

# Sistema de control de accesos

Dr. Casimiro Gómez González  
Departamento de I+D, SMARTTEST  
correo: [casimiro.gomez@smartest.mx](mailto:casimiro.gomez@smartest.mx)  
Tel: 222 707 4118

Otoño 2021



# Prólogo

El presente documento describe el proyecto del sistema de control de accesos, que ha elaborado SMARTTEST es los últimos dos años. En un esfuerzo por el desarrollo de un producto que sea confiable y robusto para el usuario. El documento en si mismo trata de ser autocontenido obviamente sin atender fundamentos básicos que se dejan al lector, el proyecto utiliza dos tecnologías emergentes: El ecosistema Elixir y El ecosistema Flutter. Dentro de elixir ocupamos los frameworks de Elixir Nerves, Elixir Phoenix y LiveView; en Flutter ocupamos su lenguaje Dart.

El autor  
Casimiro Gómez González  
Departamento de I+D, SMARTTEST



# Índice general

<b>Prólogo</b>	<b>III</b>
<b>1. El proyecto de accesos</b>	<b>1</b>
<b>2. El controlador del acceso físico: torniquete o puerta</b>	<b>3</b>
2.1. Proyecto Elixir Poncho con firmware elixir nerves y UI de Phoenix Liveview . . . . .	3
2.1.1. Estructura base del proyecto . . . . .	4
2.1.2. Sincronizando con la nube GitHub . . . . .	5
2.1.3. Agregar al subproyecto firmware el subproyecto ui como dependencia . . . . .	5
2.1.4. Configurar WiFi en el subproyecto Firmware . . . . .	5
2.1.5. Configuración del web server en el subproyecto Firmware . . .	6
2.2. Particularizando el sistema . . . . .	7
2.3. Lectura a través de HID . . . . .	10
2.3.1. hidraw . . . . .	10
2.3.2. hiddev . . . . .	11
2.3.3. Puertos HID del lector . . . . .	12
2.4. Captura de datos del lector . . . . .	19



# Capítulo 1

## El proyecto de accesos

El sistema de control de accesos de SMARTTEST esta formado por tres subsistemas funcionales intimamente relacionados:

- El servidor de base de datos.- Los datos de usuario son administrados a través de una aplicación web, la cual tiene un API Json que da servicio a la consulta de usuarios.
- El APP de usuario.- Es una APP de celular que genera los códigos QR de acceso al usuario una vez que ingresa su clave y contraseña.
- El dispositivo de acceso.- Esta formado por dos lectores, uno de entrada y otro de salida, un torniquete o puerta de control, y una pantalla o semáforo para indicar que el acceso fue permitido o negado

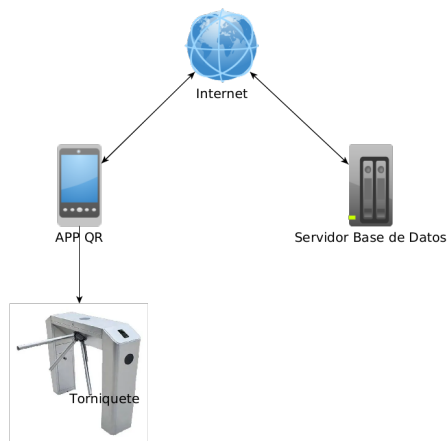


Figura 1.1: Sistema de Acceso SMARTTEST



## Capítulo 2

# El controlador del acceso físico: torniquete o puerta

El control de accesos físico, el cual puede ser un torniquete o una chapa eléctrica montada en alguna puerta, se diseñó utilizando una tarjeta raspberry pi, dicha tecnología se le montó un sistema Elixir Nerves junto con un frameworks Phoenix Liveview, ambos el elixir Nerves como el Phoenix Liveview son aglutinados utilizando la técnica “PONCHO”. Además se utilizarón dos técnicas de captura de eventos de los lectores:

- Lectura por evento.- Para lo cual se utiliza una librería que captura los eventos del sistema operativo que llegan al puerto `/dev /inputX` del dispositivo.
- Lectura por HID.- Se utiliza la propiedad de HID de los dispositivos capturando a través de un streaming las lecturas del dispositivo, en este caso a través de los puertos `/dev /hidX` del sistema operativo.

Empezaremos con describiendo de manera genérica como trabaja un proyecto poncho que glutinará los otros dos subproyectos.

