



TWeb for Uhttpd2



Contenido

Preámbulo	4
Estructura proyecto	6
Base de un código con Tweb	7
Diseño de la pantalla	7
Configuración del servidor / rutas	8
Compilación de nuestra App Web.....	10
Nociones básicas.....	13
Frontend.....	13
Backend	14
Funcionamiento del sistema de rejillas.....	14
Dessign	17
Row vs RowGroup	20
Rows Valign/Halign.....	21
Controles.....	24
Browse – Control especial.....	26
Html	29
Html inline	29
Html File	32
Sistema grid responsive.....	34
Menús	38
FrontEnd – Functions Suport	41
Web flow	41
API	42
Creación de una API.....	45
Una API varios procesos.....	46
Javascript.....	49
MsgApi() → Frontend to Backend	50

SetJS() → Backend to Frontend	51
Advanced	52
Dialogs	52
Security	56
Tokens	56
Sessions	56
Charset	58
Web charset: 'ISO-8859-1'. Editor coding: ANSI	59
Web charset: 'UTF-8'. Editor coding: ANSI	60
Web charset: 'ISO-8859-1'. Editor coding: UTF-8	61
Web charset: 'UTF-8'. Editor coding: UTF-8	62
DBF's	62
DBF's con UTF8	65
MYSQL	66
CURL	70
Pluggins	72
SSL	74
Dominio	75
Certificado	75
App web con Uhttpd2	77
Código	77
Compilado / Enlazado	77
Test en localhost	78
Test en Internet	78
Pildoras	82
Conclusiones	83

Preámbulo

Cuando uno entra en el mundo Web empieza a ver, estudiar, probar numerosos entornos que permiten crear páginas web de mil maneras diferentes, eso sí con una curva de aprendizaje muy alta y un periodo de adaptación a la nueva manera de programar, más o menos largo.

Los que venimos de entornos de programación (especialmente Windows), nos hemos curtidos en mil batallas hasta aprender, asimilar y sentirnos cómodos con este sistema operativo. En muchos lenguajes, con los años hemos aprendido las clases a fondo de nuestro lenguaje de programación y nos ha permitido crear nuestras aplicaciones de gestión, y subrayo gestión porque básicamente es el modelo que más usamos en este entorno de Harbour. Así pues, si quiero dar el salto a la web, ¿necesito ser un experto en todos los lenguajes que tendré de usar para poder generar una aplicación típica de mantenimiento y gestión?

Este ha sido el objetivo de crear esta librería, el poder facilitar el salto a la Web de una manera fácil usando nuestra filosofía de trabajo actual. No pretendo con esta librería inventar la rueda, ni hacer magia pura, simplemente que su uso nos facilite la entrada poco a poco a la Web y perder el miedo ha desarrollar nuestras aplicaciones y especialmente nuestras pantallas, ofreciéndonos una transición más fácil y suave a todo este entorno. Muchos de los que usamos librerías de programación como [FWH](#) para Windows (la mejor librería de programación para win de Antonio Linares), al principio no conocíamos las clases, ni siquiera las entendíamos. Muchas estaban escritas en C, pero allí estaban, las usábamos y funcionaban.

Con el tiempo las hemos modificado, cambiado, ... e incluso hemos desarrollado de nuevas. Pues ahora se trata de lo mismo. Usar esta metodología, sentirnos cómodos y si nos vemos capaces, ir aumentando las diferentes funcionalidades. Entrar sin miedo en este nuevo entorno y empezar a diseñar estas pantallas Web con nuestra metodología que hemos aprendido y asimilado durante años al estilo xBase, con nuestros queridos comandos. TWeb se ha basado en esta librería madurando más el concepto hasta portarlo a la Web.

Con la entrada de modharbour cree la librería Mercury que te permite estructurar perfectamente un programa dentro de este entorno web usando arquitectura MVC. El sistema está muy estabilizado y va muy bien, pero nos quedaba una pata por acabar de definir y son las vistas, como poder trabajarlas

Cuando creabas las vistas (views) las realizamos en code html/css puro y duro y aquí es donde mucho de nuestros programadores hacían el alto. Por esto he creado esta versión Tweb para modharbour usando toda la potencia del preprocesador (algo muy añorado en php) que nos ayudara de una manera brutal al diseño de estas pantallas, trabajando de una manera muy amigable.

Pero no solo se trata de saber “pintar” estas pantallas, además hemos de entender cómo funciona la web, sus flujos, lo que podemos procesar desde el frontend, desde el backend,... Esta es la otra parte que Tweb nos ayuda junto a Uhttpd2 que será nuestro servidor web y está diseñado para poder trabajar de esta manera.

Este ha sido un proceso largo, de cerca de 3 años y que ha ido mutando en cada fase. Empecé con crearla para un entorno de php, y con el tiempo y gracias al diseño del primer mod las adapté para que funcionase y se

podieran crear pantallas responsive. Con la entrada de las versiones del mod 2.1 tuve que volver a readaptarlas porque si bien el funcionamiento era prácticamente el mismo, el sistema era diferente.

Ante la falta de interés y después de reflexión, quizás nos dimos cuenta que la gente de harbour no saltaba por toda la dificultad que tiene dar este salto a la web. Es por eso que rediseñe la librería uhttpd2 del amigo Mindaugas para que tuviera una manera especial de trabajar y facilitara de nuevo el intento de poder ayudar a la gente harbour a dar este salto.

En esta versión estarán disponibles los controles más básicos, pero está diseñado para poderlo escalar fácilmente a nuevos controles y funcionalidades. Usará el framework Bootstrap uno de los más populares actualmente en el mundo web. Ser un maestro en el "maquetado" de la web lleva muchos años de aprendizaje, Bootstrap sintetiza muchos aspectos, Tweb es una capa que lo intenta facilitar aún más.

No pretendo y ni es la idea enseñar html, JavaScript ni css, por lo que como mínimo el usuario debe conocer a un nivel muy básico el lenguaje. Intentaremos poder trabajar de una manera fácil, ágil y productiva.

TWeb es el resultado de muchas horas de investigación y de ver como encajan mejor las piezas para poder disponer de este sistema.

Cualquiera está invitado a sugerir, aportar y aumentar la funcionalidad y potencia del sistema con nuevas ideas, tips, clases, conceptos, ... Intento crear una base para poder empezar a trabajar abierta a todo el mundo.

Siempre he creído que la web es de todos y nosotros debemos decidir como poderla manejar, solo quiero ofrecer ahora a toda la comunidad harbour "otro" estilo y manera de poder realizar nuestros sueños en la programación web. Lo conseguiremos ☺

Un saludo a todos y que empiece la fiesta...

Estructura proyecto

Hemos de tener claro que una aplicación web la cantidad de ficheros es muy superior a la que estamos acostumbrados a usar en aplicaciones de tipo Windows. Es por eso que necesitamos estructurar bien nuestro proyecto, para en caso de ir escalándolo lo tengamos todo bien ordenado.

La manera más rápida de empezar un proyecto es copiar la carpeta `.\samples.tpl` a la carpeta donde se ubicará tu proyecto. La otra manera es copiar las carpetas del repositorio `.\dist\files` y `.\dist\lib` a la carpeta de tu proyecto añadiendo además las carpetas `.\myproject\app` y `.\myproject\html`



Si hemos creado el proyecto con una copia del template `\sample.tpl` también se habrá copiado el fichero de compilado y el hbp. Todo preparado y listos para compilar. Mi consejo es con un click copiar en tu carpeta de proyecto el template y en segundos ya lo tienes funcionando.

Carpetas	Descripción
app	Contiene los distintos prgs para crear la app
files	Conjunto de carpetas, por defecto uhttpd2 y tweb, con los ficheros necesarios para ejecutar la web en la cara-cliente, es decir pluggins de JavaScript, css,...
html	Ficheros html que usaremos en nuestra app
lib	Ficheros que usaremos de harbour para poder compilar nuestra aplicación y ficheros de cabecera que podemos usar.
Ficheros	
go64.bat	El fichero bat preconfigurado para poder compilar y enlazar nuestra app
app.hbp	Fichero de reglas de enlazado y compilado que usamos con el hbm2 de harbour

Base de un código con Tweb

Tweb necesita una estructura básica para poder funcionar correctamente. Se encargará de cargar librerías, configurar un entorno básico y poder ejecutar unas definiciones básicas para poder “pintar” correctamente nuestra pantalla y procesar los diferentes eventos de los controles para poder dar “vida” y hacerlos funcionar.

Diseño de la pantalla

Por norma general pondremos los ficheros en la carpeta \html de nuestro proyecto. Lo mejor es ver un caso práctico y analizar dicha estructura y para ello crearemos el fichero .\html\index.html

```
<?prg
#include "lib/tweb/tweb.ch"
    LOCAL o, oWeb

    DEFINE WEB oWeb TITLE 'TWeb'

    DEFINE FORM o OF oWeb

    INIT FORM o

        SAY VALUE '<h1>Welcome to TWeb !' ALIGN 'center' GRID 12 OF o

    ENDFORM o

    INIT WEB oWeb RETURN
?>
```

Vamos a definir línea por línea el contenido de este código

- Observemos que todo el código va entre 2 tags → <?prg ... ?> que le indica a nuestro servidor que todo lo que este entre los dos tags será código harbour. Esta manera de trabajar es análogo a la que utiliza php en la que a diferencia de aquí utiliza las etiquetas → <?php ... ?>
- #include "lib/tweb/tweb.ch"
Aquí cargamos nuestro fichero de preprocesado de tweb. Como hemos comentado este tipo de librerías las pondremos dentro de la carpeta .\lib
- Definiremos 4 clausulas obligatorias
 - o DEFINE WEB oWeb TITLE 'TWeb'

Esta línea se encarga de inicializar y configurar nuestra web. Mas adelante veremos con configurar diferentes aspectos de nuestras web: lenguaje, charset, titulo, icono,... Observemos que el comando

define una variable que llamamos oWeb (o como queramos) que nos dejara más adelante cambiar propiedades de la configuración

- DEFINE FORM o OF oWeb → Definición de cómo será nuestro formulario/pantalla
- INIT FORM o → Inicio de nuestro formulario/pantalla
- ENDFORM o → Final del diseño de nuestra pantalla

En la definición del form vemos que cargamos todo sobre la variable o.
Es el mismo concepto que la variable de definición de la web.

- INIT WEB oWeb RETURN → Final de nuestra Web

- Entre los comandos INIT FORM y ENDFORM iremos definiendo los diferente componentes de nuestra pantalla. En este caso podemos observar la línea:

```
SAY VALUE '<h1>Welcome to TWeb !' ALIGN 'center' GRID 12 OF o
```

Esta es la manera de definir los controles con Tweb. Dispondremos de una serie de comando, cada uno de ellos encargado de definirlos: SAY, GET, SELECT, RADIO,... ya los veremos más adelante. Observemos que definimos el control sobre la variable o, que es la que cargamos con toda la definición del formulario

Resumen:

- ✓ Definimos una web
- ✓ Definimos un formulario con sus controles
- ✓ Cerramos la web y el servidor la envía al navegador

Con esta parte hemos definido nuestra pantalla

Configuración del servidor / rutas

Nuestro proyecto Uhttpd2 como sabéis es un servidor que se encargará del proceso de todo lo necesario para nuestra web. El fichero principal llamado app.prg (o como queráis) definiremos este servidor y nuestras rutas. Estará ubicado en la carpeta \app


```
#define VK_ESCAPE 27

request TWEB

function main()

    hb_threadStart( @WebServer() )

    while inkey(0) != VK_ESCAPE
    end

retu nil

function WebServer()

    local oServer      := Httpd2()

    oServer:SetPort( 81 )

    //    Routing...

    oServer:Route( '/' , 'index.html' )

    //    -----//

    IF ! oServer:Run()

        ? "> Server error:", oServer:cError

        RETU 1

    ENDIF

RETURN 0
```

Es un ejemplo muy simple. En la función main() se ejecuta un hilo con el arranque del servidor que esta definido en la función WebServer(). El servidor funcionará hasta que no le demos a la tecla escape.

En la función Webserver() debemos ver los siguientes dos puntos

- Creación del servidor → oServer := Httpd2()
Como se puede ver, después podemos configurar propiedades del servidor usando el objeto creado oServer. En este caso y lo usaremos en todo el manual, todos los ejemplos los construyo siempre en pre-producción con el puerto 81, para evitar conflictos con otros posibles programas que tenga instalados, como podría ser un servidor php que por defecto va por el 80.

Así podemos ver que fácilmente podemos en este caso cambiar la configuración con:

```
oServer:SetPort( 81 )
```

➤ Rutas

Este apartado es muy importante ya que es el que dará autorización a nuestro servidor de escuchar las peticiones que se nos hace desde un navegador o cliente.

```
oServer:Route( '/', 'index.html' )
```

Aquí estamos definiendo que si el usuario pone una '/' en el navegador (es la dirección/dominio por defecto) el servidor ejecutará y devolverá el fichero index.html que ya sabe que lo habrá de buscar en la carpeta .\html

Esto es una primera barrera de seguridad. SOLO se servirá lo que definamos con rutas y podemos definir tantas como necesitemos.

Si necesitamos "compartir" o dejar visible alguna carpeta, p.e para que puedan leer ficheros desde el navegador, se ha de habilitar el método:

```
oServer:SetDirFiles( <folder> )
```

En este punto ya tenemos definido nuestro servidor (con sus rutas) y nuestra página html

Compilación de nuestra App Web.

Como hemos comentado en muchas ocasiones, todo esta versión está basada en el compilador MSVC 64 de Microsoft, por lo que se suponen que ya lo debéis tener instalado. Es importante haber aprendido a compilar bien UHttpd2. Podeis encontrar mas información aquí → <https://carles9000.github.io/?search=compilar>

Antes de compilar damos un vistazo al fichero go64.bat y que deberíamos revisar las rutas del compilador MVSC 64 que tengamos instalado y la ruta donde tengamos nuestro harbour

```
@echo off

if exist "%ProgramFiles%\Microsoft Visual
Studio\2022\Community\VC\Auxiliary\Build\vcvarsall.bat" call
"%ProgramFiles%\Microsoft Visual
Studio\2022\Community\VC\Auxiliary\Build\vcvarsall.bat" amd64

del app.exe

c:\harbour\bin\hbm2 app.hbp -comp=msvc64

if errorlevel 1 goto compileerror

del app.exp
del app.lib

@cls
```

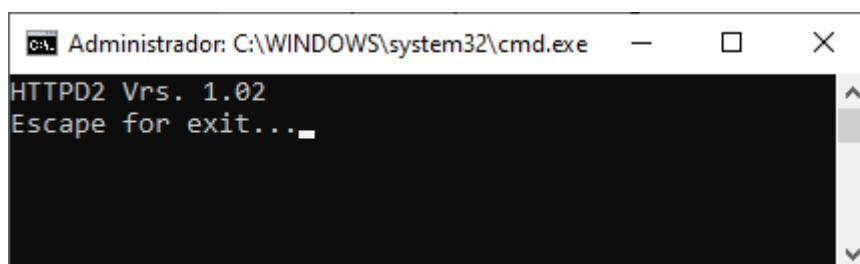
```
app.exe  
  
goto exit  
  
:compileerror  
echo *** Error  
  
pause  
  
:exit
```

Y solo queda ver el fichero app.hbp que es donde tenemos la configuración para el compilado

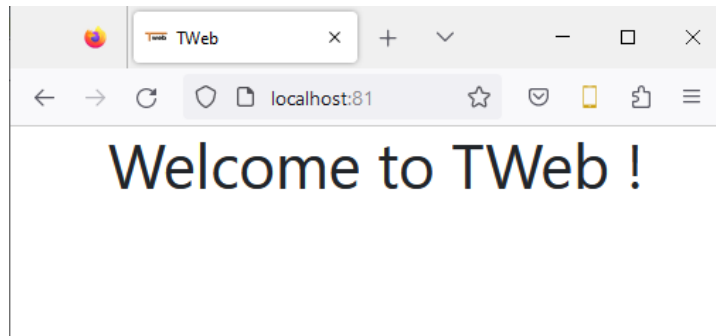
```
-n  
-w1  
-es2  
-inc  
-head=full  
-mt  
  
app\app.prg  
  
-Llib\uhtp2  
-luhtp2  
  
-Llib\tweb  
-ltweb  
  
# WAPI_OutputDebugString()  
hbwin.hbc  
xhb.hbc
```

La primera parte son los flags necesarios para compilar la aplicación. Luego indicamos el fichero/s que enlazaremos que en este caso es el app.prg. Finalmente observamos como forzamos el enlazado de las librerías uhtp2 y tweb. Las referencias del final son para el uso de uhtp2.

Ya solo queda compilar nuestro server y ponerlo en marcha. Ejecutamos go64.bat y nos debería aparecer una pantalla parecida a la siguiente:



Si aparece esta pantalla significa que el servidor esta funcionando. Si ahora nos vamos a un navegador y ejecutamos → localhost:81 nos debería aparecer la primera pantalla que hemos definido (index.html)



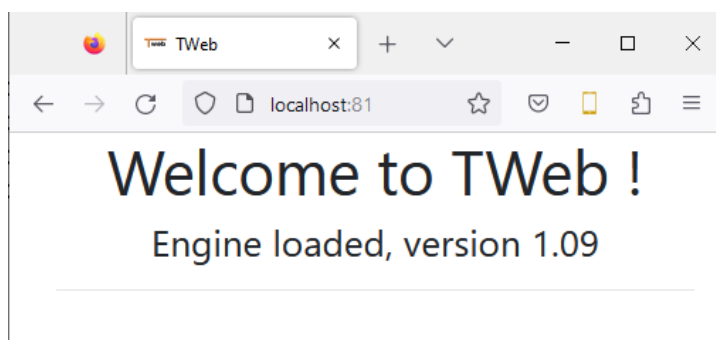
Podemos decir que ya tenemos la base de nuestro proyecto lista para funcionar con Uhttpd2 y Tweb !

Durante el manual iremos recordando bondades que tiene nuestro servidor como por ejemplo el poder usar macrosustitución usando {{ ... }}

Si cogemos el código index.html y añadimos la siguiente línea

SAY VALUE '<h4>Engine loaded, version {{ TWebVersion() }}<hr>' ALIGN 'center' GRID 12 OF o

El sistema evaluará automáticamente su contenido, en este caso devolviendo la versión de Tweb



Nociones básicas

Si bien el proyecto pretende ayudar a crear aplicación con pocos conocimientos dentro de la web, es muy importante que a medida que se pueda ir adentrándose y asimilar como mínimo de manera conceptual los diversos lenguajes y temas que se pueden encontrar en la web.

Es importante unos mínimos conocimientos en html, JavaScript, css, arquitectura, cliente, servidor, ...

Dejando al lado los numerosos puntos que con el tiempo si o si deberéis tocar si queréis hacer grandes cosas, me gustaría hacer paro en 2 conceptos que debéis por muy fáciles que parecen ser, asimilar bien: frontend y backend.

Porque esta es la base de una aplicación web y vamos a tener de programar en todos los lados y debemos saber situarnos muy bien dentro del escenario de juego.

Frontend

*“Como su nombre indica, el Frontend hace referencia a **la parte visible o interfaz de una página web o de un programa o aplicación móvil**; aquella que verán y interactúan los usuarios que accedan. Más específicamente, hace referencia a los **colores, letras, animaciones y demás elementos que pueden visualizar los usuarios**.*

*El objetivo de un desarrollador Frontend es **conseguir que la página sea intuitiva, funcional y estética para los usuarios**. Es, en definitiva, la parte del desarrollo de la que depende la calidad que el usuario percibe dentro de una página, programa o aplicación.*

El desarrollo Frontend se trabaja a través de tres lenguajes diferenciados: HTML, CSS y JavaScript. “

Los 3 Principales lenguajes Frontend en desarrollo web

1. HTML

*HTML sirve para **determinar la estructura de la página y del contenido mediante etiquetas**. Es imprescindible dominar el HTML para generar una página que se posicione correctamente en buscadores, aunque también sirve para hacerla más comprensible de cara a los usuarios. HTML tiene muchos tipos de etiquetas, cada una es para definir un segmento del código, ya sea funcional o no, ya que también hay etiquetas para insertar comentarios en HTML que no interfieren en la ejecución del código.*

2. CSS

Por otro lado, CSS es un lenguaje que tiene como objetivo **definir el estilo de la página de manera sencilla**. Todas las tareas que se llevan a cabo mediante el lenguaje CSS pueden llevarse a cabo editando el código HTML de una página, aunque hacerlo requerirá mucho más tiempo. El uso del CSS permite definir el estilo de una web en lote, **aplicando un estilo determinado a todas sus páginas de manera simultánea y, a su vez, permitiendo alterar elementos en masa**. Algunos ejemplos de estos elementos son las fuentes, el color de la letra, tamaño, imágenes de fondo...

3. JavaScript

Por último, JavaScript tiene como objetivo **permitir crear una página web interactiva**. HTML y CSS son lenguajes estáticos que no permiten añadir elementos que interactúen con otros más allá del uso de enlaces. Usar JavaScript en el diseño de una web **permite añadir botones interactivos, formularios y animaciones...** Pese a esto, también permite funcionalidades esenciales como la implementación de cookies, aunque actualmente existen otros lenguajes que también permiten crearlas. “

Backend

“El Backend hace referencia a **todos aquellos elementos de la página web, aplicación o programa, que son inaccesibles de cara a los usuarios**. Gestionar el Backend significa **crear la lógica, conectarla con el servidor y gestionar los datos o bases de datos**. También se encarga, en parte, de la experiencia del usuario a base de supervisar el rendimiento del frontend, **vigilando que la página no caiga, optimizando para que disponga de un mayor rendimiento...**”

Cita de → <https://assemblerinstitute.com/blog/backend-vs-frontend/>

Funcionamiento del sistema de rejillas

Si bien es cierto que Tweb está diseñado para utilizar Bootstrap, no es especialmente obligatorio. Bootstrap utiliza un sistema de rejillas para poder diseñar las pantallas. Hay mucha documentación acerca de cómo funciona a nivel muy detallado, pero básicamente imaginaros una pantalla formada por líneas y 12 columnas. En cada línea podemos poner nuestros controles que ocuparan tantas columnas como sea necesario de estas 12. Podemos por ejemplo crear una línea (row) y esta línea dividirla en 3 partes. Una estará formada por 6 columnas, otra por 4 y otra por 2 columnas.

