esLibre

SÍNTESIS DE AUDIO CON SHINY

Alejandro Rodríguez Antolín
Miguel Ángel Rodríguez Muiños

Arquitectura y Flujo de Ejecución en Shiny (R)

Código en R (App Shiny)

ui.R

define UI (Interfaz)

- fluidPage(), sidebarLayout(), plotOutput(). ...

server.R

define **Server Logic**

function(input, output, session) {

- reactivos: reactive(), observer()
- render*: renderPlot(), renderTable()
- lógica, cálculos, datos

}

Proceso de Ejecución

1. Lanzamiento: shiny::runApp( )
2. Inicia Servidor R (Puerto local, p.e. 3838)
3. Carga UI → Envío al Navegador (HTML/CSS/JS)
4. Espera Input del Usuario
5. Evento → input$variable cambia
6. Reactiva Server Logic (reactive( ), observe( ))
7. Genera Output (render*)
8. Envía actualización al Cliente (JSON + HTML dinámico)

Interfaz Web
(HTML/CSS/JS)

Navegador (Cliente)

Visualización Interactiva

Fuentes de Datos

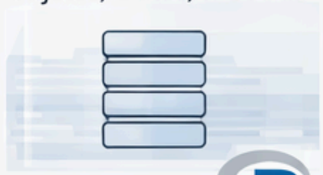
DB, Archivos, APIs

Reactivos de Shiny

- reactive( ) → Cálculos diferidos
- observe( ) → Responde a cambios
- eventReactive( ), isolate( )

Memoria (Entorno R)

Objetos, Datos, Modelos



Comunicación

- HTTP (WebSocket)
- JSON (Inputs/Outputs)
- Renderizado dinámico

Qué significa “síntesis de audio” en una app Shiny?

- Sintetizar audio significa crear una señal sonora en tiempo real (o casi real)
  - por ejemplo un beep, una senoide, una secuencia rítmica o un sonido más complejo (manipulando parámetros como envolvente, filtros, mezcla)

Cuestiones a tener en cuenta...

- El enfoque típico no es “hablar directamente con la tarjeta de sonido del usuario final”
  - En Shiny, R corre en el servidor
- Lo habitual es generar datos de audio y servir el resultado al navegador
    - Paquete **tuneR**: incluye funciones para generar ondas y construir objetos Wave
    - Paquete **seewave**: incluye funciones de síntesis (apoyadas en tuneR)
- El paquete **audio** sí ofrece reproducción contra dispositivos de audio
  - Está orientado al entorno donde se ejecuta R
    - En una app Shiny desplegada eso afecta al servidor, no al navegador del usuario

Cómo sintetizo audio en una app Shiny?

- Método 1:
  - En R, en el servidor
  - Generando muestras o un .wav y enviándolo al navegador
- Método 2:
  - En el navegador
  - Usando **Web Audio API** y dejando a R solo como controlador.

Método 1: Que el sonido lo genere R en el servidor

- R calcula y genera el vector de muestras
    - Por ejemplo una senoide
        - Frecuencia de muestreo, duración y amplitud
- Conversión a Wave
- Escritura del fichero WAV
- Shiny lo sirve al cliente
- El navegador lo reproduce

input Shiny → R calcula señal → Wave → navegador reproduce

Método 1: Ventajas

- Reproducibilidad y trazabilidad
  - Todo el código queda en R
  - Más fácilmente reutilizable (tuneR y seewave)
- Bueno para audio “offline”
  - Si el caso de uso es “pulso un botón y obtengo un sonido oarchivo”
- Menor complejidad que JavaScript.
  - La interfaz puede seguir siendo casi toda Shiny clásico.

Método 1: Desventajas

- Latencia más alta
  - Cada cambio obliga a pasar por servidor: calcular señal, servirla, reproducirla... (para controles continuos tipo slider, la sensación suele ser peor que en cliente)
- No es audio verdaderamente en tiempo real
  - El viaje R ⇄ navegador es el cuello de botella.
- Coste computacional
  - Cuantos más usuarios, más trabajo para el Servidor

