Shiny en producción: Monitor Educativo

Oscar Montañés y Natalia da Silva

Descripción General

Presentaremos desafíos y posibles soluciones en la puesta en producción de una aplicación basada en shiny que presenta diferentes niveles de complejidad.
La application fue desarrollada como monitor de uso de la plataforma educativa CREA del Plan Ceibal en Uruguay con datos de 2018 a 2021 en base a un prototipo con datos de 2015 y 2017.

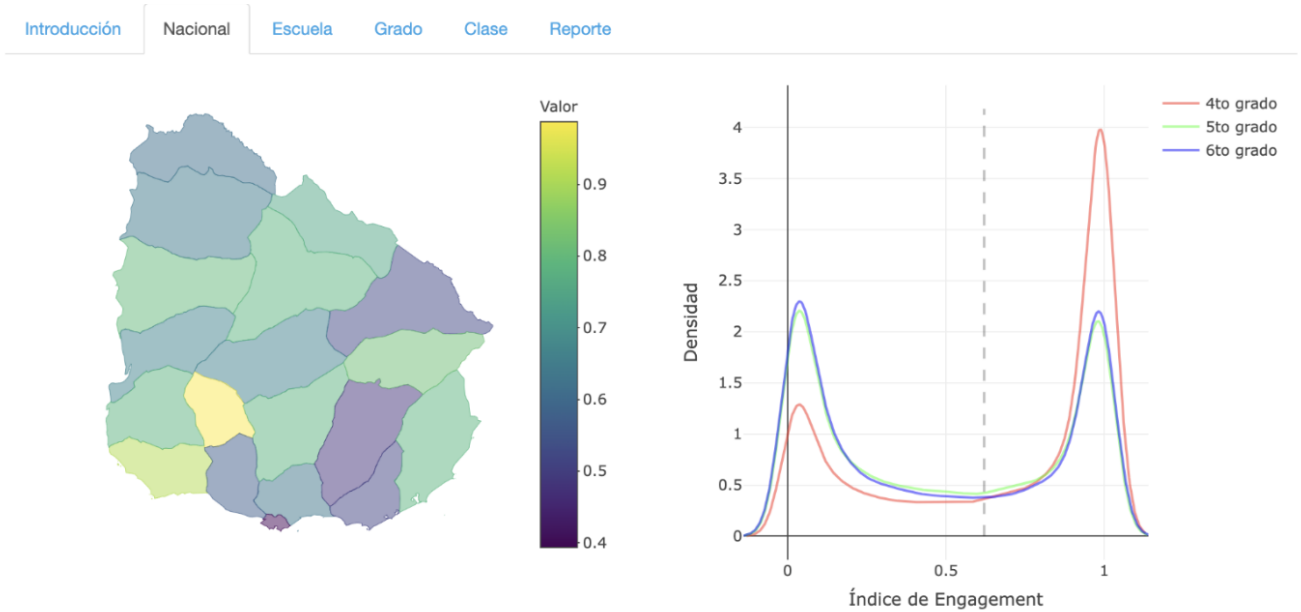
Fondo Sectorial de Educación

Desafíos

- Migrar una app shiny clásica
- Gran volumen de datos
- Cálculo complejos para un motor de base de datos
- Distintos tipo de visualizaciones
- Debe de soportar gran buena cantidad de usuarios
- Soportar cambiar el cambios a futuro de manera sencilla

Prototipo con datos 2017

http://164.73.240.157:3838/App-Ceibal/



Mapa de Uruguay para la Mediana de la distribución del índice de engagement en el año 2017

Distribución del índice de engagement en el año 2017 de 4to, 5to y 6to año de primaria

Los Datos

- Por alumno:
  - 1 fila por día
  - Aprox. 200 días
  - Aprox. 350 mil alumnos
  - 4 años, 2018 a 2021
  - ~20MM filas
- Muchas columnas numéricas con varias agregaciones (avg, count, log, etc)

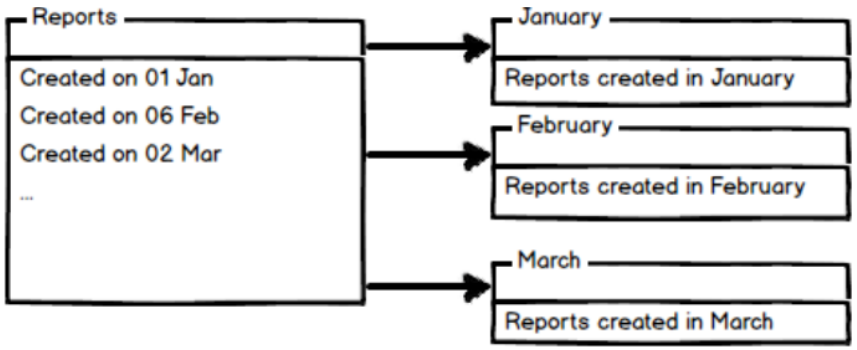
Los cálculos no se pueden hacer en caliente!