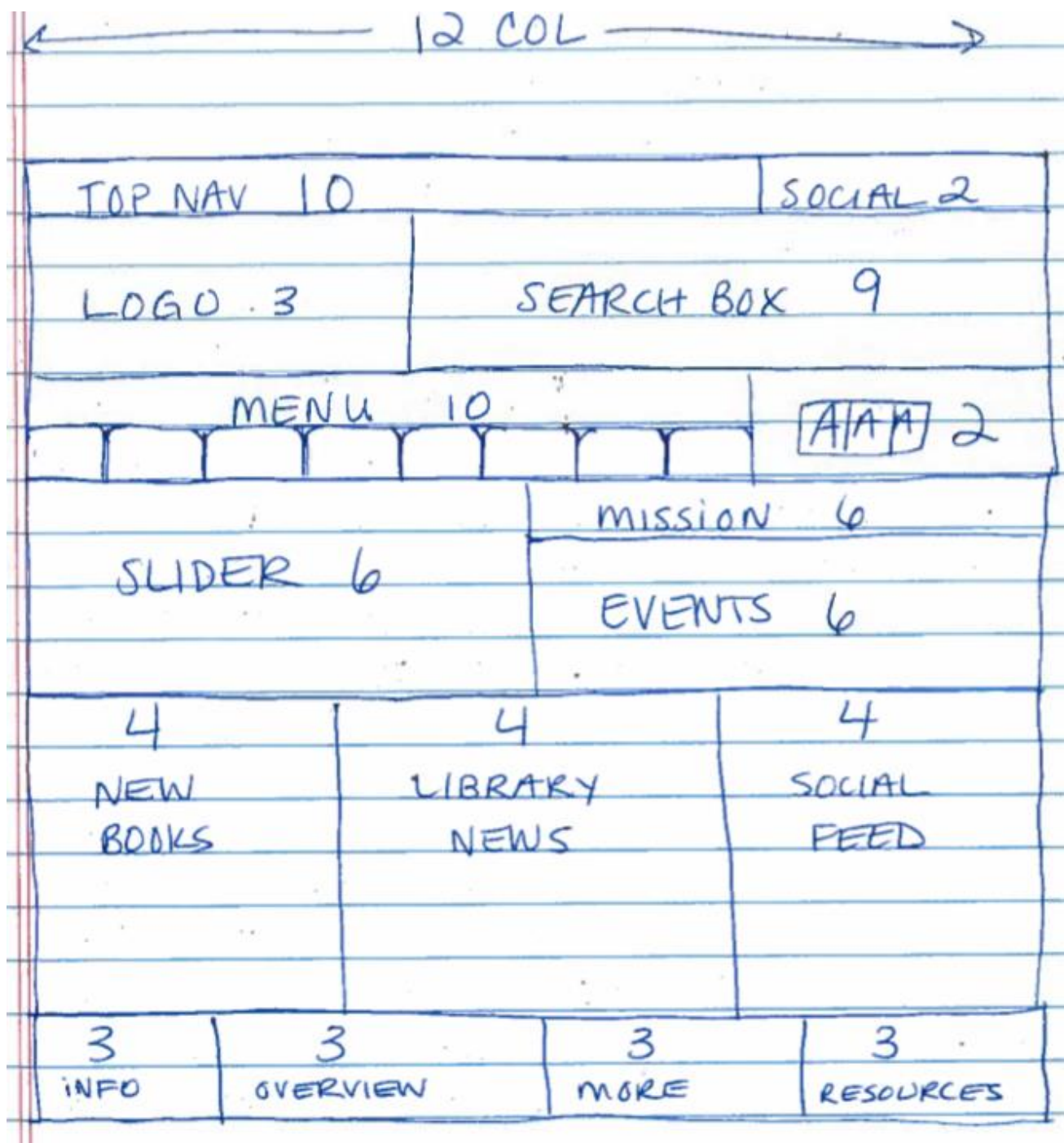
No es necesario utilizar las 12, pero estas 12 forman todo el ancho de la columna. Cada columna será como un nuevo espacio, que podremos insertar más líneas y 12 columnas...

Será el sistema Bootstrap quien se encargará de distribuir en estos espacios los diferentes controles, adaptándolos a las dimensiones de cada pantalla. Bootstrap tiene muchas funcionalidades para poder adaptar de muchas maneras nuestras pantallas, y conocerlo a fondo es tarea de mucho estudio. Tweb intenta

minimizar este impacto inicial, ayudándote de una manera inicial a crear rápidamente estas pantallas. Con el tiempo se ira asimilando estas técnicas y se podrán combinar las funcionalidades de Tweb con código nativo html/css puro y duro.



Llevado a la práctica y mirando de crear una pantalla a lápiz y papel (mockup) podríamos crear un ejemplo para entender cómo se encajan las diferentes partes en esta rejilla



Es decir, a partir de ahora habremos de pensar en clave 12 columnas, se acabó pensar en posiciones fijas de la pantalla 😊

Dessign

Lo primero que vamos a hacer es a diseñar un pequeño código y ver como Tweb nos ayuda a entender cómo trabaja bootstrap a la hora de diseñar nuestras pantallas. Todos los códigos están probados y se encuentran en la carpeta .\sample\html\tutor

tutor1.prg

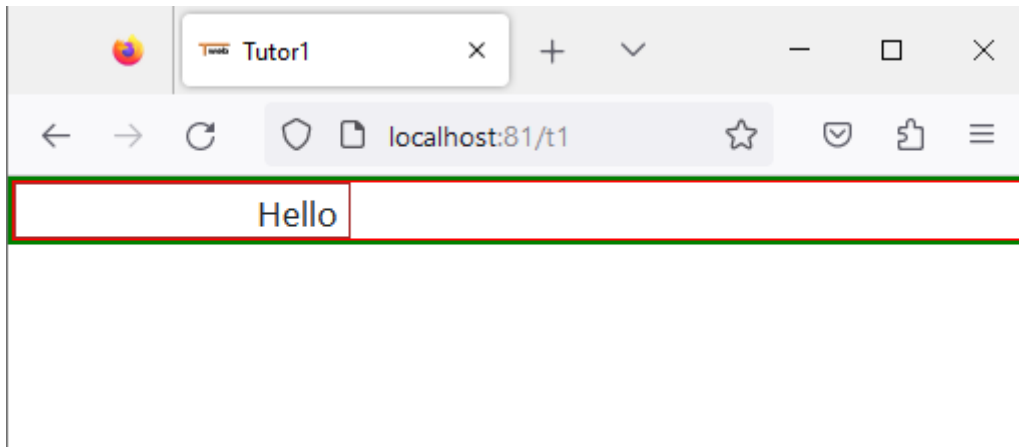
<pre><?prg #include "lib/tweb/tweb.ch" LOCAL o, oWeb DEFINE WEB oWeb TITLE 'Tutor1' DEFINE FORM o OF oWeb o:lDessign := .T. o:lFluid := .T. INIT FORM o ROW o SAY VALUE 'Hello' ALIGN 'right' OF o ENDROW o ENDFORM o INIT WEB oWeb RETURN ?></pre>	<p>Incluimos fichero de preprocesado de comandos de Tweb</p> <p>Definimos una web</p> <p>Definimos un formulario Asignamos .T. a la propiedad lDessign Asignamos .T. a la propiedad fluid</p> <p>Inicializamos el formulario</p> <p>Creamos una línea contenedora Pintamos con un control SAY "Hello" Finalizamos una línea</p> <p>Fin de formulario</p> <p>Fin de la Web</p>
---	---

Este es uno de los objetivos Tweb -> Inicializar el sistema y pintar controles !!!

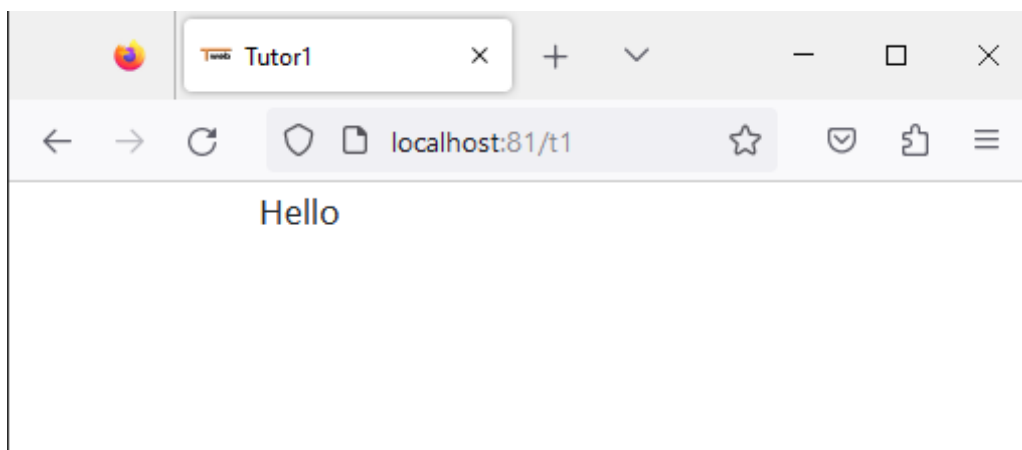
El objetivo principal es aprender a pintar la pantalla responsive. Otro tema muy importante y que vamos a seguir en este manual es el concepto de las 12 columnas de Bootstrap y por donde dibujamos nosotros nuestros controles. Es por esta razón que cuando definimos nuestro formulario podemos especificar la propiedad →IDesign = .T. que nos pintará los bordes de todo lo que especifiquemos para ver (y aprender) por donde se mueve el sistema Bootstrap 😊

A medida que vayamos poniendo controles veremos un montón de líneas (siempre que tengamos IDesign == .T.) y esto nos ayudará a entender cómo funciona el pintado (que es muy complejo en muchos casos).

Este código simple, inicializa el sistema, carga las libs, y pintamos un control say, todo en modo diseño. El resultado sería el siguiente



Como se puede observar iremos marcando las líneas para aprender como el sistema dibuja todo. Si ejecutáramos con la IDEsign := .F. podemos observar la diferencia de pintado. Lo quiero recalcar porque iré trabajando en modo diseño. En cualquier momento podéis sacar esta propiedad y ver el resultado limpio de líneas.

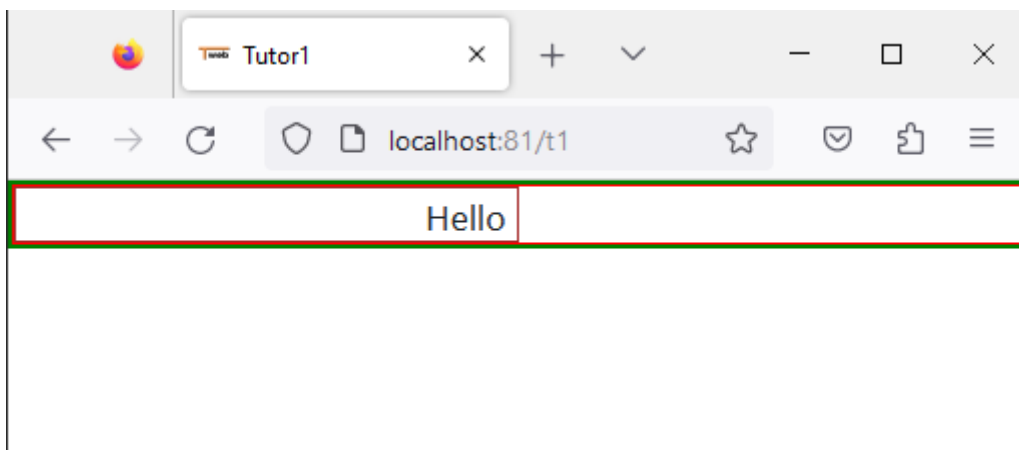


Podemos observar que realmente nos pinta, pero si no podemos entender porque en esa posición no aprenderemos como realmente actúa Bootstrap

Continuando con este primer ejemplo podemos observar una línea verde que nos marca la delimitación del formulario (DEFINE FORM) y la delimitación del objeto say (SAY value "Hello"). Por defecto los controles ocuparan 4 columnas y es lo que está marcando el recuadro rojo que vendrá a ser 1/3 parte del ancho.

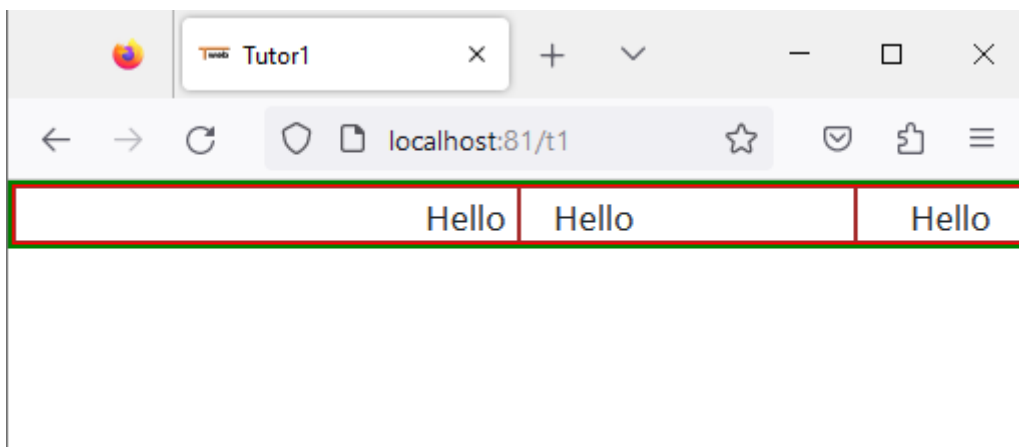
A medida que vayamos avanzando iremos aprendiendo el manejo de los comandos xBase y el posicionamiento de los diferentes controles. Por ejemplo, cada control tiene una cláusula GRID <nCols> que indicamos cuanto queremos que nos ocupe el control en la línea (recordad que tenemos 12 columnas). Si p.e ponemos en este ejemplo

```
SAY VALUE 'Hello' ALIGN 'right' GRID 6 OF 0
```



Imaginemos para acabar en este ejemplo que queremos poner 3 controles SAY que ocupen cada uno de ellos p.e 6,4 y 2 columnas (observad que suma 12) , alineados derecha, izquierda, centro

```
ROW 0  
    SAY VALUE 'Hello' ALIGN 'right' GRID 6 OF 0  
    SAY VALUE 'Hello' ALIGN 'left'  GRID 4 OF 0  
    SAY VALUE 'Hello' ALIGN 'center' GRID 2 OF 0  
ENDROW 0
```



Solo queda hablar en este primer ejemplo de la propiedad `lFluid := .T.`. Esta propiedad simplemente va ajustando todo el contenido de la pantalla a su tamaño. Si redimensionamos la pantalla vemos como se adapta, mientras que si está a `.F.` adquiere unas dimensiones pre-programadas por bootstrap que se adapta en función del tamaño del dispositivo. Son 2 maneras diferentes de hacer funcionar la pantalla.

Row vs RowGroup

Hemos visto que el comando nos crea una línea nueva. Podemos crear tantas líneas como queramos, pero el comando `ROW` nos crea una línea pegada a la siguiente, mientras que si usamos `ROWGROUP` deja un pequeño margen en la parte inferior. Vemos la diferencia de usar uno u otro con este code.

```
<?prg
#include "lib/tweb/tweb.ch"

LOCAL o, oWeb

DEFINE WEB oWeb TITLE 'Tutor3'

    DEFINE FORM o OF oWeb
        o:lDessign := .T.

    INIT FORM o

        ROWGROUP o
            SAY VALUE 'Id:' ALIGN 'right' OF o
            GET VALUE '123' OF o
        ENDROW o

        ROWGROUP o
            SAY VALUE 'Phone:' ALIGN 'right' OF o
            GET VALUE '' OF o
        ENDROW o

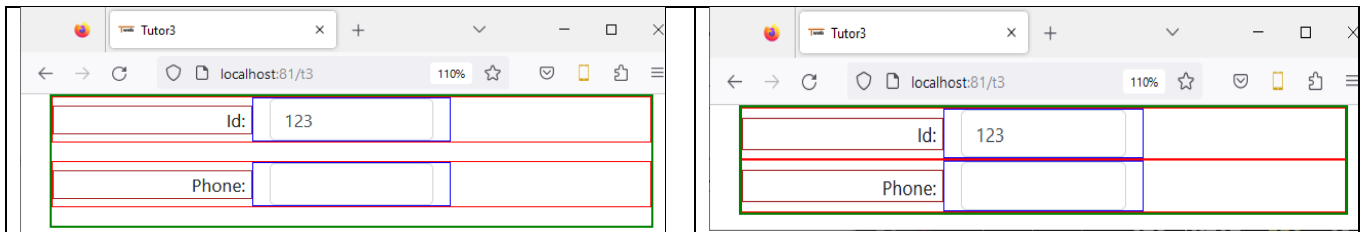
    ENDFORM o

    INIT WEB oWeb RETURN

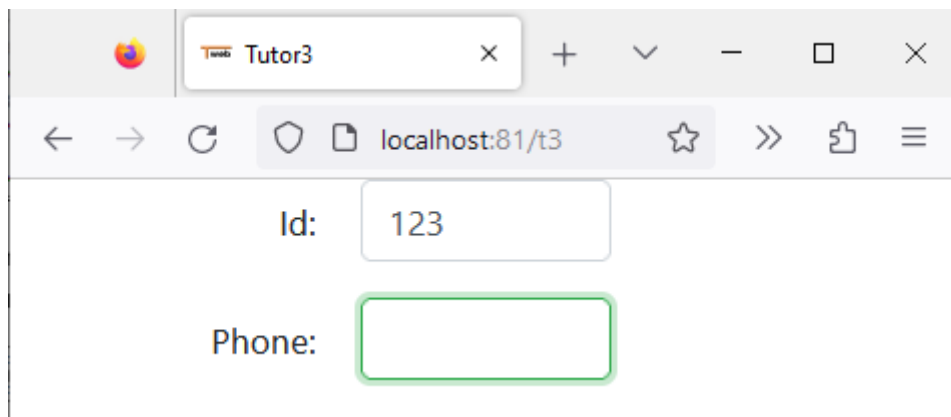
?>
```

En este pequeño código podemos observar que ponemos 2 bloques de `ROWGROUP` y cada uno tiene 2 controles, en este caso un `SAY` y un `GET`

Si hiciéramos el mismo ejemplo usando `ROW` en lugar de `ROWGROUP` no existiría el espacio entre líneas, mostrándolo todo más junto



Recordad que iremos mostrando las pantallas en este tutorial de maquetado con Tweb en modo IDesign. En cualquier momento podéis cambiar la propiedad IDesign a .F. para poder comprobar como quedaría realmente.