### 2.1. Proyecto Elixir Poncho con firmware elixir nerves y UI de Phoenix Liveview

Un proyecto poncho es una estructura para la elaboración de proyectos grandes, en donde conceptualmente se dividen las funciones, en este caso en dos partes principales como se muestra en la figura 2.1.

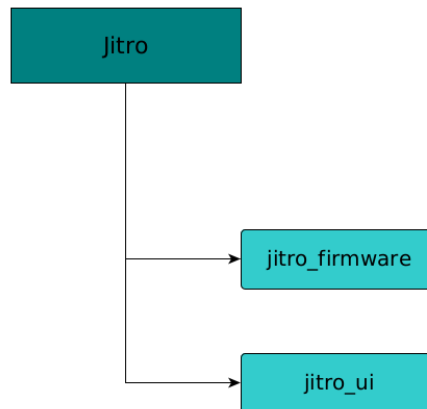


Figura 2.1: Estructura General del Proyecto JITRO de SMARTTEST

### 2.1.1. Estructura base del proyecto

Primero crearemos el directorio en el cual se harán los dos subproyectos, tanto el de firmware como el de ui.

```
1 $ mkdir Jitro
2 $ cd Jitro
```

Ahora estableceremos el nombre del proyecto, en este caso será jitro<sup>1</sup>.

```
1 $ export NOMBRE_PROYECTO=jitro
```

Crearemos también el archivo **README.md** con información para cuando se suba a github.

```
1 $ echo "Proyecto creado por el departamento de I+D de SMARTTEST\nDr
. Casimiro Gomez Gonzalez\n2021" > README.md
```

Posteriormente inicializamos el git

```
1 $ git init && git add . && git commit -m "Inicial commit"
```

Ahora se crea el firmware

```
1 $ mix nerves.new "$NOMBRE_PROYECTO"_firmware
2 $ git add . && git commit -m "Creando Nerves firmware subproyecto"
```

Procedemos a crear el subproyecto de ui

```
1 $ mix phx.new "$NOMBRE_PROYECTO"_ui --no-ecto --live
2 $ git add . && git commit -m "Creando Phoenix Liveview UI subproyecto"
```

<sup>1</sup>Por si tienen curiosidad jitro proviene del idioma construido Lojban y significa control

### 2.1.2. Sincronizando con la nube GitHub

Para la sincronización y respaldo del proyecto y su publicación se creo un proyecto vacio en github y se subirán a ese proyecto los códigos generados, es por ello que se enlaza a través de los siguientes comandos (suponiendo que nos encontramos en el directorio jitro):

```
1 $ git branch -M main
2 $ git remote add origin git@github.com:casimirogomez/jitro.git
3 $ git push -u origin main
```

### 2.1.3. Agregar al subproyecto firmware el subproyecto ui como dependencia

Para integrar el proyecto necesitamos agregar al subproyecto de firmware el subproyecto de ui como una dependencia, para ello necesitamos editar el archivo `jitro_firmware /mix.exs`.

```
1 #jitro_firmware/mix.exs
2 {:jitro_ui, path: "../jitro_ui", targets: @all_targets, env: Mix.env
  ()},
```

### 2.1.4. Configurar WiFi en el subproyecto Firmware

Para este proyecto se configura al accesos a redes a través del WiFi y de cableado con DHCP. Para ello editamos el archivo `jitro_firmware /config /target.exs`.

```
1 #jitro_firmware/config/target.exs
2 {"wlan0",
3  %{
4    type: VintageNetWiFi,
5    vintage_net_wifi: %{
6      networks: [
7        %{
8          key_mgmt: :wpa_psk,
9          ssid: System.get_env("WIFI_SSID"),
10         psk: System.get_env("WIFI_PSK")
11       }
12     ]
13   },
14   ipv4: %{method: :dhcp}
15 }
```

### 2.1.5. Configuración del web server en el subproyecto Firmware

Si estamos usando una estructura de proyecto poncho, debemos tener en cuenta que la configuración `jitro_ui` no se aplicará automáticamente, por lo que debemos importarla desde allí o duplicar la configuración requerida. Para ello, editamos el archivo `jitro_firmware /config /target.exs` agregando las líneas que se muestran a continuación:

```

1 #jitro_firmware/config/target.exs
2
3 # Import default UI config based on MIX_ENV
4 import_config "../../jitro_ui/config/config.exs"
5 import_config "../../jitro_ui/config/prod.exs"
6
7 # Override some UI config for firmware
8 config :jitro_ui, JitroUiWeb.Endpoint,
9   # Nerves root filesystem is read-only, so disable the code
   # reloader
10  code_reloader: false,
11  # Use compile-time Mix config instead of runtime environment
   # variables
12  load_from_system_env: false,
13  # Start the server since we're running in a release instead of
   # through 'mix'
14  server: true,
15  # "<firmware's hostname>.local"
16  url: [host: "nerves.local"],
17  http: [port: 80],
18  check_origin: false,
19  root: Path.dirname(__DIR__)

