

Trabajo Fin de Carrera
Grado en Ingeniería de las Tecnologías de
Telecomunicación

Desarrollo de una aplicación web con Node.js
para la monitorización en tiempo real de un
electrocardiograma

Autor: Gabriel Rodríguez Flores

Tutor: Fernando Muñoz Chavero

Departamento de Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017



Trabajo Fin de Carrera
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de una aplicación web con Node.js para la monitorización en tiempo real de un electrocardiograma

Autor:

Gabriel Rodríguez Flores

Tutor:

Fernando Muñoz Chavero

Profesor titular

Codirector:

Hipólito Guzmán Miranda

Profesor Contratado Doctor

Dep. de Teoría de la Señal y Comunicaciones

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Carrera: Desarrollo de una aplicación web con Node.js para la monitorización en tiempo real de un electrocardiograma

Autor: Gabriel Rodríguez Flores

Tutor: Fernando Muñoz Chavero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2013

El Secretario del Tribunal

A mi familia por estar siempre ahí

*A mis padres por darme la vida y
la Fe*

*A mis hermanos por toda la vida
aguantándome*

Agradecimientos

En primer lugar, agradecer a Fernando y Poli darme la oportunidad y la libertad de desarrollar el proyecto, siempre consiguiendo sacar algo de tiempo para atenderme. Por presentarme a CieNTi, a quien le doy las gracias por la base, las herramientas y todas las explicaciones relativas, o no, al contenido del proyecto.

También agradecer a mis compañeros y amigos Fran y Belén, que las intensificaciones no nos separen y hagan *merge* en el *máster*. Y por intensificaciones llegamos a Pedro y Ana, gracias por *sec* el mejor grupo de electrónicos que haya pasado por la escuela.

No puedo olvidarme de Ángela porque la conocí al entrar y ha sido fuente de inspiración para llegar hoy donde estoy.

Y por último, y más importante por ello, gracias a mis padres por los esfuerzos que hacen día a día para que haya podido cursar estos estudios y que tenga una buena vida. Gracias a mis hermanos porque están ahí, aceptándome como soy y aguantándome porque eso les ha tocado.

Resumen

Este trabajo se centra en el desarrollo Software de una aplicación web propuesta para monitorizar los resultados que recoge de una FPGA, la cual ha sido diseñada para recoger los valores de tensión a través de unos electrodos y procesar los datos para extraer unos resultados limpios y legibles. Para ello se trabaja sobre una PSoC (Programmable System-on-Chip).

La aplicación desarrollada tiene como cometido representar, de forma gráfica, los resultados de un electrocardiograma en tiempo real, aprovechando el procesamiento en paralelo de un circuito electrónico, a través del diseño realizado en la FPGA de la ZedBoard.

Para el desarrollo independiente de esta aplicación, se ha ideado un plan de simulación a través de la carga de ficheros, con el fin de que, una vez desarrollados ambos bloques Hardware y Software, la conexión entre ellos se realice de la forma más sencilla y directa posible. Por tanto, la aplicación desarrollada está estructuralmente bien organizada, siendo de fácil modificación y ampliación de sus funcionalidades (modular).

Abstract

This work focuses on the software development of a web application proposed to show the results of an FPGA, which has been designed to collect the voltage values through electrodes and process the data to extract clean and readable results . We work on a PSoC (Programmable Chip-System) to develop this project.

The application developed has the purpose of graphically representing the results of an electrocardiogram in real time, taking advantage of the parallel processing of an electronic circuit, through the design realized in the ZedBoard's FPGA.

For the independent development of this application, a simulation plan has been devised through the upload of files. Once both hardware and software blocks have been developed, the connection between them is performed as simplest as possible. Therefore, the application developed is structurally well organized, being easy to modify and extend its functionalities (modular).

Índice

Agradecimientos	ix
Resumen	xi
Abstract	xiii
Índice	xiv
Índice de Tablas	xvi
Índice de Figuras	xviii
Notación	xxi
1. Introducción	11
1.1. <i>Motivación</i>	11
1.2. <i>Estado del arte</i>	11
1.2.1. Electrocardiograma (ECG)	11
2. Alcance	14
2.1. <i>Propósito</i>	14
2.2. <i>Objetivos</i>	14
2.3. <i>Requisitos</i>	15
2.4. <i>Estructura del trabajo</i>	15
2.4.1. Introducción a la memoria	15
3. Sistema de desarrollo	16
3.1. <i>Xilinx Zynq®-7000 All Programmable SoC</i>	16
3.2. <i>ZedBoard</i>	16
3.3. <i>Xilinx</i>	16
4. Herramientas de desarrollo	17
4.1. <i>Comparativa de tecnologías</i>	17
4.1.1. Aplicación	17
4.1.2. Documentación: Markdown vs Latex vs Word	25
4.2. <i>Descripción de la tecnología escogida</i>	26
4.2.1. Lenguaje de programación: Node.js	26
4.2.2. Diseño de la web	30
4.2.3. Documentación: Markdown	31
4.2.4. Gestión proyecto: Control de versiones	31
4.2.5. Generadores de documentación	35

5. Preparación del sistema	36
5.1. Creación del entorno de desarrollo	36
5.1.1. Xillinux	36
5.1.2. Instalación de GIT	39
5.1.3. Instalación de Node	39
5.2. Estructura del proyecto	41
5.2.1. Estructura de los directorios	41
5.2.2. Estructura de las ramas de desarrollo	43
6. Aplicación Electrocardiograma	44
6.1. División en bloques	44
6.1.1. Esquema general	44
6.1.2. Intercambio de eventos	45
6.1.3. Obtención de datos: parser	46
6.1.4. Generador de datos y buffer de envío	47
6.1.5. Buffer de recepción (cliente)	48
6.2. Referencia técnica	48
6.3. Manual de usuario	48
6.4. Test y pruebas de funcionamiento	52
7. Conclusiones	54
7.1. Objetivos cumplidos	54
7.2. Mejoras	54
7.3. Dificultades	55
7.4. Ampliaciones futuras	55
Referencias	58

ÍNDICE DE TABLAS

Tabla 1 - Comparativa de lenguajes de programación para aplicaciones web	19
Tabla 2 - Comparativa de SVN frente GIT para el control de versiones	32

ÍNDICE DE FIGURAS

Figura 1 - Ondas del electrocardiograma	12
Figura 2 - Ejemplo de aspecto de una gráfica con <i>Smoothie</i>	22
Figura 3 - Ejemplo de aspecto de una gráfica con <i>CanvasJS</i>	23
Figura 4 - Menú de opciones disponibles en <i>Plotly</i>	23
Figura 5 - Ejemplo de aspecto de una gráfica con <i>Plotly</i>	24
Figura 6 - Ejemplo de aspecto de una gráfica con <i>C3</i>	25
Figura 7 - Forma tradicional de acceso al servidor	27
Figura 8 - Forma de Node.js de acceso al servidor	27
Figura 9 - Intercambio de información cliente-servidor usando Sockets	29
Figura 10 - Esquema de uso de los comandos de GIT	33
Figura 11 - Visualización del desarrollo y unión de las ramas	34
Figura 12 - Lista de modificaciones de un commit	34
Figura 13 - Menú de comandos de GitKraken	34
Figura 14 - Comandos para la gestión de las GPIO	38
Figura 15 - Ejemplo probando encender y apagar un LED de la ZedBoard	39
Figura 16 - Diagrama de bloques de la secuencia lógica del servidor	44
Figura 17 - Eventos intercambiados para la representación directa	45

Figura 18 - Eventos intercambiados para la representación directa	46
Figura 19 - Pseudocódigo simple del analizador de parsers	47
Figura 20 - Esquema de funcionamiento del buffer implementado en el cliente	48
Figura 21 - Portada de la aplicación web con <i>responsive design</i>	49
Figura 22 - Página <i>ECG</i> donde se muestra la gráfica	49
Figura 23 - Página de información de la aplicación	49
Figura 24 - Página no encontrada, muestra de información de errores	50
Figura 25 - Estado inicial de los botones	50
Figura 26 - Ejemplo de botón habilitado	50
Figura 27 - Ejemplo de botón deshabilitado	50
Figura 28 - Lista de ficheros cargados del servidor	50
Figura 29 - Contenedor precargado donde se muestra la gráfica	51
Figura 30 - Ejemplo de representación directa de la gráfica	51
Figura 31 - Ejemplo de estado representando la gráfica	52
Figura 32 - Ejemplo de gráfica parada / pausada	52

Notación

CSS	Cascade Style Sheet
DOM	Document Object Model
ECG o EKG	Electrocardiogram o Electrocardiogram
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
HTML	HyperText Markup Language
MVC	Model View Controller
NPM	Node Package Manager
SoC	System on Chip
SVG	Scalable Vector Graphics

1. INTRODUCCIÓN

If you can dream it, you can do it

Walt Disney.

1.1. Motivación

Anterior a los estudios de Grado en Ingeniería de las Tecnologías de Telecomunicación, realicé un ciclo formativo de grado superior en desarrollo de aplicaciones web. Si bien es cierto que me gustaba mucho este campo y me orientaba claramente a la intensificación de telemática, llegado el punto de elegir que hacer, escogí la intensificación de electrónica para curiosear otros temas y ver las telecomunicaciones desde un punto de vista global, donde todo esté conectado entre sí.

Así que este trabajo se me presentó como una opción para desarrollar un dispositivo completo, desde su producción hardware a través del lenguaje descriptivo *VHDL* hasta la programación de una aplicación para su visualización, sin olvidar que se implementa un procesamiento a la señal para limpiar el ruido; este proyecto unía todas las ramas, y todo esto tratando de un tema del que siempre me he sentido atraído: la telemedicina.

Puesto que el proyecto era de gran envergadura, este se dividió, y escogí encargarme de la parte de la aplicación web, donde me encontraba continuamente pensando en utilizar lo último en tecnología, tanto es así que, incluso con algo avanzado en el proyecto, volví a empezar de cero para aprender y utilizar la tecnología más indicada para esta aplicación: NodeJS.

1.2. Estado del arte

1.2.1. Electrocardiograma (ECG)

1.2.1.1. Que és / Que mide y para que sirve

El electrocardiograma (ECG o EKG) es el registro de la actividad eléctrica del corazón que se obtiene a través de unos electrodos colocados sobre la piel del paciente. Esta actividad, representada gráficamente en función del tiempo, dibuja un trazado periódico que permite la visualización de los distintos estados en los que se encuentra el órgano. Es de gran utilidad no solo para el control y monitorización del ritmo cardíaco, sino, fundamentalmente, para el diagnóstico de las enfermedades cardiovasculares, alteraciones metabólicas y la predisposición a una muerte súbita cardíaca.

1.2.1.2. Como se mide

Para poder medir esta actividad eléctrica, se usan unos electrodos que se colocan en puntos estratégicos para capturar con precisión las pequeñas ondas que forman el trazado del ECG. Para ello se registra la diferencia de potenciales eléctricos, lo cual se conoce como *derivación*. Esta diferencia de potencial será medida entre dos electrodos, derivación bipolar, o bien entre un electrodo y un punto virtual, derivaciones monopolares. Para explicar esto, en [la Web del Electrocardiograma](#) nos ponen *el simil del autobús*:

Imaginemos un autobús colocado en el centro de una nave industrial. Esta nave tiene 12 ventanas, desde las cuales, las personas que estén fuera, pueden mirar al autobús.

Si desde cada ventana se tomase una fotografía del autobús, tendríamos 12 fotografías distintas, pero todas del mismo autobús.

Algo similar son las derivaciones cardiacas en el Electrocardiograma. Cada derivación es una "fotografía" diferente de la actividad eléctrica del corazón.

Así entendemos que cada una de estas señales (derivaciones), es una visión particular, un ángulo concreto desde el cual veremos el corazón. El hecho que nos pongan de ejemplo 12 fotografías vienen porque el uso más común de un electrocardiograma son 12 derivaciones:

- Derivaciones I, II y III.
- Derivaciones aVR, aVL, aVF.
- Derivaciones V1, V2, V3, V4, V5 y V6.

1.2.1.3. Cómo se interpreta

Anteriormente se ha mencionado que la redundancia en la toma de tensiones es para medir con precisión cada una de las *ondas* que componen el trazado de tensión del corazón. Estas ondas, denominadas P, Q, R, S y T por Einthoven, son objeto de estudio, analizando sus características para el diagnóstico de las patologías y enfermedades cardiovasculares.

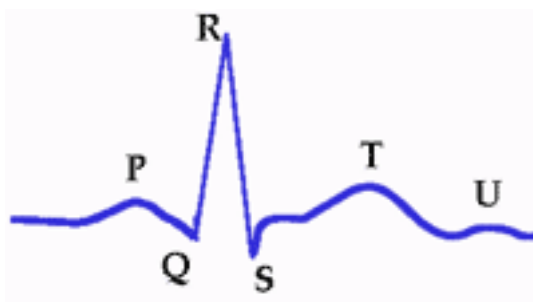


Figura 1 - Ondas del electrocardiograma

Para nuestro alcance, podemos resumir la descripción de cada onda con los siguientes intervalos:

- **Onda P** es la primera onda del ciclo cardiaco. Representa la despolarización de las aurículas.
- **Complejo QRS** está formado por un conjunto de ondas que representan la despolarización de los ventrículos.
- **Onda T** representa la repolarización de los ventrículos.

1.2.1.4. Coste, equipamiento, disponibilidad, uso, etc...

La tendencia actual de la tecnología médica es monitorizar los resultados de forma remota, aprovechando las telecomunicaciones para la transmisión de los datos que se tomen del paciente a través del dispositivo, en nuestro caso, un electrocardiógrafo.

En la actualidad, existen muchos electrocardiógrafos de diversa índole dependiendo del número de derivaciones, número de canales que se monitorizan, de si el procesamiento es analógico o digital, si la monitorización es a papel o digital sobre una pantalla, así como de las características dentro de cada uno de estas opciones, tales como la frecuencia de muestreo, filtros y/o tiempo de respuesta, que es donde se le da el mayor enfoque a este trabajo. Otro requisito es buscar economizar estos dispositivos ya que, a pesar de que existen algunos equipos portátiles y económicos, estos muestran un solo canal en una pequeña pantalla y cuentan con tres electrodos: dos de ellos *positivo* y *negativo* para la diferencia de potencial eléctrico, y un tercero como toma de referencia.

2. ALCANCE

Olvídense de lo que saben y crean en lo que ven...

Dai Vernon.

2.1. Propósito

Este trabajo es parte de un proyecto conjunto, que tiene como fin el diseño de un dispositivo para la lectura de un electrocardiograma, con su posterior visualización a través de una aplicación web. Para la división de este proyecto se ha tomado la decisión de separarlo por ámbito de desarrollo tecnológico, la primera parte dedicada al *Hardware*, es decir, al desarrollo electrónico del dispositivo, la segunda dedicada al *Software*, la aplicación final desde la que un usuario podrá ver y controlar el dispositivo electrónico desarrollado.

La primera parte realiza la captura de valores analógicos de tensión proveniente de un circuito electrónico que posee unos electrodos para poder medir las magnitudes físicas de la actividad eléctrica del corazón. Tras recibir los valores analógicos de tensión, se procesan las señales recibidas con un algoritmo concreto y una vez adquiridos datos distinguibles, se procede a convertir estos valores de tensión a valores digitales a través de un conversor analógico-digital. Todo esto implementado a través de un circuito digital descrito en una FPGA.

La segunda parte se encarga del despliegue y creación de una aplicación o página web. Esta aplicación montada sobre la placa de desarrollo, recibe los datos que se generan en formato digital, los gestiona realizando el menor número de operaciones posibles y las envía al cliente de la web. Estas muestras serán representadas en una gráfica dinámica, apta para mostrar varios canales.

Estas dos partes irán comunicadas a través de dos FIFO (bloques *First In First Out*) una para lectura del micro sobre la FPGA y otra para escritura. Todo esto montado sobre una placa de desarrollo que nos permite el despliegue y conexión de estas dos partes independientes del proyecto.

2.2. Objetivos

En este documento se detalla el desarrollo *Software* del proyecto mencionado, la segunda parte de la división que acabamos de comentar.

El objetivo de este trabajo de fin de grado consiste en el desarrollo de una aplicación web que sea modular, que esté bien estructurada y sea dinámica, de fácil ampliación funcional; y *responsive*, es decir, que se adapte al dispositivo con el que se ejecute la aplicación como si fuera diseñada para ese tamaño concreto.