Rows Valign/Halign

Cuando creamos una línea (row) podemos poner varios controles, textos, ... y cada uno puede ocupar más espacio que otro ya sea verticalmente u horizontalmente. Por defecto una row alinea en la parte superior ('top'), pero podemos indicar que se alinea verticalmente en el 'center' o en la parte inferior ('bottom') con la cláusula VALIGN <cValue>

```
<?prg
#include "lib/tweb/tweb.ch"
#define IS_DESSIGN .T.

LOCAL o, oWeb, oSelect
LOCAL aTxt := { 'Volvo', 'Seat', 'Renault' }
```

```

LOCAL aKey := { 'V', 'S', 'R' }

DEFINE WEB oWeb TITLE 'Tutor 7'

  DEFINE FORM o OF oWeb
    o:lDessign := IS_DESSIGN
    o:cSizing  := 'sm'           // SM/LG

  INIT FORM o

    ROWGROUP o VALIGN 'top'      // Default
      SEPARATOR o LABEL 'Align top'
      SELECT oSelect LABEL 'Cars' PROMPT aTxt VALUES aKey GRID 6 OF o
      SELECT oSelect              PROMPT aTxt VALUES aKey GRID 6 OF o
    ENDROW o

    ROWGROUP o VALIGN 'center'
      SEPARATOR o LABEL 'Align center'
      SELECT oSelect LABEL 'Cars' PROMPT aTxt VALUES aKey GRID 6 OF o
      SELECT oSelect              PROMPT aTxt VALUES aKey GRID 6 OF o
    ENDROW o

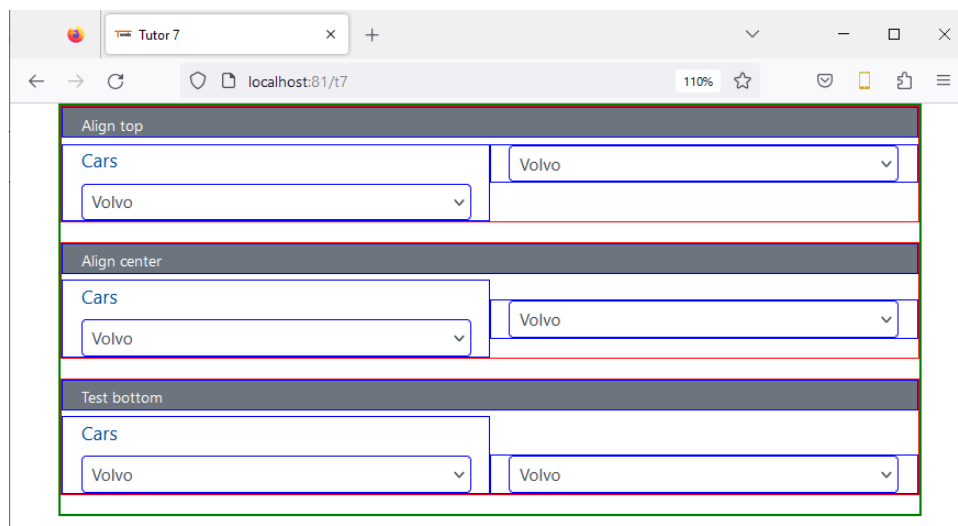
    ROWGROUP o VALIGN 'bottom'
      SEPARATOR o LABEL 'Test bottom'
      SELECT oSelect LABEL 'Cars' PROMPT aTxt VALUES aKey GRID 6 OF o
      SELECT oSelect              PROMPT aTxt VALUES aKey GRID 6 OF o
    ENDROW o

  ENDFORM o

INIT WEB oWeb RETURN

```

?>



De la misma manera podemos alinear a nivel horizontal, por defecto se alinea por la izquierda usando la cláusula HALIGN

tutor8.prg

```
<?prg
#include "lib/tweb/tweb.ch"
#define IS_DESSIGN      .T.

LOCAL o, oWeb

DEFINE WEB oWeb TITLE 'Tutor 8'

  DEFINE FORM o OF oWeb
    o:lDessign := IS_DESSIGN
    o:lFluid   := .T.

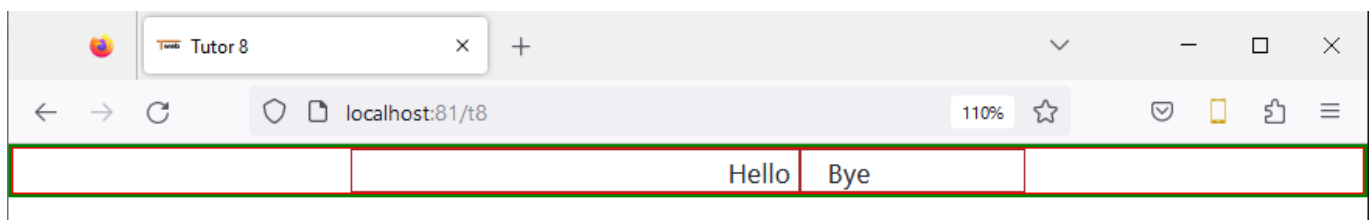
  INIT FORM o

    ROW o HALIGN 'center'
      SAY VALUE 'Hello' GRID 4 ALIGN 'right' OF o
      SAY VALUE 'Bye'   GRID 2 OF o
    ENDROW o

  ENDFORM o

INIT WEB oWeb RETURN

?>
```



Podéis ver que tenemos un control de 4 y otro de 2, total 6 alineados horizontalmente en la row por el centro

Siempre que definimos un formulario internamente se trataran los controles con un tamaño normal por defecto, per podemos especificar que se traten con un tamaño pequeño o grande. Para ello utilizaremos la cláusula cSizing := 'sm' para pequeños (small) o cSizing := 'lg' para grandes (large)

```
<?prg
#include "lib/tweb/tweb.ch"
#define IS_DESSIGN .T.

LOCAL o, oWeb

DEFINE WEB oWeb TITLE 'Tutor 9'

DEFINE FORM o OF oWeb
o:lDessign := IS_DESSIGN
o:cSizing := 'sm' // SM/LG (small/large)

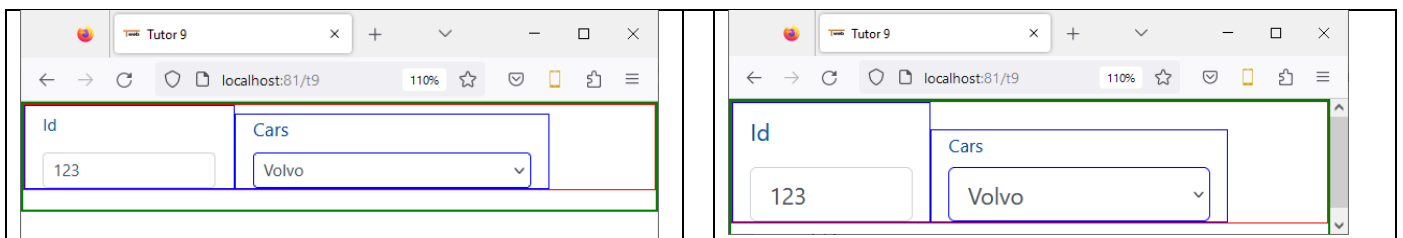
INIT FORM o

ROWGROUP o VALIGN 'bottom'
GET VALUE '123' LABEL 'Id' OF o
SELECT LABEL 'Cars' PROMPT 'Volvo', 'Renault' VALUES 'V', 'R' GRID 6 OF o
ENDROW o

ENDFORM o

INIT WEB oWeb RETURN

?>
```



Controles

Llamaremos controles a los diferentes inputs, etiquetas y componentes del DOM usando sintaxis xBase para un uso más familiar y amigable. Así pues, tenemos los siguientes controles:

SAY, GET, BUTTON, CHECKBOX, FOLDER, FORM, MEMO, RADIO SELECT, SWITCH, BROWSE,...

Muchos de estas clases comparten propiedades que son comunes a todas y otras tienen propiedades propias, pero básicamente el conocimiento de un control, ayuda a entender las demás.

Indirectamente ya hemos visto en los apartados anteriores una breve pincelada de cómo funcionan los controles, los cuales los definimos y especificamos a que contenedor están ubicados.

Cuando creamos un control, este llevara un ID para identificarlo. No es obligatorio pero necesario si queremos interactuar posteriormente por ejemplo con javascript, pues es el id que identifica el elemento en el dom. Un ejemplo seria este:

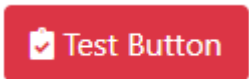
```
GET ID 'myid' LABEL 'Id.' VALUE '' GRID 6 OF oForm
```

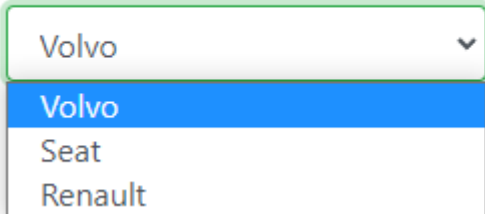
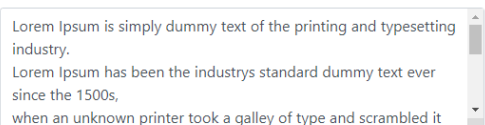
Los parámetros pueden ser opcionales o obligatorios dependiendo de cada tipo de control pero siempre tendremos de especificar el control a que contenedor o ámbito pertenece, en este caso <oForm>, por lo que la cláusula OF <oContenedor> siempre será obligatoria.

Un control por defecto tendrá un ancho de 4 columnas pero con la cláusula GRID <n> podremos variar. Muchos de los controles tienen cláusulas idénticas para su mejor uso. A continuación de describen las mas comunes

Cláusula	Descripción
ID <id>	Identificador del control
GRID <n>	Numero de las 12 columnas que se van a tomar el control
OF <contenedor>	Referencia al contenedor (Obligatorio)
ONCHANGE	Evento de conexión con el API. Se verá más adelante
ALIGN <cAlign>	Algunos controles pueden llevar esta cláusula de alineamiento horizontal: left, center, right
FONT <cFont>	Objeto Font que usamos para definir la fuente del control
CLASS <cClass>	Regla CSS que se añadirá al control en caso de querer usarla
VALUE <uValue>	Valor
DISABLED	Inicializamos el control desactivado
HIDDEN	Inicializamos el control oculto
STYLE <cStyle>	Permite especificar un style al control → 'border: 1px solid gray;margin: 10px;padding: 10px;background-color: lightgray;'
PROP <cProp>	Permite especificar una propiedad al control → maxlength="10"

Para poder ver una primera manera aproximada de cómo se debe codificar algunos controles, aquí hay algún pequeño ejemplo para podernos hacer una idea.

Controles	Código
<p>Identificador</p> <input type="text" value="11"/>	<pre>GET ID 'myid' VALUE '11' GRID 6 PLACEHOLDER 'Id.' BUTTON '<i class="fas fa-search"></i>' ACTION 'GetId()' OF o</pre>
	<pre>BUTTON ID 'btn' LABEL ' Test Button' ACTION 'alert(123)' GRID 4 ICON '<i class="fas fa- clipboard-check"></i>' CLASS 'btn-danger btnticket' OF o</pre>

<input checked="" type="checkbox"/> Ready	SWITCH ID 'onoff' VALUE .T. LABEL 'Ready' OF o
Cars 	SELECT oSelect ID 'cars' LABEL 'Cars' PROMPT 'Volvo', 'Seat', 'Renault' VALUES 'V', 'S', 'R' GRID 6 ONCHANGE 'Select()' OF o
<input checked="" type="checkbox"/> Is Active ?	CHECKBOX oCheck ID 'active' LABEL 'Is Active ?' GRID 3 ON ACTION 'active' OF o
<input checked="" type="radio"/> Soltero <input type="radio"/> Casado	RADIO oRadio ID 'status' PROMPT 'Soltero', 'Casado' VALUES 'S', 'C' GRID 6 INLINE ONCHANGE 'status' OF o
Loren Ipsum 	GET oMemo MEMO ID 'memo' LABEL 'Loren Ipsum' VALUE cTxt ROWS 5 GRID 12 ONCHANGE 'showmemo' OF o

La mejor manera de ver las posibilidades de manejar cada control es probar las decenas de ejemplos que se encuentran en las carpetas samples de Tweb. Es posible que se vayan aumentando o modificando las diferentes clausulas por lo que siempre viene bien también echarle un vistazo al fichero de pre-procesado \lib\tweb.ch donde están definidos todos los comandos de todos los controles.

El control más complejo de usar a la vez que potente es el browse, que se comentará en un capítulo a parte y se basa en el uso del plugin Tabulator, y es el que viene también integrado con la librería Uhttpd2 por defecto.

Browse – Control especial

El Browse o grid como llaman muchos es quizás el control más especial por sus múltiples funcionalidades, opciones, combinaciones,... Es a la vez muy complicado tener un control total del browse y muchas veces es posible quedarnos en el camino de cosas que nos gustaría hacer y no sabemos cómo.

Y si a eso le añadimos que no queremos tocar html, js, css, pues más complejo.

Tweb ha elegido usar el browse integrado en Uhttpd2 que se basa en el plugin Tabulator, un fantástico plugin que nos permite edición en línea y hacer muchas cosas increíbles

<https://tabulator.info/>

La idea que creo que es la mejor, es encapsular en Tweb la definición del browse como control y usar unas pocas funciones básicas que nos ayudaran a manejarlo. Esto, juntamente con la posibilidad de poder quien quiera añadir diferentes funcionalidades a medida que se necesiten vía js o html, permitirá escalar muy bien.

Pero básicamente a nivel de codificación para hacernos una idea, el browse se basa en dos partes obligatorias y una opcional:

- Configuración del browse (opcional)
- La definición del propio browse
- La definición de las columnas y propiedades del browse

Y evidentemente el comportamiento a nivel de “pintado” es igual que cualquier otro control.

```
<?prg
#include "lib/tweb/tweb.ch"

LOCAL o, oWeb, oBrw, aOptions
LOCAL aData := {
    { 'FIRST' => 'Homer'          , 'LAST' => 'Hoaks', 'AGE' => 38 },,
    { 'FIRST' => 'Jean-Piere'    , 'LAST' => 'Thain', 'AGE' => 58 },,
    { 'FIRST' => 'Steve'         , 'LAST' => 'Staro', 'AGE' => 26 } ;
}

DEFINE WEB oWeb TITLE 'Browse'

HTML oWeb FILE 'tpl\header_brw.tpl' PARAMS 'Basic'

DEFINE FORM o OF oWeb

INIT FORM o

    ROW o

        // https://tabulator.info/docs/5.4/options

        aOptions := { ;
            'height' => '300px', ;
            'index' => 'id' ; // Default id
        }

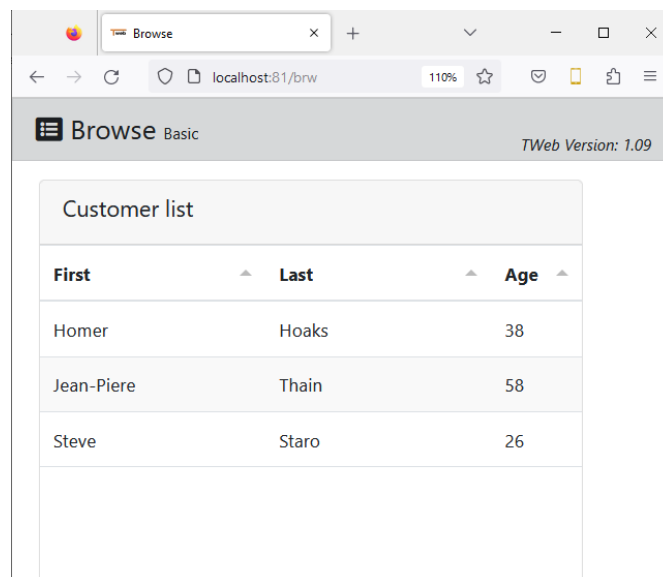
        DEFINE BROWSE oBrw ID 'mytable' OPTIONS aOptions ;
            TITLE '<h5>Customer list</h5>' OF o

            // https://tabulator.info/docs/5.4/columns

            COL oCol TO oBrw CONFIG {'title' => "First", 'field' => "FIRST" }
            COL oCol TO oBrw CONFIG {'title' => "Last", 'field' => "LAST" }
            COL oCol TO oBrw CONFIG {'title' => "Age", 'field' => "AGE" }
```

```
INIT BROWSE oBrw DATA aData  
  
ENDROW o  
  
ENDFORM o  
  
INIT WEB oWeb RETURN  
?>
```

Este simple code ya nos permite mostrar una simple tabla con los datos que tengamos definidos en un array.



The screenshot shows a web browser window titled 'Browse' with the address bar displaying 'localhost:81/brw'. The browser shows a table titled 'Customer list' with three columns: 'First', 'Last', and 'Age'. The table contains three rows of data: Homer Hoaks (38), Jean-Piere Thain (58), and Steve Staro (26). The browser interface includes standard navigation buttons and a zoom level of 110%.

First	Last	Age
Homer	Hoaks	38
Jean-Piere	Thain	58
Steve	Staro	26

A partir de aquí es empezar a volar y como siempre, practicar...

En los ejemplos de la carpeta .\samples hay un capítulo dedicado al manejo del browse:

- Basic example
- Load Data from backend
- Format Cell
- Select Rows
- Movable rows & cols
- Quick Print
- Add events
- Browse Styles
- Filtering
- Order/Detail event change
- Edit - CRUD
- Edit & save on-fly

Html

Html inline

TWeb nos permite mezclar nuestro código xBase con código normal y fácilmente podemos combinarlo usando el comando HTML. Tenemos 2 variantes:

- Usar en una misma línea un pequeño código → HTML o INLINE '<h3>Test Form</h3><hr>'
- Insertar todo el código que queramos y al finalizar poner el comando ENDTEXT

```
<?prg
#include "lib/tweb/tweb.ch"

LOCAL o, oWeb

DEFINE WEB oWeb TITLE 'Tutor4'

DEFINE FORM o OF oWeb
o:lDessign := .T.

HTML o INLINE '<h3>Test Html</h3><hr>'

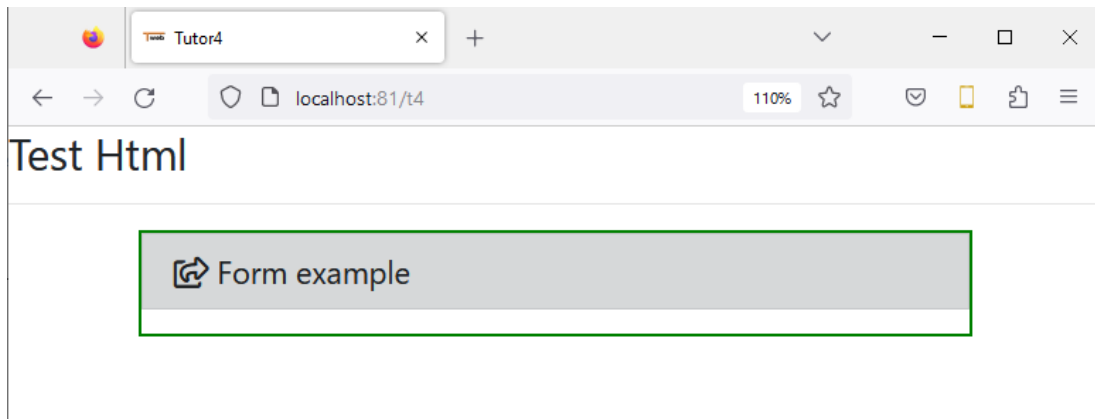
INIT FORM o

HTML o
<div class="row alert alert-dark form_title" role="alert">
<h5 style="margin:0px;">
<i class="far fa-share-square"></i>
Form example
</h5>
</div>
ENDTEXT

ENDFORM o

INIT WEB oWeb RETURN
?>
```

De esta manera fácilmente podemos poner el código html/css/JavaScript dentro de nuestro diseño , dándonos el control total sobre la página.



Una de las ventajas de usar Tweb es que es una librería independiente para harbour que no necesita el uso de otros programas para generar nuestras páginas, multiplataforma (Windows/Linux) y no necesita compilaciones. Escribir código y ejecutar, no hay más. Otros frameworks muestran un abanico de opciones para poder ver las propiedades de cada elemento del DOM, pero creo que la mejor manera es usar el propio inspector que lleva el propio navegador para ir probando, modificando, ... y luego aplicar los cambios directamente en nuestro código.

Quizás es un nivel más pero a medida que vayáis avanzando seguro usareis el inspector (porque además es vital), pero de esto ya se hablará en otro momento...

En este punto del manual ya debemos entender como iremos definiendo nuestras pantallas. Este pequeño ejemplo podemos observar cómo vamos mezclando los diferentes conceptos, añadiendo rows, controles, código html puro y duro,...

tutor5.prg

```
<?prg
#include "lib/tweb/tweb.ch"
#define IS_DESIGN .T.

local o, oWeb
local cLoren := "<h2>Why do we use it?</h2>It is a long established fact
that a reader will be distracted by the readable content of a page when looking at
its layout. The point of using Lorem Ipsum is that it has a more-or-less normal
distribution of letters, as opposed to using 'Content here, content here', making
it look like readable English. Many desktop publishing packages and web page
editors now use Lorem Ipsum as their default model text, and a search for 'lorem
ipsum' will uncover many web sites still in their infancy. Various versions have
evolved over the years, sometimes by accident, sometimes on purpose (injected
humour and the like)."
```

```
DEFINE WEB oWeb TITLE 'Tutor5'
```

```
DEFINE FORM o OF oWeb
  o:lDessign := IS_DESIGN
  o:lFluid   := .f.

  HTML o INLINE '<h3>Test Form</h3><hr>'

  INIT FORM o

    HTML o
      <div class="row alert alert-dark form_title" role="alert">
        <h5 style="margin:0px;">
          <i class="far fa-share-square"></i>
          Form example
        </h5>
      </div>
    ENDTEXT

    ROWGROUP o
      SAY VALUE 'Id:' ALIGN 'right' OF o
      GET VALUE '' OF o
    ENDROW o

    ROWGROUP o
      SAY VALUE 'Phone:' ALIGN 'right' OF o
      GET VALUE '' PLACEHOLDER "Enter phone number" OF o
    ENDROW o

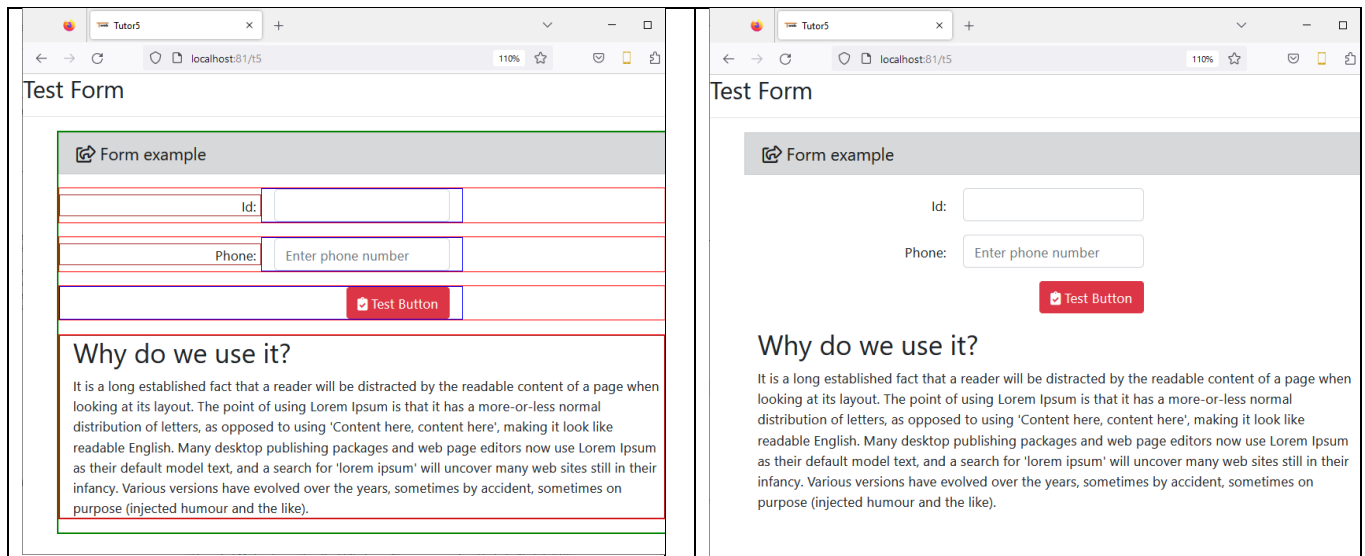
    ROWGROUP o
      BUTTON LABEL ' Test Button' ACTION "alert( 'Hi!' )" GRID 8 ;
      ALIGN 'right' ICON '<i class="fas fa-clipboard-check"></i>' ;
      CLASS 'btn-danger btnticket' OF o
    ENDROW o

    ROWGROUP o
      SAY VALUE cLoren GRID 12 OF o
    ENDROW o

  ENDFORM o

  INIT WEB oWeb RETURN
```

?>



Html File

Esta segunda manera de procesar código html es muy poderosa y ya se enmarca en usuarios más avanzados. Se trata de utilizar el concepto Views. Podemos cargar templates prediseñados que usamos continuamente y que nos permite optimizar mucho el código.