Estrategia a nivel de datos

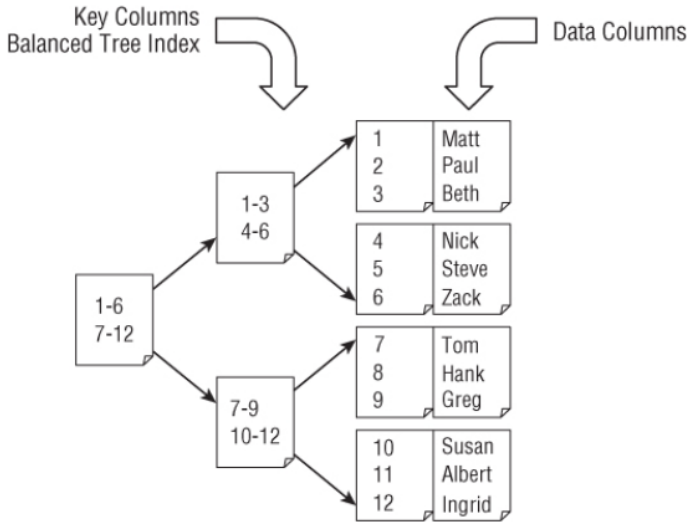
- **Particionar tablas por mes**
  Partir una tabla de 20MM de filas, a tablas por mes, esto facilita la búsqueda por (en vez de buscar en 48 meses, buscamos sólo en 1 mes)
- **Realizar pre-cálculos**
  Pre Calcular los índices de engagement por centro y departamento, esto ahorra uso de CPU. La app sólo "busca y muestra"
- **Indexar cálculos por centro educativo y año**
  En vez de buscar línea centro a centro, usamos un índice "más chico" y con estrategias de búsqueda (ej árboles)

Estrategia a nivel de datos

Particionamiento



Índices



Modularización

Con aplicaciones grandes, es fundamental para el mantenimiento estructural adecuadamente el código mediante archivos y directorios. R viene con las funciones `library()` y `source()`, pero su funcionalidad es limitada cuando se trata de dividir su código en módulos y expresar sus dependencias.

Rhino vs golem

	Vanilla shiny	**Golem**	**Rhino**
Fiabilidad de la base del código	Simple, mientras no escale	Existe desde hace bastante tiempo.	Se centra en gran medida en el lado de ingeniería de SW
Desarrollo	Esfuerzo relativamente bajo	Es como tomar una colección de scripts a un paq. ordenado	Se basa en el concepto de cajas, no de paquetes.
Estructura del proyecto	Típicamente … Spaghetti Code	Estructura familiar de un paquete R	Vista-Controlador, donde cada cosa es un módulo.
Experiencia general en desarrollo.	No es muy eficiente	Convención de nomenclatura y estructura de carpetas más rígida	Extraño al principio, pero luego es muy rápido y entendible.
Capacidad de prueba	No está dentro de las prioridades	Lo mismo que testear un paquete	Integrable con Cypress.
Otros	Difícil de modularizar	Poco actualizado	Hay una industria atras

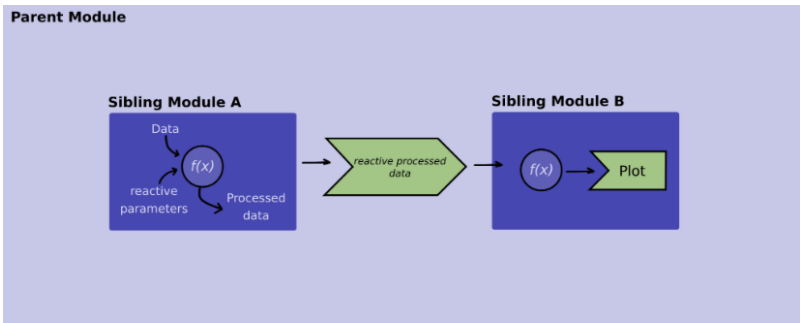
Lógica del proyecto

- Logica VC: Vista-Controlador.
- Otros recursos en distintas carpetas: JS, Test, Static, etc
- Integra Renv: Se copian los paquetes de dependencies (Garantiza reproducibilidad.)

```
.
├── app
│   ├── js
│   │   └── index.js
│   ├── logic
│   │   └── __init__.R
│   ├── static
│   │   └── favicon.ico
│   ├── styles
│   │   └── main.scss
│   ├── view
│   │   └── __init__.R
│   └── main.R
├── tests
│   ├── cypress
│   │   └── integration
│   │       └── app.spec.js
│   ├── testthat
│   │   └── test-main.R
│   └── cypress.json
├── app.R
├── RhinoApplication.Rproj
├── dependencies.R
├── renv.lock
└── rhino.yml
```