Está previsto que la aplicación ofrezca las opciones de configuración del dispositivo, permita seleccionar diferentes opciones de monitorización, tanto en la actividad en tiempo real como en la carga de datos de un fichero o base de datos, e implemente diversas funcionalidades de análisis de resultados.

En adición a la base de la aplicación, se pretende conseguir que las operaciones sean lo más simples posibles, aprovechando que todo el proceso de la señal se realice en el circuito electrónico, con la ventaja que supone tratar los diferentes canales simultáneamente, en un procesamiento individual paralelo, algo que no es posible con un microprocesador.

2.3. Requisitos

- Funcionalidad
 - **F.1** Muestre en una gráfica los datos en tiempo real
 - **F.2** Guarde y cargue los datos en distintos formatos
 - **F.3** Analice los datos para mostrar resultados al momento
 - Mostrar el nivel de tensión
 - Mostrar la frecuencia cardíaca
 - **F.4** Gestión de usuarios para el control de los registros
- Desarrollo

En cuanto al desarrollo se requiere una elaboración sencilla y ordenada, que sea de fácil ampliación. Por lo tanto, es necesario un desarrollo modular, es decir, la división de la lógica de la programación en módulos independientes, como si fueran subprogramas, de manera que cada uno de ellos realice una función reutilizable. Así mismo, esta idea llevada al conjunto del proyecto, hace necesaria el desarrollo en bloque MVC (del inglés *Model-View-Controller*) que separa el trabajo en bloques de desarrollo: la vista para el aspecto de la aplicación, el modelo para el acceso a la base de datos y el controlador el eje principal del programa que se encarga de llamar a los módulos para ser ejecutados cuando correspondan.

2.4. Estructura del trabajo

2.4.1. Introducción a la memoria

En este documento se describe, en primer lugar, el sistema con el que se ha desarrollado la aplicación, tanto las herramientas físicas como las tecnologías que han intervenido en el desarrollo de la misma. En segundo lugar, se detallan las entrañas del proyecto, la lógica, las funciones implementadas, el control, los eventos y todo lo que se ha desarrollado en él.

Tras conocer el proyecto, se muestra, a modo de manual, una guía de uso de la aplicación desarrollada. Para finalizar se concluye con las propuestas de mejora, el trabajo que ha quedado pendiente y las causas del incumplimiento de los requisitos del trabajo.

3. SISTEMA DE DESARROLLO

A wizard is never late, nor is he early, he arrives precisely when he means to
Gandalf - The Lord of the Rings: The Fellowship of the Ring

En este apartado se introduce el sistema físico sobre el que vamos a trabajar, y puesto que el contenido de este trabajo no toca en profundidad el uso de ese equipo, pero sí usa de él y el objetivo de desarrollarlo en su procesador, se describe brevemente. Para ello, cito el trabajo de fin de grado de *Jesús Fernández Manzano*, en el cual explica al detalle las características de esta placa de desarrollo.

3.1. Xilinx Zynq®-7000 All Programmable SoC

La familia Zynq®-7000 All Programmable SoC integran un procesador ARM® con un hardware programable de una FPGA, permitiendo de esa manera tener la funcionalidad mixta de los análisis hardware integrando una CPU en un solo dispositivo. En concreto, la Zynq-7000 está equipada con el procesador ARM Cortex-A9 de doble núcleo, que integra la lógica programable de Artix-7 o Kintex®-7 con una arquitectura de 28nm. Estos dispositivos SoC son altamente escalables, lo que los hace ideales para el desarrollo de prototipos.

3.2. ZedBoard

ZedBoard es una placa de desarrollo de bajo coste para la Zynq®-7000 de Xilinx, que contiene todo lo necesario para crear un entorno, o sistema operativo. Cuenta con muchos conectores, interruptores para el control de entrada/salida tanto físicas como lógicas de fácil acceso, tanto desde la FPGA, como desde el procesador.

3.3. Xilinx

Xilinx es una distribución del sistema operativo Linux, el cual está basado en Ubuntu 12.4 LTE. Este permite integrar el manejo del procesador y la FPGA de forma gráfica como si de un ordenador común se tratara. Para implementar la comunicación entre estos dos elementos, se realiza a través de dos bloques FIFO, una hacia una dirección (escritura desde la FPGA, lectura desde el procesador) y la otra en dirección contraria.

4. HERRAMIENTAS DE DESARROLLO

All that is gold does not glitter

J. R. R. Tolkien

Como el trabajo a desarrollar es con fines académicos, se detalla a continuación no solo las tecnologías usadas para el desarrollo de la web, sino también aquellas que forman parte de lo aprendido durante el desarrollo del proyecto y que son de utilidad para otros futuros, y que han sido relevantes en el curso del trabajo.

4.1. Comparativa de tecnologías

4.1.1. Aplicación

4.1.1.1. Lenguaje de programación

Esta es la primera pregunta que hay que hacerse a la hora de comenzar un proyecto de esta índole: *¿Qué lenguaje de programación debo de usar?* Para ello existen muchos factores a tener en cuenta. Principalmente la naturaleza de la aplicación final dará pistas sobre que lenguaje es más afín, pero en definitiva no existe el lenguaje perfecto definitivo, de ser así, sólo existiría uno. A continuación, se listan los lenguajes de programación más usados para el ámbito que estamos abarcando, una aplicación que se ejecute en un navegador web.

4.1.1.1.1. PHP

Es la opción más rápida y sencilla, *PHP* se creó como una herramienta para añadir funcionalidad y dinamismo a una página web. Por su antigüedad, la mayoría de las páginas web usan de este potente lenguaje, incluso páginas de tan envergadura como Facebook o Wikipedia.

– *Ventajas:*

- Software gratuito lanzado bajo la licencia PHP
- Fácil de aprender (curva de aprendizaje corta)
- Gran comunidad de usuarios y desarrolladores
- Proporciona una amplia base de datos de apoyo
- Ofrece un gran número de extensiones y códigos de fuente disponibles
- Permite la ejecución de código en entornos restringidos
- Ofrece administración de sesiones nativas y API de extensión
- Una gran alternativa para competidores como ASP de Microsoft (Active Server Pages)
- Se puede implementar en la mayoría de los servidores web
- Funciona en casi todos los sistemas operativos y plataformas
-

– *Inconvenientes:*

- No es adecuado para realizar aplicaciones de escritorio
- El manejo de errores es tradicionalmente pobre
- Los parámetros de configuración globales pueden cambiar la semántica del lenguaje, complicando el despliegue y la portabilidad
- Generalmente se considera menos seguro que los otros lenguajes de programación

4.1.1.1.2. Ruby

Es uno de los lenguajes de programación que mayor crecimiento ha experimentado en los últimos años, es un lenguaje de propósito general que se volvió extremadamente famoso gracias al framework web Ruby On Rails.

– *Ventajas:*

- De código abierto
- Funciona en múltiples plataformas
- Puede ser integrado en Hypertext Markup Language (HTML)
- Ofrece encapsulación de métodos de datos dentro de objetos
- Los programas escalables y grandes escritos en Ruby son fáciles de mantener
- Tiene una sintaxis limpia, fácil y flexible
- Capacidad para escribir aplicaciones multi-threaded con una API sencilla
- Capaz de escribir bibliotecas externas en Ruby o C
- Mejores características de seguridad
- Potente manejo de cadenas

– *Inconvenientes:*

- Carece de recursos informativos
- Menor tiempo de procesamiento (tiempo de CPU) en comparación con otros lenguajes de programación
- El desarrollo y las actualizaciones son más lentos

4.1.1.1.3. Python

El código de Python se asemeja al pseudo-código como todos los lenguajes de scripting. El diseño elegante y las reglas de sintaxis de este lenguaje de programación lo hacen bastante legible incluso entre los equipos de desarrollo multi-programador. Apoya múltiples formas de construir la estructura y los elementos de los programas informáticos, incluyendo la programación orientada a objetos y funcional

– *Ventajas:*

- Fácil y rápido de aprender
- Se ejecuta en múltiples sistemas y plataformas
- Sintaxis legible y organizada
- Ofrece prototipado rápido y capacidades semánticas dinámicas
- Gran apoyo comunitario
- Reutilización mediante la implementación cuidadosa de paquetes y módulos

- Orientado a Objetos
- *Inconvenientes:*
 - Realmente no funciona el multi-procesador muy bien
 - Pequeño grupo de desarrolladores de Python en comparación con otros lenguajes
 - Limitaciones de la capa de acceso a la base de datos
 - Es más lento que otros lenguajes

4.1.1.1.4. Node

Node.js es una librería y entorno de ejecución de E/S dirigida por eventos y por lo tanto asíncrona que se ejecuta sobre el intérprete de JavaScript creado por Google V8. La idea principal de este lenguaje es el uso no-bloqueante para permanecer ligero y eficiente en la superficie del uso intensivo de datos en tiempo real de las aplicaciones que se ejecutan en dispositivos distribuidos.

- *Ventajas:*
 - Es rápido, es muy rapido
 - El IO asíncrono impulsado por eventos ayuda a la gestión simultánea de peticiones
 - Como no necesita de un servidor debajo, puede arrancar varias aplicaciones en el mismo servidor
 - Tiene una gran comunidad de desarrollo muy activa (en GitHub tienen muchos códigos compartidos)
 - Node package modules (npm) tiene un gran número de módulos y sigue creciendo.
 - Escalabilidad de red gracias a las funciones de *callback*
 - El uso de un mismo lenguaje tanto para la programación en el cliente (Front-End) y en el servidor (Back-End)
 - Optimiza los recursos del servidor
 - Un dato curioso: la web *Linkedin* redujo de 30 a 3 servidores con la migración de Ruby a Node
 - Ofrece un entorno de tiempo de ejecución de código abierto, por lo que admite el almacenamiento en caché de módulos individuales
- *Inconvenientes:*
 - Inadecuado para aplicaciones web grandes y complicadas, no es compatible con ejecuciones multi-thread
 - No soporta base de datos relacionales
 - Nuevos conceptos con las funciones callback

Tabla 1 - Comparativa de lenguajes de programación para aplicaciones web

	Node.js	Ruby	Python	PHP
Lenguaje	JavaScript	Ruby	Python	PHP
Motor	V8	YARV	cPython	Apache
Entorno	Módulos del core	Librerías estandar	Librerías estandar	Librerías estandar
Framework	Express	Rails	Django	Laravel

4.1.1.2. Lenguajes descriptivos

4.1.1.2.1. Estructura de la web

Sin poder decir que es una comparación, vamos a evaluar dos formas posibles de describir el contenido de la aplicación web. No es una comparación puesto que *HTML* es el lenguaje estándar de descripción para páginas web, en cambio *PUG* es tan solo un motor de plantilla que posee Node.js para escribir lo mismo de manera más simple, rápida y sencilla, con lo que construirá el *HTML* tal como si lo hubiéramos escrito como tal.

Además de simplificar, *PUG* nos permite definir variables y bloques de contenido que nos facilita dinamizar y hacer más modular la estructura de la web.

Para no entrar mucho en detalles, pues es ciertamente fácil e intuitivo, se expone un simple ejemplo a modo de explicar lo mucho que se simplifica.

- Donde en *HTML* escribiríamos:

```
<body>
  <div id="id" class="class">
    <p>Hola</p>
  </div>
</body>
```

- En *PUG* es equivalente a

```
body
  #id.class
  p Hola
```

4.1.1.2.2. Diseño de la web (Responsive Design)

Al igual que pasa con HTML, para dar formato a una página, existe el estándar CSS, el cual tan solo se ha usado en el proyecto de forma nativa para dar un estilo específico a algunos elementos concretos que se escapan de las librerías CSS utilizadas. De entre las librerías *responsive* vamos a comparar las dos siguientes:

1.1.1.1.1 Bootstrap

La librería bootstrap es la más popular para el desarrollo de *responsive design*, está muy extendida y es utilizada en la mayoría de páginas. Además, posee una extensa cantidad de plantillas, algunas gratuitas, otras que pueden ser adquiridas mediante la compra de los derechos de uso.

– *Ventajas:*

- es un framework gratuito para un desarrollo web más rápido y fácil
- Incluye plantillas de diseño basadas en HTML y CSS, así como complementos JavaScript opcionales
- Da la capacidad de crear fácilmente diseños *responsive*
- Se adapta a las pantallas de los teléfonos, tablets y ordenadores de escritorio
- Permite diseñar la página por defecto para un tamaño móvil, dando a la web un aspecto como si fuera una *app* (aplicación móvil)

- Es compatible con todos los navegadores modernos (Chrome, Firefox, Internet Explorer, Safari y Opera)
- *Inconvenientes:*
 - Parte de Bootstrap es manejado por Javascript (no solo CSS)
 - Usa de la librería JQuery de Javascript

1.1.1.1.2 W3CSS

Es un nuevo framework creado por la web *W3Schools* construido para *responsive design*

- *Ventajas:*
 - Más pequeño y más rápido que otros frameworks CSS
 - Más fácil de aprender y más fácil de usar que otros frameworks CSS
 - Utiliza sólo CSS estándar (no usa jQuery o JavaScript)
 - Acelera y simplifica el desarrollo web, ideal para prototipos
 - Es compatible con todos los navegadores modernos (Chrome, Firefox, Internet Explorer, Safari y Opera)
 - Se adapta a las pantallas de los teléfonos, tablets y ordenadores de escritorio
 - Es de W3School, es de un valor añadido por ser sinónimo de calidad y tener detrás una comunidad de desarrollo
- *Inconvenientes:*
 - Es muy básica, está muy limitada.

4.1.1.3. Librería para las gráficas

Se podría pensar que este apartado es meramente gráfico, y como tal debería estar incluido en el apartado anterior, pero he decidido ponerlo a parte debido a su importancia para la aplicación porque, dejando aparte el aspecto gráfico en el que se mostrarán los datos, cada una de estas gráficas almacena los datos de manera distinta, teniendo que cambiar el formato de los datos adaptándonos a librería que vayamos a utilizar.

Librería de este tema hay muchísimas, de las cuales se ha hecho una selección, se ha probado cada una de ellas para hacer el estudio de simplicidad, eficiencia y personalización. En las distintas ramas de GIT, de las que se hablará más adelante de su organización, se pueden ver las pruebas que se han realizado, su funcionamiento y la estética que se consigue.

4.1.1.3.1. Smoothie

Smoothie aparece como una librería para Javascript orientada a la representación en streaming, es decir, para una transmisión continua de datos sin interrupción. Es simple, se configura rápido, pero es poco personalizable, orientada a visualizar los datos recibidos en tiempo real, añadiendo varias líneas y colores al trazado.

Incluir *Smoothie* es sencillo, basta con importar la librería:

```
<script type="text/javascript" src="javascripts/smoothie.js"></script>
```

Crear un elemento 'canvas' que será donde mostraremos la gráfica:

```
<canvas id="mycanvas" width="400" height="100"></canvas>
```

Recoger el elemento y añadir los datos (variable *random*):

```
var chart = new SmoothieChart();
chart.addTimeSeries(random, {
  strokeStyle: 'rgba(0, 255, 0, 1)',
  fillStyle: 'rgba(0, 255, 0, 0.2)',
  lineWidth: 4
});
chart.streamTo(document.getElementById("chart"), 500);
```

El resultado que obtenemos tiene el siguiente aspecto:

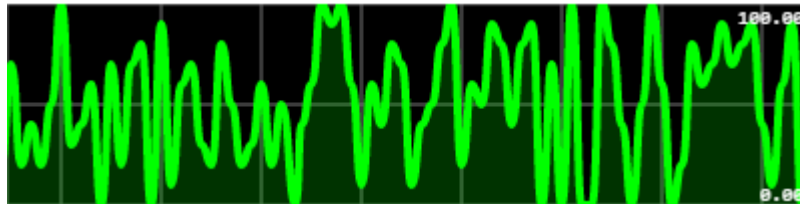


Figura 2 - Ejemplo de aspecto de una gráfica con *Smoothie*

4.1.1.3.2. CanvasJS

Acercándonos más a lo que estamos buscando, aparece *CanvasJS*, una librería mucho más completa que la anterior, muy usada por grandes empresas conocidas y que presume de ser rápida y eficiente.