Si el hecho de que 10 páginas de nuestra aplicación cargan el mismo template:

- Evitamos tener que mantener el código de cada página
- Solo modificando el template afecta directamente a las diferentes páginas. La productividad aumenta

Los templates también se encontraran dentro de la carpeta `.\html` y por defecto están en otra subcarpeta llamada `.\tpl`. Un ejemplo básico de uso seria el siguiente.

```
<?prg
#include "lib/tweb/tweb.ch"

local oWeb
local a := 'My App'
local b := '(Vrs.1.23b)'
local c := 'Charly'

DEFINE WEB oWeb TITLE 'Tpl 1'
```



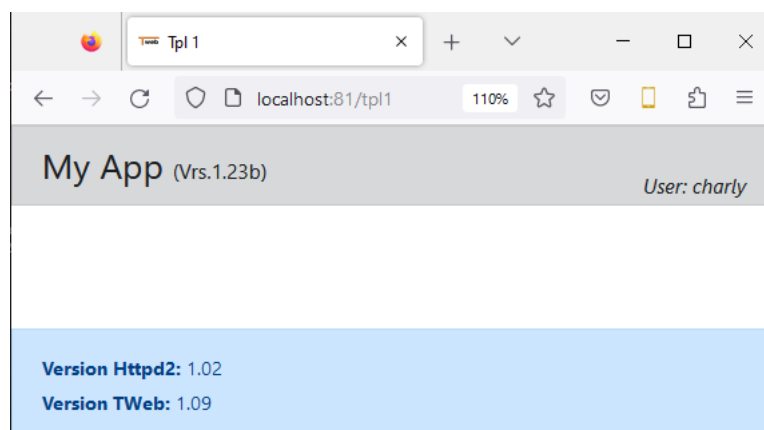
```
HTML oWeb FILE 'tpl\header.tpl' PARAMS a, b, c  
HTML oWeb FILE 'tpl/footer.tpl'
```

```
INIT WEB oWeb RETURN  
?>
```

Aquí usamos 2 templates, uno para crear una cabecera y otro para un pie de página. Se puede observar que en el primero usamos la cláusula PARAMS y nos permite pasar parámetros a nuestro template. El código del template podría ser algo similar a este:

```
<div class="alert alert-dark form_title" role="alert">  
  <h4 style="margin:0px;">  
    {{ pvalue(1) }}  
    <span style="font-size:14px;">{{ pvalue(2) }}</span>  
    <span style="font-size:14px;float: right;margin-top:20px;font-style:  
italic;">User: {{ pvalue(3) }}</span>  
  </h4>  
</div>
```

Quien quiera usar código nativo y crear templates es tan fácil como insertar su código y entre {{ ... }} recuperar si es necesario los parámetros recibidos.



Podemos complicar estas views tanto como queramos hasta el punto de crear módulos que usamos siempre en nuestras aplicaciones como p.e. llamar a una pantalla de login.

Sistema grid responsive

Hasta ahora hemos visto como hemos de combinar con estos pocos comandos para situar nuestros controles en pantalla y que se adapte bien. Nos queda ver la manera en que podemos controlar (si queremos o necesitamos) las columnas a medida que redimensionamos la pantalla. Para ello usaremos este código para poder entender este capítulo.

Definiremos una row y dentro de ella pondremos 2 columnas con texto cada una de ellas de 6 de ancho. La idea es que se se distribuyen a lo ancho de la pantalla, pero queremos controlar a la hora de redimensionar la pantalla cuando la hacemos más pequeña (o se use estas pantallas en una table o smartphone) que una columna se posicione debajo de la otra para poder ver mejor la información.

Tutor10.prg

```
<?prg
#include "lib/tweb/tweb.ch"
#define IS_DESSIGN .T.

LOCAL o, oWeb, oGet
LOCAL cTxt := ''

DEFINE WEB oWeb TITLE 'Tutor10'

DEFINE FORM o OF oWeb
o:lDessign := .F.
o:cType := 'md' // xs,sm,md,lg
o:cSizing := 'sm' // sm,lg

HTML o
<div class="alert alert-dark form_title" role="alert">
<h5 style="margin:0px;">
<i class="far fa-share-square"></i>
Test Memo (Resizing screen...)
</h5>
</div>
ENDTEXT

TEXT TO cTxt
Lorem Ipsum is simply dummy text of the printing and typesetting industry.
Lorem Ipsum has been the industrys standard dummy text ever since the 1500s,
when an unknown printer took a galley of type and scrambled it to make a type
It has survived not only five centuries, but also the leap into electronic
typesetting, remaining essentially unchanged. It was popularised in the 1960s with
the release of Letraset sheets containing Lorem Ipsum passages, and more recently
with desktop publishing software like Aldus PageMaker including versions of Lorem
Ipsum.
ENDTEXT

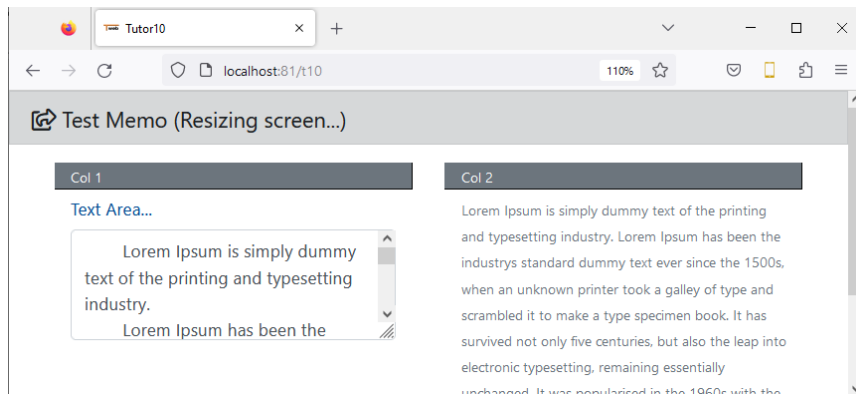
INIT FORM o
```

```
ROW o VALIGN 'top'  
  
COL o GRID 6  
  
SEPARATOR o LABEL 'Col 1'  
  
GET oGet MEMO LABEL 'Text Area...' VALUE cTxt GRID 12 OF o  
  
ENDCOL o  
  
COL o GRID 6  
  
SEPARATOR o LABEL 'Col 2'  
  
SMALL o LABEL cTxt GRID 12  
  
ENDCOL o  
  
ENDROW o  
  
ENDFORM o  
  
INIT WEB oWeb RETURN  
?>
```

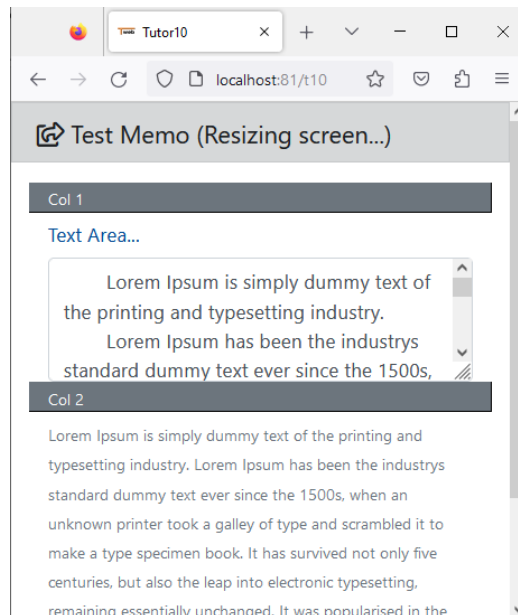
Esta pantalla vista en pantalla de escritorio tendría una apariencia similar a esta



Por defecto a medida que la hacemos mas estrecha mantiene la definición que hemos diseñado



Pero podemos controlar el flujo para decir que, a partir de un ancho, cabalgue y se ponga una columna encima de la otra



Este punto final de control lo haremos con la propiedad cType del objeto Form, y podrá tener los siguiente valores: xs, sm, md, lg (extra small, small, médium, large)

En función del valor el sistema recolocará los controles de una manera u otra. Se trata de ir probando con estos 4 valores e ir redimensionando el tamaño del navegador para ver y entender cómo funciona el sistema. Por defecto Tweb no tiene valor, por lo que el sistema mantendrá nuestras columnas como las hemos definido. Evidentemente esto no significa que se vea bien la pantalla cuando esta se vea p.e desde un smartphone, por lo que para cada caso de pantalla tendrá un diseño u otro.

De esta manera tan fácil podremos controlar el flujo de las diferentes partes de nuestra pantalla para que tenga un comportamiento responsive

Bootstrap ofrece muchas maneras de ejercer este tipo de control y dentro de su complejidad inicial te permite junto a la manipulación del css diseñar pantallas que se adapten perfectamente en función de dispositivo y contenido. Podéis leer más sobre el tema en este enlace → <https://getbootstrap.com/docs/4.0/layout/grid/>

Y este es objetivo como hemos dicho de Tweb, ayudar a crear rápidamente y sobre todo fácilmente pantallas responsive de aspecto profesional.

The screenshot shows a web browser window with the address bar displaying 'localhost/go/spagetti.prg'. The page features a header image of a city skyline with the text 'Friends of mod harbour' in blue, followed by 'Slack group project.' and 'Spagetti code example... All in one !'. Below this is a 'Contact Us' section with a grey header. The form contains several input fields: 'Alias' (placeholder 'Entra tu alias...'), 'Nombre' (placeholder 'Entra tu nombre...'), 'Fecha de Nacimiento' (placeholder 'dd/mm/aaaa' with a calendar icon), 'Mail de contacto' (placeholder 'Entra tu mail...'), and 'Clase de Bono' (a dropdown menu with 'Normal' selected). There is also a paragraph of placeholder text in Catalan: 'En la indústria editorial i en disseny gràfic, lorem ipsum és un text de farciment que s'usa habitualment per a mostrar els elements gràfics d'un document, com ara la tipografia o la composició.' Below the text is a toggle switch labeled 'Acepto condiciones' which is currently turned on. At the bottom of the form are two buttons: 'Enviar' with a paper plane icon and 'List' with a list icon.

Hasta aquí hemos visto la base de Tweb dentro del apartado de diseño y como fácilmente podemos plantear la composición de nuestras pantallas.

Menús

Este apartado es simple y solo pretende mostrar como en un par de líneas podemos crear un menú sencillo y en segundos tenemos armado el disparador de nuestro módulos de la aplicación.

Este es un ejemplo de cómo crear un Nav y un par de items de menú

```
<?prg
#include "lib/tweb/tweb.ch"

LOCAL o, oWeb

DEFINE WEB oWeb TITLE 'Menus'

    NAV oNav ID 'nav' TITLE '&nbsp;Menus' ;
        LOGO 'files/images/mini-mercury.png' ;
        ROUTE 'menu' WIDTH 30 HEIGHT 30 OF oWeb // BURGUERLEFT

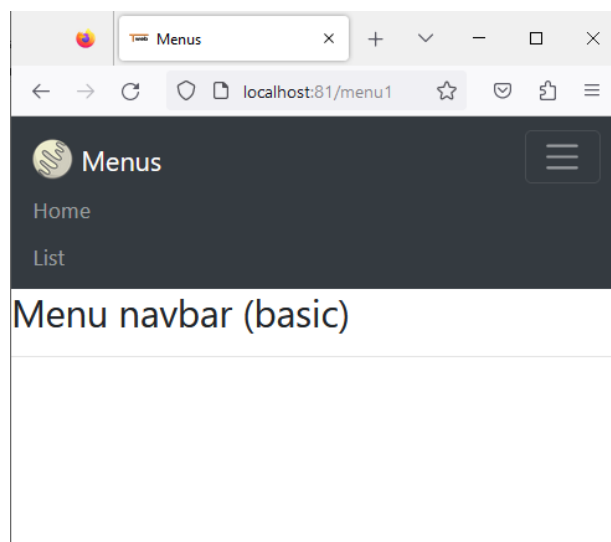
    MENUITEM 'Home' ROUTE 'menu' OF oNav
    MENUITEM 'List' ROUTE 'brw' OF oNav

    DEFINE FORM o OF oWeb

        HTML o INLINE '<h3>Menu navbar (basic)</h3><hr>'

    INIT WEB oWeb RETURN

?>
```



A partir de aquí podemos crear añadiendo solo la cláusula SIDEBAR un menú con este estilo tan usado. Podemos poner tantos items como queramos, iconos,... siguiendo nuestra metodología de codificación

```
<?prg
#include "lib/tweb/tweb.ch"

LOCAL o, oWeb
LOCAL cSideBar := ''

DEFINE WEB oWeb TITLE 'Menu'

    NAV oNav ID 'nav' TITLE '&nbsp;Menus' ;
        LOGO 'files/images/mini-mercury.png' ;
        ROUTE 'menu' WIDTH 30 HEIGHT 30 SIDEBAR OF oWeb

        MENU GROUP 'General' OF oNav

            MENUITEM 'Home' ICON '<i class="fa fa-home" aria-hidden="true"></i>'
ROUTE 'menu' OF oNav
            MENUITEM 'List' ICON '<i class="fa fa-list" aria-hidden="true"></i>'
ROUTE 'brw' ACTIVE OF oNav

        ENDMENU OF oNav

        MENU 'Tools' ICON '<i class="fa fa-times-circle" aria-hidden="true"></i>'
OF oNav
            MENUITEM 'My Submenu' ICON '<i class="fa fa-list" aria-
hidden="true"></i>' ROUTE 'menu' OF oNav
            MENUITEM 'My Submenu2' ICON '<i class="fa fa-list" aria-
hidden="true"></i>' ROUTE 'menu' OF oNav
            MENUITEM SEPARATOR OF oNav
            MENUITEM 'My Submenu3' ICON '<i class="fa fa-list" aria-
hidden="true"></i>' ROUTE 'menu' OF oNav
            ENDMENU OF oNav

            MENUITEM SEPARATOR OF oNav
            MENUITEM 'List' ICON '<i class="fa fa-list" aria-hidden="true"></i>' ROUTE
'brw' OF oNav

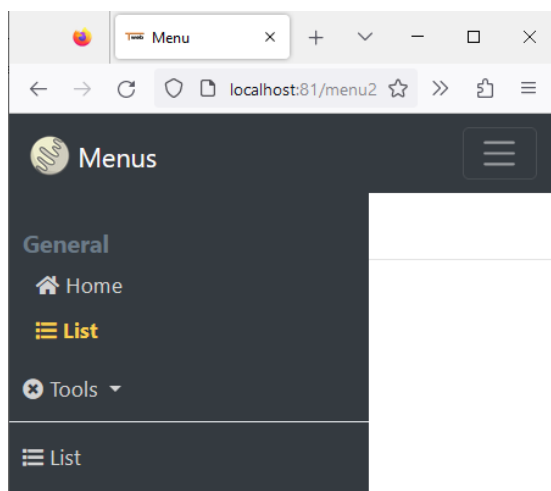
    DEFINE FORM o OF oWeb

        HTML o INLINE '<h3>Menu Sidebar (basic)</h3><hr>'

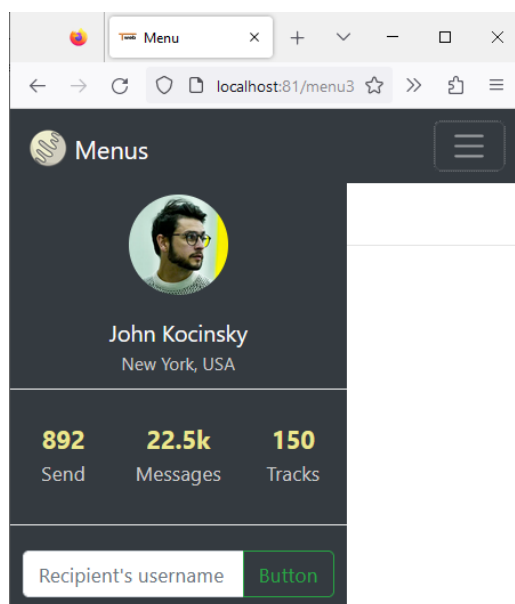
    ENDFORM o

    INIT WEB oWeb RETURN
?>
```

Autor Carles Aubia
 Fecha 01/04/2023
 Versión 2.0a



Y como hemos podido ver, Tweb nos permite insertar código nativo si lo necesitamos por lo que con poco que se haga, fácilmente y en cuestión de minutos tienes tu menú armado



FrontEnd – Functions Suport

Tweb dispone de unas sencillas funciones para poderlas usar desde JavaScript en caso de querer usar este lenguaje para alguna acción en particular.

Función	Descripción
MsgInfo(cMsg, fCallback, cTitle, clcon)	Mensaje asincrono usando dialogo modal cMsg – Mensaje fCallback – funcion a ejecutar cuando finalice el mensaje cTitle – Titulo del Diálogo clcon – Icono del titulo
MsgError(cMsg, fCallback, cTitle, clcon)	Mensaje de Error. Parámetros iguales que MsgInfo()
MsgGet(cInput, fCallback, cTitle, clcon)	Mensaje de Get. Parámetros iguales que MsgInfo()
MsgYesNo(cMsg, fCallback, cTitle, clcon)	Mensaje de YesNo. Parámetros iguales que MsgInfo()
MsgLoading(cMessage, cTitle, clcon, lHeader)	Mensaje de carga... /* Icons Animated "fas fa-spinner fa-spin" "fas fa-circle-notch fa-spin" "fas fa-sync fa-spin" "fas fa-cog fa-spin" "fas fa-spinner fa-pulse" "fas fa-stroopwafel fa-spin" */
MsgNotify(cMsg, cType, clcon, lSound)	Mensaje de notificación. cType = success, info, danger, warning Examples -> http://bootstrap-growl.remabledesigns.com/
TWebIntro(cId, fFunction)	Forzar un control a partir de su ID que cuando se pulse la tecla Intro salte a una función
MsgServer(cUrl, fCallback, oValues)	Funcion para enviar a una url <cUrl> y la respuesta llega a la funcion callback <fCallback>. Se pueden especificar parámetros <oValues>

Web flow

Este apartado lo que he querido nombrar así porque es el que se encargara Tweb de controlar en nuestra aplicación para poder gestionar las diferentes peticiones de nuestra página web a nuestro servidor y que este a su vez nos permita gestionar parte de nuestros controles que se encuentran en el navegador

API

En este capítulo se explica otra de las magias de Tweb y es la conexión de la pantalla con la web en si, la interacción con los controles, el crear un flujo de la web enviando y recibiendo continuamente datos...De esto se va a tratar en este tema.

Para poder entender esta parte retomaremos el primer ejemplo que creamos y añadiremos 2 controles

```
<?prg
#include "lib/tweb/tweb.ch"

LOCAL o, oWeb

DEFINE WEB oWeb TITLE 'TWeb'

    DEFINE FORM o OF oWeb
        o:lDessign := .f.

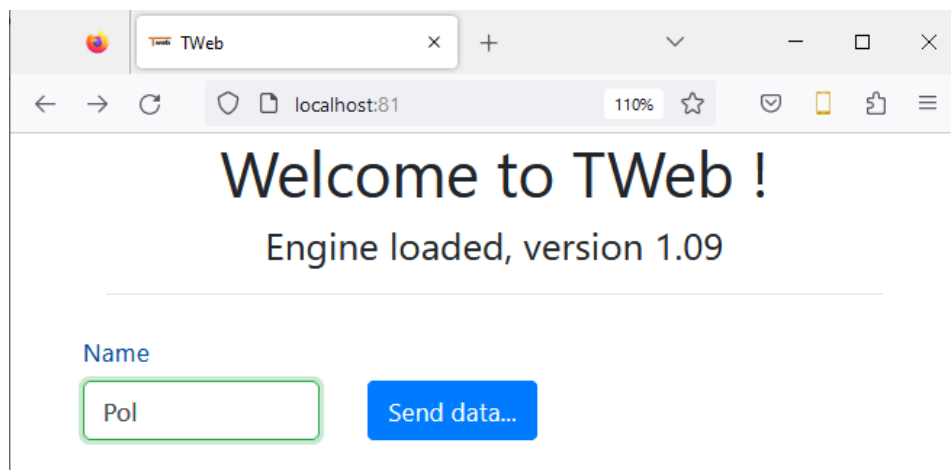
    INIT FORM o
        SAY VALUE '<h1>Welcome to TWeb !' ALIGN 'center' GRID 12 OF o

        SAY VALUE '<h4>Engine loaded, version {{ TWebVersion() }}<hr>' ;
            ALIGN 'center' GRID 12 OF o

        ROWGROUP o VALIGN 'bottom'
            GET VALUE '' LABEL 'Id:' OF o
            BUTTON LABEL 'Send data...' OF o
        ENDROW o

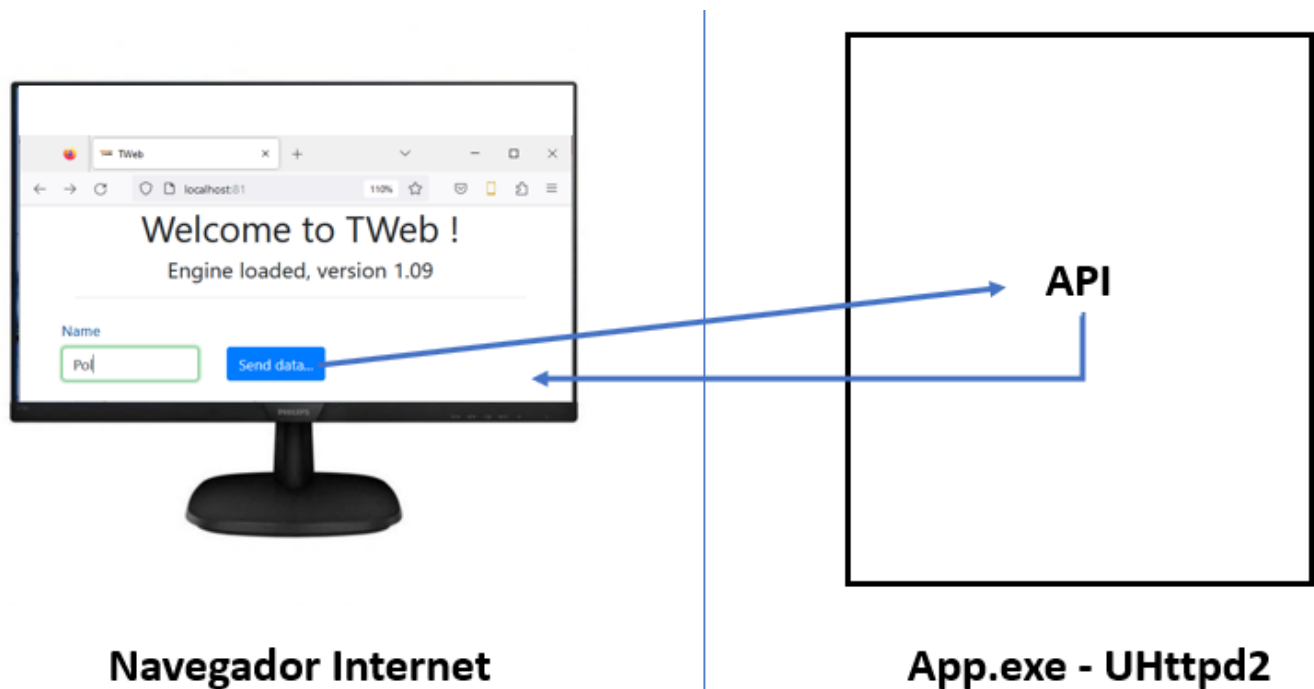
    ENDFORM o

INIT WEB oWeb RETURN
?>
```



Hasta aquí hemos reproducido lo que sabemos ya para empezar a “maquetar” pantallas, pero el resultado es una bonita pantalla, pero muerta, no tiene vida, no hace nada, ... El concepto de uhttpd2 es crear una serie de API's que ayudan a controlar la gestión de cada pantalla. Cada control de una pantalla podrá tener unos eventos que a cualquier disparo de ellos los enviará a dicha api que procesará lo que necesita y devolverá de nuevo una respuesta a la pantalla.

