viyash
vedha@appsilon.com

Thank you



Appsilon