Para echarla a andar necesitamos incluir la librería:

```
<script type="text/javascript" src="javascripts/canvasjs.min.js"></script>
```

Crear un elemento 'div', a diferencia de la otra librería que usaba el elemento canvas:

```
<div id="#myDiv"></div>
```

Y añadir los datos de la siguiente manera:

```
var chart = new CanvasJS.Chart("myDiv",{
  title :{
    text: "Live Random Data"
  },
  data: [{
    type: "line",
    dataPoints: [
      {x:1,y:1},
      {x:2,y:2}
    ]
  }
]
});
```

El aspecto de esta gráfica sería el siguiente:

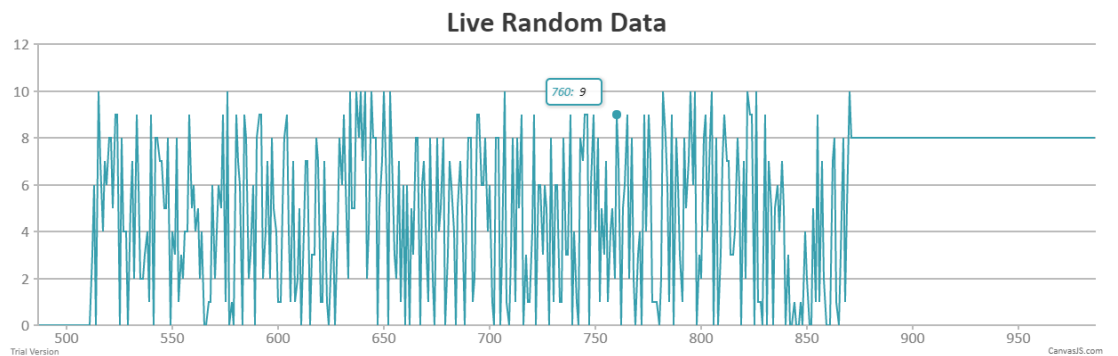


Figura 3 - Ejemplo de aspecto de una gráfica con *CanvasJS*

4.1.1.3.3. Plotly

La librería *Plotly* nos ofrece, de entre todas, el acabado más completo y profesional con menos esfuerzo, puesto que genera una gráfica con todo tipo de funcionalidades, algunas conocida como zoom, autoscale, exportar gráfica como imagen, son ejemplos de lo que podemos hacer con muy pocas líneas de código. Basado en la librería *D3.js*, esta librería usa de *SVG* para generar las gráficas. La contra de esta librería es su carga en tamaño y procesamiento, ya que contiene, como hemos dicho, múltiples funcionalidades.



Figura 4 - Menú de opciones disponibles en *Plotly*

Empezamos como siempre, incluyendo la librería:

```
<script type="text/javascript" src="javascripts/plotly-latest.min.js"></script>
```

Y al igual que la librería anterior, creamos un elemento 'div' que recogeremos con su ID:

```
<div id="#myDiv"></div>
```

Tan sencillo como llamar a una función con dos parámetros: el ID del elemento donde crear la gráfica y los datos, que será un array con los objetos de las trazas que vayamos a representar, rellenas como mostramos a continuación:

```
var trace1 = {
  x: [1, 2, 3, 4],
  y: [10, 15, 13, 17],
  type: 'scatter'
};
var trace2 = {
  x: [1, 2, 3, 4],
  y: [16, 5, 11, 9],
  type: 'scatter'
};
var data = [trace1, trace2];
Plotly.newPlot('myDiv', data);
```

Con estas pocas líneas obtenemos un resultado tan espectacular como este:

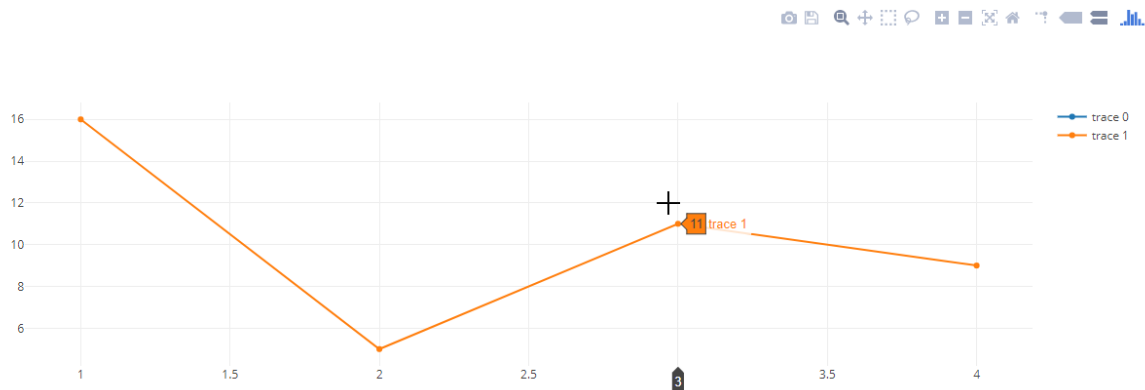


Figura 5 - Ejemplo de aspecto de una gráfica con *Plotly*

Pero buscamos simplicidad y eficiencia para el proyecto, por lo que seguimos buscando la gráfica más acorde a nuestra aplicación.

4.1.1.3.4. C3

C3 es otra librería que también está montada sobre D3.js, que, según su página oficial, es *cómoda, personalizable y controlable*. Es una librería reciente, salió hace apenas 3 años, pero las últimas actualizaciones se remontan a menos de un mes, donde van ampliando las funcionalidades y solventando errores.

Como todas las librerías, incluimos las librerías necesarias, en este caso requerimos de añadir la librería D3.js de la que depende

```
<script type="text/javascript" src="javascripts/d3.v3.min.js"></script>
<script type="text/javascript" src="javascripts/c3.min.js"></script>
<link href="/stylesheets/c3.min.css" rel="stylesheet" type="text/css"/>
```

Creamos un elemento 'div' que recogeremos con su ID:

```
<div id="#myDiv"></div>
```

Añadimos los datos en el objeto 'data', en el cual podemos añadirlo de la siguiente manera:

```
var chart = c3.generate({
  bindto: '#myDiv',
  data: {
    columns: [
      ['data1', 30, 200, 100, 400, 150, 250],
      ['data2', 50, 20, 10, 40, 15, 25]
    ]
  }
});
```

El resultado se muestra así:

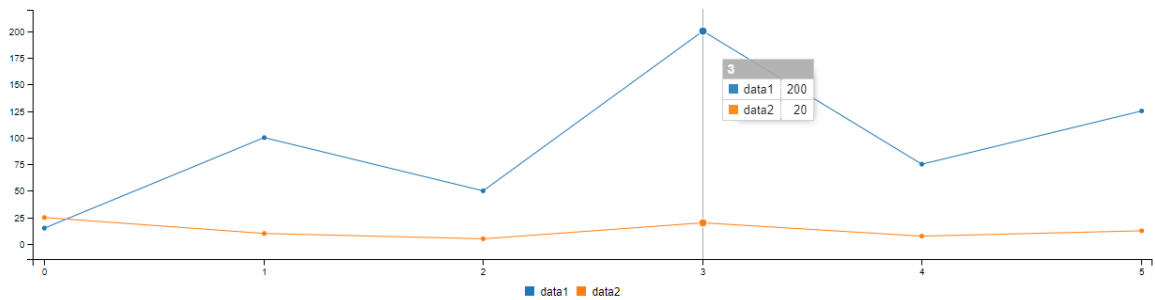


Figura 6 - Ejemplo de aspecto de una gráfica con C3

4.1.1.3.5. NVD3

NVD3 se nos presenta como una librería muy parecida a la anterior, más nueva todavía, que también depende de la librería D3, que pretende simplificar su uso sin perder la potencia de esta librería base. De esta librería hay que decir que actualmente está muy verde, le faltan muchas implementaciones en las que, en la API oficial nos redirige a la librería D3, *lo que no puedes hacer con NVD3, hazlo en D3*.

Exactamente igual que C3, necesitamos importar la librería base más la librería propia de NVD3:

```
<script type="text/javascript" src="javascripts/d3.v3.min.js"></script>
<script type="text/javascript" src="javascripts/nv.d3.min.js"></script>
<link href="/stylesheets/nv.d3.css" rel="stylesheet" type="text/css"/>
```

Creamos un elemento 'div' y dentro de este un elemento 'svg' donde generaremos la gráfica:

```
<div id="#myDiv">
  <svg></svg>
</div>
```

Hasta aquí la similitud con C3, puesto que la creación de la gráfica y la inclusión de los datos es completamente distinta.

```
nv.addGraph(function() {
  var chart = nv.models.lineChart();
  var myData =[
    {
      values: [{x: 1, y: 1},{x: 2, y: 2}],
      key: 'Trace name',
      color: '#ff7f0e'
    }
  ];
});
```

Como esta gráfica ha sido la escogida, mostraremos los resultados con esta librería más adelante.

4.1.2. Documentación: Markdown vs Latex vs Word

4.1.2.1. Word

Como es bien conocido, Microsoft Word es un procesador de texto que permite la realización de actividades ofimáticas para la creación de documentos de texto, creando contenido con unas herramientas de diseño y personalización muy elaboradas. Este potente programa tiene por contra la necesidad de ser usado para poder abrir los documentos sin perder calidad en el diseño, no siendo totalmente compatible, en la mayoría de los casos, con otros programas similares.

4.1.2.2. Latex

LaTeX es un sistema de preparación de documentos con el cual puedes preparar manuscritos, artículos de revista, cartas, tesis, presentaciones y cualquier tipo de documento que queramos imprimir en papel o mostrar en pantalla con un diseño final muy profesional, con una excelente calidad de imprenta. Uno de los puntos fuertes de LaTeX es que permite generar desde ecuaciones y gráficas, hasta pentagramas musicales, esquemas y tablas de manera fácil y rápida.

4.1.2.3. Markdown

Markdown es una herramienta extremadamente intuitiva, que, al igual que LaTeX tiene la finalidad de facilitar crear contenido, pero este de manera mucho más sencilla de escribir, con un diseño legible en texto plano fácil, rápido e intuitivo para manejar el contenido. La gran ventaja de ser un texto plano es la compatibilidad de abrir el archivo con cualquier editor, sin problemas de formatos o diseño, los cuales se aplicarán a través de un conversor, como *PANDOC*, el cual nos permite exportar nuestro contenido a múltiples formatos.

4.2. Descripción de la tecnología escogida

4.2.1. Lenguaje de programación: Node.js

La elección de Node.js como lenguaje de programación para el desarrollo del proyecto no es sólo el utilizar un solo lenguaje para programar tanto el cliente como el servidor, no es más que una ventaja. El hecho definitivo es la naturaleza propia de este entorno de ejecución, Node.js es concebido como un entorno de ejecución de JavaScript orientado a eventos asíncronos, diseñado para construir aplicaciones en red escalables y de rápida comunicación y ejecución de las entradas y salidas.

Según la web oficial:

Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

La característica más importante de NodeJS, y que ahora otra serie de lenguajes están aplicando, es la de no ser bloqueante, es decir, si durante la ejecución de un programa hay peticiones que requieren un tiempo para producirse la respuesta, Node no detiene el hilo de ejecución del programa esperando que esa parte acabe, sino que continúa procesando las siguientes instrucciones que no requieren los resultados de el proceso lento. Cuando este termina, realiza las instrucciones que fueran definidas para realizar con los resultados recibidos.

El repositorio de módulos **NPM** de Node.js no solo es grande, también es de fácil uso. Este puede ser instalado conjuntamente con Node, y la instalación de los módulos es automatizada, como se verá más adelante, con un archivo de configuración incluido en el proyecto.

4.2.1.1.1. Concurrencia

Node.js funciona con un modelo de evaluación de un único hilo de ejecución, usando entradas y salidas asíncronas las cuales pueden ejecutarse concurrentemente en un elevado número de peticiones sin incurrir en costos asociados al cambio del hilo de ejecución. Esta programación asíncrona es una de las grandes ventajas que ya hemos visto de Node con Javascript, pero

también es parte de los dolores de cabeza para los programadores, tanto para los novicios como los más expertos. En definitiva, un *callback* es pasar una función como parámetro para que dicha función se encargue de ejecutar nuestro parámetro.

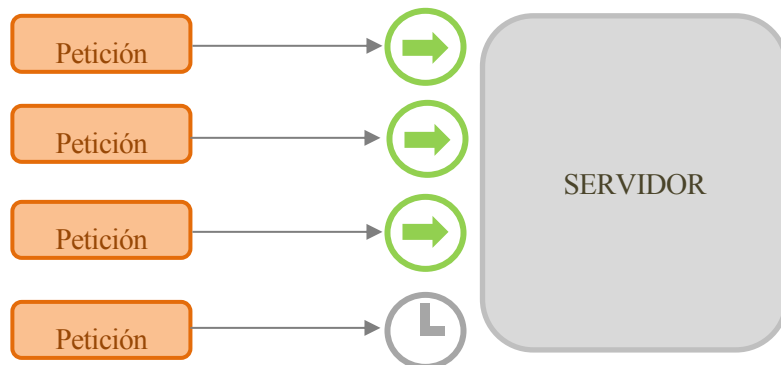


Figura 7 - Forma tradicional de acceso al servidor

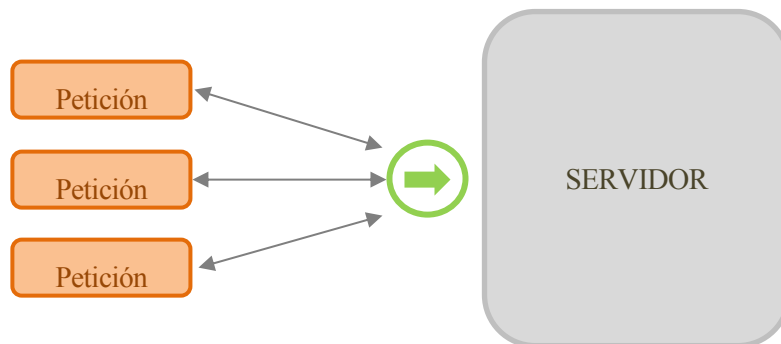


Figura 8 - Forma de Node.js de acceso al servidor

4.2.1.1.2. Motor V8

V8 es el entorno de ejecución para JavaScript creado para Google Chrome. Es software libre desde 2008, está escrito en C++ y compila el código fuente JavaScript en código de máquina en lugar de interpretarlo en tiempo real.

Node.js contiene libuv para manejar eventos asíncronos. Libuv es una capa de abstracción de funcionalidades de redes y sistemas de archivo en sistemas Windows y sistemas basados en POSIX como Linux, Mac OS X y Unix.

4.2.1.1.3. Eventos

Node.js se registra con el sistema operativo y cada vez que un cliente establece una conexión se ejecuta un callback. Dentro del entorno de ejecución de Node.js, cada conexión recibe una pequeña asignación de espacio de memoria dinámico, sin tener que crear un hilo de ejecución. A diferencia de otros servidores dirigidos por eventos, el bucle de gestión de eventos de Node.js no es llamado explícitamente, sino que se activa al final de cada ejecución de una función callback. El bucle de gestión de eventos se termina cuando ya no quedan eventos por atender.

4.2.1.1.4. Módulos

Node cuenta con una serie de módulos nativos compilados en el propio nativo. Pero además cuenta con la posibilidad de incluir módulos externos, los cuales se implementan siguiendo la especificación CommonJS para módulos, utilizando una variable de exportación para dar a estos scripts acceso a funciones y variables implementadas por los mismos. La vía más rápida, fiable y

sencilla para la implementación de estos módulos está en el instalador de paquetes NPM (del inglés *Node Package Modules*) que hemos mencionado con anterioridad. Según su página oficial:

npm es el gestor de paquetes para JavaScript y el mayor registro de software del mundo.

Usa npm para instalar, compartir y distribuir código; gestione las dependencias en sus proyectos; y comparta y reciba la opinión con otros.

4.2.1.2. Express

Express es una librería que nos permite la creación de un servidor web de manera rápida y sencilla. Si bien es cierto que Node ya nos ofrece una librería nativa 'http' que también nos permite la creación de este servidor en pocas líneas de código, pero con el mismo esfuerzo, con *Express* vamos a conseguir mucho más, preparado para manejar solicitudes complejas, configuración de rutas y trabajar cómodamente con las cabeceras de HTTP.

Para incluirlo en nuestro proyecto, lo realizaremos a través del instalador de paquetes NPM:

```
npm install socket.io --save
```

Una vez incluido, la configuración a nivel de código quedaría:

1. Importamos el módulo:

```
var express = require('express');
```

2. Inicializamos la aplicación:

```
var app = express();
```

3. Definimos el puerto del servidor:

```
app.listen(3000, function() {  
  console.log('Servidor funcionando en http://localhost:3000');  
});
```

Con esto ya tenemos nuestro servidor arrancando y listo para atender peticiones, ahora tan solo nos queda gestionar estas peticiones como rutas, para ello configuramos el comportamiento del servidor según la ruta que se le requiera:

```
app.get('/', function(req, res) {  
  res.send('Ruta por defecto');  
});
```

```
app.get('/ruta/:nombre', function(req, res) {  
  res.send('En esta pagina hay un parametro: ' + req.params.nombre + '!!!');  
});
```

En esta segunda ruta, tenemos el parámetro 'nombre' que puede tomar cualquier valor y recibir la cadena a través del objeto *request.param*.