```

Una aplicación web basada en Phoenix ahora está lista para ejecutarse en nuestro dispositivo integrado basado en Nerves. Al separar el proyecto basado en Phoenix del proyecto basado en Nerves, permitimos que los equipos trabajen en la funcionalidad principal y el código de la interfaz de usuario incluso sin tener hardware físico. También minimizamos el esfuerzo de integración de hardware / software al administrar tanto el software central como la infraestructura de implementación del firmware en un solo proyecto de poncho o paraguas.

Al desarrollar la interfaz de usuario, simplemente podemos ejecutar el servidor Phoenix desde el directorio del proyecto `jitro_ui`:

```

1 # Suponiendo que nos encontramos en el directorio raíz $HOME/Jitro
2 $ cd jitro_ui
3 $ iex -S mix phx.server

```

*Nota 2.1.1.* Si al ejecutar el comando anterior existe un error donde nos indique que ejecutemos en el subdirectorio `assets` el comando `npm install` se debe a incompatibilidad entre el `nodejs` y el `npm` instalado en el sistema operativo. En estos casos lo recomendable es instalar tanto el `nodejs` como el `npm` que vienen con la distribución de linux, eso resuelve cualquier conflicto de incompatibilidad, así lo recomendable es borrar el subproyecto `ui` y volver a crearlo una vez ya se halla instalado el nuevo `nodejs` y `npm` de la distribución.

Ahora es momento de compilar el firmware, esto lo podemos realizar desde el directorio `jitra_firmware` de nuestro proyecto:

```
1 # Suponiendo que nos encontramos en el directorio raíz $HOME/Jitra
2 $ cd jitra_firmware
3 $ export MIX_TARGET=rpi4
4 $ mix deps.get
```

*Nota 2.1.2.* En el caso de que marque error de “environment variable `SECRET_KEY_BASE` is missing” debemos ejecutar los siguientes comandos:

```
1 # Suponiendo que nos encontramos en el directorio raíz $HOME/Jitra
2 $ cd jitra_firmware
3 $ export SECRET_KEY_BASE=proyecto_jitra
4 $ mix deps.get
5 $ mix phx.gen.secret
6 $ export SECRET_KEY_BASE=TaKFCrxwz1Yjvtu0sJx2rxgaJG/8e+d+1
   XpaM90CyLr1NcJpEtmrqbeLcV2/+xVc (resultado del comando anterior)
7 $ mix deps.get
```

después de lo cual simplemente ejecutamos:

```
1 $ mix firmware
```

## 2.2. Particularizando el sistema

Ahora se cambiarán las imágenes para que muestren los logos `smartest`. Para ello se copian los logos (como se muestran en la figura) en el directorio `jitra_ui/assets/static/images`

Ahora editaremos el archivo `jitra_ui/lib/jitra_ui_web/templates/layout/root.html.leex`

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8"/>
5     <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
```

*Departamento de I+D, SMARTEST  
Elaboró: Dr. Casimiro Gómez González*

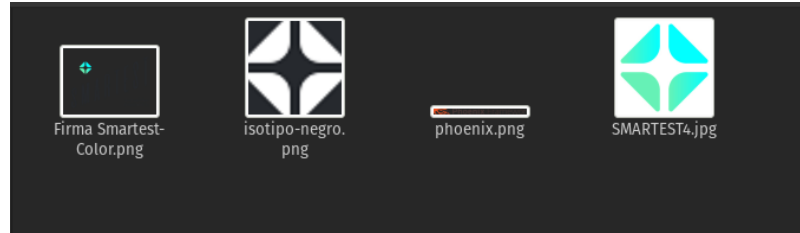


Figura 2.2: Logos de SMARTTEST en el directorio `jitro_ui/assets/static/images`