Método 2: Que el sonido lo genere la API Web Audio

- R no manda muestras de audio, sino parámetros:
  - “toca una onda senoidal a 440 Hz durante 200 ms”, ...
  - “lanza secuencia”
- En Shiny:
  - R usa session$sendCustomMessage("playTone", list(freq=440, dur=0.2))
  - JavaScript recibe el mensaje
  - JS usa AudioContext, OscillatorNode, GainNode, ...
  - El navegador sintetiza localmente

input Shiny → R manda parámetros → JS/Web Audio sintetiza en el navegador

Método 2: Ventajas

- Menos latencia
  - La síntesis ocurre en el cliente
- Más capacidades nativas para síntesis.
  - Osciladores, buffers, filtros, ganancias, routing y scheduling encajan de forma natural.
- Mejor para timing fino
  - La programación temporal de eventos de audio suele ser más robusta en el motor de audio del navegador que en Shiny

Método 2: Desventajas

- Más complejidad técnica
  - Además de R, necesitas JavaScript
- Se aleja de la filosofía/metodología de programación en R
- Dificultad de reutilizar código ya escrito en R
  - tuneR, seawave, audio, ...

Método 1 vs Método 2

- **Generar en R cuando...**
  - El audio es resultado de un cálculo más que de una interacción inmediata
  - Necesitas exportar WAV o dejar rastro reproducible
  - Tu pipeline ya vive en tuneR/seewave
  - La app no requiere tocar, modular o secuenciar en tiempo real
- **Generar con Web Audio API cuando...**
  - Quieres respuesta inmediata
  - Habrá sliders, teclas, secuencias, envelopes o efectos
  - Buscas una experiencia tipo instrumento o interactividad
  - El servidor no debería cargar con el audio de todos los usuarios

Recomendación

- Si el usuario debe oír el cambio mientras mueve un control, usar Web Audio API

- Si el usuario pulsa “Generar” y espera un resultado reproducible o descargable, usar R

Ejemplo:
DRUM MACHINE

melisagal.shinyapps.io/DrumMachine/

Drum Machine

BPM (TEMPO)