Vamos a basarnos en nuestro un ejemplo, tenemos 2 controles uno de tipo GET (input en html) y otro de tipo BUTTON (button en html). Podremos definir una ACTION en el button para que realice o mande realizar algo a nuestra api. En este caso lo que queremos es que envíe el contenido de nuestro GET a la API y la api recupere el valor y mande un saludo de bienvenida a nuestro navegador. La API evidentemente está en nuestro servidor, así que ya debemos pensar en clave web y situar cada cosa en su sitio, y siguiendo pensando podemos tener el servidor en la India y nosotros estamos con un navegador en Francia. Es importante esto, porque siempre he notado que la mayoría les cuesta asimilar este escenario



En resumen, una pantalla es un “pintado” que ha hecho nuestro navegador, pero cada interacción enviaremos un mensaje a nuestro servidor (uhttpd2) y será allá donde se decida todo lo que se ha de hacer. Muchos pensarán ahora que desde el propio navegador y con JavaScript podemos realizar ciertas acciones y es cierto, pero en esta parte del manual y partiendo de la base que es un proyecto con 0 JavaScript o casi nada, seguiremos este concepto.

- Para poder configurar este sistema será necesario Identificar todos los controles que queramos gestionar con un id.
- Definiremos una acción a ejecutar con el button que se hará con la cláusula ACTION. A modo de ejemplo la llamaremos "send_data"
- Identificaremos el FORM con un ID y especificaremos el nombre de la API que podremos llamar en este caso "myapi"

Nuestro código quedaría de esta manera y diríamos que a nuestra pantalla que teníamos definida ahora le añadimos una funcionalidad y le damos vida.

```
<?prg
#include "lib/tweb/tweb.ch"

LOCAL o, oWeb

DEFINE WEB oWeb TITLE 'TWeb'

    DEFINE FORM o ID 'myform' API 'myapi' OF oWeb

    INIT FORM o
        SAY VALUE '<h1>Welcome to TWeb !' ALIGN 'center' GRID 12 OF o

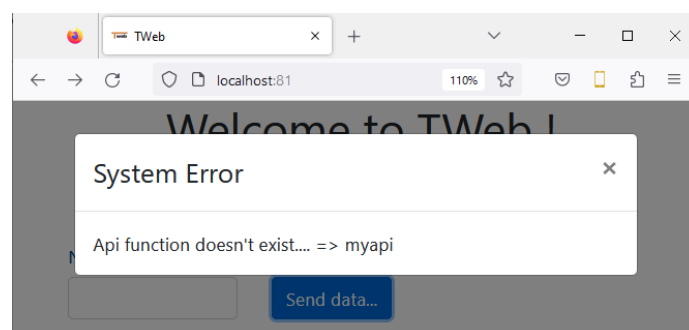
        SAY VALUE '<h4>Engine loaded, version {{ TWebVersion() }}<hr>' ;
            ALIGN 'center' GRID 12 OF o

        ROWGROUP o VALIGN 'bottom'
            GET ID 'myname' VALUE '' LABEL 'Id:' OF o
            BUTTON ID 'mybtn' LABEL 'Send data...' ACTION 'send_data' OF o
        ENDROW o

    ENDFORM o

INIT WEB oWeb RETURN
?>
```

Si cargamos de nuevo nuestra pantalla y clickamos al button nos debería salir un mensaje de error !



El sistema nos dice que no existe la api “myapi”. Y es así porque no la hemos definido aun

Creación de una API

Una api ser una función/conjunto de funciones que se encargaran de procesar estas peticiones/respuestas a la web. Y como bien se entenderá estas funciones irán enlazadas en nuestra aplicación.

Vamos a crear una api que pondremos en nuestra carpeta .\app que llamaremos myapi.prg. Todas las apis recibirán un parámetro que será un objeto oDOM. Es te objeto hace referencia al DOM del navegador y nos ayudará a recuperar información de la pantalla del navegador y ejecutar acciones sobre elementos del DOM y /o navegador.

Intento ir paso a paso y quizás ahora estamos en un punto que nos suene raro todo, pero es muy fácil, cuando tengamos el ejemplo montado y analizado veréis como el flujo es muy sencillo.

\app\myapi.prg

```
function myapi( oDom )  
  
    local cName := oDom:Get( 'myname' )  
  
    oDom:SetMsg( 'Hello ' + cName + ' at ' + time() )  
  
    retu oDom:Send()
```

Son 4 líneas pero importantísimas para entender el concepto de la api.

1. La función la hemos llamado myapi, igual que la referencia que hemos hecho en nuestra web con la cláusula API ‘myapi’. Esta función recibe un objeto oDom
2. Podemos observar cómo RECUPERAR un valor del navegador desde nuestra API que esta en el servidor. Es uno de los muchos métodos que tiene este objeto y que están descritos en el anexo. En este caso con el método Get(<Id>) podemos recuperar el valor de ese control
3. Con SetMsg(<cMsg>) ejecutamos un método que muestra un mensaje en nuestro navegador.
4. Retu oDom:Send() es la mas importante. Nuestra api finaliza y con este último método enviamos al navegador nuestra/s respuesta/s

Como bien sabies, ya solo queda enlazar este fichero a nuestro proyecto. Para ello lo añadiremos en nuestro fichero app.hbp. Podemos añadir tantas apis como necesitemos.

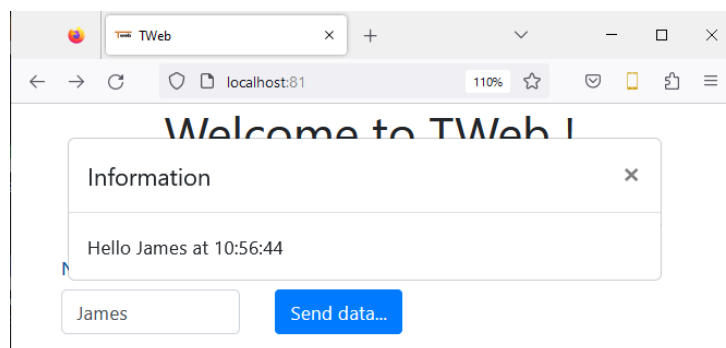
```
-n  
-w1  
-es2  
-inc  
-head=full  
-mt  
  
app\app.prg  
app\myapi.prg
```

```
-Llib\uhttpd2
-luhttpd2

-Llib\tweb
-ltweb

# WAPI_OutputDebugString()
hbwin.hbc
xhb.hbc
```

Si enlazamos la aplicación y volvemos a pulsar el button el sistema nos tendría de funcionar y mostrar el mensaje



Y esta es la magia y la base de TWEB !!! Un sistema que nos permite crear nuestras pantallas y gestionar nuestras apis para su control de una manera rápida y sencilla. Hasta aquí hemos visto como no hemos usado ningún servidor como apache (sino nuestro uhttpd2), ni JavaScript, ni html,... Y este es el objetivo de este proyecto ! con un mínimo de esfuerzo poder crear módulos web sencillos de programar sin necesidad de absorber grandes conocimientos en lenguajes, protocolos, técnicas de arquitectura,...

Una API varios procesos...

Para finalizar este capítulo y de entender la conexión de la pantalla con nuestra api, hemos de saber que podemos tener otros controles y que cada uno puede procesar su propio mensaje. Imaginemos que añadimos un par de líneas mas iguales a las que tenemos, un get y un button. Este button queremos procesar un mensaje para que se vaya al servidor y nos de la hora actual y la escriba en nuestro get. Llamaremos a nuestra acción del button "get_time"

```
ROUGROUP o VALIGN 'bottom'
  GET ID 'mytime' VALUE '' LABEL 'Now' DISABLED OF o
  BUTTON ID 'mybtn2' LABEL 'Get Time...' ACTION 'get_time' OF o
ENDROW o
```

Observamos que se ha definido una nueva acción, es decir, nuestra pantalla la preparamos para procesar 2 mensajes a nuestra api.

Ahora es cuando yo propongo este diseño que sirve para aglutinar 2 o 2000 mensajes de proceso.

```
function myapi( oDom )

    do case
        case oDom:GetProc() == 'send_data' ; DoHello( oDom )
        case oDom:GetProc() == 'get_time' ; DoSetTime( oDom )

        otherwise
            oDom:SetError( "Proc doesn't exist: " + oDom:GetProc() )
    endcase

retu oDom:Send()

// ----- //

function DoHello( oDom )

    local cName := oDom:Get( 'myname' )

    oDom:SetMsg( 'Hello ' + cName + ' at ' + time() )

retu nil

// ----- //

function DoSetTime( oDom )

    local cNow := DToc(date()) + ' - ' + time()

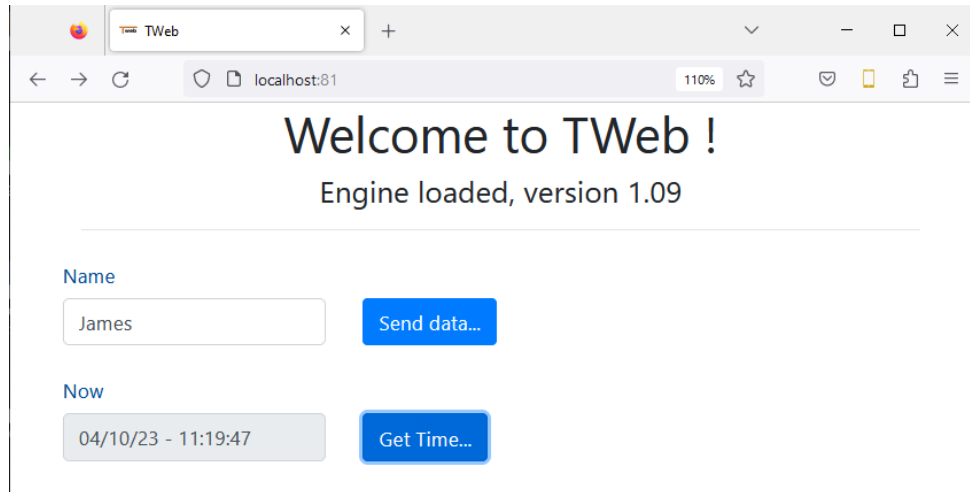
    oDom:Set( 'mytime', cNow )

retu nil
```

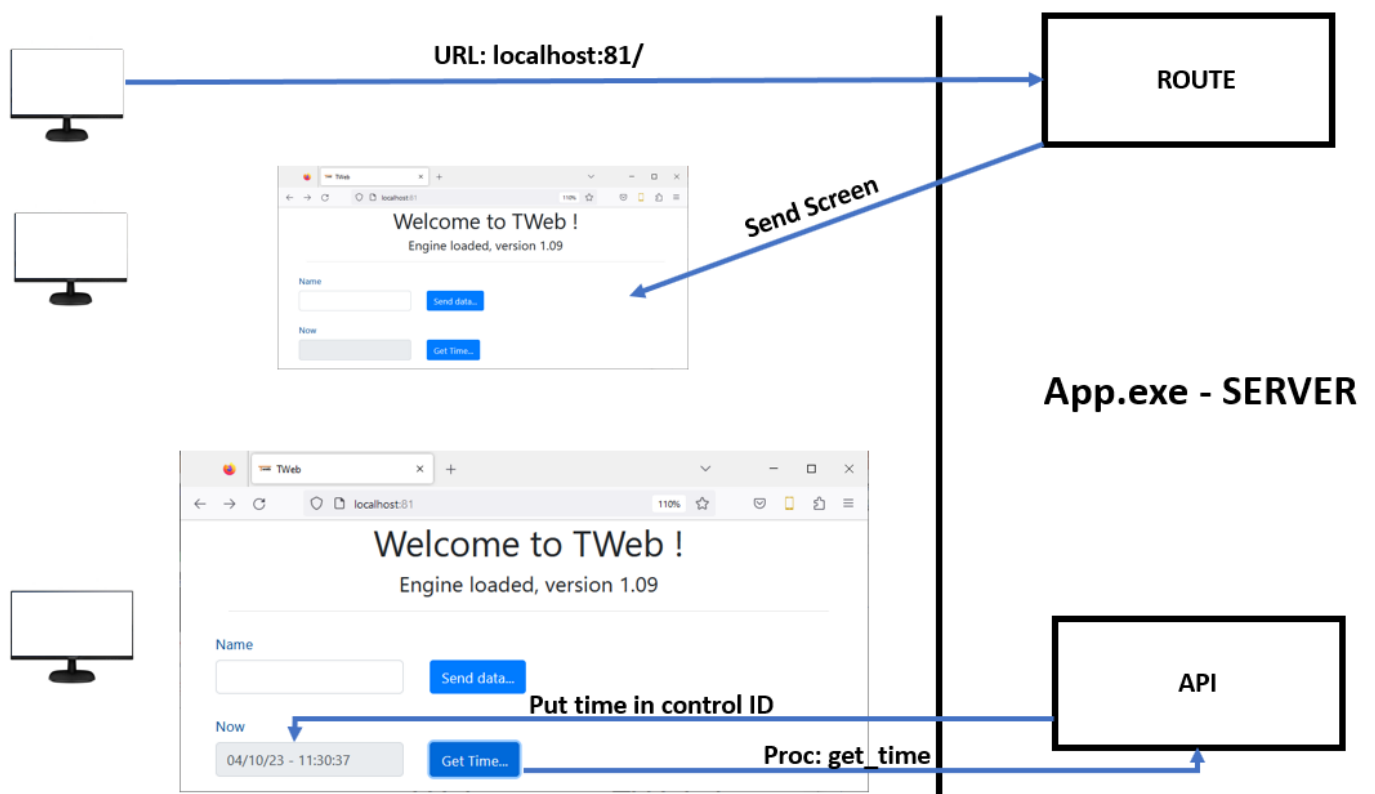
La parte más importante es la de la función api myapi(). Ahora esta función está diseñada para recuperar que proceso están pidiendo, con el método oDom:GetProc(). Una vez sepamos que proceso nos piden desde el navegador, ejecutaremos un proceso asociado a él. Fácil !!!

En el caso de que no tengamos definido ninguna función para el proceso que nos piden, enviaremos al navegador un mensaje de error. Esto nos ira muy bien en tiempo de diseño para saber que nos falta aun alguna pieza a programar.

Esta es la base de la api, y con este diseño es muy fácil escalar a las funciones que necesitemos para esta pantalla. Bien, a reflexionar que una api no ha de ser exclusiva de una sola pantalla/formulario. Podríamos usar si nos conviene una misma api en varios sitios, pero dejemos que fluya todo a medida que lo necesitéis...



En este punto debemos tener ya una visión de cómo se comporta el sistema. Básicamente tenemos una parte de carga de pantallas que gestionaremos con la definición de nuestras rutas y otra de gestión de mensajes de proceso de la pantalla de la cual se encargara nuestra api.



Javascript

Como se ha comentado anteriormente TWEB no es una librería inclusiva y restrictiva. Permite en el caso de que uno tenga conocimientos inyectar tanto html como JavaScript, css,...

La manera en que se ha diseñado Uhttpd2 permite al usuario poder realizar como hemos visto antes conexiones con nuestro backend usando la api que permite que nuestros controles se conecten de manera automática a nuestro sistema.

```
<?prg
#include "lib/tweb/tweb.ch"

LOCAL o, oWeb

DEFINE WEB oWeb TITLE 'Tutor10'

DEFINE FORM o ID 'mydlg' API 'api_dialog' OF oWeb

INIT FORM o

  ROWGROUP o
    GET ID 'myid' VALUE '11' GRID 4 BUTTON 'Search' ACTION 'GetId()' OF o
  ENDROW o

  HTML o
    <script>
      function GetId() {
        var cId = $('#mydlg-myid').val()

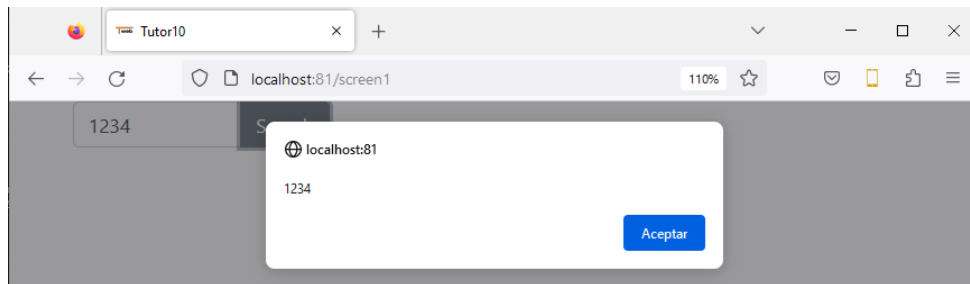
        alert( cId )
      }
    </script>
  ENDTEXT

ENDFORM o

INIT WEB oWeb RETURN
?>
```

Es importante tener en cuenta que TWEB convierte internamente el ID que definimos del control en la suma del ID del FORM (en este ejemplo 'mydlg') mas el ID del control (en este ejemplo 'myid'). Así pues en la función de JavaScript nos refiramos a este id como "mydlg-myid")

Si ejecutamos el ejemplo al pulsar el button dispararía la function GetId()



Pero siempre existe la necesidad de que necesitamos por el motivo que sea ir por otro camino, y este es en este caso el camino standard, el uso de JavaScript.

Como hemos visto en el tema del api, nosotros debemos especificar el nombre del proceso que queremos ejecutar con la cláusula ACTION, y el sistema junto al nombre de la API especificado en el nombre del FORM ya se encargará de realizar las peticiones pertinentes.

Cuando TWEB comprueba que action lleva un nombre entre comillas, p.e. "get_time" lo tratará como el nombre del procedimiento a ejecutar en el api. Sin embargo si especificamos paréntesis validara que exista la función en JavaScript y la ejecutará. Es decir, tiene la capacidad de tomar 2 caminos diferentes en función de nuestras necesidades.

Evidentemente este sistema ya solo lo podrán tomar programadores que tengan nociones de JavaScript.

MsgApi() → Frontend to Backend

Si dentro de nuestra lógica de programa y habiéndonos saltado el camino natural del uso de nuestra API, estamos dentro de una function javascript y necesitaríamos acceder a una api y ejecutar un proceso, Tweb provee de la función MsgApi() que nos dará la opción de crear esta conexión.

Los parámetros de esta función son:

MsgApi(<API>, <PROC>, [<oParam>])

oParam es un parámetro opcional que podríamos añadir parámetros en la llamada a la api.

Siguiendo el ejemplo anterior, podríamos modificar la función para que hiciera una llamada directa a nuestra api

```
function GetId() {  
    var oPar = new Object()  
    oPar[ 'myname' ] = 'Jean-Paul'
```

```
MsgApi( 'api_dialog', 'hello', oPar )  
}
```

SetJS() → Backend to Frontend

Si estamos procesando en el backend, podemos ejecutar una función javascript y enviarle incluso parámetros.