```

6   <meta name="viewport" content="width=device-width, initial-scale
    =1.0"/>
7   <%= csrf_meta_tag() %>
8   <%= live_title_tag assigns[:page_title] || "Jitro", suffix: "
    Sistema de acceso" %>
9   <link phx-track-static rel="stylesheet" href="<%= Routes.
    static_path(@conn, "/css/app.css") %>" />
10  <script defer phx-track-static type="text/javascript" src="<%=
    Routes.static_path(@conn, "/js/app.js") %>"></script>
11 </head>
12 <body>
13   <header>
14     <section class="container">
15       <nav role="navigation">
16         <ul>
17           <li><a href="http://www.smartest.mx/">Pagina de inicio</
            a></li>
18           <%= if function_exported?(Routes, :live_dashboard_path,
            2) do %>
19             <li><%= link "LiveDashboard", to: Routes.
                live_dashboard_path(@conn, :home) %></li>
20           <% end %>
21         </ul>
22       </nav>
23       <a href="https://www.smartest.mx/" class="phx-logo">
24         " alt="Logo SMARTEST"/>
25       </a>
26     </section>
27   </header>
28   <%= @inner_content %>
29 </body>
30 </html>

```

El resultado se muestra en la figura 2.3.

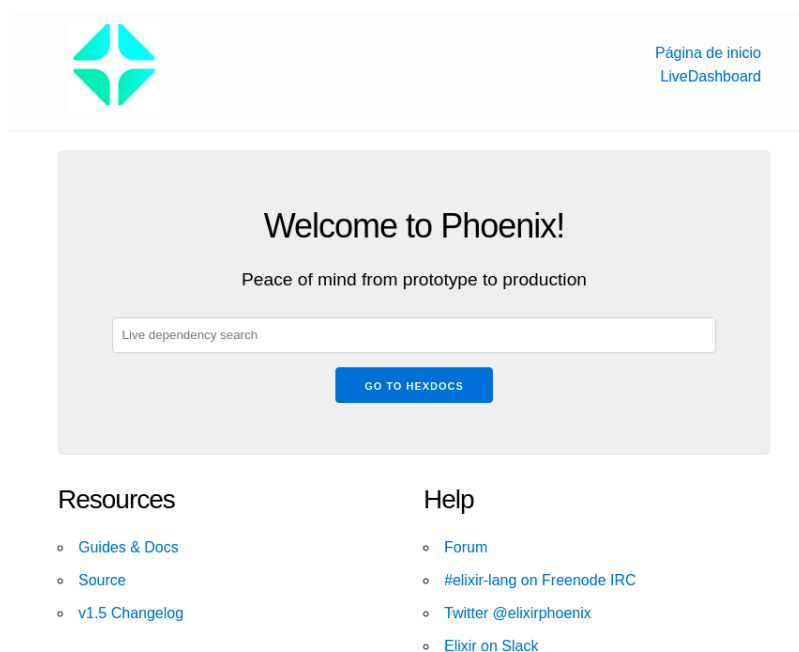


Figura 2.3: Pantalla de accesos después de la modificación

Ahora eliminamos la bienvenida de Phoenix para colocar nuestros letreros, para ello editamos el archivo `jito_ui /lib /jito_ui_web /live /page_live.html.leex`, como se muestra:

```
1 <section class="phx-hero">
2   <h1><%= gettext "Bienvenido a %{name}!", name: "Jitro" %></h1>
3   <p>El sistema de control de accesos de SMARTTEST</p>
4
5
6 </section>
7
8 <section class="row">
9   <article class="column">
10    <h2>Recursos</h2>
11
12   </article>
13   <article class="column">
14    <h2>Ayuda</h2>
15
16   </article>
```

*Departamento de I+D, SMARTTEST*  
*Elaboró: Dr. Casimiro Gómez González*

17 </section>

Quedando como se muestra en la figura 2.4



Figura 2.4: Pantalla de accesos después de la modificación de bienvenida

## 2.3. Lectura a través de HID

Los dispositivos HID (*Human Interface Devices*) pueden ser de dos tipos principales:

- Los HIDRAW.- El controlador hidraw proporciona una interfaz sin formato para dispositivos de interfaz humana USB y Bluetooth. Se diferencia de hiddev en que los informes enviados y recibidos no son analizados por el analizador HID, sino que se envían y reciben del dispositivo sin modificar.
- Los HIDDEV.-

### 2.3.1. hidraw

Hidraw debe utilizarse si la aplicación del espacio de usuario sabe exactamente cómo comunicarse con el dispositivo de hardware y puede construir los informes HID manualmente. Este suele ser el caso cuando se crean controladores de espacio de usuario para dispositivos HID personalizados.



Hidraw también es útil para comunicarse con dispositivos HID no conformes que envían y reciben datos de una manera que no concuerda con sus descriptores de informes. Debido a que hiddev analiza los informes que se envían y reciben a través de él, comparándolos con el descriptor de informes del dispositivo, dicha comunicación con estos dispositivos no conformes es imposible utilizando hiddev. Hidraw es la única alternativa, además de escribir un controlador de kernel personalizado, para estos dispositivos no compatibles.

Un beneficio de hidraw es que su uso por parte de las aplicaciones del espacio de usuario es independiente del tipo de hardware subyacente. Actualmente, Hidraw está implementado para USB y Bluetooth. En el futuro, a medida que se desarrollen nuevos tipos de bus de hardware que utilicen la especificación HID, hidraw se ampliará para agregar soporte para estos nuevos tipos de bus.

Hidraw usa un número mayor dinámico, lo que significa que se debe confiar en udev para crear nodos de dispositivo hidraw. Udev normalmente creará los nodos del dispositivo directamente debajo de / dev (por ejemplo: / dev / hidraw0). Como esta ubicación depende de la distribución y de las reglas de udev, las aplicaciones deben usar libudev para ubicar los dispositivos hidraw conectados al sistema.

### 2.3.2. hiddev

Además de los dispositivos HID de tipo de entrada normal, USB también utiliza los protocolos de dispositivo de interfaz humana para cosas que no son realmente interfaces humanas, pero que tienen necesidades de comunicación similares. Los dos grandes ejemplos de esto son los dispositivos de alimentación (especialmente las fuentes de alimentación ininterrumpidas) y el control del monitor en monitores de gama alta.

Para admitir estos requisitos dispares, el sistema USB de Linux proporciona eventos HID a dos interfaces separadas:

- el subsistema de entrada, que convierte los eventos HID en interfaces de dispositivos de entrada normales (como teclado, mouse y joystick) y una interfaz de eventos normalizada; consulte `Documentation / input / input.rst`
- la interfaz hiddev, que proporciona eventos HID bastante crudos

El flujo de datos para un evento HID producido por un dispositivo es similar al que se muestra en la figura 2.5

Además, otros subsistemas (además de USB) pueden potencialmente alimentar eventos en el subsistema de entrada, pero estos no tienen ningún efecto en la interfaz del dispositivo oculto.

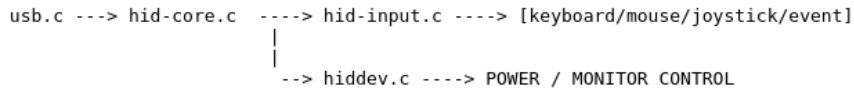


Figura 2.5: Flujo de datos para un evento HID

La interfaz hiddev es una interfaz de caracteres que utiliza el USB mayor normal, con los números menores comenzando en 96 y terminando en 111. Por lo tanto, necesita los siguientes comandos:

```

1 mknod /dev/usb/hiddev0 c 180 96
2 mknod /dev/usb/hiddev1 c 180 97
3 mknod /dev/usb/hiddev2 c 180 98
4 mknod /dev/usb/hiddev3 c 180 99
5 mknod /dev/usb/hiddev4 c 180 100
6 mknod /dev/usb/hiddev5 c 180 101
7 mknod /dev/usb/hiddev6 c 180 102
8 mknod /dev/usb/hiddev7 c 180 103
9 mknod /dev/usb/hiddev8 c 180 104
10 mknod /dev/usb/hiddev9 c 180 105
11 mknod /dev/usb/hiddev10 c 180 106
12 mknod /dev/usb/hiddev11 c 180 107
13 mknod /dev/usb/hiddev12 c 180 108
14 mknod /dev/usb/hiddev13 c 180 109
15 mknod /dev/usb/hiddev14 c 180 110
16 mknod /dev/usb/hiddev15 c 180 111

```

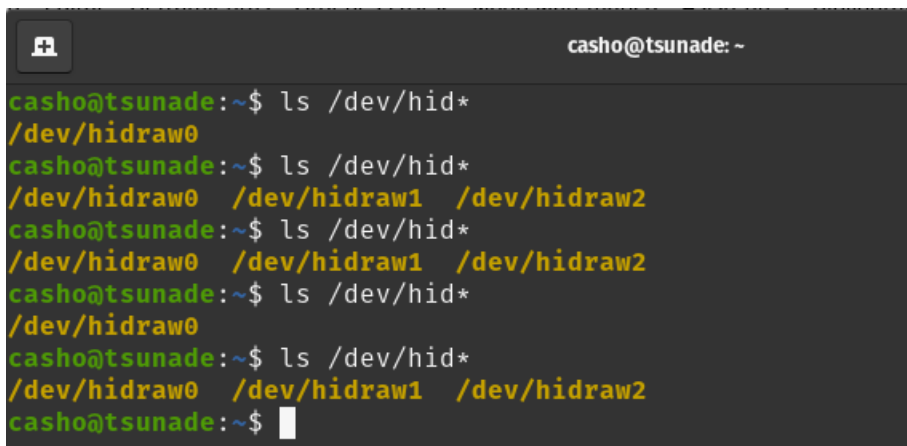
### 2.3.3. Puertos HID del lector

Cuando conectamos el lector de QR, RFID y NFC en un puerto USB, podemos detectarlo usando el comando que se muestra en la figura 2.6.

En la figura 2.6 se muestra el resultado del comando **ls /dev/hid\*** cuando se conecta y desconecta un lector. Debe revisarse en cual puerto de los dos que activa el lector, envia los datos de sus tres métodos de lectura: QR, RFID y NFC.

La captura de las lecturas a través del puerto **/dev/hidrawX** se realizan utilizando dos Genservers y un programa:

- Programa **lector.ex** .- El programa tiene dos mapas y dos rutinas. Los mapas convierten el número hexadecimal que lee el puerto HID y lo convierte en un carácter utf-8, el mapa **@hid** los convierte el minúsculas y el **@hid2** los convierte en mayúsculas. La función **decode\_hid** lee la secuencia hexadecimal de un