60 120 180

60 72 84 96 108 120 132 144 156 168 180

120

VOLUMEN

0 70 100

0 10 20 30 40 50 60 70 80 90 100

70%

▶ PLAY

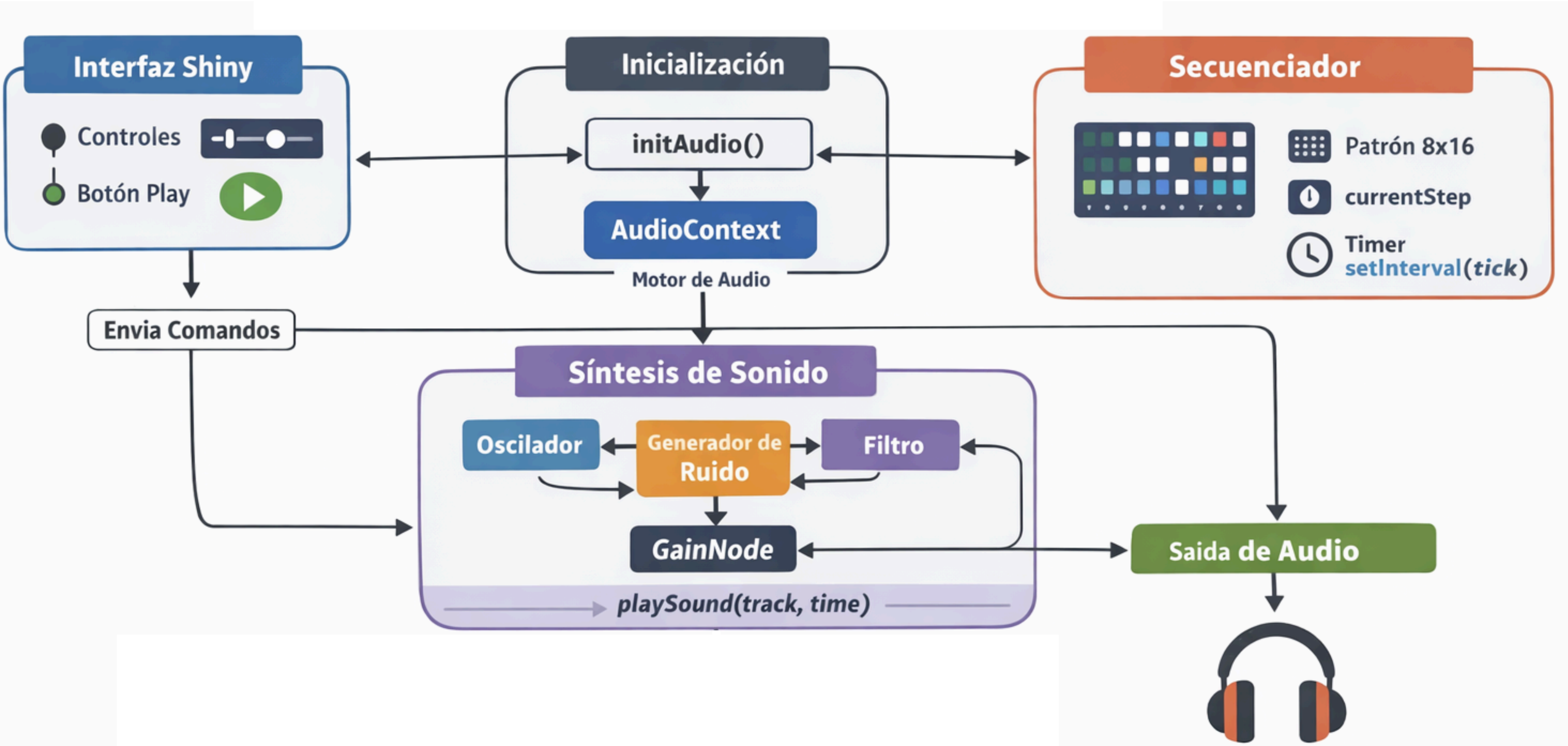
CLEAR

PRESETS:

Basic Beat | Rock | Hip Hop | Electronic

Kick	■				■				■				■			
Snare					■								■			
Hi-Hat	■		■		■		■		■		■		■		■	
Clap																
Tom High																
Tom Low																
Crash																
Rim																

Web Audio API en Shiny Drum Machine



Uso de Web Audio API en app Shiny Drum Machine

- Esquema general:
  - Aplicación Shiny → interfaz y control
    - UI (sliders, botones, grid)
    - Envía valores (BPM, volumen)
  - JavaScript + Web Audio API → generación de sonido
  - Navegador (cliente)
    - Ejecución y salida de audio

- Inicialización del audio
    - Se crea un único AudioContext
    - Función initAudio()
- Componentes Web Audio usados
    - OscillatorNode → genera tonos
    - AudioBufferSourceNode → genera ruido
    - GainNode → controla volumen
    - BiquadFilterNode → modifica el timbre
    - destination → salida (altavoces)

- Síntesis de sonido
  - No se usan samples → sonido generado en tiempo real
    - Kick → oscilador + caída de frecuencia
    - Snare → ruido + filtro high-pass
    - Hi-hat → ruido corto + band-pass
    - Clap → múltiples disparos de ruido
    - Tom → oscilador con frecuencia variable
    - Crash → ruido largo filtrado
    - Rim → onda cuadrada