Podemos definir en nuestra pantalla la siguiente función

```
HTML o  
  
<script>  
  
    function myCollector( u ) {  
  
        MsgInfo( 'Message from my func myCollector.<br>It was executed from  
backend.<br>Check console' )  
  
        console.log( 'Parameters received from backend', u )  
    }  
  
</script>  
  
ENDTEXT
```

Esta simple función la podríamos ejecutar desde nuestro backend usando nuestro objeto oDom

```
function DoJS( oDom )  
  
    local hData := {=>}  
  
    hData[ 'name' ]      := 'Johtn Kocinsky'  
    hData[ 'age' ]       := 37  
    hData[ 'in' ]        := date() - 1000  
  
    oDom:SetJS( 'myCollector', hData )  
  
retu nil
```

Advanced

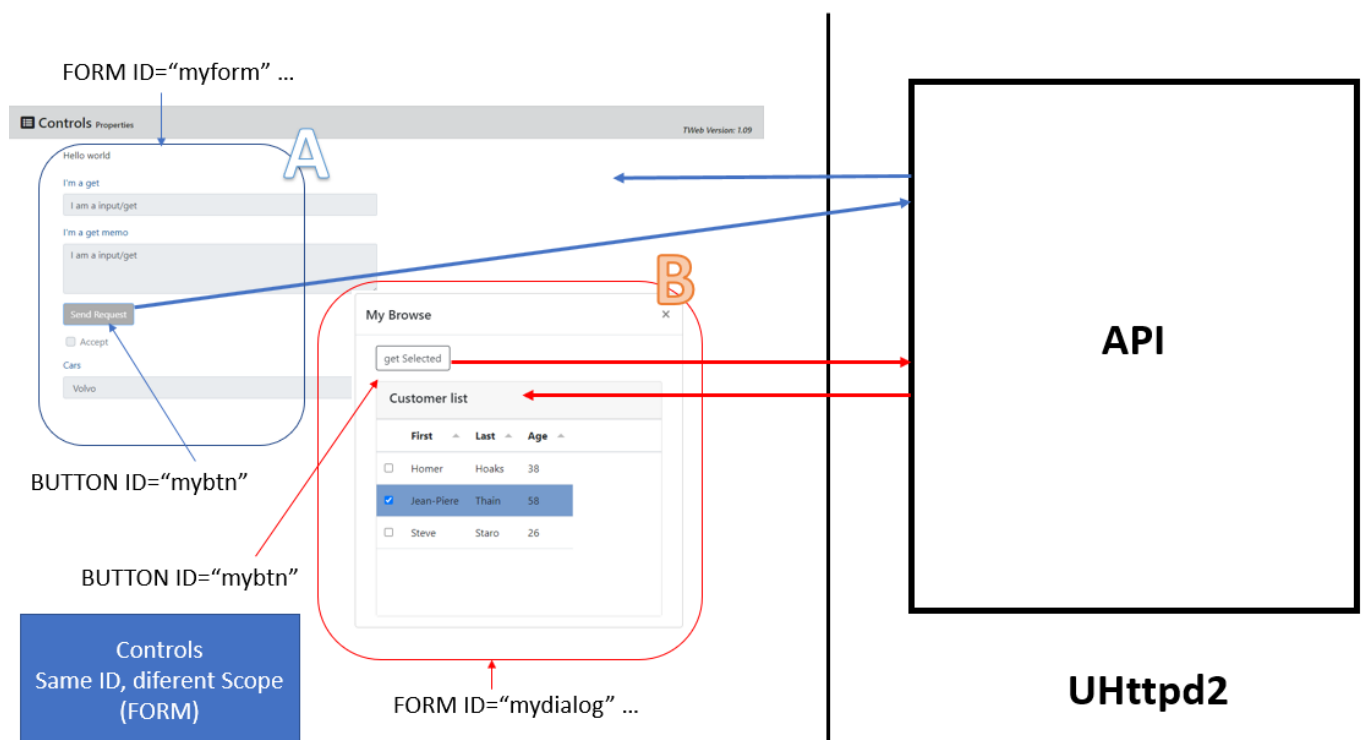
Dialogs

TWeb ofrece la posibilidad de crear fácilmente diálogos que nos ayuda fácilmente a cumplimentar nuestras pantallas de datos. El tema aquí no se trata de crear un dialogo en controles sino en que dicho dialogo pueda seguir funcionando con la técnica que usamos en uhttpd2.

Por eso es quizás importante entender el siguiente diagrama. Básicamente pretende ser una entrada de datos (A) que en un momento dado puedas invocar y crear un diálogo (B). Hasta ahora estábamos diseñando en A, y podemos imaginar que uno de nuestro botones tiene un ID="mybtn".

Si nosotros creamos un diálogo (B) y casualmente o no existe un button con un ID="mybtn", como sabrá nuestro api si queremos ejecutar una acción sobre cual de los 2 actuar ?. La diferencia estriba en el scope, el ámbito, el FORM. Un button pertenece al FORM myform y el otro a mydialog.

Y es este quizás solo lo que debemos tener en cuenta a la hora de seguir trabajando con diálogos. Todo esto lo veremos en este capítulo.



Realmente crear un dialogo es muy fácil con pocos conocimientos de javascript, pero siguiendo con nuestro sistema será nuestro backend y no nuestro frontend quien se encargue de crear estos diálogos.

Para crear un diálogo desde nuestro backend y gracias a nuestro objeto oDom, ejecutar el método SetDialog().

oDom:SetDialog(<id>, <html>, [<title>], [<options>])

Parámetro	Descripción
id	ID del FORM del Dialogo que crearemos
html	Contenido del dialogo. Código html. Este código puede volverse a crear usando TWEB
title	Opcional. Título del diálogo
options	Opcional. Parámetros que se pueden pasar al diálogo. TWeb usa l plugin de bootbox para manejar estos diálogos, por lo que la mayoría son parámetros relacionados con él. Mirando los diferentes ejemplos podemos ver las distintas opciones que podemos usar en la construcción del diálogo.

Imaginaros una api sencilla que si le llega el proceso "pinta" dibuja un dialogo. A ver si podemos seguir la secuencia

```
#include "../lib/tweb/tweb.ch"

function Api_Dialog( oDom )

    do case
        case oDom:GetProc() == 'pinta' ; DoPinta( oDom )

        otherwise
            oDom:SetError( "Proc don't defined => " + oDom:GetProc() )
    endcase

    retu oDom:Send()

// ----- //

static function DoPinta( oDom )

    local cHtml := MyScreen()

    oDom:SetDialog( 'xxx', cHtml )

    retu nil

// ----- //

static function MyScreen()
```

```
LOCAL o, oDlg

DEFINE DIALOG oDlg

    DEFINE FORM o ID 'abc' API 'api_dialog' OF oDlg

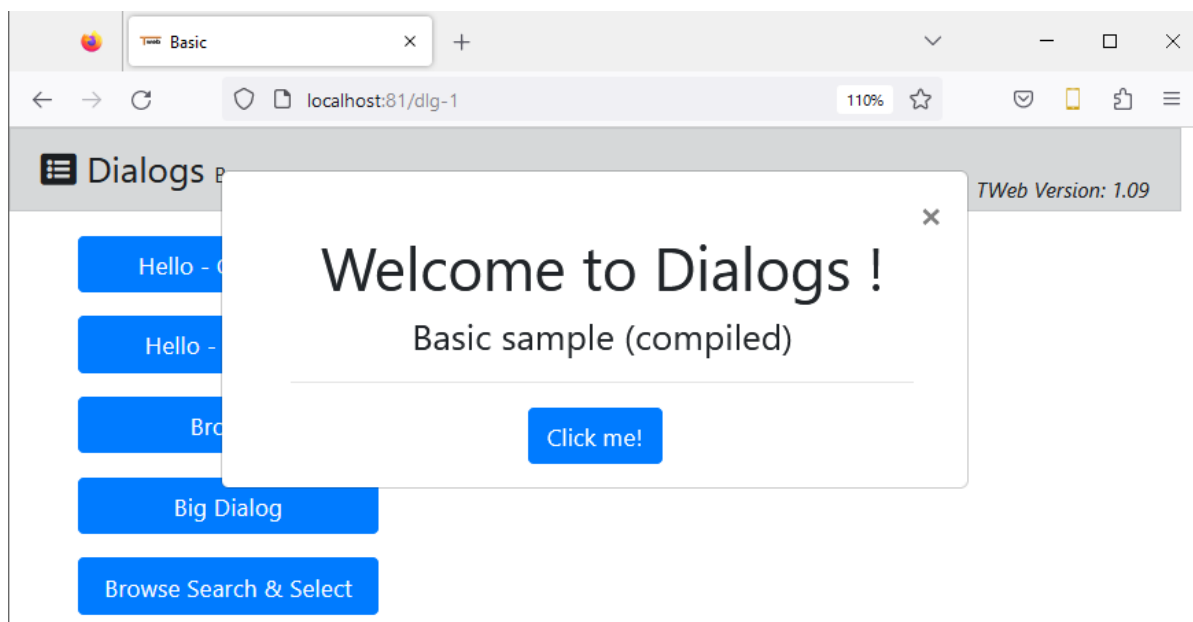
    INIT FORM o
        SAY VALUE '<h1>Welcome to Dialogs !' ALIGN 'center' GRID 12 OF o
        SAY VALUE '<h4>Basic sample (compiled)<hr>' ALIGN 'center' GRID 12 OF o

        BUTTON ID 'btn' LABEL 'Click me!' ALIGN 'center' GRID 12 ACTION 'hello' OF
o
    ENDFORM o

INIT DIALOG oDlg RETURN

retu nil
```

Este sencillo código e incluso la parte de definición de la pantalla que es un código que ya nos suena, es toda la técnica a usar.



Estos ejemplos están dentro del apartado diálogos de samples. Os aconsejo que los reviséis y los experimentéis. En este simple ejemplo destacar estos puntos.

1.- Cuando definamos un dialogo que enviaremos a una pantalla ya construida del navegador, en lugar de usar las clausulas DEFINE WEB / ACTIVATE WEB usaremos las cláusulas DEFINE DIALOG / ACTIVATE DIALOG

2.- Fijaros que el código se crea a partir de una funcion MyScreen() que ejecuta código Tweb puro y duro. Este código como os podéis imaginar está dentro del prg y esta compilado, es decir que en este caso nadie lo podría manipular desde fuera el servidor

En lugar de usar esta técnica podríamos usar la función UloadHtml()

```
static function DoPinta( oDom )

    local cHtml := UloadHtml( 'dialog\screen1.html' )
    local o      := {=>}

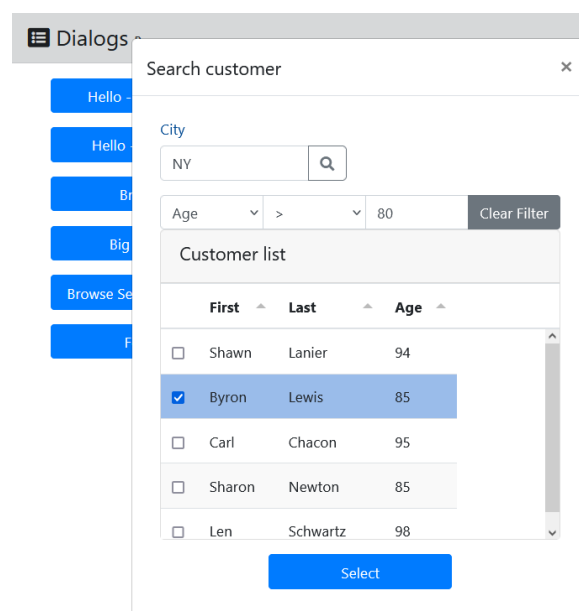
    //o[ 'backdrop' ]      := .t.
    //o[ 'onEscape' ]      := .f.
    //o[ 'closeButton' ]    := .f.
    //o[ 'className' ]      := 'bounceIn fadeOutRight'
    o[ 'title' ]           := 'Search customer'
    o[ 'size' ]            := '90%'

    oDom:SetDialog( 'xxx', cHtml, nil, o )

retu nil
```

En este caso la API carga el fichero .\html\screen1.html por lo que siempre podremos modificar sin tener que tocar para nada nuestro servidor.

Son distintas maneras de poder gestionar este punto en el diseño de nuestros diálogos, y como podemos observar seguimos usando Tweb sin ningún cambio adicional. Podemos tener resultados espectaculares entendiendo estos puntos de manera fácil y rápida.



Security

En este apartado haremos un breve vistazo a las funciones que tiene Uhttpd2 para poder usar en temas de seguridad como son el uso de Tokens y de Sesiones

Tokens

No es el propósito aquí explicar para que sirven los tokens pero si como fácilmente usar si necesitamos estos tokens en nuestras aplicaciones. La manera más fácil de explicar que es un token es la de un dato encriptado y en especial en el caso de Uhttpd2 podemos usar datos complejos, es decir de un string a un array o hash de datos.

Para poder crear un token usaremos la función

```
cToken := USetToken( hData )
```

Para recuperar su valor

```
hVar := UGetToken( cToken )
```

En los ejemplos de .\samples podréis encontrar un test de seguridad y podéis ver el fácil manejo de los tokens.

Sessions

Las sesiones son una forma sencilla de almacenar datos para usuarios de manera individual usando un ID de sesión único. Esto se puede usar para hacer persistente la información de estado entre peticiones de páginas.

El uso de sesiones en este caso es exclusivo con Tweb, por lo que solo con la librería Uhttpd2 no se podrá usar.

De manera básica podemos activar cuando queramos una sesión. A partir de aquel momento, cualquier variable la podremos almacenar de forma local en el servidor, de tal manera que queda fuera del alcance local de cualquier navegador local en la que puede ser accedida y/o manipulada.

Las sesiones ya están preconfiguradas con una serie de valores, pero es a la hora de levantar el servidor cuando se inicializan estos valores. Aquí están todos los parámetros configurables

```
Config Sessions !
-----
oServer:cSessionPath      := '.sessions' // Default path session ./sessions
oServer:cSessionName      := 'USESSID'   // Default session name USESSID
```


oServer:cSessionPrefix	:=	'sess_'	//	Default prefix sess_
oServer:cSessionSeed	:=	'm!PaswORD@'	//	Password default ...
oServer:nSessionDuration	:=	3600	//	Default duration session time 3600
oServer:nSessionGarbage	:=	1000	//	Default totals sessions executed for garbage
oServer:nSessionLifeDays	:=	3	//	Default days stored for garbage 3
oServer:lSessionCrypt	:=	.F.	//	Default crypt session .F.

A continuación, se describen las funciones básicas para el uso de sesiones.

Función	Descripción
UsessionReady()	Devuelve lógico. Si existe para la conexión una session creada
UsessionStart()	Inicia una Session en el servidor para esa conexión
Usession(<cVar>, <uData>)	Setter/Getter. Si se especifica <cVar> y <uData> se almacenara en la session el valor uData Si solo se especifica <cVar> se recupera
UgetSession()	Devuelve todos los datos almacenados mas los datos propio de gestion de la session. Si creais una salida _d(UgetSession()) podreis ver todos los valores del sistema de gestion de la session: path, name, perfix, duration, expired,...
UsessionEnd()	Destruye la session en el servidor

Ejemplos de uso:

Crear Session y añadir datos

```
static function DoSession_Init( oDom )

    local hData := {}

    hData[ 'name' ] := oDom:Get( 'myname' )
    hData[ 'age' ] := oDom:Get( 'myage' )
    hData[ 'date' ] := oDom:Get( 'mydate' )

    UsessionStart()
    Usession( 'data_user' , hData )
    Usession( 'data_in' , dtoc( date() ) + ' - ' + time() )

    oDom:SetMsg( 'Session created !' )

retu nil
```

Verificar session y recuperar datos

```
static function DoSession_Load( oDom )

    local hData := {}

    if USessionReady()

        hData := USession( 'data_user' )

    endif

    retu nil
```

Destruir Session

```
static function DoSession_End( oDom )

    USessionEnd()

    oDom:SetMsg( 'Session was deleted !' )

    retu nil
```

Charset

Este capítulo es uno de los más importantes y de los que da más quebradores de cabeza a muchos con la codificación y manipulación de los caracteres. Antes de empezar hemos de saber que la web se codifica con un mapa de caracteres “charset”, que el código lo escribimos con una codificación según editor y que...nuestras dbf's están también codificadas con su charset. Todos estos factores, todos, influirán en la codificación y correcta visualización de nuestra web. Para hacerlo un poco más complicado quizás añadir el escenario de usar una dbf que ya está siendo usada por una aplicación externa por ejemplo de Windows y que ha de coexistir con nuestra web.

Todos los ejemplos están en los .\samples y también hay una píldora (video) en el que se explica toda esta casuística y que es muy interesante para acabar de entender estos escenarios.

La web la podemos codificar de muchas maneras y trabajar de otras muchas, es por eso que Tweb se ha diseñado para trabajar de una manera que considera la más óptima independientemente de que se pueda parametrizar de una u de otra.

Por defecto nuestro primer comando para definir una web

```
DEFINE WEB oWeb TITLE 'Charset'
```

Ya viene con su configuración y por defecto usará UTF-8 para el manejo de los charset. Como ya comentamos el objeto que creamos, en este caso oWeb puede tener una serie de propiedades a configurar y una es charset

```
oWeb: cCharset = 'ISO-8859-1'
```

De esta manera ya tenemos asignado el nuevo charset.

Intentaré ser un poco práctico mostrando resultados de hacerlo de una manera o de otra

Si nosotros vamos a usar nuestros caracteres tradicionales Latin-1 podríamos sin ningún problema codificarla como ISO-8859-1. Es más, incluso la mayoría de nuestras bases de datos en dbf no tendríamos de codificar a cada lectura a utf8.

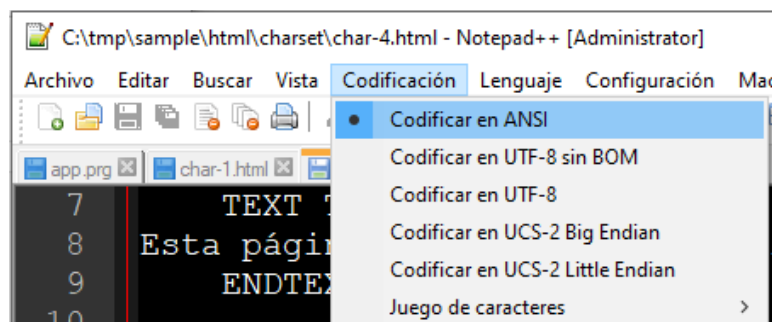
Web charset: 'ISO-8859-1'. Editor coding: ANSI



ANSI characters: àâèéíòóúñç

Caràcteres en otro codepage, p.e. Chino no lo puedo escribir porque mi editor no lo soporta en ANSI

Atención cuando especifico que nuestro editor codifica en ANSI. A saber, que todos los editores puedes configurarlos para que lo hagan de una manera u otra.



Web charset: 'UTF-8'. Editor coding: ANSI

Charset

TWeb Version: 1.09

Esta página está codificada en ANSI con nuestro editor. Nuestra web tiene un charset = 'UTF-8'

ANSI characters: 

Caracteres en otro codepage, p.e. Chino no lo puedo escribir porque mi editor no lo soporta en ANSI

Aquí ya podemos ver y entender que ocurre. Si abres el código (recuerdo que todo está en .\samples\charset) observamos que esta codificado en ANSI y correctamente escrito

```
TEXT TO cTxt
Esta página está codificada en ANSI con nuestro editor. Nuestra web tiene un
charset = 'UTF-8'
ENDTEXT


DEFINE WEB oWeb TITLE 'Charset'
oWeb:cCharset = 'UTF-8' // Default

HTML oWeb FILE 'tpl\header_examples.tpl' PARAMS 'Charset', ''

DEFINE FORM o ID 'myform' API 'api_charset' OF oWeb

INIT FORM o

HTML o FILE 'tpl\msg2.tpl' PARAMS cTxt, 'secondary', .f.

HTML o
ANSI characters: 
<br>
Caracteres en otro codepage, p.e. Chino no lo puedo escribir porque mi editor no
lo soporta en ANSI
ENDTEXT

ENDFORM o
```

Entonces que ocurre? Que el servidor envía a nuestro navegador un fichero perfectamente escrito y codificado en ANSI, pero que nuestra web tiene por defecto activada UTF-8. Entonces intenta recodificar estos símbolos especiales y es cuando NO se muestra bien.

Web charset: 'ISO-8859-1'. Editor coding: UTF-8

Charset

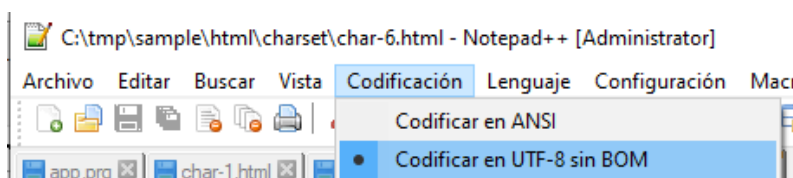
TWeb Version: 1.09

Esta página está codificada en UTF-8 con nuestro editor. Nuestra web tiene un charset = 'ISO-8859-1'

ANSI characters: Å Ä;Ã"Ã©Ã²Ã³Ã°Ã±Ã§

CarÃ cteres en otro codepage, p.e. Chino: è;™æ~ç""ä,æ—†â™çš,

En este caso codificamos con nuestro editor con UTF-8 sin BOM (**IMPORTANTISIMO** especificar sin BOM), y podemos observar cómo en nuestro editor podemos incluir caracteres chinos



```

DEFINE WEB oWeb TITLE 'Charset'
    oWeb:cCharset = 'ISO-8859-1'

    HTML oWeb FILE 'tpl\header_examples.tpl' PARAMS 'Charset', ''

    DEFINE FORM o ID 'myform' API 'api_charset' OF oWeb

    INIT FORM o

        HTML o FILE 'tpl\msg2.tpl' PARAMS cTxt, 'secondary', .f.

        HTML o
ANSI characters: àáâëíðòóúñç
<br>
Caràcteres en otro codepage, p.e. Chino: 这是用中文写的
    ENDTXT

ENDFORM o

```

Evidentemente cuando esta en UTF-8 y la web la tenemos codificada en 'ISO-8859-1'

Web charset: 'UTF-8'. Editor coding: UTF-8

Charset

TWeb Version: 1.09

Esta página está codificada en UTF-8 con nuestro editor. Nuestra web tiene un charset = 'UTF-8'

ANSI characters: àáèéíòóúñç

Caràcteres en otro codepage, p.e. Chino: 这是用中文写的

Este es el caso perfecto para usar todo lo que queramos, con caracteres de cualquier charset.