```

casho@tsunade:~$ ls /dev/hid*
/dev/hidraw0
casho@tsunade:~$ ls /dev/hid*
/dev/hidraw0 /dev/hidraw1 /dev/hidraw2
casho@tsunade:~$ ls /dev/hid*
/dev/hidraw0 /dev/hidraw1 /dev/hidraw2
casho@tsunade:~$ ls /dev/hid*
/dev/hidraw0
casho@tsunade:~$ ls /dev/hid*
/dev/hidraw0 /dev/hidraw1 /dev/hidraw2
casho@tsunade:~$

```

Figura 2.6: Lista de dispositivos HID cuando conectas un lector

caracter, la cual viene formada por muchos ceros y algún código hexadecimal de la tecla enter (un solo número diferente) o un número hexadecimal diferente que representa una caracter o bien dos número hexadecimal que representan un letra mayúscula (ya que uno de esos dos números es el código de la tecla shift). La función **generar** primero filtra los ceros del lector, posteriormente convierte los códigos hexadecimal leídos en caracteres utf-8 y por último filtra aquellos caracteres vacíos.

- Genserver **lector\_hid1.ex**.- Este genserver abre el puerto para leer los hexadecimales de entrada del lector y va creando en su estado la lista de caracteres utf-8 que corresponden a cada lista de hexadecimales leídos. Además concatena la lista de caracteres para formar la cadena de lectura. Envía al genserver **cadena\_actual** la cadena leída y si esta corresponde a un lector de entrada o de salida, basándose en el número de puerto HID que hizo la lectura.
- Genserver **cadena\_actual.ex**.- Este genserver mantiene en su estado la última cadena leída y que lector la realizó: si el de entrada o salida.

A continuación se muestra el código del programa **lector.ex**

### Código del programa lector.ex

```

1 defmodule JitroUi.Lector do
2   # ***** Teclas Especiales *****
3   #define KEY_MOD_RSHIFT 0x20

```

*Departamento de I+D, SMARTTEST  
Elaboró: Dr. Casimiro Gómez González*

```
4  #define KEY_MOD_LSHIFT 0x02
5  @shift_derecha      0x20
6  @shift_izquierda    0x02
7  @enter              0x28
8
9  @hid %{ 0x04 => 'a',
10         0x05 => 'b',
11         0x06 => 'c',
12         0x07 => 'd',
13         0x08 => 'e',
14         0x09 => 'f',
15         0x0A => 'g',
16         0x0B => 'h',
17         0x0C => 'i',
18         0x0D => 'j',
19         0x0E => 'k',
20         0x0F => 'l',
21         0x10 => 'm',
22         0x11 => 'n',
23         0x12 => 'o',
24         0x13 => 'p',
25         0x14 => 'q',
26         0x15 => 'r',
27         0x16 => 's',
28         0x17 => 't',
29         0x18 => 'u',
30         0x19 => 'v',
31         0x1A => 'w',
32         0x1B => 'x',
33         0x1C => 'y',
34         0x1D => 'z',
35         0x1E => '1',
36         0x1F => '2',
37         0x20 => '3',
38         0x21 => '4',
39         0x22 => '5',
40         0x23 => '6',
41         0x24 => '7',
42         0x25 => '8',
43         0x26 => '9',
44         0x27 => '0',
45         0x2C => '□',
46         0x2D => '-',
47         0x2E => '=',
48         0x2F => '[',
49         0x30 => ']' ,
```