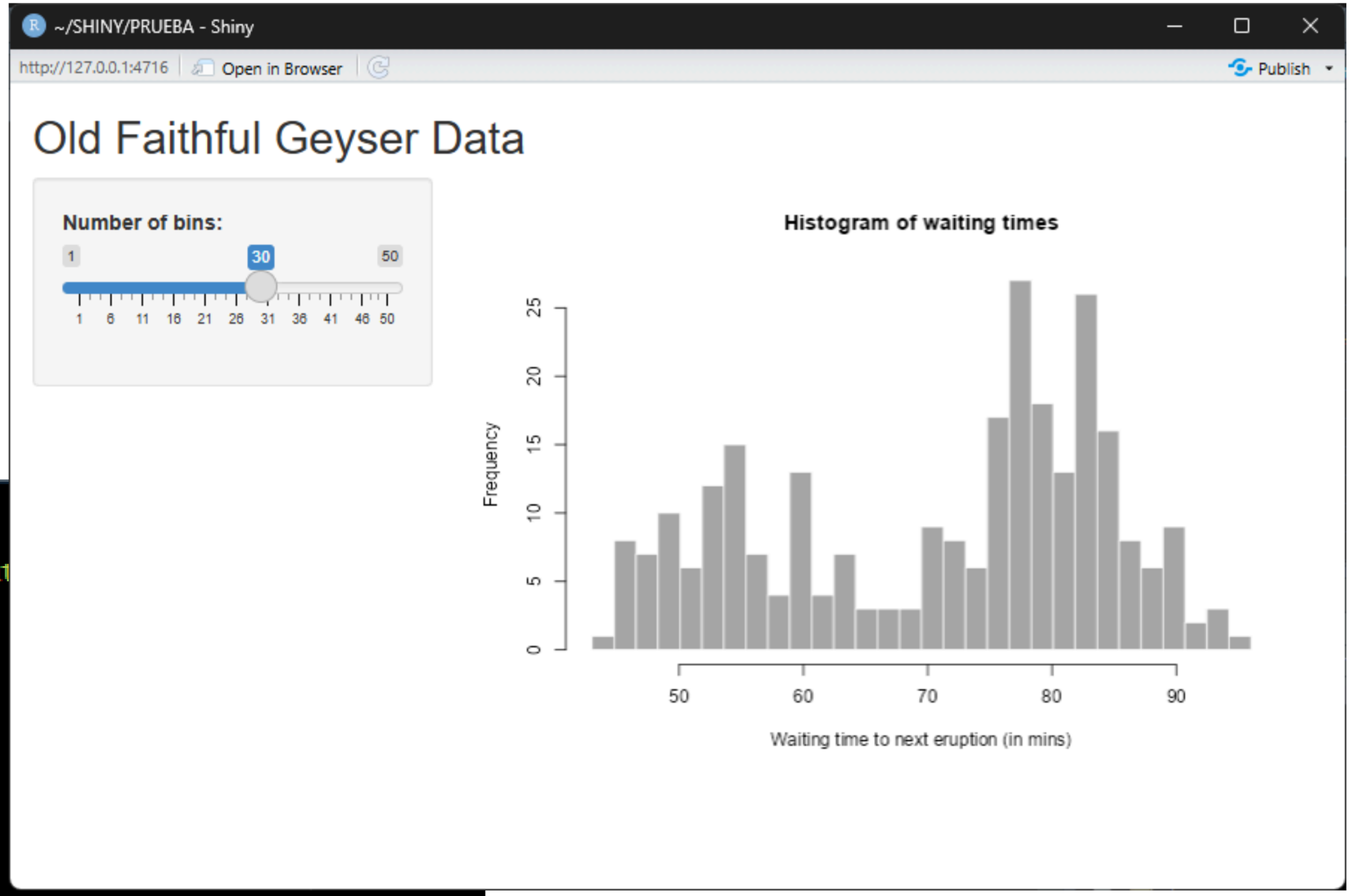
- Secuenciador (lógica rítmica)
  - Matriz: pattern[track][step] (8 × 16)
  - Variable: currentStep
- Función tick():
  - Avanza el paso
  - Revisa qué celdas están activas
  - Llama a playSound()
- Tempo
  - Intervalo (60 / BPM) * 1000 / 4
  - Usa setInterval()
  - Cada paso = semicorchea (16 pasos por compás)