Hasta aquí los 4 casos básicos y parece que todo pueda estar claro. Quizás la 4 opción, edición en UTF-8 y web en UTF-8 y arreando, es lo mejor y para nosotros nos puede ir muy bien, peeeeero ahora solo faltaría acabar de solucionar el tema de las dbf's.

DBF's

Si en el 99% esta codificada en ANSI, hemos visto que cuando hemos codificado en ansi y mostrado en utf-8 pues no iba del todo bien, entonces que debemos hacer. Hemos de recodificar de ANSI → UTF8 cuando leamos de la tabla, y esto lo podemos hacer con la función hb_strtoUtf8().

Si creamos un formulario puro y duro y queremos guardar caracteres especiales en una dbf, en principio no tendremos ningún problema. Si leéis este documento interesante:

<http://harbourlanguage.blogspot.com/2010/06/harbour-codepage.html>

"theorically DBF are created to contain oem code page. But actually the great part of windows applications save these files as ANSI code page without problem or conflict."

Charset. Ansi TWeb Version: 1.09

Load Clean

Customer list

First	Last
Homer	Simpson
ñçääëéíòóú ÀÄÄËË	ñçäÄääëéíòóú
Jean-Piere	ñçaaääÄÈÖÄ

Lister (dbfviewer) - [C:\tmp\sample\data\te

File Edit Options Encoding Help

Table: test.dbf

ID	FIRST	LAST
1000	Homer	Simpson
1001	ñçääëéíòóú ÀÄÄËË	ñçäÄääëéíòóú
1002	Jean-Piere	ñçaaääÄÈÖÄ

Solo falta la correcta manipulación de la tabla. Muchos desconocen o no han necesitado por el motivo que sea usar con el comando de harbour la cláusula CODEPAGE <cPage> y ahora es vital para poder trabajar correctamente y sin dolores de cabeza.

No vamos a entrar en detalle de los codepages pero añadido solo este tip

Code page Description

1258 Vietnamese
1257 Baltic
1256 Arabic
1255 Hebrew
1254 Turkish
1253 Greek
1252 Latin1 (ANSI)
1251 Cyrillic
1250 Central European
950 Chinese (Traditional)
949 Korean
936 Chinese (Simplified)
932 Japanese
874 Thai
850 Multilingual (MS-DOS Latin1)
437 MS-DOS U.S. English

In general OEM, as opposed to ANSI (i.e., cp1252), denotes the codepage that most international versions of Windows have.

It is one of the OEM codepages from <http://www.microsoft.com/globaldev/reference/cphome.mspx>, and is used for the 'DOS boxes', to support legacy applications.

A German Windows version for example usually uses ANSI codepage 1252 and OEM codepage 850.

A equivalent Unicode characters. Standard ASCII and ISO 8859-1 (Latin-1) character

Para abrir una tabla usando el 850 debemos especificarlo de esta manera

```
use ( cPathFile ) new alias (cAlias) VIA 'DBFCDX' CODEPAGE 'CP850'
```

Para poder ejecutarlo correctamente, debéis hacer un request antes en vuestro programa principal

```
REQUEST HB_CODEPAGE_ES850
REQUEST HB_LANG_ES
REQUEST HB_CODEPAGE_ESWIN
```

Si vuestra aplicación va a trabajar toda de la misma manera, indicando en la configuración de vuestro programa el siguiente seteo ya será suficiente y no habré de poner más la cláusula CODEPAGE

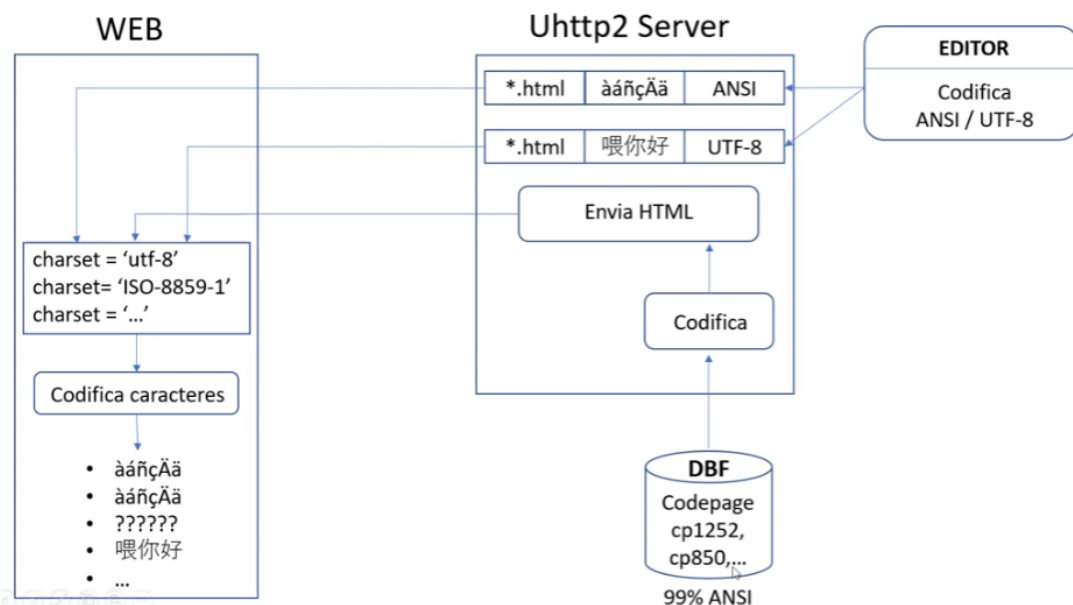
```
HB_LANGSELECT('ES')
HB_SetCodePage ( "ESWIN" )

SET( _SET_DBCODEPAGE, 'ESWIN' )
```

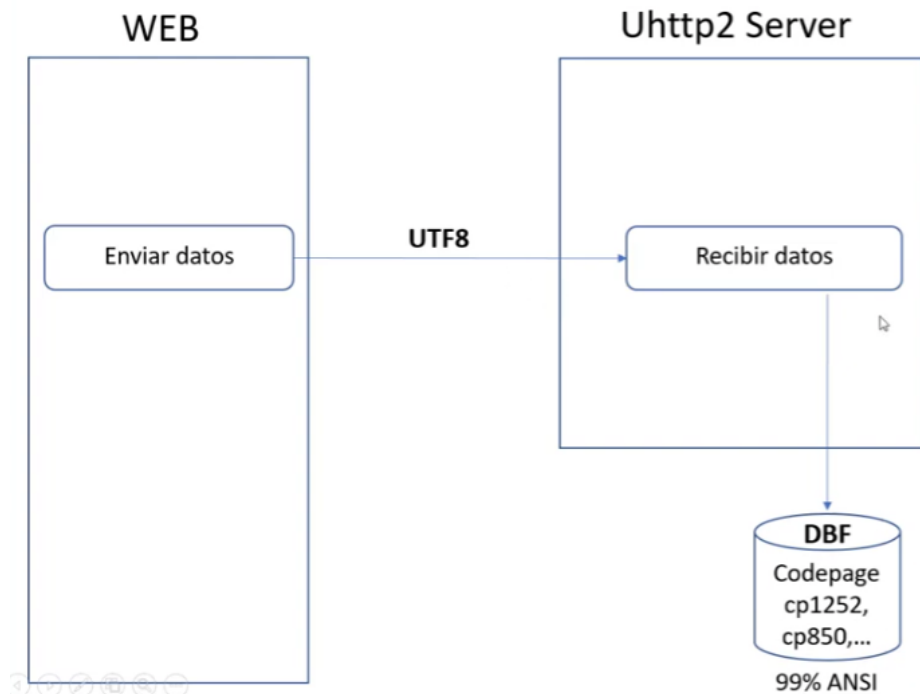
Con esta parametrización SOLO es necesario convertir en las lecturas de la dbfs a UTF8 para traspasarlo a la web. Todo lo que nos viene de la web al servidor para salvar en una dbf nos llega en formato utf8, PERO como nosotros le indicamos a nuestras tablas cual es nuestro codepage, ya se encarga el propio harbour de convertirlo a su codepage de turno !!!

Extraigo estas 2 capturas de una de las sesiones que hice que resumen los 2 caminos de una manera gráfica y sencilla

Server → Web



Web → Server



Quizás lo más relevante es que en la parte de carga hacia la web hay un proceso que debemos de hacer de recodificación mientras que en la parte de recepción de datos en el servidor y salvado de datos NO se hace nada porque ya se encarga el propio Harbour y sus codepages.

DBF's con UTF8


Y será posible tener nuestras DBF's con UTF-8 ? Por supuesto, la respuesta es SI. De la misma manera que hemos comentado que con los codepage (nuestro charsets) podemos gestionar nuestras dbf's podríamos usar el de UTF-8 para conseguir este sistema.

Si hacemos un REQUEST HB_CODEPAGE_UTF8EX al inicio de nuestro programa para que lo enlace y especifiquemos este codepage por defecto o en la abertura de la tabla, es posible. Solo debemos especificar un parámetro en la definición de nuestro server y es

```
oServer:Utf8 := .T.
```

Este flag lo que hace es que este UTF8 real que puede llegar de nuestra aplicación lo fuerzo en nuestras dbfs, es decir convierte utf8 a string para poderlos salvar correctamente

Hay un ejemplo en los samples que uso una tabla en varios idiomas

 **Charset.** UTF-8 TWeb Version: 1.09

Para poder ejecutar correctamente este ejemplo debes iniciar el servidor con la propiedad IUTF8 a .T.
-> oServer:IUtf8 := .t.

In order to run this example correctly, you must start the server with the property IUTF8 to .T. ->
oServer:IUtf8 := .t.

Load

Clean

Customer list

Nick	Name
Charly	English
カルロス	Japones
كارلوس	Arabe
카를로스	Coreano

Este ejemplo puede perfectamente editar los valores de las celdas en Coreano, japonés, árabe,... pero en este caso al ser todo el sistema UTF8 activamos simplemente este flag

Independientemente del uso del charset conociendo como funciona un poco mejor todo el sistema podemos entender que el sistema trabaja perfectamente con otros sistemas de base de datos como por ejemplo MySQL.

MYSQL

Poco a referirse al tema del uso de MySQL o cualquier otra base de datos. Seguimos insistiendo que el límite está en la capacidad de harbour en usar todo este tipo de librerías que nos permiten el uso de todas estas herramientas.

A manera de ejemplo rápido muestro como sería un sencillísimo ejemplo el que podemos hacer queries de una manera sencilla con Tweb. Esto ya es un módulo completamente funcional y se ve un poco de todo lo visto en el manual.

FrontEnd

```
<?prg
#include "lib/tweb/tweb.ch"

LOCAL o, oWeb, oBrw

DEFINE WEB oWeb TITLE 'Mysql'

HTML oWeb FILE 'tpl\header_examples.tpl' PARAMS 'Mysql', 'SQL command'

DEFINE FORM o ID 'myform' API 'api_mysql' OF oWeb
o:lDessign := .f.

INIT FORM o

HTML o
<h5>
    Usamos la base de datos dbharbour con tablas para estudiar como:
customer,          states, sellers
    <hr>
    We use the dbharbour database with tables to study such as: customer,
states, sellers
</h5><br>
ENDTEXT

ROWGROUP o

    GET ID 'sql' VALUE "select * from customer where age > 90 and state =
'NY'" LABEL 'Sql' ;
    BUTTON 'Execute', 'Clear' ACTION 'execute', 'brwclean' GRID 12 OF o

ENDROW o

ROWGROUP o

    COL o GRID 12

    aOptions := { 'height' => '300px' }

    DEFINE BROWSE oBrw ID 'mytable' OPTIONS aOptions TITLE '<h5>Sql
command</h5>' OF o

    INIT BROWSE oBrw

    ENDCOL o

    ENDROW o

ENDFORM o

INIT WEB oWeb RETURN
?>
```

BackEnd

```
static function DoExecute( oDom )

    local aRows      := {}
    local aCols      := {}
    local cSql       := oDom:Get( 'sql' )
    local lError     := .f.
    local oWDO       := WDO_MYSQL():New( "localhost", "harbour", "hb1234",
"dbHarbour", 3306 )
    local hRes, a, n, hConfig, aStruct

    IF ! oWDO:lConnect

        oDom:SetError( oWDO:cError )
        retu nil

    ENDIF

    IF oWDO:lConnect

        IF !empty( hRes := oWDO:Query( cSql, @lError ) )

            while ( !empty( a := oWDO:Fetch_Assoc( hRes ) ) )
                Add( aRows, a )
            end

            oWDO:Free_Result( hRes )

        ENDIF

    ENDIF

    if lError
        oDom:SetError( oWDO:cError )

        hConfig := { 'columns' => {}, 'data' => {} }


        oDom:TableInit( 'mytable', hConfig )
        oDom:TableTitle( 'mytable', '' )
    else
        aStruct := oWDO:DbStruct()

        for n := 1 to len( aStruct )
            Add( aCols, { 'field' => aStruct[n][1], 'title' => aStruct[n][1]} )
        next

        hConfig := { ;
            'columns' => aCols, ;
            'data' => aRows;
        }

        oDom:TableTitle( 'mytable', '<h5>Coincidencias: ' + str(len(aRows)) +
```

```
'</h5>' )  
    oDom:TableInit( 'mytable', hConfig )  
  
    endif  
  
retu nil
```

 **MySQL** SQL command TVWeb Version: 1.0e

Usamos la base de datos dbharbour con tablas para estudiar como: customer, states, sellers

We use the dbharbour database with tables to study such as: customer, states, sellers

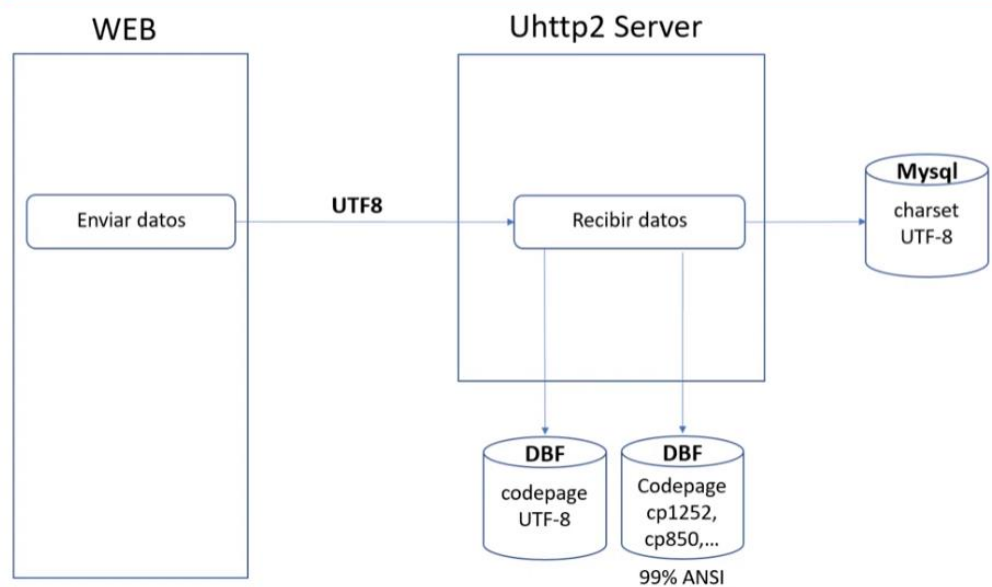
Sql

Coincidencias: 158

id	first	last	street	city	state	married	age
848	Hoechst	Spray	Milwaukee Avenue	Honeoye Falls	NY	1	97
985	Oved	Louman	S Mitchell Dr	Las Vegas	NY	1	91
1037	Christian	Volz	West Davis Street	Richardson	NY	0	99
3031	Sandy	Kline	S. Lake Shore Drive	Palm Desert	NY	1	96
3546	Hans	Warren	W. Bullard	Bromma	NY	0	99

Para terminar este apartado y añadiendo un punto final podríamos siguiendo el último diagrama sobre acceso a base de datos y dbf's

Web → Server



CURL

Curl es una librería de funciones para conectar con servidores para trabajar con ellos. El trabajo se realiza con formato URL. Es decir, sirve para realizar acciones sobre archivos que hay en URLs de Internet, soportando los protocolos más comunes, como http, ftp, https, etc.

Tweb viene con ejemplos para el uso de CURL y no tiene mas relevancia que el uso de CURL por parte de quien quiera usarlo.

Ejemplo de como recuperar imágenes de un servicio host

```

function DoBrwSetData( oDom )

    local cPage          := oDom:Get( 'mypage' )
    local h               := {}
    local aRows          := {}
    local cUrl, uValue, hCurl, n, cError

    cUrl := "https://picsum.photos/v2/list?page=" + cPage + "&limit=10"

    if ! empty( hCurl := curl_easy_init() )

        curl_easy_setopt( hCurl, HB_CURLOPT_URL, cUrl )
        curl_easy_setopt( hCurl, HB_CURLOPT_TIMEOUT, 10 )
        curl_easy_setopt( hCurl, HB_CURLOPT_DL_BUFF_SETUP )
        curl_easy_setopt( hCurl, HB_CURLOPT_USERAGENT, 'Chrome/79' )
    
```

```

curl_easy_setopt( hCurl, HB_CURLOPT_DL_BUFF_SETUP )

curl_easy_setopt( hCurl, HB_CURLOPT_SSL_VERIFYPEER, .f. )
curl_easy_setopt( hCurl, HB_CURLOPT_SSL_VERIFYHOST, .f. )
curl_easy_setopt( hCurl, HB_CURLOPT_NOPROGRESS, .t. )

curl_easy_setopt( hCurl, HB_CURLOPT_FOLLOWLOCATION

n := curl_easy_perform( hCurl )

if n == 0

    uValue := curl_easy_dl_buff_get( hCurl )

    aRows := hb_jsonDecode( uvalue )

else

    uValue := curl_easy_strerror( n )

    oDom:SetError( uValue )


endif

curl_easy_cleanup(hCurl)

endif




oDom:TableSetData( 'mytable', aRows )

retu nil
    
```

 Curl Get images

TWeb Version: 1.09

1 ▾
Load
Clean

Customer list					
Id	Author	Width	Height	Image	Url
2	Alejandro Escamilla	5000	3333		https://unsplash.com/photo
3	Alejandro Escamilla	5000	3333		https://unsplash.com/photo
4	Alejandro Escamilla	5000	3333		https://unsplash.com/photo

Pluggins

Cuando codificamos nuestra web, TWEB por defecto se encarga con la primera línea de definición cargar una serie de librerías y configuraciones iniciales para poder empezar a funcionar.

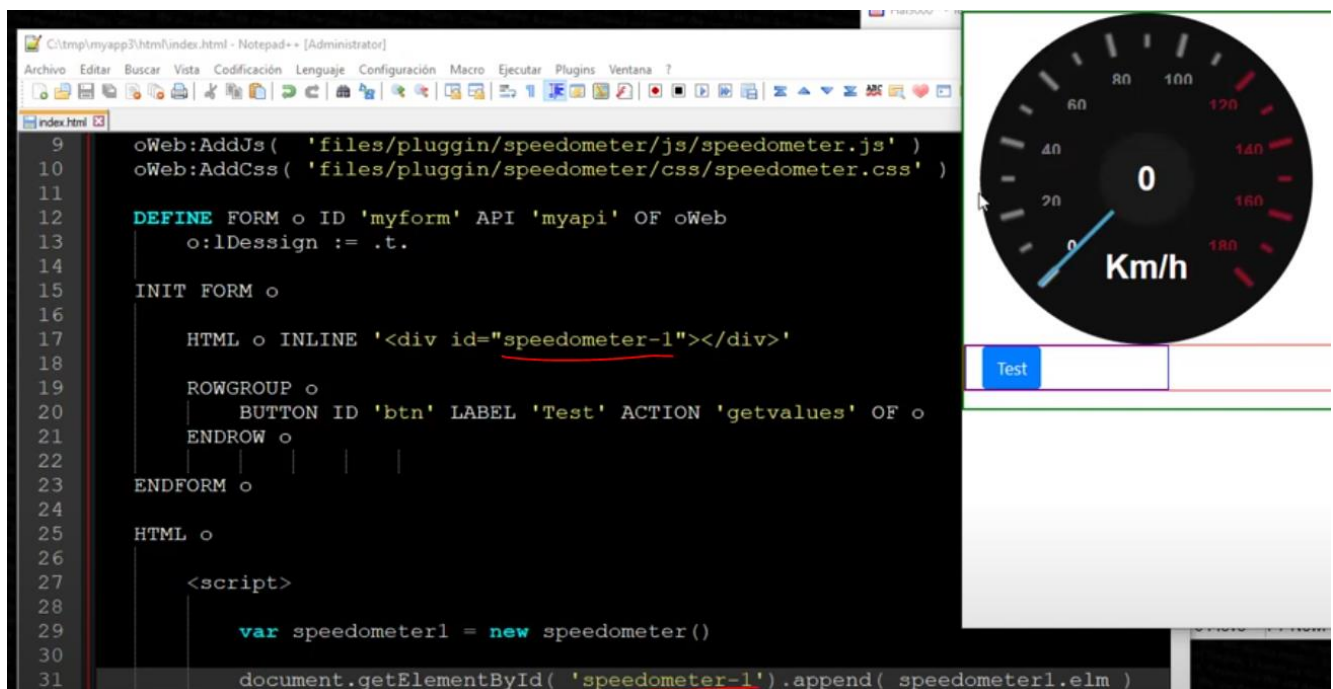
```
DEFINE WEB oWeb TITLE 'Menu'
```

Pero evidentemente queramos realizar una carga de más librerías, muchas de ellas diferentes pluggins que nos gustaría poder usar en nuestra aplicación. Para poder realizarlo tan fácil como hacer las cargas de los ficheros y lo podemos hacer con los métodos del objeto oWeb SetCss() y SetJs()

```
DEFINE WEB oWeb TITLE 'TWeb'

oWeb:AddJs( 'files/pluggin/speedometer/js/speedometer.js' )
oWeb:AddCss( 'files/pluggin/speedometer/css/speedometer.css' )
```

Y dependiendo de cada uno, lo normal es crear un contenedor, y posteriormente vía JavaScript inicializarlo



The screenshot shows a Notepad++ window with the following code:

```
9 oWeb:AddJs( 'files/pluggin/speedometer/js/speedometer.js' )
10 oWeb:AddCss( 'files/pluggin/speedometer/css/speedometer.css' )
11
12 DEFINE FORM o ID 'myform' API 'myapi' OF oWeb
13   o:lDessign := .t.
14
15 INIT FORM o
16
17   HTML o INLINE '<div id="speedometer-1"></div>'
18
19   ROWGROUP o
20     BUTTON ID 'btn' LABEL 'Test' ACTION 'getvalues' OF o
21   ENDROW o
22
23 ENDFORM o
24
25 HTML o
26
27   <script>
28
29     var speedometer1 = new speedometer()
30
31     document.getElementById( 'speedometer-1' ).append( speedometer1.elm )
```

To the right of the code editor, a speedometer UI is displayed. It features a circular gauge with a black face, white markings, and a blue needle pointing to 0. The text 'Km/h' is centered below the needle. A 'Test' button is located below the speedometer.