4.2.1.3. SocketIO

SocketIO es otra de las librerías estrella de Node, que nos permite sincronizar la comunicación entre el cliente y el servidor minimizando en tiempo e intercambio de datos para conseguir una comunicación en tiempo real. Las posibilidades que nos ofrece *SocketIO* son muy amplias, bajo la idea de dinamizar una página, actualizando su contenido sin necesidad de refrescar ni volver a cargar esta.

Para todo ello, *SocketIO* se encarga de manejar los *WebSockets*, no es más que una librería que simplifica, con un uso de estos a más alto nivel, el cual permite establecer una conexión bilateral cliente-servidor de manera síncrona, en la que ambos extremos envían y están preparados para recibir cuando requieran hacerlo. La diferencia que existe con otros métodos, por ejemplo *AJAX*, quien también permite la actualización de la página sin recargarla, es la sincronidad, donde el cliente siempre necesita hacer una petición para que el servidor le responda y le envíe los datos. En cambio, con *WebSockets* el servidor tiene la capacidad de decidir y enviar paquetes sin requerir la petición por parte del cliente.

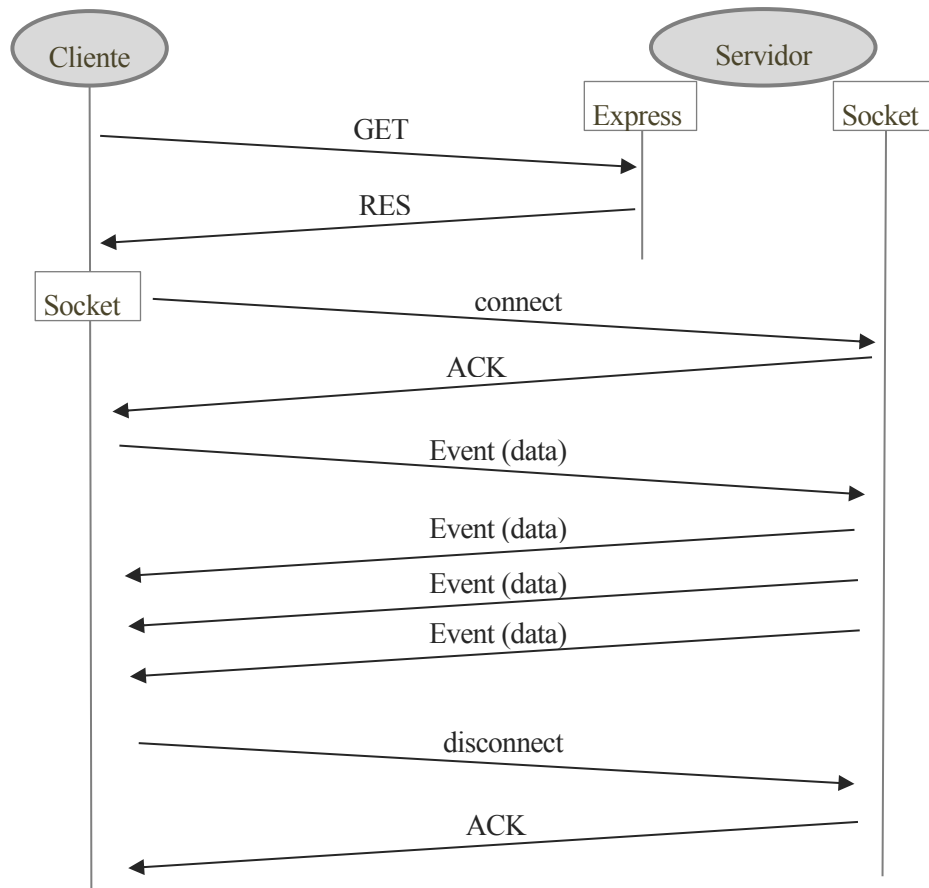


Figura 9 - Intercambio de información cliente-servidor usando Sockets

Al igual que Express, podemos instalarlo a través de NPM, e incluirlo en nuestro proyecto:

```
npm install socket.io --save
```

Una vez lo tengamos, creamos el servidor para los sockets, nosotros como ya tenemos un servidor web creado, se lo pasamos para incluir este servicio:

```
var io = require('socket.io')(server);
```

Pero aún nos falta realizar que el cliente abra la conexión, para ello **el cliente** deberá incluir la librería y ejecutar:

```
<script src="/socket.io/socket.io.js"></script>
```

```
var socket = io('http://localhost:3000'); // Si no ponemos la dirección tomará la del servidor como predeterminada
```

Ya teniendo nuestro servidor para sockets y la petición del cliente, creamos el evento de conexión, es decir, lo que vamos a realizar cuando un usuario se conecta a nuestro servidor:

```
io.on('connection', function (socket) {  
  console.log("socket connected");  
});
```

De la misma forma que hemos creado nuestro evento de conexión, podemos crear eventos particulares, que serán ejecutados cuando sean llamados desde el otro extremo. De esta forma podemos intercambiar eventos, por ejemplo:

En el servidor

```
io.on('connection', function (socket) {  
  /* Emite el evento al cliente */  
  socket.emit('Nombre del evento', variable_a_enviar);  
});
```

En el cliente

```
/* AL conectarse lanza el evento 'connection' */  
var socket = io('http://localhost:3000');  
/* Prepara el evento para cuando sea emitido por el servidor */  
socket.on('Nombre del evento', function(variable_recibida) {  
  alert('El evento ha enviado: ' + variable_recibida);  
})
```

4.2.1.4. Nodemon

Nodemon es una herramienta de Node utilizada exclusivamente para el desarrollo, a través de la cual nos da la comodidad de trabajar editando los archivos del proyecto y automáticamente reinicia el servidor actualizando los cambios quitándonos la necesidad de hacerlo nosotros mismos. Para ello instalaremos este módulo ya bien, en su versión global (recomendado), o añadiéndolo a la dependencia del proyecto, en el archivo package.json, pero sólo para el desarrollo, añadiendo --dev en lugar de --save, ya que no es una dependencia del proyecto por tanto no lo guardaremos como tal en el package.json. Como he dicho, lo instalamos en su versión global:

```
npm install -g nodemon
```

Para ejecutarlo nos dirigimos al directorio raíz del proyecto y ejecutamos:

```
nodemon {nombre del archivo} o nodemon si tenemos bien configurado nuestro package.json
```

4.2.2. Diseño de la web

4.2.2.1. JQuery

JQuery es una de las librerías para Javascript más usada en el mundo, apareció hace muchos años con la filosofía "*escribir menos para hacer más*", y se ha convertido en un referente y casi un estándar para el acceso al DOM (Document Object Model). Según la página oficial:

jQuery es una biblioteca JavaScript rápida, pequeña y rica en funciones. Hace cosas como el desplazamiento y la manipulación de documentos HTML, el manejo de eventos, animación y Ajax mucho más sencillo con una API fácil de usar que funciona a través de una multitud de navegadores. Con una combinación de versatilidad y extensibilidad, jQuery ha cambiado la forma en que millones de personas escriben JavaScript.

Esta librería usa de la nomenclatura típica de CSS para la selección de los elementos de la página, a los cuales puede modificar su contenido, añadir y eliminar elementos, así como asignarle eventos y alterar sus parámetros y estilos.

4.2.2.2. W3CSS

W3CSS es la librería escogida para maquetar la aplicación que vamos a desarrollar. Los motivos de esta selección son claros, un diseño rápido, fácil y sencillo, que requiera de los menores recursos posibles para realizar un primer prototipo de la web, mostrar su funcionalidad y la forma en que se adapta a todo tipo de pantallas. Si avanzado el proyecto, se requiere de algún diseño concreto, es fácilmente reemplazable, seleccionando entonces, la librería más oportuna o cargando una plantilla ya desarrollada.

Esta librería tan solo usa de un fichero CSS, el cual debemos añadir en la cabecera de la página, en el archivo que será común a todas las páginas creadas. Así, añadimos el fichero de la siguiente manera:

```
<link href="/stylesheets/w3.css" rel="stylesheet" type="text/css"/>
```

A partir de aquí, solo nos queda añadir las clases predeterminadas en la [API](#) de W3CSS, para saber la nomenclatura de cada estilo. Eso sí, muy intuitiva, todas tienen un nombre seguido de 'w3-'.

4.2.2.3. NVD3

NVD3 como una librería que aún no está completa, está emergiendo y actualizándose a diario, pero que cumple con lo que necesitamos para el alcance de nuestra aplicación.

La configuración que hemos dado a la gráfica, así como la conversión de los datos a través del parámetro *.tickFormat* pueden verse en el archivo */public/javascripts/graph.js*

4.2.3. Documentación: Markdown

Como ya hemos dicho, el contenido de la memoria ha sido redactado en texto plano, utilizando una sintaxis conocida como *Markdown*, y adaptándonos a *CommonMark* que propone un estándar para esta forma de escribir, ya que de por sí, *Markdown* no especifica una sintaxis concreta sin ambigüedad, existiendo múltiples maneras de escribir lo mismo (palabras en cursiva o en negrita tienen varias aceptaciones).

4.2.4. Gestión proyecto: Control de versiones

Un sistema de control de versiones es un software que permite gestionar los cambios que se van realizando sobre un conjunto de ficheros de forma que se puedan recuperar en cualquier momento versiones válidas del código, etiquetar versiones específicas, y permitir el trabajo simultáneo de diferentes desarrolladores.

Existen dos tipos de estos sistemas:

- **centralizados:** como SVN, en los cuales existe un almacén central de datos (el repositorio) accesible a todos los usuarios. El modo en que funcionan es simple, los archivos están en el servidor y no pueden ser modificados por dos usuarios al mismo tiempo, cuando un usuario accede al fichero en cuestión, el sistema lo marca inmediatamente con permisos de sólo lectura para el resto de los usuarios. Este tipo de sistemas requiere de tener acceso al servidor (ya sea remoto o local) para poder funcionar.

- **descentralizados:** como GIT, donde además del repositorio donde se almacena el proyecto, estos sistemas trabajan con copias locales. Un usuario recoge los datos y los almacena en local, lugar donde puede modificar todo cuanto desee. Una vez modificados, estos datos son devueltos al servidor, donde serán almacenados y registrados si no hay conflicto. Un conflicto surge cuando el proyecto original que hemos descargado no coincide con el actual en el servidor, debido a que otro usuario ha realizado otra modificación, en tal caso, se vuelve a descargar el proyecto y si coincide los ficheros modificados localmente con los modificados del servidor, no podremos subir el proyecto hasta que no resolvamos las incidencias (en ese momento hay que decidir qué edición se queda).

En esta tabla comparativa se muestran los dos sistemas de control de versiones que hemos nombrado en la anterior explicación:

Tabla 2 - Comparativa de SVN frente GIT para el control de versiones

	SVN	GIT
Control de versiones	Centralizada	Distribuida
Repositorio	central donde se generan copias de trabajo	Copias locales del repositorio en las que se trabaja directamente
Autorización de acceso	Dependiendo de la ruta de acceso	Para la totalidad del directorio
Seguimiento de cambios	Basado en archivos	Basado en contenido
Historial de cambios	Solo en el repositorio completo, las copias de trabajo incluyen únicamente la versión más reciente	Tanto el repositorio como las copias de trabajo individuales incluyen el historial completo
Conectividad de red	Con cada acceso	Solo necesario para la sincronización

4.2.4.1. GIT

GIT, como acabamos de ver, es un software que permite realizar un seguimiento de los cambios realizados en un proyecto con el tiempo. Git trabaja registrando los cambios que realiza en un proyecto, almacenando esos cambios y permitiéndole hacer referencia a ellos según sea necesario. Como vamos a ver más adelante en nuestro proyecto, *GIT* permite el uso de ramas de desarrollo, esto es, en definitiva, guardar el estado del proyecto actual y seguir desarrollando para que, en un momento dado, podamos volver hacia atrás o aplicar los cambios. Pero además podemos abrir varias ramas y usarlas como vías de desarrollo paralelo, cada una con un objetivo distinto.

Como sistema descentralizado que es *GIT*, es necesario todo el proceso de descarga del proyecto del repositorio y almacenamiento y modificación en local. para todo ello *GIT* usa de ciertos comandos para su implementación. A la hora de trabajar con un repositorio, realizaremos los siguientes pasos:

- Para actualizar el repositorio local:
 - *pull*: Actualiza el repositorio local con el último contenido del repositorio remoto. Equivale a realizar *fetch + merge(local)*

- Para subir los cambios al repositorio:
 - *status*: Ver el estado de los repositorios [no es necesario, solo para saber si algo ha cambiado]
 - *add*: Añadir un archivo nuevo o modificado preparándolos para ser añadidos al repositorio
 - *commit*: Añadir al repositorio **local** los archivos añadidos
 - *push*: Actualizar los cambios del *commit* al repositorio **remoto**
 - Es posible que del fallo que hemos comentado anteriormente, que el repositorio se haya modificado mientras editábamos, si ocurre esto realizaríamos un *pull* para actualizar el repositorio local.

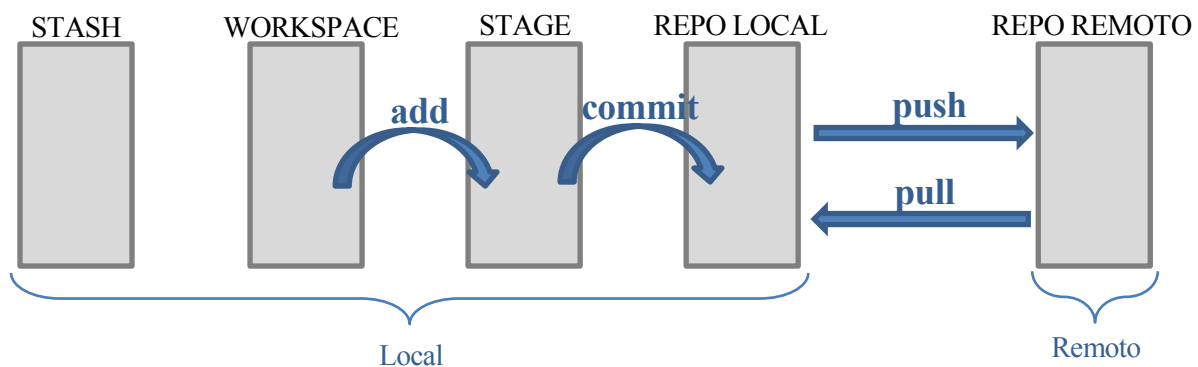


Figura 10 - Esquema de uso de los comandos de GIT

Además de estos comandos, *GIT* tiene otros muchos que son de gran utilidad en ciertos momentos en el desarrollo del proyecto. Algunos de ellos son:

- *stash*: Recoge los cambios modificados que no se han hecho *commit* y los guarda como una lista de cambios.
- *apply*: Coge un *stash* y aplica sus cambios a donde sea que estes pidiendo hacerlo.
- *pop*: Equivale a *apply* + borrar el *stash*.
- *log*: Muestra el registro de los *commit*.
- *blame*: Comando que sirve para ver información de las líneas de un fichero; quién creó una línea y cuándo lo hizo.
- *bisect*: Sirve para encontrar los fallos a traves de un historial, nosotros debemos indicar al menos, un *commit* que sea correcto y otro que no lo sea, y *git bisect* va reduciendo el numero de *commit* que puedan funcionar. Tras varias selecciones de este tipo, obtendremos en que *commit* se produjo el error.

4.2.4.2. GITHUB

GitHub es un servidor habilitado para tener un repositorio propio personal. Bajo la idea de *GIT*, *GitHub* ofrece un aspecto visual al desarrollo del proyecto, el cual puedes compartir y ver los proyectos de los demás usuarios, siempre y cuando sean públicos, que es el servicio gratuito ofrecido por esta plataforma.

Según la página oficial:

GitHub es una plataforma de desarrollo inspirada en la forma de trabajar. Es de código abierto orientado al negocio, ofrece la posibilidad de alojar y revisar código, gestionar proyectos y crear software junto con millones de otros desarrolladores.

4.2.4.3. GITKRAKEN (entorno gráfico muy intuitivo)

GitKraken es una herramienta de apoyo a un repositorio de GIT, es tan solo un framework, una aplicación que simplifica y ayuda a la visualización del proyecto sobre el que se trabaja. Esta herramienta es increíblemente intuitiva, y con unos pocos conocimientos en GIT podemos ponerla en marcha, y operar con GIT rápida y cómodamente.

