

Appsilon

Estructura de un proyecto Shiny



Agustin Perez Santangelo
agustin@appsilon.com

Hoja de Ruta

- Contexto/motivación
- Estructura mínima
- Herramientas
 - **renv** para controlar dependencias (paquetes)
 - init, restore, snapshot, status
 - implicit/explicit
 - dependencies.R (facilitar deployment con rsconnect)
 - **testthat** para unit-testing
 - arquitectura
 - filosofía (e.g. TDD)
 - **covr** para trackear testing
 - **config** para manejar perfiles de constantes
 - test/producción
 - **lintr** para monitorear código
 - **styler** para aplicar estilo consistente (usar con cuidado)

Contexto

Nuestra **primera** Shiny app



```
library(shiny)

ui <- fluidPage(
  title = "Super App",
  "¡Hola, mundo! :)"
)

server <- function(input, output, session) {
}

shinyApp(ui, server)
```

```
primera-app/
├─ app.R
```

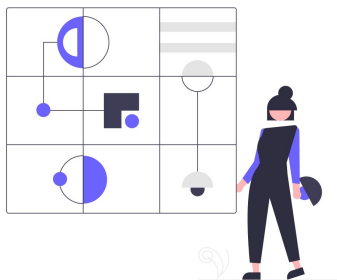
```
primera-app/
├─ server.R
├─ ui.R
```



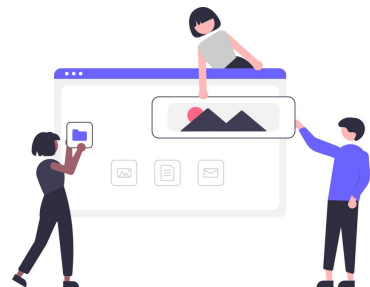
0_primera-app

Contexto

Apps más complejas



Desarrollo en equipo



Comerciales/producción



Contexto

Proyecto

- Requerimientos
- Herramientas

Estructura

Mejor
organización



Desarrollo
más rápido

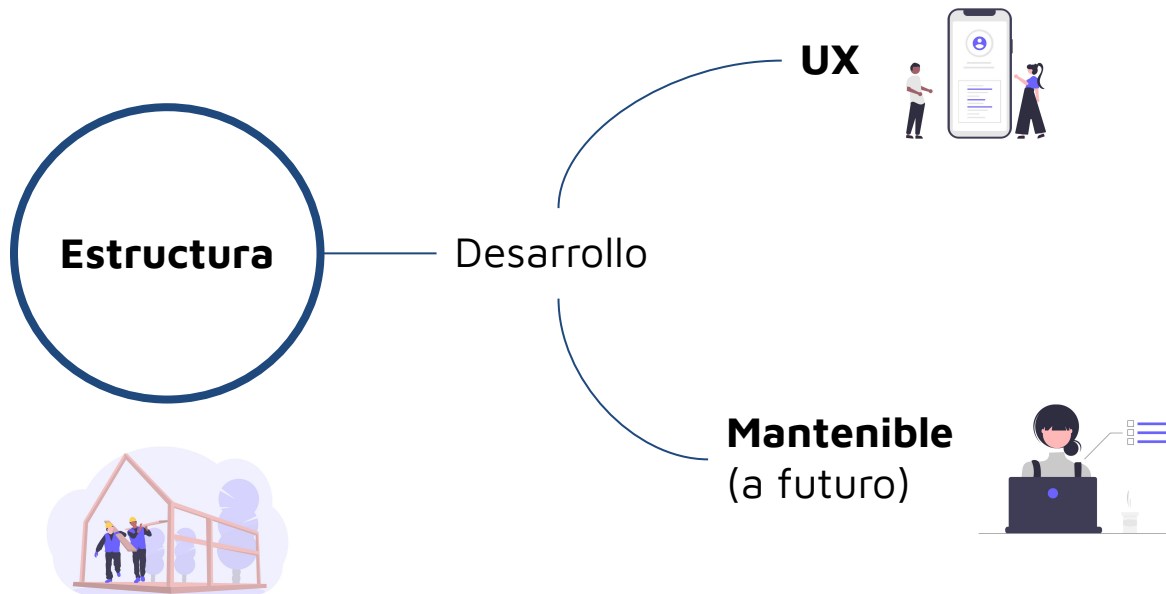


Cooperación
más sencilla



Estructura básica

Premisas



- Escalable
- Eficiente
- Sin bugs
- Extensible
- Disfrutable

Estructura básica

- Cada proyecto tiene su idiosincrasia
 - Punto de partida
- Un **proyecto** “típico” que incluya
- Datos
 - Scripts de carga y procesamiento de datos
 - App
 - UI + Server
 - www (aka assets)
 - Imágenes
 - Audio
 - [CSS]
 - [JavaScript]
 - [Tests]

```
mi-proyecto-tipico/  
├── app/  
│   ├── data/  
│   │   └── data.csv  
│   ├── app.R  
│   ├── helpers.R  
│   └── www/  
│       ├── mi_css.css  
│       ├── mi_js.js  
│       ├── imagen.png  
│       └── sonido.wav  
├── scripts/  
│   ├── raw_data/  
│   │   └── raw.csv  
│   └── preprocess.R  
└── tests
```