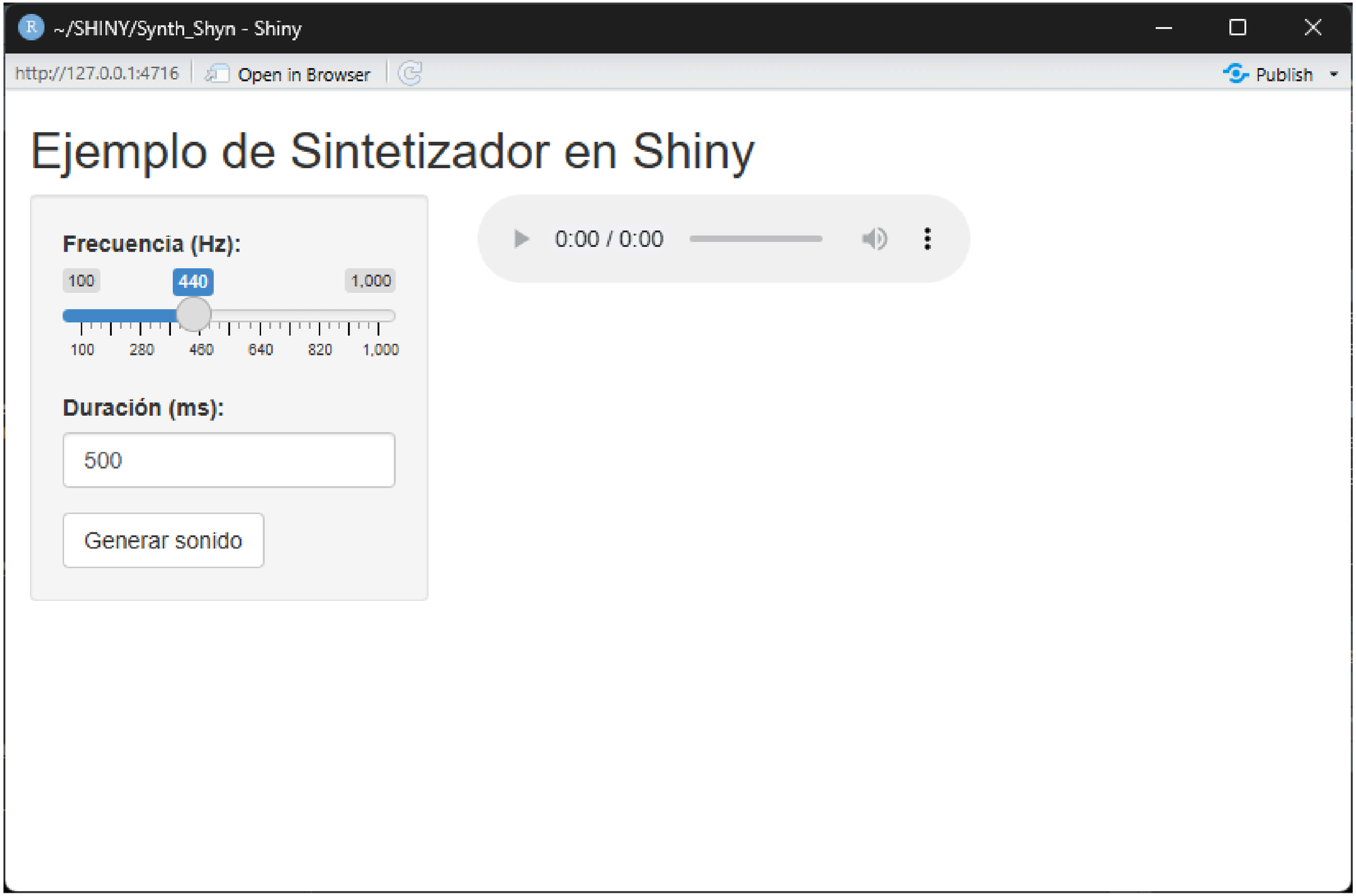
- Resumen:
  - Usuario pulsa Play
  - Se activa AudioContext
  - Se inicia el loop (setInterval)
  - tick() recorre el patrón
  - Si hay un paso activo → se genera sonido
  - El sonido pasa a Web Audio
  - Se escucha en los altavoces