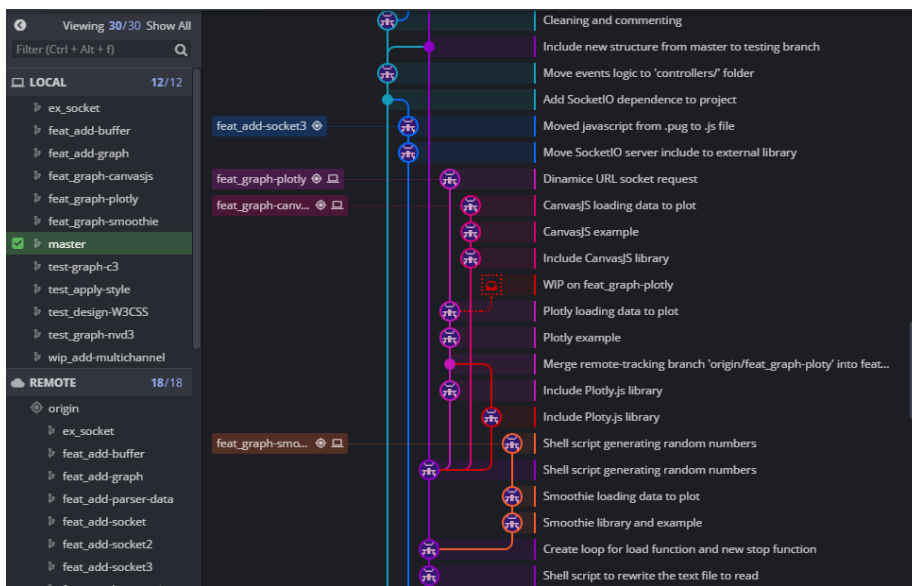


Figura 11 - Visualización del desarrollo y unión de las ramas

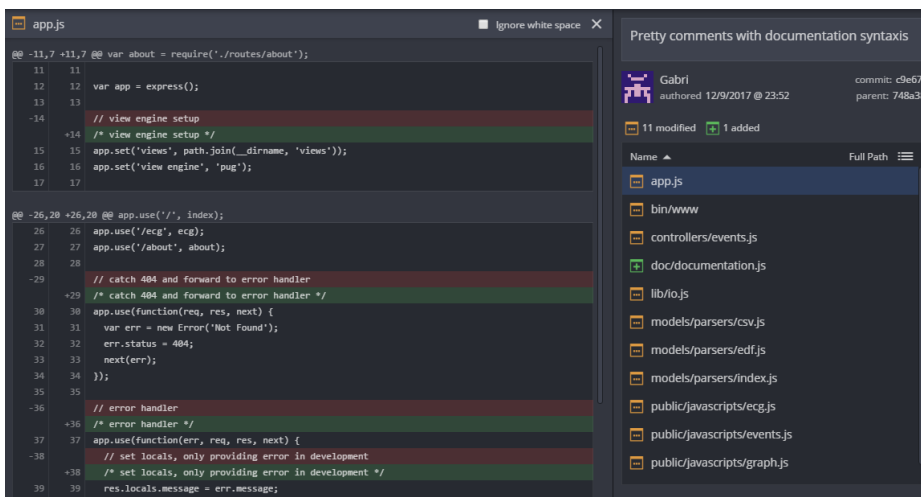


Figura 12 - Lista de modificaciones de un commit

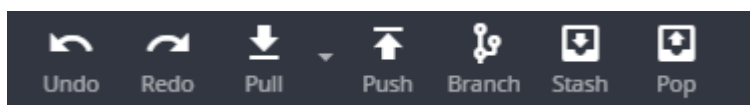


Figura 13 - Menú de comandos de GitKraken

4.2.5. Generadores de documentación

Para la documentación de un código, existen múltiples herramientas, que, a través de una sintaxis determinada, generan automáticamente la descripción redactada en el comentario y la definición de los parámetros que hemos definido con un aspecto atractivo y ordenado. Lo cierto es que hay varias herramientas que generan esta documentación, como *Doxygen* o *JSDoc*, pero no todas integran funcionalidades para Node.js. El código ha sido comentado siguiendo la sintaxis estándar para su generación, así, una vez se despliegue y configure uno de estos programas, se podrá realizar rápidamente la documentación técnica del proyecto.

5. PREPARACIÓN DEL SISTEMA

Todo es facil cuando deja de ser dificil

Roberto Giobbi

5.1. Creacion del entorno de desarrollo

5.1.1. Xilinx

5.1.1.1. Instalación

Para la instalación de Xilinx necesitamos una tarjeta SD, que es donde irá almacenado todo el sistema operativo, además necesitaremos también los archivos, nos dirigimos a la [página oficial de xilinx](#) para descargarlos.

Cuando tengamos los archivos, podemos seguir la guía de instalación que ellos mismos nos proporcionan en el enlace debajo del que acabamos de entrar para la descarga, en esta [guía](#) nos vienen tanto los pasos para la instalación del sistema operativo, como los primeros pasos para la configuración elemental y los problemas que podamos tener durante la instalación. La instalación se resume a los siguientes pasos:

1. Descargar los archivos
2. Descomprimir "the boot partition kit"
3. Cambiar datos del archivo `vhdl/src/xillydemo.vhd`
 - Eliminando o comentando:

```
PS_CLK : IN std_logic;
PS_PORB : IN std_logic;
PS_SRSTB : IN std_logic;
```
 - Descomentando:

```
-- signal PS_CLK : std_logic;
-- signal PS_PORB : std_logic;
-- signal PS_SRSTB : std_logic;
```
4. Abrir el programa Vivado y ejecutar, sin abrir ningún proyecto, el script tc1 de la carpeta `/vhdl`
5. Comprobar que todo haya salido correctamente
6. Ir al menú de la izquierda y seleccionar "Generate Bitstream", aparecerá una ventana emergente, daremos a ok y luego, hacer click en 'Yes'
7. Comprobar que el archivo con extensión ".bit" se ha generado

Ya hemos generado el archivo de configuración que tenemos que incluir en la SD, pero esta tarjeta tiene que estar formateada para poder ser ejecutada en el arranque del sistema de la

ZedBoard, por tanto, hay que incluirlo como archivo de imagen a la SD. De nuevo, en la guía explica cuidadosamente cada paso a ejecutar, incluso nombra el uso de distintas herramientas. La configuración de esta tarjeta ha sido realizada con el software 'USB Image Tool', se explican los pasos brevemente:

1. Descargar el software necesario, por ejemplo 'USB Image Tool'
2. Continuar con los pasos para seleccionar y darle a 'revert'
3. Extraer la SD (sacar y meter es necesario para actualizar la tabla de particiones)
4. Volver a poner la SD y ver que dentro solo hay un archivo 'ulmage'
5. Añadir los 3 archivos necesarios que hemos descargado, así como el que hemos generado en los pasos anteriores.
 - ulmage
 - boot.bin
 - devicetree.dtb
 - xillydemo.bit

Ya tenemos la tarjeta lista para arrancar. Si todo va según lo previsto, el sistema ejecutará sin ningún problema en el modo consola. Para arrancar el servidor X de entorno gráfico, ejecutamos:

```
startx
```

Para poder entrar en el modo U-Boot, será preciso conectar la ZedBoard a través de la UART a nuestro ordenador, entrando por línea serie y pulsando cualquier tecla mientras la placa esté enciendiendo antes de cargar el sistema. Seguimos los siguientes pasos:

1. Conectar el cable a la entrada UART de la ZedBoard y a un puerto USB del ordenador
2. A través de putty realizamos una conexión por el puerto serie a 115200 de velocidad
3. Reiniciamos el sistema y pulsamos alguna tecla durante el arranque para entrar en el modo U-Boot

Aquí podremos modificar variables del arranque, así como la dirección MAC de nuestra ZedBoard.

5.1.1.2. Configuración

La guía mencionada en el apartado anterior recomienda realizar una serie de configuraciones iniciales para un mejor rendimiento del sistema y una primera puesta a punto, ya que sería arriesgado modificar archivos de configuración una vez tengamos avanzado el proyecto, cabe el riesgo de cometer algún error y perder el trabajo que tengamos. Por lo que se aconseja realizar estos pasos nada más instalar el sistema.

La primera configuración y más importante es realizar un reparticionado de la tarjeta SD, aumentando así su capacidad total borrando y aprovechando el espacio que dejan los ficheros de configuración que ya no vamos a necesitar.

Por temas de seguridad, también es recomendable cambiar la contraseña de root.

Otra de las tareas que se han realizado ha sido la instalación de un servidor SSH, estableciéndole una dirección IP estática conocida, ya que si va a actuar de servidor no queremos que cambie. Los pasos ejecutados han sido:

1. [opcional] nos movemos a la carpeta de configuración de red

```
cd /etc/network/
```

2. Crear una copia de la configuración actual

```
cp /etc/network/interfaces /etc/network/interfaces.old
```

3. Modificamos el archivo "interfaces"

```
nano /etc/network/interfaces
```

4. El contenido del fichero será el siguiente

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 192.168.1.127
    netmask 255.255.255.0
    # gateway 192.168.1.1
```

5. Guardar y reiniciar el sistema

```
reboot
```

5.1.1.3. Controles de I/O

En nuestra placa ZedBoard contamos con varios pines de Entrada/Salida configurables para el usuario. Todos los pines tienen un número asociado, por ejemplo, los pines que corresponden a los diodos led van desde el número GPIO61 al GPIO64, para los diodos nombrados LED4 al LED7 respectivamente.

Para manejar estas GPIO se procede de la siguiente manera:

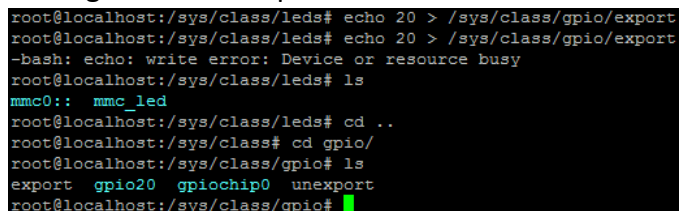
1. Realizando este comando se activa la gestión del GPIO20

```
echo 20 > /sys/class/gpio/export
```

2. Ahora se habrá generado la carpeta

```
/sys/class/gpio/gpio20/
```

3. Donde nos permitirá gestionar este pin



```
root@localhost:/sys/class/leds# echo 20 > /sys/class/gpio/export
root@localhost:/sys/class/leds# echo 20 > /sys/class/gpio/export
-bash: echo: write error: Device or resource busy
root@localhost:/sys/class/leds# ls
mmc0:: mmc_led
root@localhost:/sys/class/leds# cd ..
root@localhost:/sys/class# cd gpio/
root@localhost:/sys/class/gpio# ls
export gpio20 gpioclip0 unexport
root@localhost:/sys/class/gpio#
```

Figura 14 - Comandos para la gestión de las GPIO

4. Dentro de esta carpeta:

- En el archivo *direction* podemos definir si es de **entrada o de salida**
- En el archivo *value* podremos ver y editar su **valor**

Nota: Solo si definimos el GPIO como "out" nos dejara editar el archivo "value"

```
root@localhost:/usr/bin# echo 62 > /sys/class/gpio/export
root@localhost:/usr/bin# cd /sys/class/gpio/gpio62
root@localhost:/sys/class/gpio/gpio62# echo out > direction
root@localhost:/sys/class/gpio/gpio62# echo 1 > value
root@localhost:/sys/class/gpio/gpio62# echo 0 > value
root@localhost:/sys/class/gpio/gpio62# echo 1 > value
root@localhost:/sys/class/gpio/gpio62#
```

Figura 15 - Ejemplo probando encender y apagar un LED de la ZedBoard

5. Con el siguiente comando liberamos el GPIO

```
echo 20 > /sys/class/gpio/unexport
```

5.1.1.4. Lectura/Escritura en la FPGA

Para leer y escribir en las entradas y salidas de la FPGA, se hace en el directorio `/dev/`. Para este trabajo vamos a recibir datos desde la FIFO, por lo tanto, solo vamos a leer el archivo `/dev/xillybus_datastream` que será donde estarán los datos actualizados disponibles.

5.1.2. Instalación de GIT

Con la posibilidad delante de tener un repositorio disponible para usar, nos disponemos a instalar la herramienta ya conocida *GitKraken* para usar de manera cómoda y visual el repositorio. Una vez instalada esta herramienta, procedemos a su configuración. En el caso de disponer del repositorio en una cuenta en GitHub, tan solo tendremos que iniciar sesión con nuestro usuario y contraseña. En nuestro caso, el repositorio se encuentra en un servidor privado propio, por tanto, la configuración que tenemos que realizar es la siguiente:

1. Establecer una **clave SSH**, la cual se puede generar directamente desde el programa, donde quedará asignada.
2. Copiar el contenido del archivo generado de extensión ***.pub** al archivo `~/.ssh/authorized_keys`
 - `cat id_rsa.pub >> ~/.ssh/authorized_keys`
1. Establecer los permisos de acceso al usuario a la carpeta y al archivo
 - `chmod 700 ~/.ssh`
 - `chmod 600 ~/.ssh/authorized_keys`

Nota: El contenido ha de ser de una sola línea, del estilo a: `ssh-rsa XXXXDDDWWW...`

5.1.3. Instalación de Node

La máquina o servidor donde vamos a instalar Node.js va a ser un entorno virtual, para un desarrollo más cómodo. Pero como la final ejecución de esta aplicación va a ser realizada en nuestra ZedBoard sobre Xillinux, que está basado en Ubuntu 12.04 LTE, nuestro sistema operativo virtual va a ser el mismo, el cual, para la instalación de Node.js, requiere de un repositorio especial del cual instalar los paquetes. Este repositorio lo podemos incluir ejecutando el siguiente comando:

```
apt-add-repository ppa:chris-lea/node.js
```

Actualizamos el repositorio con:

```
apt-get update
```

Una vez configurado el repositorio, instalamos Node.js:

```
apt-get install nodejs
```

Para asegurar la correcta instalación, y ver la versión en la que vamos a trabajar, podemos ejecutar:

```
node -v
```

En nuestro caso, nos encontramos en la versión:

```
v6.11.0
```

5.1.3.1. Instalación NPM

Una vez instalado Node.js con el comando anterior, se habrá instalado el instalador de módulos *NPM*, podemos ver la versión actual ejecutando:

```
npm -v
```

en nuestro caso la salida nos devuelve la siguiente versión:

```
3.10.10
```

En el caso en que npm no se encuentre instalado, podemos probar a instalarlo individualmente con el siguiente comando:

```
apt-get install npm
```

pero para nuestra aplicación aún necesitamos más cosas, todos los módulos que serán instalados a través de este gestor de módulos.

5.1.3.2. Instalación del proyecto

Ahora vamos a instalar las dependencias del proyecto, aquellos módulos que vamos a necesitar tanto para el desarrollo y como para la ejecución de la aplicación. Esta instalación puede realizarse de forma casi automática, en el caso que el archivo **package.json** esté bien configurado. Este archivo, que ya se considera un estándar, es una especificación de los paquetes de Javascript, en el cual se añaden las dependencias del proyecto. Tan solo es necesario crear el archivo en el directorio raíz con la siguiente estructura:

```
{
  "name": "ecg-node",
  "version": "0.0.0",
  "dependencies": {}
}
```

También existen más objetos, por ejemplo *"description"* para añadir una descripción a nuestro proyecto o paquete, o *"script:start"* para indicar cual es el archivo que arranca node por defecto, como más adelante veremos que lo tenemos implementado.

Así, una vez tengamos nuestro archivo *package.json* creado, no vamos a tener la necesidad de incluir manualmente las dependencias, pues pueden ser incluidas en el momento de su instalación añadiendo la opción *--save* a nuestro comando. Por ejemplo, para dos de los módulos esenciales de nuestro proyecto *Express* y *SocketIO* la instalación y registro de estos se realizaría de la siguiente manera:

```
npm install express --save
```

```
npm install --save socket.io {es completamente indiferente la posición de la opción save}
```

Así vamos creando nuestro archivo *package.json* a medida que vamos instalando las dependencias.

Una vez queramos importar un nuevo proyecto, en el directorio raíz de este ejecutaremos:

```
npm install
```

Y tendremos instaladas todas las dependencias definidas en el *package.json*.

5.2. Estructura del proyecto

Ahora pasamos a la organización de un proyecto usando la estructura modular MVC, existe muchas formas de organizar estos archivos, muchos son los blogs que explican su forma de organizar el proyecto, y abren debates para poner las ideas en conjunto e intentar llegar a una conclusión de "la mejor estructura modular". En cambio, no hay ninguna norma o estándar de organización en Node mientras esté todo bien organizado.

Nosotros vamos a partir de un esqueleto aprovechando el módulo *Express*, para ello, vamos a instalar este módulo en todo el sistema, puesto que lo vamos a utilizar fuera del proyecto, de hecho, aún no existe el proyecto. Así ejecutaremos la instrucción de instalación con el parámetro *-g* de *global*, y este módulo será accesible para todos los proyectos (esto no lo almacena en *package.json*):

```
npm install express-generator -g
```

Una vez instalado, podemos generar nuestro nuevo esqueleto con la ejecución del siguiente comando:

```
express --view=pug ecg_node/
```

Que nos generará los archivos base para arrancar directamente un servidor con *express*, eso sí, instalando primero las dependencias con el comando que ya conocemos:

```
npm install
```

Y arrancando el servidor con el comando *node* o bien *nodemon* si lo hemos instalado previamente.

Como podemos ver en el repositorio de trabajo, en el segundo *commit* está la creación del esqueleto en el cual aparecen los archivos generados y su contenido. A partir de esta estructura hemos ido construyendo el proyecto, añadiendo directorios modularizando todo el contenido, hasta tener la estructura final.

5.2.1. Estructura de los directorios

En cuanto a nuestra aplicación, cada carpeta tiene su función, pues este concepto de *estructura modular* consiste en la separación del proyecto en bloques funcionales como ya hemos explicado.

- **.git/**: La carpeta del GIT con todos los registros del proyecto
- **bin/www**: La aplicación principal, esta es la que se ejecuta

- **controllers/**: La clásica carpeta controlador de MVC, aquí van los archivos que gestionan el curso de la aplicación y realizan la llamada de cada función
 - *events.js*: En este archivo estan los eventos definidos de SocketIO que se usan en la aplicación
- **lib/**: Las librerías de las funciones de la aplicación
 - *io.js*: Librería para arrancar el servidor de socketIO
- **models/**: Tradicionalmente los archivos que configuran y realizan el acceso a la base de datos
 - **parsers/**: Los archivos que leen los ficheros de datos
 - *csv.js*: conversor de archivos de extensión .csv al paquete de datos de la aplicación
 - *edf.js*: conversor de archivos de extensión .edf al paquete de datos de la aplicación (solo es un parser de prueba)
 - *index.js*: El controlador de los parser, analiza la carpeta para saber de cuales dispone
- **node_modules/**: la carpeta con todas las librerías de los modulos de Node, tanto los nativos como los añadidos
- **public/**: Los archivos compartidos al cliente
 - **images/**: Carpeta de imágenes de la web
 - **javascripts/**: librerías Javascript
 - *jquery-3.2.1.js*: Librería JQuery
 - *d3.v3.min.js*: Librería D3 para gráficas
 - *nv.d3.min.js*: Librería NVD3 para gráficas (framework de D3)
 - *graph.js*: Aquí estan las funciones para la representación de la gráfica
 - *ecg.js*: Aquí las funciones de la lógica de la página
 - *events.js*: Aquí estan la conexión y los eventos de socketIO
 - **stylesheets/**: Librería de estilos CSS
 - *w3.css*: Librería W3CSS para responsive_design
 - *nv.d3.css*: Librería NVD3 para gráficas
 - *graph.css*: Librería para dar formato a la gráfica
 - *style.css*: Aplicación de estilos específicos de la web, fuera de las librería
- **routes/**: Los controladores de la web, aquí se inicia la ejecución de cada página mandando la orden de renderizar los archivos de extensión .pug
 - *about.js*: Recoge la petición del cliente y renderiza about.pug
 - *ecg.js*: Recoge la petición del cliente y renderiza ecg.pug
 - *index.js*: Recoge la petición del cliente y renderiza index.pug
- **test/**: -- Carpeta que contiene los ficheros de datos [Esto no debería estar aqui una vez desplegada la aplicación]
 - *drivedb.csv*: Ejemplo de datos ECG bajo conducción en formato CSV
 - *route.csv*: Ejemplo de datos ECG en estado de apnea en formato CSV
 - *route_9ch.blah*: Prueba multicanal 9 canales
 - *route_15ch.blah*: Prueba multicanal 15 canales

- **views/**: La vista de la estructura MVC, donde van los archivos que definen la estructura de cada página
 - *about.pug*: Página de información adicional de la web
 - *ecg.pug*: Página donde se muestra la gráfica (la aplicación en sí)
 - *error.pug*: Para definir una página cuando ocurre algún error en la web
 - *index.pug*: Página de inicio de la aplicación
 - *layout.pug*: Es la plantilla genérica de todas las páginas, todas requieren de ellas (otra estrategia modular)
- *.gitignore*: Se definen que archivos o carpetas van a ser ignorados por GIT a la hora de comprobar y almacenar los archivos modificados
- *app.js*: -- Se encarga de redireccionar las peticiones del cliente para llamar al *route* correspondiente
- **package.json**: Archivo descriptivo del proyecto donde se definen la dependencia de los módulos del proyecto
- *README.md*: Descripción y documentación del proyecto

5.2.2. Estructura de las ramas de desarrollo

Como ya hemos comentado, una de las características de GIT es la creación de ramas de trabajo, que se puede paralelizar en su desarrollo, y de las que podemos cambiar de una rama a otra sin ninguna complicación. Observando el repositorio, podemos apreciar las distintas ramas y la naturaleza de su creación, es decir, por qué se han creado, para qué y hasta donde. Para ello hemos seguido la siguiente nomenclatura para las ramas:

- *master*: Por supuesto la rama principal, en la cual tendremos siempre la aplicación sin errores y completamente funcional, a la que se irán uniendo las ramas de desarrollo una vez depuradas, ordenadas y listas para funcionar.
- *feat_*: Esta rama será creada cuando queramos añadir alguna funcionalidad a la aplicación, por ejemplo, añadir una nueva funcionalidad: *feat_add-socketio*
- *bug_*: Esta rama tiene por finalidad describir un problema, pero no la solución, que es para lo que vamos a desarrollar la rama, para que revisando los commit podamos ver como hemos dado solución a dicho problema.
- *wip_*: Se suele usar como 'work in progress' para algo que es de largo recorrido, fuera de especificaciones, o que no corre prisa. generalmente es un desarrollo continuado del que se van haciendo *merges* a *master* cuando haya algo concreto
- *test_*: Generalmente denominado 'junk' en esta forma de estructurar las ramas, se utiliza para pruebas de desarrollo, como comparar distintas tecnologías con el fin de descartar o seleccionar una de ellas. En nuestro proyecto podemos ver el ejemplo de las librerías para las gráficas que hemos probado, aquellas ramas con nombre precedidos por *test_graph*

De esta forma, cada rama tiene un propósito funcional, de manera que cuando finalice el desarrollo y el motivo por el que fue creada, realizará un *merge* a *master* y desaparecerá.

6. APLICACIÓN ELECTROCARDIOGRAMA

Vivir resultaría terriblemente aburrido si todo se hiciese siempre de la manera más eficiente

Guy Hollingworth

En este apartado se describe el conjunto de todo el proyecto desde los distintos puntos de vista, en primer lugar, una visión estructural por bloques, la documentación técnica, comunmente denominada *API*, y una guía de uso para el usuario de la aplicación.

6.1. División en bloques

6.1.1. Esquema general

La aplicación se basa en la comunicación cliente-servidor, en el cual el cliente hace una petición y el servidor actúa sobre esta petición. Así que tenemos dos partes:

- Por un lado, se pone en marcha el servidor cargando la configuración que requiera y que atiende a las peticiones del cliente.
- En el otro extremo nos conectaremos a la dirección del servidor y le solicitaremos los datos para representarlos en una gráfica.

Para la simulación en tiempo real de los datos cargados de un fichero, el servidor realiza los siguientes pasos. Para la lectura del fichero, pasa el contenido del fichero elegido por un bloque *parser*, que analiza los datos dependiendo su extensión y genera un objeto json genérico sin importar la codificación del archivo origen. Tras cargar estos datos, ya pueden ser enviados al cliente pasando por un bloque *generador* que recoge todos estos datos y los envía uno a uno a otro bloque, *buffer*, que se encarga de formar un paquete para optimizar el intercambio de los datos. Una vez lleno el buffer, envía el paquete al servidor.

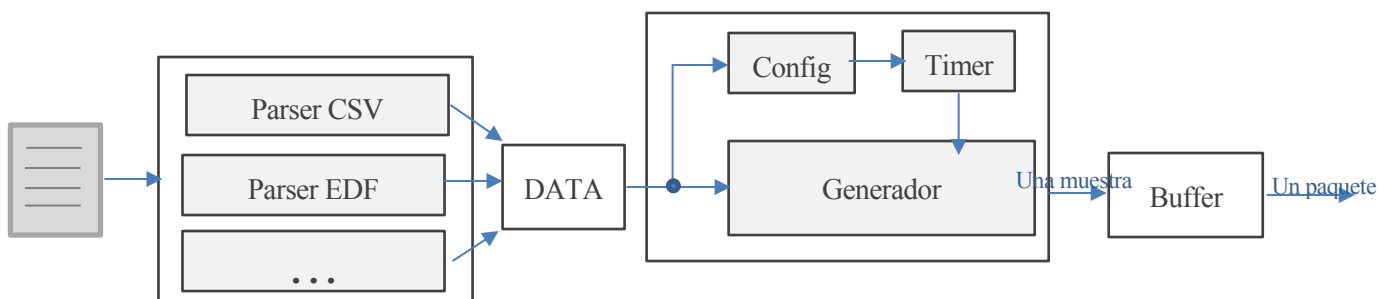


Figura 16 - Diagrama de bloques de la secuencia lógica del servidor

6.1.2. Intercambio de eventos

Cuando el cliente accede a la página de la aplicación, se realiza una conexión socket, que queda abierta para la comunicación de los dos extremos. Para la interacción de uno sobre otro, se intercambian eventos. Explicamos las dos representaciones implementadas:

Mostrar todos los datos de una vez

1. Se ha creado un botón que al ser pulsado llama a una función que envía el evento *Load folder*. Dicho evento solicita al servidor cargar los ficheros disponibles.
2. Al recibir este evento, el servidor carga los archivos disponibles y los envía al cliente a través del evento *File list*
3. El cliente recibe y muestra estos ficheros. Cuando uno de ellos es seleccionado, se envía el evento *Load data* al servidor con la ruta del archivo como parámetro.
4. El servidor carga el archivo seleccionado, enviando al cliente los datos de configuración de la gráfica a través del evento *Configure graph*.
5. El cliente al recibir estos datos, construye el plano donde se va a representar la gráfica ajustando los ejes a la configuración recibida.
6. Con otra interacción por parte del usuario, pulsaremos otro botón y solicitaremos que envíe todos los datos enviando el evento *Plot graph*
7. El servidor recibirá el evento y enviará los datos a través del evento *data*.
8. El cliente recoge los datos y los añade a la gráfica.

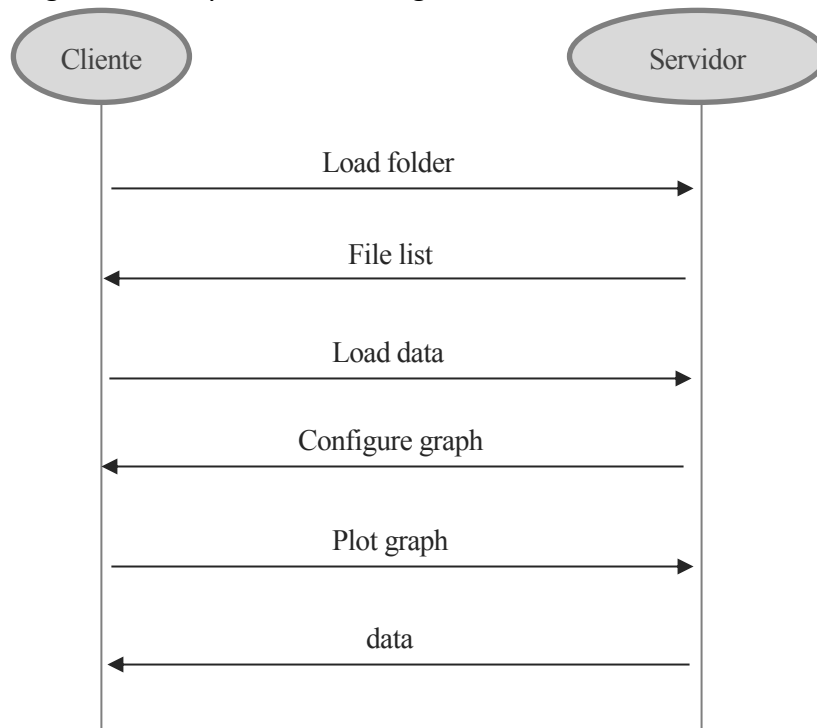


Figura 17 - Eventos intercambiados para la representación directa

Representación de los datos muestra a muestra

1. Como se puede ver, los primeros 5 pasos descritos anteriormente son comunes para ambos procedimientos, continuamos desde el paso 5.
2. En este caso, hacemos click en otro botón distinto que enviará el evento *Start graph* al servidor para iniciar la transmisión de los datos.

3. El servidor atiende a este evento enviando paquetes de datos periódicamente a través del evento *Data packet*.
4. El cliente recoge el paquete de datos e incluye estos uno a uno a la gráfica.

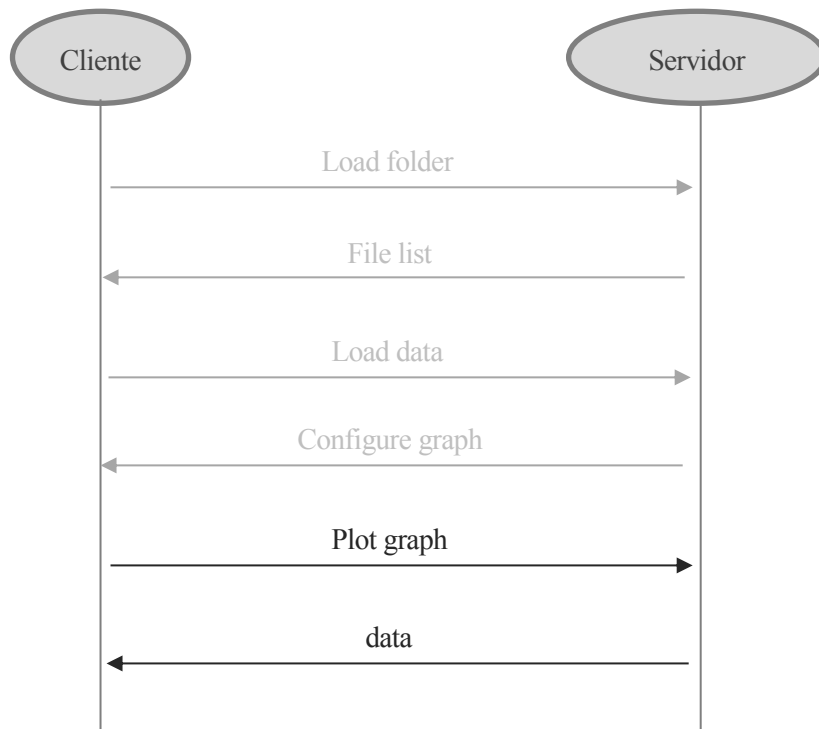


Figura 18 - Eventos intercambiados para la representación directa

6.1.3. Obtección de datos: parser

Para automatizar la inclusión de nuevos formatos de archivo, se ha implementado una lógica de análisis de parsers disponibles, de manera que tan solo con incluir el nuevo fichero javascript en la carpeta *parsers/* podemos aceptar ese nuevo formato sin modificar nuestro código. Para realizar esto, se analizan los ficheros de la carpeta y se añade a un objeto la llamada a estos si son considerados como parser.

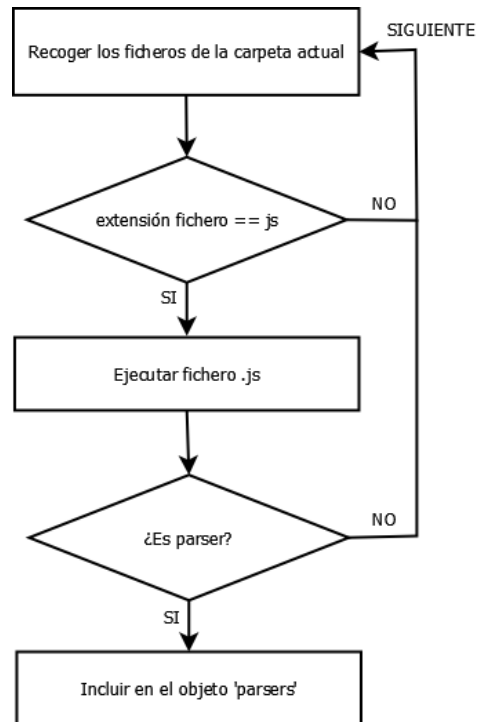


Figura 19 - Pseudocódigo simple del analizador de parsers

Todos estos archivos *parser* deben exportar una variable json con varios objetos. Esta variable ha de tener, al menos, una función *parse* que devuelva los datos en la estructura fijada, genérica para todos los parser. Esta estructura de datos es la siguiente:

```

"data": {
  "ts": 0,
  "n_samples": 0,
  "n_channels": 0,
  "samples": {}
}

```

Donde el objeto 'samples' contendrá un vector de tiempo y las muestras de cada canal. Un ejemplo de 2 canales con 3 muestras quedaría:

```

"samples": {
  "t": [0,1,2],
  "channel_0": [1,3,5],
  "channel_1": [2,4,6]
}

```

6.1.4. Generador de datos y buffer de envío

Este bloque recibe el objeto 'data' relleno de los datos que se han leído del archivo procesado por el parser correspondiente. Con la cabecera de los datos, en la que está la frecuencia de muestreo, puede configurar el timer que controlará este bloque. Este timer es un intervalo de tiempo para llamar periódicamente al generador y que recoja una sola muestra de los datos y la saque al siguiente bloque.

Este dato podría ser enviado directamente al servidor, pero no sería eficiente debido a que cuando mandamos un paquete con muy poco contenido la relación de los datos útiles respecto a la cabecera del mensaje es muy baja. Para que la transmisión se óptima, recogemos un número de muestras determinadas, establecida en 10 muestras para el desarrollo funcional del prototipo,

de manera que cuando llenemos ese número de muestras, enviamos un paquete de datos al cliente, mejorando la relación de datos útiles en el mensaje. De esta forma hemos implementado un buffer de salida, con el cual es cierto que sufrimos un pequeño desfase en cuanto a la representación de los datos en tiempo real, pero en vista a que tenemos una alta frecuencia de muestreo este desfase resulta imperceptible.

6.1.5. Buffer de recepción (cliente)

El cliente es el encargado de mostrar los datos muestra a muestra, pero el cliente recibe estos en paquetes de muestras. Para gestionar esto se ha implementado un buffer de recepción, en el cual se van añadiendo los paquetes en el momento en el que se reciben. Sobre este buffer se sacará la primera muestra (recoger la muestra y eliminarla del buffer), y se enviará a la función encargada de añadirla a la gráfica.

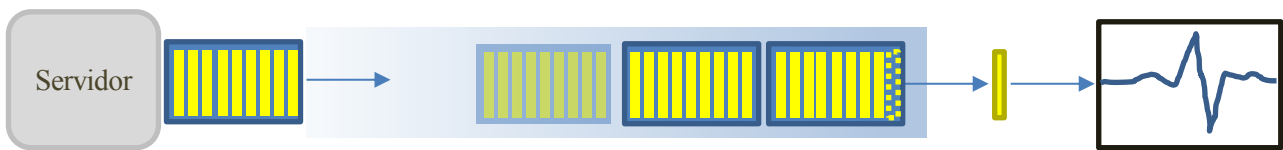


Figura 20 - Esquema de funcionamiento del buffer implementado en el cliente

6.2. Referencia técnica

Como se ha comentado en el apartado de uso de las herramientas de desarrollo, para realizar una documentación técnica de manera rápida, ordenada y automatizada, se ha utilizado un generador de documentación de código, donde, respetando una sintaxis y utilizando la herramienta que más nos guste (por funcionalidad o diseño de la documentación), podemos crear una referencia técnica que recoja la descripción de cada una de las partes del código más importantes, aquellas como funciones o estructuras de control, que sean de importancia para la comprensión de todo el funcionamiento de la aplicación.

Esta referencia se encuentra incluida en los archivos del proyecto.

6.3. Manual de usuario

La aplicación es sencilla de utilizar y muy intuitiva, cuenta con 4 direcciones definidas en el menú superior, en el cual, el logo también es un enlace que redirige a la página principal.

La división del diseño se ha realizado con un menú, un cuerpo de contenido y un pie de página que se mantendrá fijo en la parte inferior exista o no contenido.

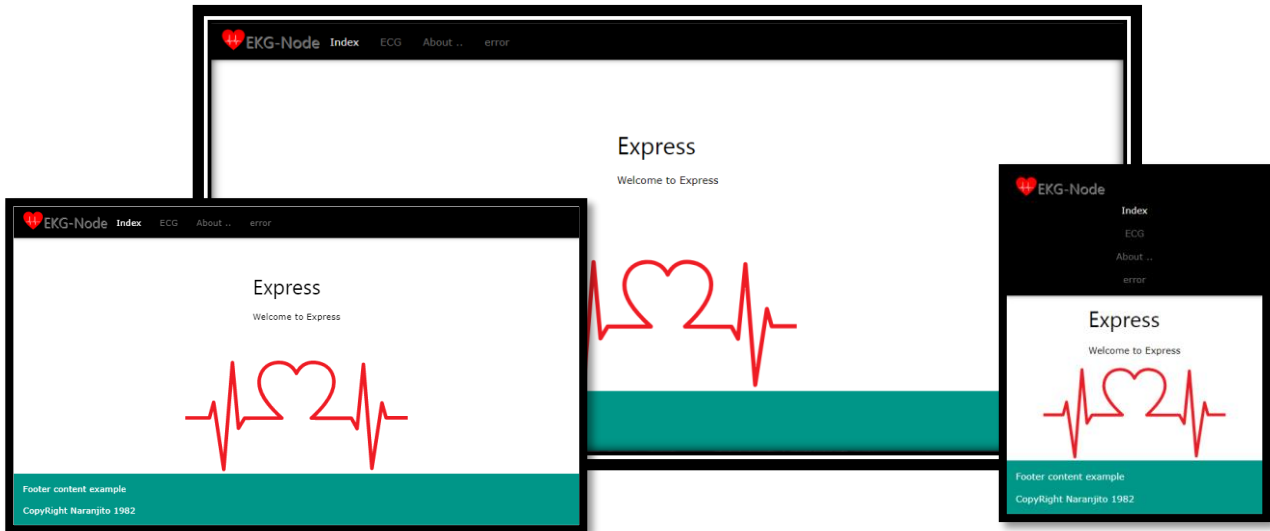


Figura 21 - Portada de la aplicación web con *responsive design*

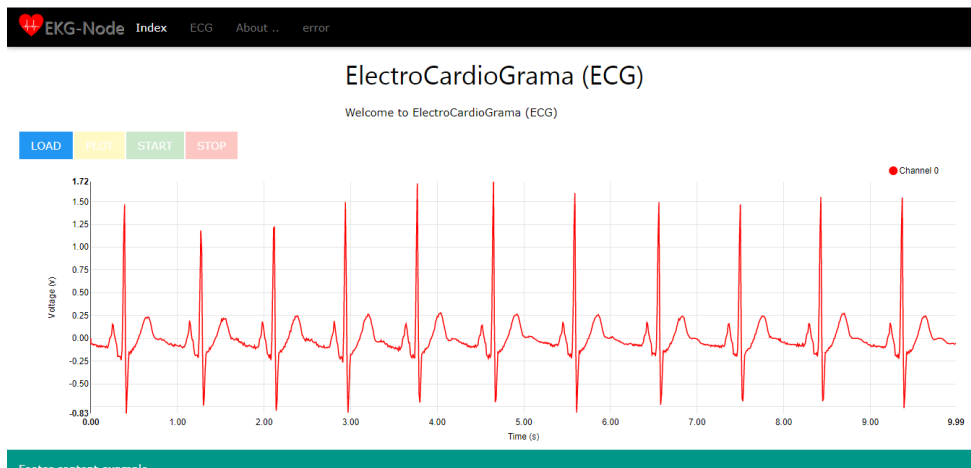


Figura 22 - Página *ECG* donde se muestra la gráfica

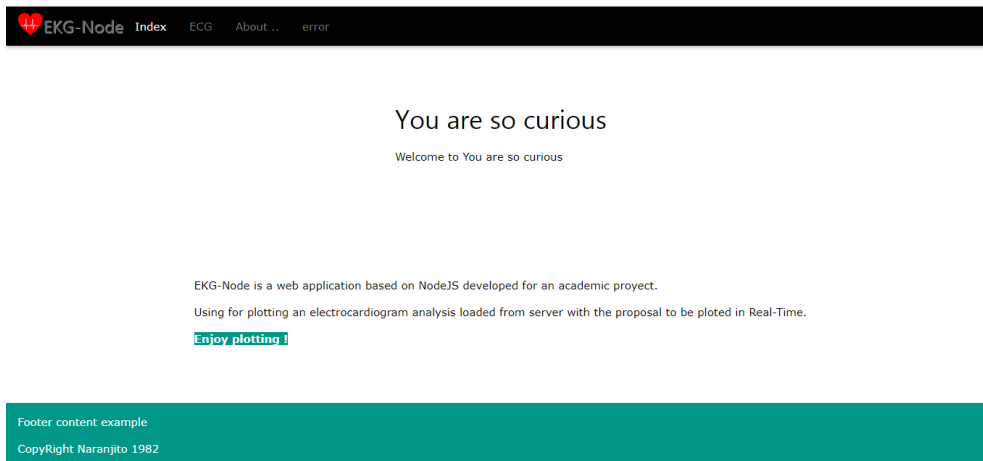


Figura 23 - Página de información de la aplicación

Not Found

404