Finalmente, y con la técnica que desde el backend también podemos inyectar datos, podemos fácilmente manipular nuestro plugin desde nuestro frontend como desde nuestro backend.

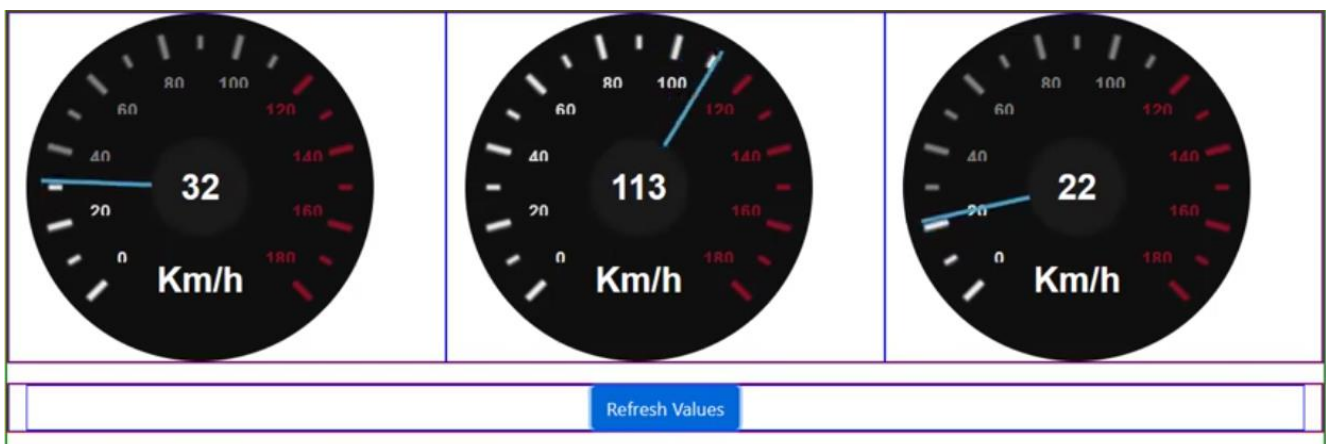
Brevemente y recordando el tema, imaginarnos ahora una simple función en JavaScript

```
function MyControl( dat ) {  
    speedometer1.setPosition( dat[ 'site-a' ] )  
}
```

Desde nuestro backend podríamos llamarla de esta manera

```
14 function DoGetValues( oDom )  
15  
16     local hValues := {}  
17  
18     hValues[ 'site-a' ] := hb_RandomInt( 10, 180 )  
19  
20     oDom:SetJS( 'MyControl', hValues )  
21  
22  
23     return nil
```

Finalmente, podrás gestionar como quieras cada plugin pudiendo de una manera fácil, rápida y limpia este tipos de casuísticas



SSL

Quizás sea este el último tema a tocar, pues entiendo yo que se ha de aplicar cuando ya tienes una aplicación lista para la puesta en marcha y en producción.

Es algo que tenía aparcado por lo comentado y porque no puedes poner un ejemplo operativo, ya que cada uno de ellos necesita su propio certificado que va ligado al servidor y la máquina donde lo uses.

Añadir que no es que sea complicado, pero para la generación de un certificado intervienen varios factores desde dominios, routers, ip, ... y puede ser un poco engorroso. Y si a esto le añadimos la parametrización del firewall, la codificación a tener en cuenta, etc,.... Puede ser un poco costoso al principio, pero solo se trata de tener en cuenta todos estos conceptos.

Siempre comento que este es el último paso a hacer en algún proyecto nuestro. Es bueno hacer pruebas y ver que somos capaces y podemos llegar a este stage, pero para programar la aplicación, sin el SSL es todo más fácil. Al final el concepto es licenciar un proyecto y prepararlo para que salga de nuestra casa.

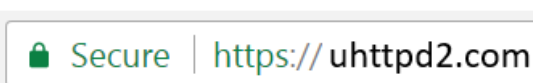
Siguiendo la línea del proyecto enfocado en clave Harbour, quiero seguir haciendo las cosas lo más fácil posible para el usuario, intentando aparcar conceptos y temas técnicos para quien más lo necesite o quiera inmiscuirse.

Existe mucha literatura que explica muy entendible lo que es el SSL y por lo que he visto no tendréis problemas en encontrar información fiable y buena.

Básicamente para entenderlo, cuando vemos en la url `https://` y un candado sabemos que esa url está protegida. Esta protección se basa en un cifrado de punta a punta (server-cliente) y su contenido es inteligible si sniffan la transmisión, todo está encriptado.

Así pues resulta comprensible que nuestras aplicaciones necesiten si o si estos certificados para poder ofrecer garantías a nuestros usuarios.

Using SSL



Not Using SSL



Dominio

Para nuestra aplicación necesitaremos también un dominio. Siguiendo la línea vamos a usar un servido gratuito para crear nuestros dominios. Después de probar unos pocos y hablándolo con programadores que utilizan estos tipos de servicios, me decanto al final por <https://duckdns.org/> por su extrema facilidad de gestionar el dominio a parte que podemos ir renovándolo de muchas maneras incluso desde nuestro propio programa en harbour

Para la siguiente prueba creo un domino p.e → hbweb.duckdns.org

Certificado

Después de mirar varias maneras de realizarlo y para hacerlo de una forma lo más fácil posible para tener un punto de entrada y poder experimentar, me decanto por usar certbot

<https://certbot.eff.org/>

Sencillamente porque es muy fácil crear un certificado que nos sirva. Os aconsejo que os miréis la página, pero a modo resumen os explico como podéis crear un certificado para Windows.

- 1.- Bajaros el software para crear el certificado de https://dl.eff.org/certbot-beta-installer-win_amd64.exe
- 2.- Ejecutar el cmd como administrador
- 3.- Ejecutamos el siguiente comando:

```
C:\WINDOWS\system32> certbot certonly -standalone
```

Importante: Tener el puerto 80 para tu máquina, abriendo puerto de tu router

```
C:\WINDOWS\system32>certbot certonly --standalone
Microsoft Windows [Versión 10.0.19045.2604]
(c) Microsoft Corporation. Todos los derechos reservados.

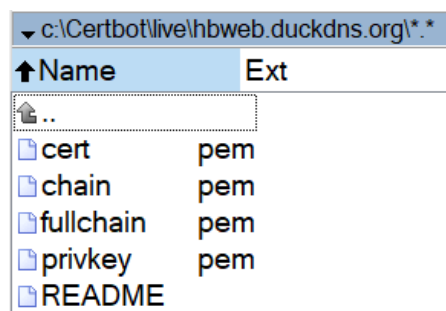
C:\WINDOWS\system32>certbot certonly --standalone
Saving debug log to C:\Certbot\log\letsencrypt.log
Please enter the domain name(s) you would like on your certificate (comma and/or
space separated) (Enter 'c' to cancel): hbweb.duckdns.org
Requesting a certificate for hbweb.duckdns.org

Successfully received certificate.
Certificate is saved at: C:\Certbot\live\hbweb.duckdns.org\fullchain.pem
Key is saved at: C:\Certbot\live\hbweb.duckdns.org\privkey.pem
This certificate expires on 2023-05-28.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

-----
If you like Certbot, please consider supporting our work by:
* Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate
* Donating to EFF: https://eff.org/donate-le
-----

C:\WINDOWS\system32>
```

El sistema instala el certificado en este caso en → C:\Certbot\live\hbweb.duckdns.org\



Y eso es todo para crear un certificado. Podéis usar otras herramientas y experimentar de varias maneras, pero creo que esta es la forma más fácil para empezar con vuestra aplicación.

Copiaremos en el directorio raíz de nuestra aplicación:

- privkey.pem → privatekey.pem
- cert.pem → certificate.pem

App web con Uhttpd2

A continuación, se detalla lo que debemos tener en cuenta en nuestra aplicación para que funcione con SSL

Código

Cuando creamos el servidor en nuestro código, deberemos tener en cuenta 3 cosas

Acción	Código
Indicar el puerto 443	<code>oServer:SetPort(443)</code>
Indicar el nombre del certificado y del fichero de clave privada	<code>oServer:SetCertificate(<privatekey.pem>, <certificate.pem>)</code>
Indicar que la aplicación usara SSL	<code>oServer:SetSSL(.T.)</code>

Compilado / Enlazado

Para usar aplicaciones con HTTPS necesitaremos tener estas 2 dll en la carpeta de nuestra aplicación y las podéis encontrar en la carpeta dll.

- libcrypto-1_1-x64.dll
- libssl-1_1-x64.dll

Esto implica que el exe las necesitará para poder SSL. Si faltase una nos daría error.

En el fichero hbp deberemos especificar que use las librerías SSL

~~-luhttpds2~~

~~-lhssl~~

~~-lhssls~~

~~-llibcrypto-1_1-x64~~

~~-llibssl-1_1-x64~~

Observad que la librería uhttpd pasa a ser la uhttpds2 que es la versión para compilar y usar SSL

En un ejemplo básico el fichero hbp seria el siguiente

```
-n
-wl
-es2
-inc
-head=full
-mt

app\app.prg
app\web_basic.prg

-lIlib
-luhttpds2

-lhbssl
-lhbssls
-lcrypto-1_1-x64
-lssl-1_1-x64

# WAPI_OutputDebugString()
hbwin.hbc
xhb.hbc
```

Test en localhost

Una vez enlazado podemos probar nuestro servidor poniendo en la url

<https://localhost>

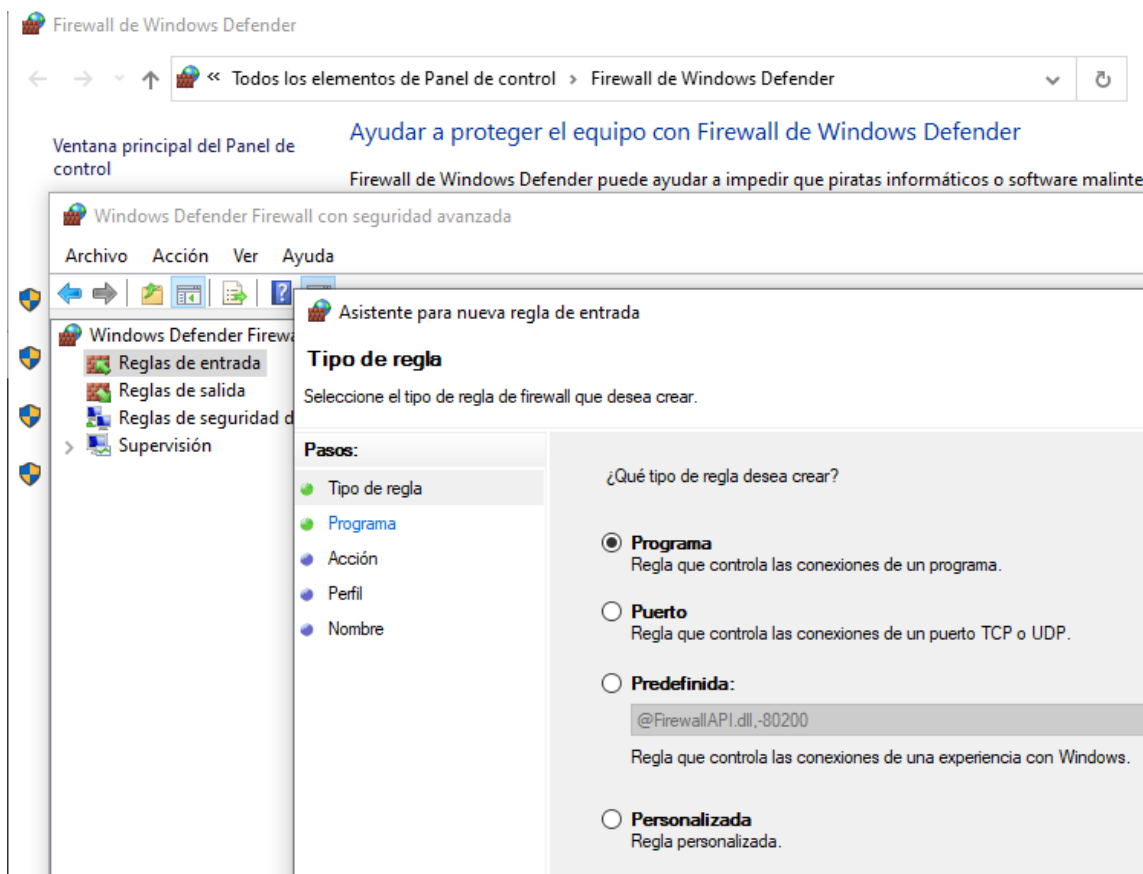
Si se ejecuta es que el server se ha ejecutado correctamente.

Test en Internet

Para poder acceder desde internet:

- Debemos abrir el puerto de nuestro router 443 a nuestra máquina
- Habilitar en nuestro firewall permisos para que esta aplicación pueda tener paso desde fuera a nuestra máquina

En el caso de que queremos ejecutar el server que tengamos ubicado en c:\uhttpd2\samples\20-SSL\app.exe podremos definir el firewall de esta manera.



Asistente para nueva regla de entrada

Programa

Especifique la ruta completa y el nombre del archivo ejecutable del programa con el que coincide esta regla.


Pasos:

- ☒ Tipo de regla
- ☒ Programa
- ☐ Acción
- ☐ Perfil
- ☐ Nombre

¿Se aplica esta regla a todos los programas o a uno específico?

- ☐ **Todos los programas**
La regla se aplica a todas las conexiones en el equipo que coinciden con otras propiedades de reglas.
- ☒ **Esta ruta de acceso del programa:**

 Ejemplo: c:\path\program.exe
 %ProgramFiles%\browser\browser.exe

 Asistente para nueva regla de entrada

Acción

Especifique la acción que debe llevarse a cabo cuando una conexión coincide con las condiciones especificadas en la regla.

Pasos:


- Tipo de regla
- Programa
- Acción
- Perfil
- Nombre

¿Qué medida debe tomarse si una conexión coincide con las condiciones especificadas?

☒ **Permitir la conexión**
Esto incluye las conexiones protegidas mediante IPsec y las que no lo están.

☐ **Permitir la conexión si es segura**
Esto incluye solamente las conexiones autenticadas mediante IPsec. Éstas se protegerán mediante la configuración de reglas y propiedades de IPsec del nodo Regla de seguridad de conexión.

☐ **Bloquear la conexión**

 Asistente para nueva regla de entrada

Perfil

Especifique los perfiles en los que se va a aplicar esta regla.

Pasos:


- Tipo de regla
- Programa
- Acción
- Perfil
- Nombre

¿Cuándo se aplica esta regla?

☒ **Dominio**
Se aplica cuando un equipo está conectado a su dominio corporativo.

☒ **Privado**
Se aplica cuando un equipo está conectado a una ubicación de red privada, como una red doméstica o del lugar de trabajo.

☒ **Público**
Se aplica cuando un equipo está conectado a una ubicación de redes públicas.

 Asistente para nueva regla de entrada

Nombre

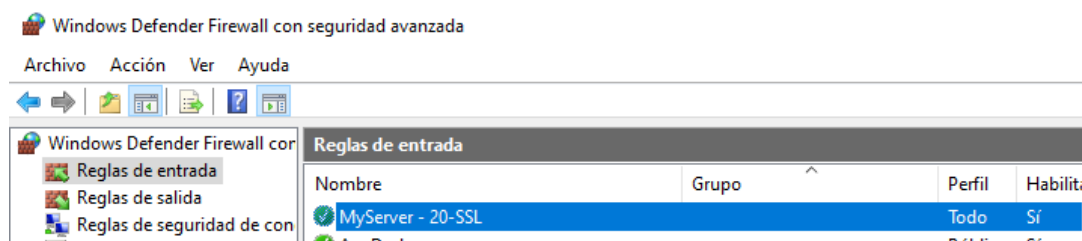
Especifique el nombre y la descripción de esta regla.

Pasos:

- Tipo de regla
- Programa
- Acción
- Perfil
- Nombre

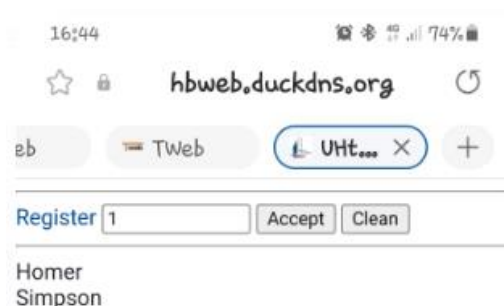
Nombre:

Se habrá creado una regla de entrada en nuestro firewall

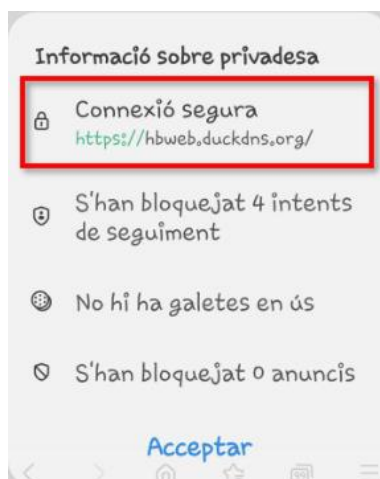


Una vez abierto desde cualquier máquina o nuestro móvil (probadlo con datos, no con vuestra wifi) especificáis vuestro dominio:

<https://hbweb.duckdns.org>















Si pulsamos en el candado podremos ver que es un certificado válido y que nuestra aplicación se ejecuta correctamente en SSL.



Y si todo está correcto, esto debería ser todo lo mínimo necesario para poder tener nuestras aplicaciones con certificado SSL.

Píldoras

En la página del repositorio de documentación → <https://carles9000.github.io/> podreis encontrar mucha información del proyecto, pero pongo aquí las píldoras (sesiones que hice en video) donde reflejen de manera práctica muchos de los puntos hablados y otros que son importantes a tener en cuenta.

Descripción	Tmp.	Píldora
Como compilar una app con Uhttpd2	3:28	
Como compilar una app con Uhttpd2 y TWeb	2:33	
TWeb – Diseño de pantallas	46:16	
TWeb – Conectar pantallas con la API	45:21	
TWeb – Conectar con API via Javascript	9:45	
TWeb – Conectar con Javascript via API	17:15	
TWeb – Manejo de browse	50:30	
TWeb – Manejo de dialogos	27:02	
TWeb – Seguridad – Token	41:22	
TWeb – Seguridad – Sessions	1:03:00	
TWeb – Codepage	46:03	
TWeb – Menus	10:37	

Conclusiones

La web es muy dura de aprender a controlar, muchas cosas a saber, controlar.... Es un monstruo que te sale a atacar y no tienes armas para poder luchar con ella. La web tiene su código, su estructura, su arquitectura, su manera de ser. Y no me gusta ver a una generación de grandes programadores como se quedan al lado y se les aparta del camino.

Tweb es una librería diseñada para ayudar a los programadores Harbour que dan el salto a la web. Mientras la usamos nos iremos familiarizando con diferentes conceptos de la web, css, funcionamiento de Bootstrap... Llegará el momento en que ya no la necesitemos o la queramos seguir usando juntamente con código nativo. La velocidad de diseño de páginas y la productividad que nos ofrece es muy alta, por lo que el gran problema que teníamos al inicio de todo el proceso ya lo tendremos controlado.

Esta versión de TWeb para Uhttpd2 es fruto de la evolución que ha ido tomando todo el proyecto que empecé con el mod-harbour, acabando con uhttpd2, en la que todo el sistema ofrece un servidor, diseñador de pantallas y control de flujo de la web que hace una manera fácil, rápida y real de poder crear tu aplicación para la web.

No es fácil crear todo un sistema como esta y encajar las piezas para que les pueda ser de ayuda a otros y menos crear un manual que no se haga largo, aburrido, sin sentido...

Espero que podáis sacar provecho y podáis crear estas aplicaciones que tanto os demandan de una manera fácil, rápida y divertida. Este ha sido a mi aportación a harbour programando el mod, el uhttpd2 y tweb. Me voy en paz 😊

Lo hemos comentado y creo que cualquiera que venga de Harbour ya puede de manera muy fácil crearse sus primeros módulos funcionales sin tener idea de web, de sus flujos, de sus servidores, de html, de javascripts, css,...

Estoy contento , quizás estos últimos 3 años han valido la pena para poder ayudar a unos cuantos 😊



Saludos y mucha suerte en vuestras apps.



Charly Aubia Floresví