Propuesta

Estructura básica

- **0. Proyecto como .Rproj**
 - Ideal para compartimentalizar trabajo
 - Inicia nueva sesión
 - Lee .Renviron (env variables)
 - `source(".Rprofile")`
 - Setea directorio



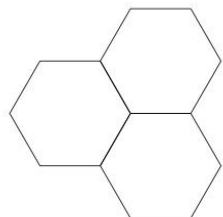
Estructura básica

- 1. Controlar entorno de desarrollo con **renv**

- Versión de R y dependencias (paquetes)
- Reproducibilidad
 - mismo código, mismos resultados
 - colaboración más sencilla
- Facilita deployment
- Cache global de paquetes
 - Carga más rápida
 - Ahorra espacio en disco



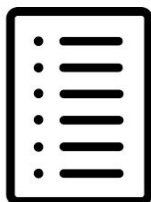
Project library



snapshot(*)



renv.lock



restore()



```
# Inicializar renv en el proyecto
renv::init()

# Salvar library a `renv.lock`
renv::snapshot()

# Cargar library desde `renv.lock`
renv::restore()
```

.Rprofile
 renv.lock
 renv/activate.R
 renv/library



1_renv_init



Estructura básica

- 1. Controlar entorno de desarrollo con **renv**
 - Tipos de snapshot (registro en lock file)
 - *explicit*: dependencias declaradas en DESCRIPTION
 - *implicit*: la intersección entre library y código (default con init())
 - *all*: toda la library



Explicit es ideal para tener mayor control, pero trae problemas para deploy con rsconnect (shinyapps.io, RStudio Connect).

Solución:

- Implicit snapshots
- **.renvignore** (sigue misma lógica que .gitignore)
- dependencies.R
 - listamos pkgs llamando a library()



```
# Solo usar `dependencies.R` para inferir dependencias.  
*  
!dependencies.R
```



1_renv_init2



Estructura básica

- **2. Extraer constantes con `config`**
 - Permite crear perfiles de constantes
 - test
 - dev
 - production
 - Ahorra tiempo al momento de cambiar valores de constantes.
 - Menos números mágicos y hard-coding en el código de nuestra app.





2_config

Estructura básica

- 3. Unit testing con **testthat** + tracking con **covr**
 - Chequear funcionalidad
 - Ciclo virtuoso con implementación
 - ~Test-Driven Development
 - **covr**
 - trackear testing
 - generar reportes

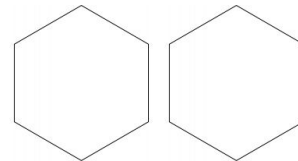




3_testthat-covr

Estructura básica

- 4. Código limpio y consistente con **lintr** y **styler**
 - Seguir [guía de estilo tidyverse](#) (convención adoptada en la comunidad)
 - Minimizar chances de bugs/errores
 - Código más legible y entendible para un humano





4_lintr-styler

Uniendo todo

```

mi-proyecto-tipico/
├── app/
│   ├── data/
│   │   └── data.csv
│   ├── app.R
│   ├── helpers.R
│   └── www/
│       ├── mi.css.css
│       ├── mi.js.js
│       ├── imagen.png
│       └── sonido.wav
├── scripts/
│   ├── raw data/
│   │   └── raw.csv
│   └── preprocess.R
└── tests
  
```

```

mi-proyecto/
├── app/
│   ├── constants/
│   │   └── config.yaml
│   ├── data/
│   │   └── data.fst
│   ├── js/
│   │   └── index.js
│   └── r/
│       ├── main.R
│       ├── module_1.R
│       ├── module_2.R
│       └── utils.R
├── styles/
│   └── main.scss
├── www/
│   ├── css/
│   │   └── app.min.css
│   ├── js/
│   │   └── app.min.js
│   ├── favicon.ico
│   ├── image.png
│   └── sound.wav
└── ...
  
```

Parte 3

Parte 2

```

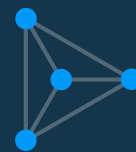
...
├── renv/
│   ├── library/
│   ├── staging/
│   ├── activate.R
│   └── settings.dcf
├── scripts/
│   ├── raw data/
│   │   └── raw.csv
│   └── preprocess.R
├── tests/
│   └── testthat/
│       ├── test-main.R
│       ├── test-module_1.R
│       ├── test-module_2.R
│       └── test-utils.R
├── tools/
│   ├── coverage
│   ├── unit-test
│   └── style
│       └── lint-r
├── .lintr.yaml
├── .renvignore
├── .Rprofile
├── app.R
├── dependencies.R
├── README.md
├── renv.lock
└── src.Rproj
  
```

Todo esto...y más!



- [Documentacion](#)
- 🎓 [Tutorial](#)
- 📄 [Workshop](#)

¿Preguntas?



Appsilon

¡Gracias!

Referencias

- [.Rproj](#) projects
- [renv](#)
- [config](#)
- [testthat](#)
- [covr](#)
- [lintr](#)
- [styler](#)
- [rhino](#)