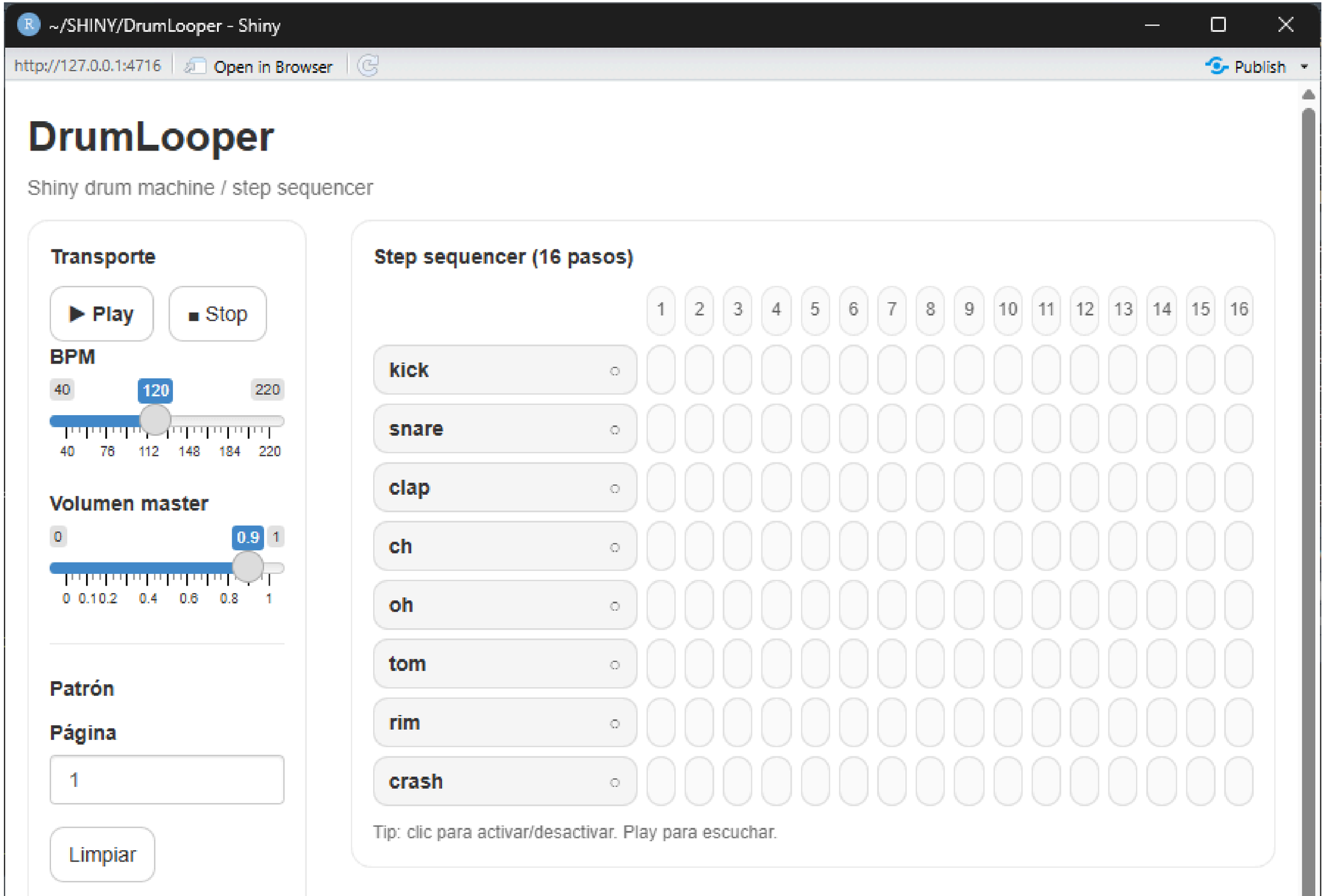
El diseño de la interfaz

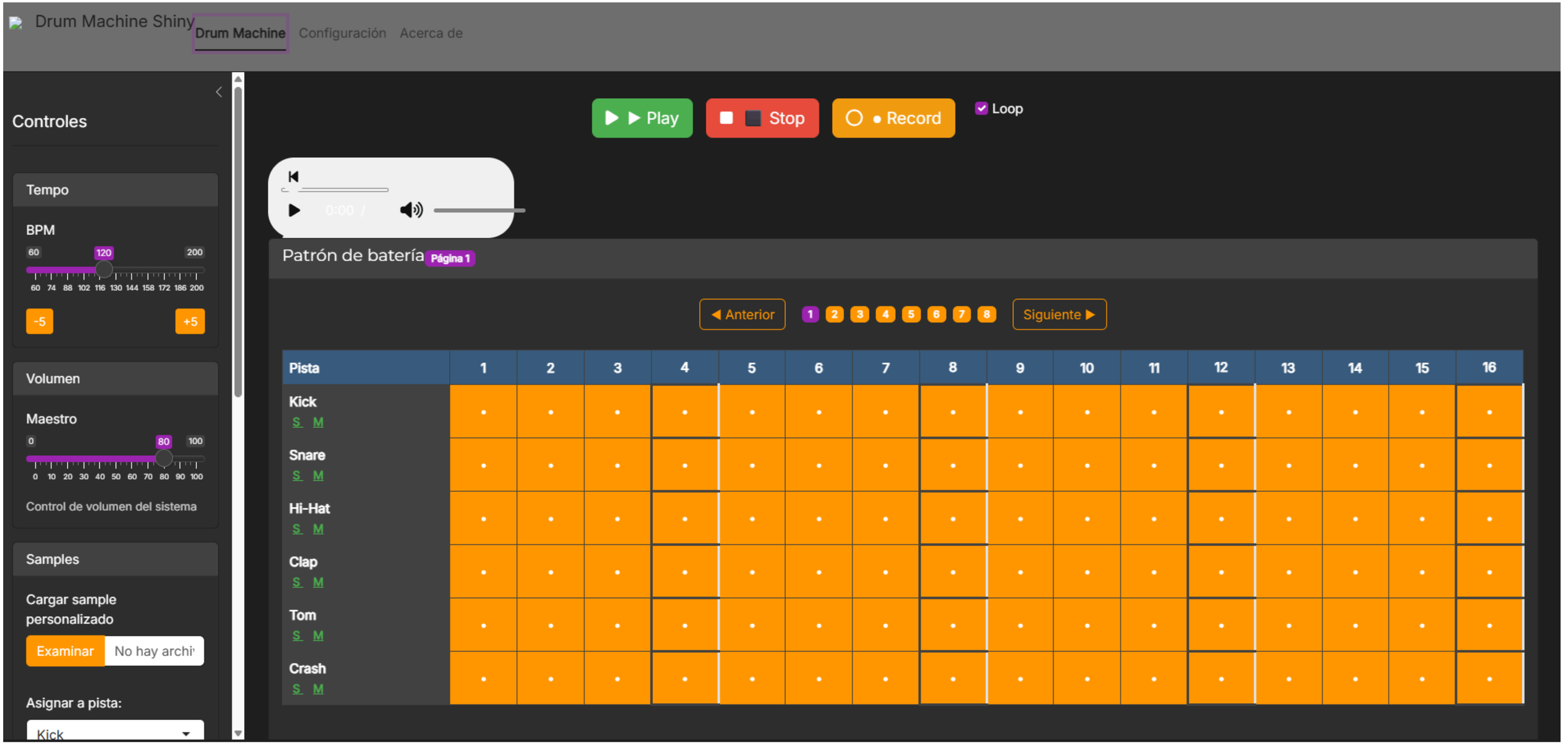
El “Hello World!” de Shiny

```
1  library(shiny)
2  ui <- fluidPage(
3    titlePanel("Old Faithful Geyser Dat
4    sidebarLayout(
5      sidebarPanel(
6        sliderInput("bins",
7                    "Number of bins:",
8                    min = 1, max = 50,
9      mainPanel(plotOutput("distPlot")
10 server <- function(input, output) {
11   output$distPlot <- renderPlot({
12     x    <- faithful[, 2]
13     bins <- seq(min(x), max(x), length.out = input$bins + 1)
14     hist(x, breaks = bins, col = 'darkgray', border = 'white',
15          xlab = 'Waiting time to next eruption (in mins)',
16          main = 'Histogram of waiting times') }) }
17 shinyApp(ui = ui, server = server)
18
```