Lógica del proyecto

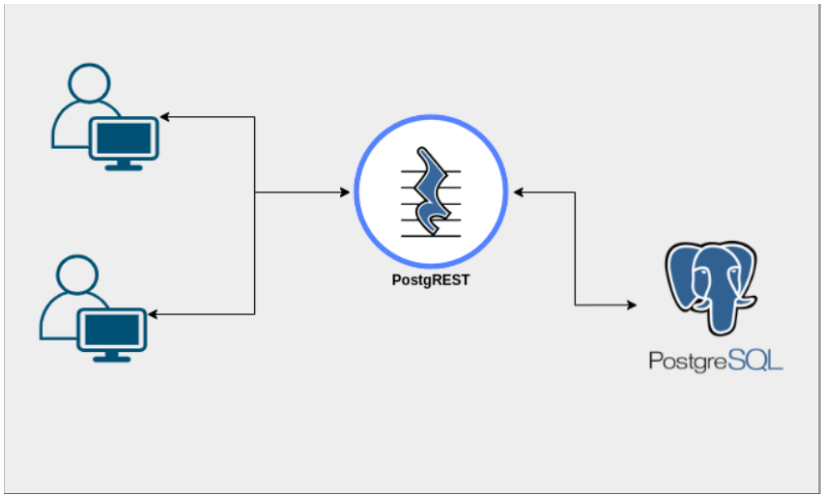
- Cada módulo es un box.
- Se comunican mediante reactives y llamadas de funciones.




```
# app/view/greet_module.R
box::use(
  shiny[moduleServer, NS, renderText, div, textOutput, req],
  shiny.semantic[textInput],
)

box::use(
  app/logic/greet[greet],
)

#' @export
ui <- function(id) {
  ns <- NS(id)
  div(
    textInput(ns("name"), "Name"),
    textOutput(ns("message"))
  )
}

#' @export
server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$message <- renderText({
      req(input$name)
      greet(input$name)
    })
  })
}
```

SQL y Pool de conexiones

- No es deseable hacer consultas SQL en código
- En R, no es fácil manejar el concepto de pool de conexiones



Interfaz de Usuario

¿Qué usar?

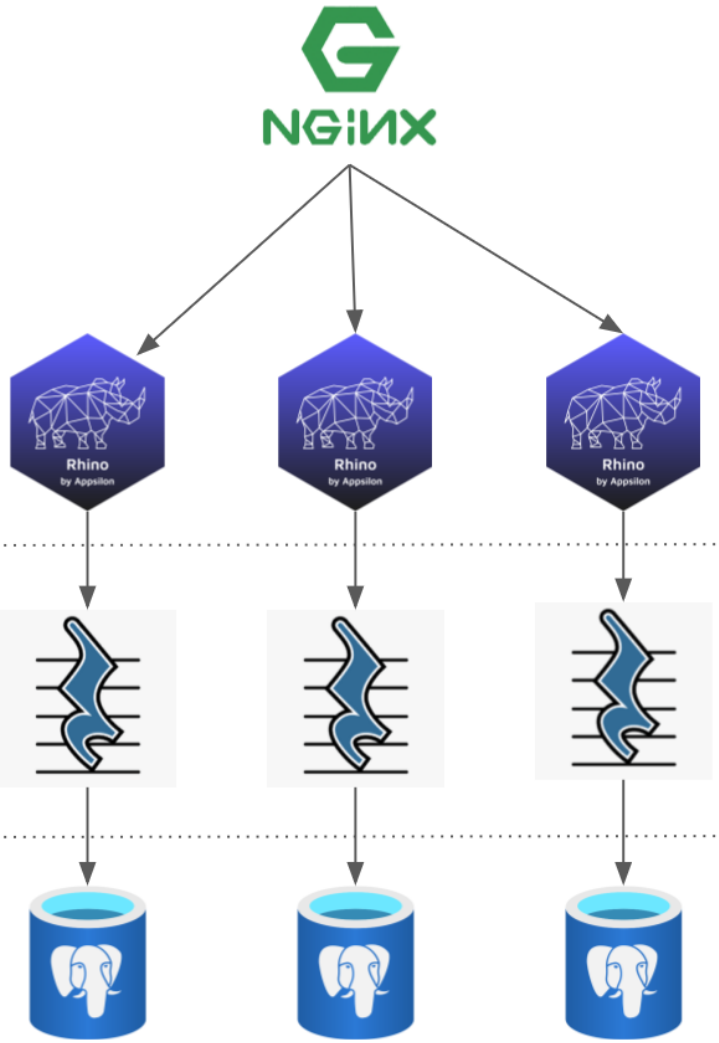
- vainilla: simple con limitantes.
- shinydashboard: muy repetido y desactualizado.
- shiny.fluent: excelente, aunque hay que codear mucho css y js
- semantic.dashboard: no es compatible con muchos otros paquetes.
- bslib: cambia bastante la lógica y sintaxis respecto a los otros.
- b4Dash: Una evolución de shinydashboard

Todo Junto

UI y visualizaciones

Pool de conexiones y WS

Datos y cálculos




MUESTRANOS LA APP!
imgflip.com