```

50      0x31 => '\\',
51      0x33 => ';',
52      0x34 => '\\',
53      0x35 => ',',
54      0x36 => ', ',
55      0x37 => '. ',
56      0x38 => '/ ',
57      0x28 => ' ' }
58 @hid2 %{ 0x04 => 'A',
59      0x05 => 'B',
60      0x06 => 'C',
61      0x07 => 'D',
62      0x08 => 'E',
63      0x09 => 'F',
64      0x0A => 'G',
65      0x0B => 'H',
66      0x0C => 'I',
67      0x0D => 'J',
68      0x0E => 'K',
69      0x0F => 'L',
70      0x10 => 'M',
71      0x11 => 'N',
72      0x12 => 'O',
73      0x13 => 'P',
74      0x14 => 'Q',
75      0x15 => 'R',
76      0x16 => 'S',
77      0x17 => 'T',
78      0x18 => 'U',
79      0x19 => 'V',
80      0x1A => 'W',
81      0x1B => 'X',
82      0x1C => 'Y',
83      0x1D => 'Z',
84      0x1E => '!',
85      0x1F => '@',
86      0x20 => '#',
87      0x21 => '$',
88      0x22 => '%',
89      0x23 => '^',
90      0x24 => '&',
91      0x25 => '*',
92      0x26 => '(',
93      0x27 => ')',
94      0x2C => ' ',
95      0x2D => '-'

```

```

96         0x2E => '+',
97         0x2F => '{',
98         0x30 => '}',
99         0x31 => '|',
100        0x33 => ':',
101        0x34 => '"',
102        0x35 => '~',
103        0x36 => '<',
104        0x37 => '>',
105        0x38 => '?',
106        0x28 => ' ' }
107    def generar(mensaje) do
108        mensaje
109        |> Enum.filter(fn y -> y > 0 end )
110        |> decode_hid
111        |> Enum.filter(fn y -> y != [] end )
112    end
113
114    @spec decode_hid(nonempty_maybe_improper_list, any) :: [[1..255]]
115    def decode_hid(lista, shift \\ false) do
116        if (lista != []) do
117            valores= Enum.reduce(lista, 0, fn x, acc -> x + acc end)
118            [h|t] = lista
119            if (h == @enter ) do
120                ''
121            else
122                {cadena, shift2} = if (h == @shift_derecha || h ==
123                                     @shift_izquierda) && (valores > 0x20) do
124                    {'', true}
125                else
126                    if shift == true do
127                        case Map.fetch(@hid2, h) do
128                            {:ok, dato} -> {dato, false}
129                            _ -> {'', false}
130                        end
131                    else
132                        case Map.fetch(@hid, h) do
133                            {:ok, dato} -> {dato, false}
134                            _ -> {'', false}
135                        end
136                    end
137                [cadena | decode_hid(t, shift2)]
138            end
139        else
140            ''

```

```
141     end
142   end
143 end
```

Ahora el código del genserver `lector_hid1.ex`

### Código del genserver `lector_hid1.ex`

```
1 defmodule JitroUi.LectorHid1 do
2   use GenServer
3   alias JitroUi.Lector
4   alias JitroUi.CadenaActual
5
6   @nombre :lectorHID1_server
7   @valor_inicial []
8
9   def start_link(_opts) do
10     GenServer.start_link(__MODULE__, @valor_inicial, name: @nombre)
11   end
12   def init(lista_inicial) do
13     _pid = Port.open('/dev/hidraw1', [:stream, :eof])
14     {:ok, lista_inicial}
15   end
16   def actualizar() do
17     GenServer.call @nombre, :actualizar
18   end
19
20   def actual() do
21     GenServer.call @nombre, :actual
22   end
23   def handle_call(:actual, _from, ultima_lectura) do
24     {:reply, ultima_lectura, ultima_lectura}
25   end
26   def handle_call(:actualizar, _from, ultima_lectura) do
27     {:reply, ultima_lectura, ultima_lectura}
28   end
29   def handle_info(mensaje, estado) do
30     {_puerto, {:data, caracter}} = mensaje
31     nuevo_caracter = caracter
32     |> Enum.filter(fn y -> y > 0 end )
33     nuevo_estado = if nuevo_caracter != [] do
34       if Enum.any?(caracter, fn y -> y == 40 end) do
35         IO.puts "\n_Cadena_Salida:_ "
36         IO.puts (inspect estado)
37         CadenaActual.actualizar({estado, "entrada" })
38       end
39     end
39   end
```