~/SHINY/Synth_Shyn - Shiny
http://127.0.0.1:4716
Open in Browser
Publish
Ejemplo de Sintetizador en Shiny
Frecuencia (Hz):
100
440
1,000
100
280
460
640
820
1,000
Duración (ms):
500
Generar sonido
0:00 / 0:00


~/SHINY/DrumLooper - Shiny
http://127.0.0.1:4716
Open in Browser
Publish
DrumLooper
Shiny drum machine / step sequencer
Transporte
▶ Play
■ Stop
BPM
40
120
220
40 76 112 148 184 220
Volumen master
0
0.9
1
0 0.10.2 0.4 0.6 0.8 1
Patrón
Página
1
Limpiar
Step sequencer (16 pasos)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
kick
snare
clap
ch
oh
tom
rim
crash
Tip: clic para activar/desactivar. Play para escuchar.


Drum Machine Shiny
Drum Machine
Configuración
Acerca de
Controles
Tempo
BPM
60
120
200
60 74 88 102 116 130 144 158 172 186 200
-5
+5
Volumen
Maestro
0
80
100
0 10 20 30 40 50 60 70 80 90 100
Control de volumen del sistema
Samples
Cargar sample personalizado
Examinar
No hay archi
Asignar a pista:
Kick
Play
Stop
Record
Loop
0:00 /
Patrón de batería
Página 1
Anterior
1 2 3 4 5 6 7 8
Siguiente
Pista
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Kick
S M
Snare
S M
Hi-Hat
S M
Clap
S M
Tom
S M
Crash
S M

Cómo resolverlo?

R + CSS + JS + HTML

Construcción de la Interfaz Gráfica en App Shiny Drum Machine

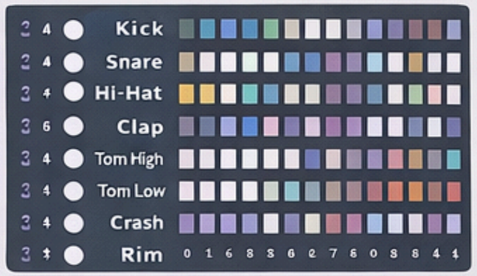
Contenedor Principal
fluidPage(...)

Estilo Visual
- tags$head(...)
- tags$style(...)
- Fondo oscuro
- Botones personalizados

Estructura Visual
- h1(Drum Machine)
- BPM / Tempo 120
- Volumen 80%

Control de Temporización Volumen
- div.controls(...)
- BPM / Tempo 120
- Volumen 80%

Secuenciador
- div(class = "sequencer"), uiOutput("sequencerUI")



Acciones buttons
- tags$button(...)
- ▶ PLAY
- ■ CLEAR

Presets
- div.preset-section(...)
- Basic Beat ✓
- Rock ✓
- Hip Hop ✓
- Electronic ✓

Pie de Página
- Instrucciones
- Enlace al código
- Atribución

Relaciones

R
- slidersInput(),
- textOutput(...)
- uiOutput(renderUI')

CSS
- slidersInput(),
- textOutput(...)
- JS callbacks
- uiOutput(renderUI')
- JS callbacks

JS
- Detectar clicks
- Cambiar clases
- Manejar presets

Drum Machine

BPM (TEMPO)

60 120 180

60 72 84 96 108 120 132 144 156 168 180

120

VOLUMEN

0 70 100

0 10 20 30 40 50 60 70 80 90 100

70%

▶ PLAY

CLEAR

PRESETS:

Basic Beat | Rock | Hip Hop | Electronic

Kick	■				■				■				■			
Snare					■								■			
Hi-Hat	■		■		■		■		■		■		■		■	
Clap																
Tom High																
Tom Low																
Crash																
Rim																

Una última vuelta de tuerca...

RetroTracker

melisagal.shinyapps.io/RetroTracker/

RETRO TRACKER

AudioFLOSS Team >> 4 canales · ticks por fila · FX 0/1/2/A/B/C/D/E/V/F · PAL/NTSC · Duración por nota

▶ Patrón | ▶▶ Canción | ■ Stop

Tempo (Fxx > 1F): 125 | Speed (ticks): 6 | Reloj: PAL 50Hz

Filas patrón: 26

CANALES / SONIDO

SINTETIZADOR + SAMPLE

- **Canal 1** — Pan automático · Vol/FX/Dur por nota — Seleccionado
- **Canal 2** — Pan automático · Vol/FX/Dur por nota — Disponible
- **Canal 3** — Pan automático · Vol/FX/Dur por nota — Disponible
- **Canal 4** — Pan automático · Vol/FX/Dur por nota — Disponible

sine | square | sawtooth | triangle

☐ Usar sample en vez de oscilador

Examinar... No se ha sel...gún archivo.

Ningún sample cargado (se usará el sintetizador)

El sample se afina según la nota (raíz aproximada C4).

En cada celda: **Nota** (C-4), **Vol** (00–0F), **FX**, **Dur** (ticks).
FX soportados:
0xy arpegio · 1xx/2xx slide tono · Axy vol slide · Vxx/Cxx volumen (C00 sin nota = corte) · Bxx position jump · Dxx pattern break · EEx pattern delay · ECx note cut en tick x · Fxx speed/BPM.
Dur en blanco = nota normal; si pones un número, la nota se corta tras ese número de ticks (gate).

PATRÓN ACTUAL

26 FILAS · 4 CANALES

Fila	Canal 1 Nota · Vol. · FX · Dur	Canal 2 Nota · Vol. · FX · Dur	Canal 3 Nota · Vol. · FX · Dur	Canal 4 Nota · Vol. · FX · Dur
00	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
01	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
02	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
03	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
04	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
05	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
06	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
07	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
08	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
09	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
10	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
11	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
12	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
13	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
14	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
15	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
16	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
17	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
18	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
19	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
20	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
21	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
22	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
23	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
24	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -
25	--- -- --- -	--- -- --- -	--- -- --- -	--- -- --- -

CANCIÓN / PATRONES

ORDER LIST

Patrón 1 / 1 | ◀ Anterior | Siguiente ▶

+ Nuevo patrón | Duplicar patrón | Eliminar patrón

Pos	Patrón #
00	0

+ Añadir posición | − Quitar última | Reset canción

GUARDAR / CARGAR CANCIÓN

JSON + NAVEGADOR

Guardar en navegador | Cargar del navegador

Exportar a JSON | Importar desde JSON

```
Pulsa "Exportar a JSON" para volcar aquí la canción completa (patrones + order list + ajustes), o pega un JSON y pulsa "Importar desde JSON".
```

Tick ≈ 2.5/BPM segundos (estilo ProTracker). PAL (50Hz) vs NTSC (60Hz) ajustan el tiempo del tick. Notas desde tabla fija C-2..B-5. Dur controla el "gate" en ticks.

Código fuente de RetroTracker

```
library(shiny)

ui <- fluidPage(
tags$iframe(src = "index.html", style = "width:100%; height:100vh; border:none;")
)

server <- function(...) {}

shinyApp(ui = ui, server = server)
```

Proyecto AudioFLOSS

audiofloss.melisa.gal

esLibre

GRACIAS

por vuestra atención

audiofloss@gmail.com