```

39         else
40             #IO.puts Lector.generar(caracter)
41             generado=Lector.generar(caracter)
42             #IO.puts "#{estado}+#{generado}"
43             #IO.puts (inspect caracter)
44             [estado | generado]
45         end
46     else
47         estado
48     end
49     {:noreply, List.flatten( nuevo_estado)}
50 end
51 end

```

Por último el código de `cadena_actual.ex`

### Código del genserver `cadena_actual.ex`

```

1 defmodule JitroUi.CadenaActual do
2   use GenServer
3
4   @nombre :cadenaActual_server
5   @valor_inicial %{:lectura => "Aun no lee ", :direccion => "Sin
6     direccion "}
7
8   def start_link(_opts) do
9     GenServer.start_link(__MODULE__, @valor_inicial, name: @nombre)
10  end
11  def init(lista_inicial) do
12    {:ok, lista_inicial}
13  end
14  def actual() do
15    GenServer.call @nombre, :actual
16  end
17  def actualizar(lectura) do
18    GenServer.call @nombre, {:actualizar, lectura}
19  end
20  def handle_call(:actual, _from, respuesta) do
21    {:reply, respuesta, respuesta}
22  end
23  def handle_call({:actualizar, {cadena, direccion}}, _from, _lector)
24    do
25      estado_actual=%{:lectura => cadena, :direccion => direccion}
26      {:reply, estado_actual, estado_actual}
27    end
28  end
29 end

```



## 2.4. Captura de datos del lector

Para la captura de los datos del lector es necesario editar el archivo `jitro_ui /lib /jitro_ui_web /live /page_live.ex` y borrar las rutinas no utilizadas, quedando solo la rutina **mount** la cual editamos y permitimos la transferencia de un mapa que llamaremos `ultima_lectura` que tendrá la siguiente estructura:

```
1 %{
2     :lectura => cadena,
3     :direccion => direccion
4 }
```

El archivo el archivo `jitro_ui/lib/jitro_ui_web/live/page_live.ex` queda de la siguiente fomra:

```
1 defmodule JitroUiWeb.PageLive do
2   use JitroUiWeb, :live_view
3
4   @impl true
5   def mount(_params, _session, socket) do
6     if connected?(socket), do: Process.send_after(self(), :
7       actualizar, 100)
8
9     {:ok, assign(socket, resultado: %{ :lectura => "Aun no lee", :
10       direccion => "Sin direccion"})}
11   end
12
13   def handle_info(:actualizar, socket) do
14     Process.send_after(self(), :actualizar, 100)
15     lectura = JitroUi.CadenaActual.actual
16     {:noreply, assign(socket, :resultado, lectura)}
17   end
18 end
```

Asi mismo, es necesario modificar el archivo `jitro_ui /lib /jitro_ui_web /live /page_live.html.leex`

```
1 <section class="phx-hero">
2   <h1><%= gettext "Bienvenido a %{name}!", name: "Jitro" %></h1>
3   <p>El sistema de control de accesos de SMARTTEST</p>
4
5
6 </section>
7
8 <section class="row">
9   <article class="column">
```

*Departamento de I+D, SMARTTEST*  
Elaboró: Dr. Casimiro Gómez González

```
10 <h2>Lectura</h2>
11 <%= @resultado.lectura %>
12 </article>
13 <article class="column">
14 <h2>Direccion</h2>
15 <%= @resultado.direccion %>
16 </article>
17 </section>
```

Para que los **genservers** de lectura esten activos es necesario darlos de alta en el archivo `jitro_ui /lib /jitro_ui /application.ex` quedando de la siguiente forma:

```
1 ...
2 def start(_type, _args) do
3   children = [
4     JitroUi.CadenaActual,
5     JitroUi.LectorHid1,
6
7     # Start the Telemetry supervisor
8     JitroUiWeb.Telemetry,
9     # Start the PubSub system
10    {Phoenix.PubSub, name: JitroUi.PubSub},
11    # Start the Endpoint (http/https)
12    JitroUiWeb.Endpoint
13    # Start a worker by calling: JitroUi.Worker.start_link(arg)
14    # {JitroUi.Worker, arg}
15  ]
16 ...
```