```
Error: Not Found
at /mnt/hgfs/GIT/ecg_node/app.js:31:13
at Layer.handle [as handle_request] (/mnt/hgfs/GIT/ecg_node/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/mnt/hgfs/GIT/ecg_node/node_modules/express/lib/router/index.js:317:13)
at /mnt/hgfs/GIT/ecg_node/node_modules/express/lib/router/index.js:284:7
at Function.process_params (/mnt/hgfs/GIT/ecg_node/node_modules/express/lib/router/index.js:335:12)
at next (/mnt/hgfs/GIT/ecg_node/node_modules/express/lib/router/index.js:275:10)
at /mnt/hgfs/GIT/ecg_node/node_modules/express/lib/router/index.js:635:15
at next (/mnt/hgfs/GIT/ecg_node/node_modules/express/lib/router/index.js:260:14)
at Function.handle (/mnt/hgfs/GIT/ecg_node/node_modules/express/lib/router/index.js:174:3)
at router (/mnt/hgfs/GIT/ecg_node/node_modules/express/lib/router/index.js:47:12)
```

Figura 24 - Página no encontrada, muestra de información de errores

Comenzando la guía de uso, la aplicación de representación de la gráfica se encuentra en la ruta *ecg*, en la que encontramos 4 botones de colores. Al principio podemos observar que tan solo uno de ellos, el botón azul *LOAD* está habilitado, puesto que nuestro primer paso en la aplicación será cargar los datos.

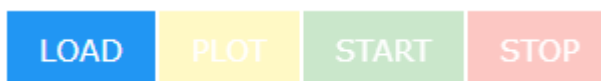


Figura 25 - Estado inicial de los botones

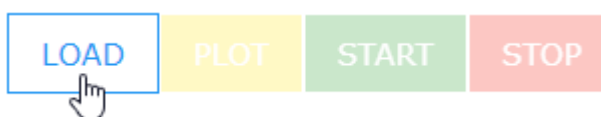


Figura 26 - Ejemplo de botón habilitado



Figura 27 - Ejemplo de botón deshabilitado

Si pulsamos en este botón, aparecerá una pantalla sobrepuesta con una lista de nombres de ficheros. Estos son los ficheros disponibles que hemos cargado del servidor.

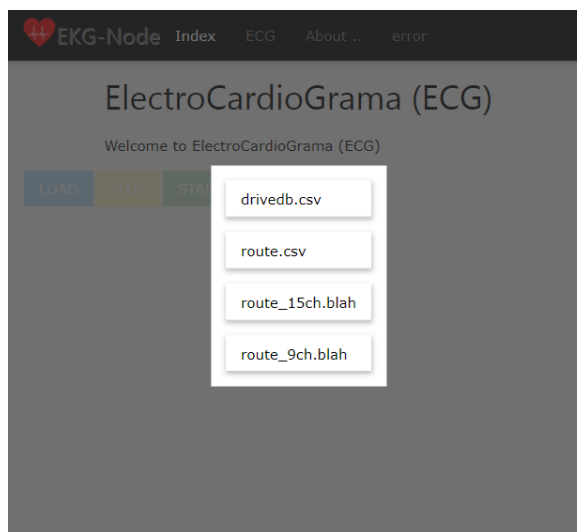


Figura 28 - Lista de ficheros cargados del servidor

Seleccionando uno de ellos, veremos como aparece el plano en el que vamos a representar la gráfica. Además, podemos ver como se han activado los botones *PLOT* (amarillo) y *START* (verde) con los cuales podremos representar los datos.

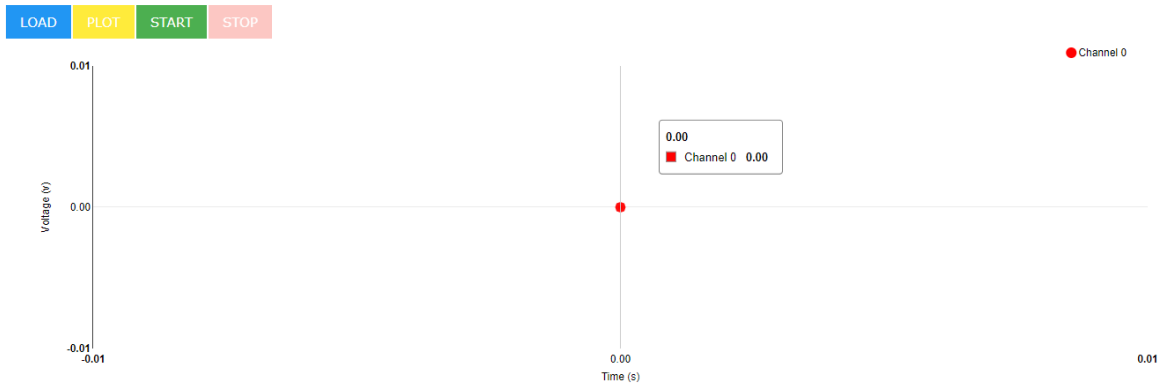


Figura 29 - Contenedor precargado donde se muestra la gráfica

Existen dos opciones de representar los datos:

- A) Pulsando el botón *PLOT* realizaremos una petición al servidor de que nos envíe todos los datos del archivo para representarlos todos de una vez

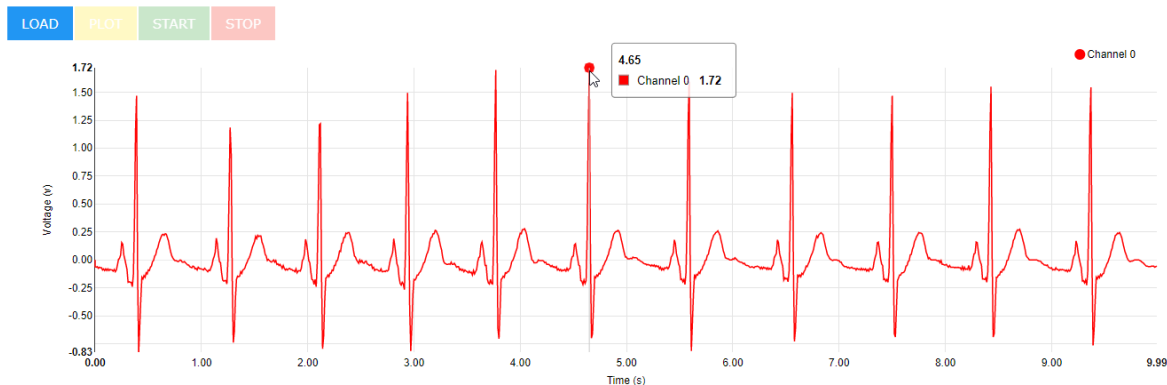


Figura 30 - Ejemplo de representación directa de la gráfica

- B) Pulsado en botón *START* el servidor nos enviará, en este caso, los datos en paquetes de 10 muestras, los cuales se mostrarán uno a uno en la gráfica. Podemos ver como se llena la gráfica hasta llegar a las 200 muestras, punto en el cual veremos la gráfica desplazándose para mostrar los nuevos datos que van llegando.

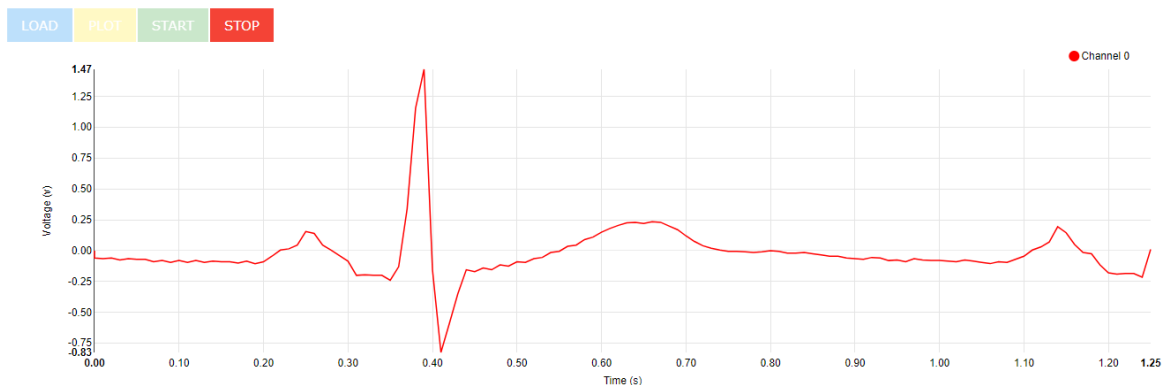


Figura 31 - Ejemplo de estado representando la gráfica

Como podemos ver, al pulsar el botón *START* se ha habilitado el botón rojo *STOP*, el cual, si pulsamos, detendrá la representación de los datos y cambiará a deshabilitado, modificando también el botón *START*, que ahora aparece habilitado y con el texto *CONTINUE* para continuar con la representación en el mismo punto si lo pulsamos.

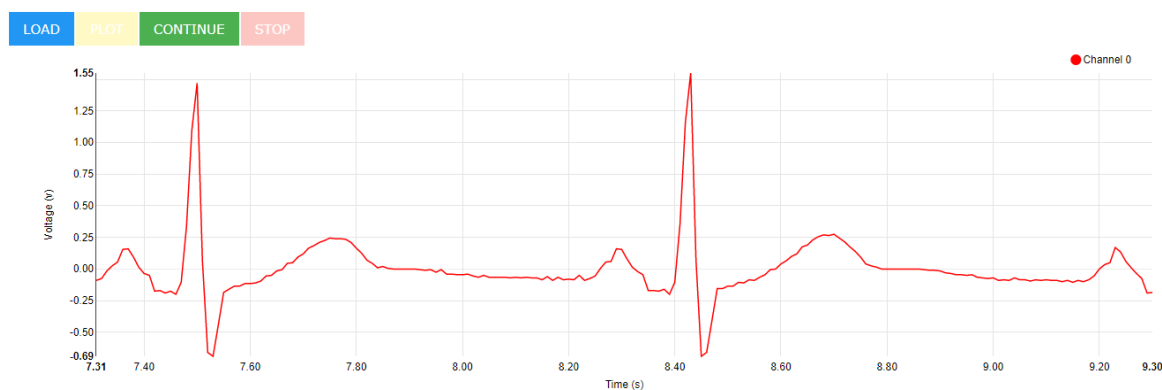


Figura 32 - Ejemplo de gráfica parada / pausada

Como podemos ver, durante la representación de la gráfica no podremos cargar otro archivo, tendremos que parar la ejecución para realizar dicha acción.

6.4. Test y pruebas de funcionamiento

Para poder probar el funcionamiento de la gráfica, se han recogido ficheros de extensión *csv* con los resultados de análisis reales de electrocardiografía. Estos datos han sido recogidos de una web, [PhysioBank](#), que contiene acceso a más de 50 bases de datos con registros como los que estamos tratando. Además, esta página muestra y exporta el contenido en diversos formatos, realizando también una representación de los datos. Por ello se ha implementado en el proyecto el bloque de *parsers* para aceptar los múltiples formatos en los que puedan venir los datos. Así hemos recogido los datos en formato *csv* por simplicidad, diseñando el parser para procesarlos.

Además de estos ficheros *csv*, también se han realizado una serie de pruebas con datos aleatorios, con el fin de probar el funcionamiento de la implementación multicanal automatizada. Estos ficheros son los incluidos en el proyecto que tienen la extensión inventada *.blah* y que se procesan con el mismo parser que los *csv*.

7. CONCLUSIONES

Dientes, pipí y a la cama
Mi padre
momentos después de cenar

7.1. Objetivos cumplidos

Los objetivos base que se marcaron a principios del proyecto se han cumplido, en el cual se ha conseguido desarrollar una base sobre una aplicación, realizando toda su estructura modular, que nos permite ampliar y dividir el trabajo de desarrollo en ficheros funcionales. Así, se ha llegado hasta implementar un esquema de lectura de ficheros y representación, tanto directa como muestra a muestra, con el fin de mostrar la funcionalidad directa de la aplicación, estableciendo una estructura por la cual continuar.

Partiendo de esta base dinámica, capaz de adaptarse a distintos tipos de formato, nos quedaría implementar las funcionalidades que se plantearon y que no se han llegado a implementar, las cuales se reflejan como mejoras en el siguiente apartado.

También se propuso, y que se ha realizado a modo de prototipado, el diseño de la página de forma *responsive*, que cumple con el cometido que se requería, adaptarse a los distintos dispositivos.

7.2. Mejoras

Algunos de los objetivos que no se han llegado a cumplir o modificaciones que pueden aplicarse a los objetivos cumplidos con fin de su optimización son los siguientes:

- **Ajustar de forma dinámica la escala de la gráfica:** Para implementar esta mejora, es necesario que tengamos la información de muestreo y conversión de los datos, es decir, cual es la frecuencia de muestreo y cual es la resolución con la que se han convertido los datos a digital.
- **Adaptar el diseño a los requisitos del cliente:** Como se ha explicado, el uso de W3CSS que hemos empleado a modo de prototipo es simple y liviano, si para el aspecto final se requiere más animaciones u otro tipo de diseño habría que ajustar este a las necesidades que se requieran. Esto podría realizarse sobre esta misma librería o con otras, como Bootstrap mencionada en la comparativa.
- **Añadir un panel de configuración,** donde podamos controlar la representación de la gráfica, entre los distintos parámetros se proponen:
 - para la representación en tiempo real
 - *La frecuencia de muestreo:* para mayor o menor calidad de los datos, así no requerimos tanta carga de procesado y tamaño del registro de los datos si no

- nos interesa demasiada precisión, por ejemplo, para solo ser observada y no analizar los valores numéricos que nos devuelven.
- *La escala de representación:* Al igual que hemos descrito como implementar el ajuste de la escala de forma dinámica, se podría programar, para una visualización más cómoda, personalizada para el usuario, que esta escala fuera configurable.
- para simulación
- *Velocidad de la representación:* Dado que los datos ya están almacenados, podríamos reproducir la gráfica a la velocidad que querramos, así, si el usuario decide observar más detenidamente un periodo de los datos, o un pulso en concreto, se podría añadir la funcionalidad de reproducir más lento con ajustes predeterminados (como x0.5 o x0.25), o si quiere observarlo de forma general, acelerar su representación (x2 o x3).

7.3. Dificultades

Tras realizar todo el desarrollo de un proyecto de Software por primera vez, la mayor dificultad ha surgido por temas de organización: *¿Qué hago? ¿Por dónde empiezo?*, desde el primer momento, para la selección de las tecnologías a utilizar, con la enorme cantidad de lenguajes de programación distintos que existen, y el elegir una tecnología desconocida, con nuevos conceptos y formas de optimizar el código. Y con todo ello realizar un proyecto ordenado, modular y apto para ser ampliado cómodamente en un futuro.

Entre las dificultades durante el desarrollo, cabe destacar el planteamiento para realizar las simulaciones de representación en tiempo real, donde se comenzó realizando pequeños programas que actualizaban un archivo de texto tal como lo haría la FIFO en el sistema completo. Pero esta forma de simular cambió su perspectiva cuando aprovechamos esta idea para ser implementada como una funcionalidad para la aplicación, de ahí la representación muestra a muestra desde la recodiga de datos de un fichero.

En cuanto a las dificultades a nivel de código, la mayoría fueron solventadas a través de comunidades de desarrollo, como *Stackoverflow*, o estudiando el código de otros desarrolladores, desde *GitHub* u otras páginas orientadas al aprendizaje de programación, así como blogs técnicos personales de los usuarios de internet.

Podemos destacar uno de los problemas, que apareció con el fin de realizar las pruebas de funcionamiento de los límites de la representación. Se comprobó que en la representación muestra a muestra, cuando el número de puntos representados era muy elevado, la ejecución se relantizaba, incluso llegando a bloquearse. Tras eso implementamos el uso del buffer, que mejoró la respuesta, pero seguía teniendo el mismo inconveniente. Para solventar el posible fallo, se tomó la decisión de eliminar los datos en el cliente tras un número de muestras, consiguiendo así el efecto de movimiento de la representación y una fluidez durante todo su recorrido.

7.4. Ampliaciones futuras

A medida que se avanzaba en el curso del proyecto, iban surgiendo ideas de funcionalidades, las cuales no se han implementado por no ser parte del alcance de este proyecto. Estas ideas se

reflejan como modificaciones y nuevas funcionalidades que realizar en el proyecto, algunas de estas son:

- Permitir el uso completo del dispositivo a tiempo real

Se ha pretendido dejar el proyecto lo más organizado posible para poder añadir un bloque que se encargue la recogida de datos del archivo binario `archivo`. Este bloque solo ha de encargarse de leer estos datos y enviarlos al buffer, que recibe los datos en el objeto predefinido.

- Registro de usuarios y Base de datos

Avanzando sobre el tema de la telemedicina, y aspirando un poco más allá del alcance de este proyecto, cabe la posibilidad de implementar un sistema de usuarios con permisos, en el cual, un médico pudiera ver o modificar los datos de sus pacientes, y estos pacientes sólo pudiesen ver sus propios datos. De esta manera se podría tener un registro digital de estos análisis.

Este registro puede hacerse mediante el uso de una tecnología con la que hemos topado, pero no ha sido incluida debido que escapa al contenido del proyecto. El uso de los JWT (Json Web Tokens) nos brinda la oportunidad de realizar un inicio de sesión sin almacenar al usuario en una base de datos propia, sino identificandolo a través de las redes sociales 'Facebook' o su cuenta e 'Google'. JWT almacena unos datos encriptados en el registro local del cliente que solo entiende el servidor, de manera que cuando este acceda a él, sin importar el tiempo que haya transcurrido, el servidor lo reconozca e interactúen ambos extremos. De esta manera evitamos la conocida 'política de privacidad de las cookies', pues todo el intercambio de información personal es consentida y privada.

- Comparativa de resultados

Con el fin de realizar el seguimiento a un paciente, se propone añadir la posibilidad de cargar los datos de dos registros y mostrarlos simultáneamente para ver variaciones de un tiempo a otro.

- Imprimir información del estado de la grafica o exportarla a imagen

Esta funcionalidad viene a agilizar la revisión de los datos por parte de un médico o paciente, tener los datos de la gráfica de forma física (impresa) o almacenada (en imagen) sin la necesidad de volver a acceder a la aplicación para generar y ver los datos.

- Alertas inteligentes según los resultados

Analizando el comportamiento de cada onda, según las anomalías que pueden presentar, se puede programar un sistema inteligente de avisos al médico. Como por ejemplo estableciendo un comportamiento base, de manera que cuando algún dato sea fuera de lo común, guarda los resultados y envía un correo al médico con estos datos o mostrándole directamente la gráfica

REFERENCIAS

- [1] Fernández Manzano, J. (2016). *Codiseño HW/SW en System-on-Chip programable de última generación* (Trabajo Fin de Grado). Escuela Técnica Superior de Ingeniería. Universidad de Sevilla. Disponible en: <http://bibing.us.es/proyectos/abreproy/90549/fichero/Trabajo+Fin+de+Grado.pdf>. [Último acceso: 14/09/2017]
- [2] Guzmán Miranda, H. (2016). *Buenas Prácticas en el Desarrollo de Proyectos* (Seminario). Escuela Técnica Superior de Ingeniería. Universidad de Sevilla.
- [3] La Web del Electrocardiograma. Disponible en: <http://www.my-ekg.com/>. [Último acceso: 14/09/2017]
- [4] BrunoProg64. *Proyecto de Tesis de ECG - Parte I*. Disponible en: <https://brunoprog64.wordpress.com/2010/05/04/proyecto-de-tesis-de-ecg-parte-1/>. [Último acceso: 14/09/2017]
- [5] Node JS. Disponible en: <https://nodejs.org/es/>. [Último acceso: 14/09/2017]
- [6] Express JS. Disponible en: <http://expressjs.com/es/> [Último acceso: 14/09/2017]
- [7] SocketIO. Disponible en: <https://socket.io/>. [Último acceso: 14/09/2017]
- [8] Bootstrap. Disponible en: <https://www.w3schools.com/bootstrap/>. [Último acceso: 14/09/2017]
- [9] W3CSS. Disponible en: <https://www.w3schools.com/w3css/>. [Último acceso: 14/09/2017]
- [10] Plotly. Disponible en: <https://plot.ly/>. [Último acceso: 14/09/2017]
- [11] Smoothie. Disponible en: <http://smoothiecharts.org/>. [Último acceso: 14/09/2017]
- [12] CanvasJS. Disponible en: <https://canvasjs.com/>. [Último acceso: 14/09/2017]
- [13] C3 JS. Disponible en: <http://c3js.org/>. [Último acceso: 14/09/2017]
- [14] D3 JS. Disponible en: <https://d3js.org/>. [Último acceso: 14/09/2017]
- [15] NVD3 JS Disponible en: <http://nvd3-community.github.io/nvd3/examples/documentation.html>. [Último acceso: 14/09/2017]
- [16] CommonMark. Disponible en: <http://commonmark.org/>. [Último acceso: 14/09/2017]
- [17] Stack overflow. Disponible en: <https://stackoverflow.com/>. [Último acceso: 14/09/2017]
- [18] GitHub. Disponible en: <https://github.com/>. [Último acceso: 14/09/2017]
- [19] GitKraken. Disponible en: <https://www.gitkraken.com/>. [Último acceso: 14/09/2017]
- [20] Xilinx Zynq-7000. Disponible en: <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>. [Último acceso: 14/09/2017]
- [21] Xillybus. *Xillinux*. Disponible en: <http://xillybus.com/xillinux>. [Último acceso: 14/09/2017]
- [22] NPM. Disponible en: <https://www.npmjs.com/>. [Último acceso: 14/09/2017]
- [23] Git. Disponible en: <https://git-scm.com/>. [Último acceso: 14/09/2017]
- [24] Xillybus Ltd. *Getting started with Xillinux for Zynq-7000 EPP*. Disponible en: http://xillybus.com/downloads/doc/xillybus_getting_started_zynq.pdf. [Último acceso: 14/09/2017]
- [25] 1webdesigner. *PHP vs Ruby vs Python: The Three Programming Languages in a Nutshell*. Disponible en: <https://1stwebdesigner.com/php-vs-ruby-vs-python/>. [Último acceso: 14/09/2017]

- [26] Bhagat, V. *PHP vs Python vs Ruby: Detailed Comparison*. Disponible en: <https://www.pixelcrayons.com/blog/web/php-vs-python-vs-ruby-comparison/>. [Último acceso: 14/09/2017]
- [27] Kumar, H. *PYTHON vs PHP vs RUBY*. Disponible en: <http://www.findalltogether.com/post/python-vs-php-vs-ruby/>. [Último acceso: 14/09/2017]
- [28] Nodemon. Disponible en: <https://nodemon.io/>. [Último acceso: 14/09/2017]
- [29] JQuery. Disponible en: <https://jquery.com/>. [Último acceso: 14/09/2017]
- [30] 1and1. *Git vs. SVN: ¿cuál es el mejor sistema de control de versiones?*. Disponible en: <https://www.1and1.es/digitalguide/paginas-web/desarrollo-web/git-vs-svn-una-comparativa-del-control-de-versiones/>. [Último acceso: 14/09/2017]
- [31] PhysioBank ATM. Disponible en: <https://physionet.org/cgi-bin/atm/ATM>. [Último acceso: 14/09/2017]
- [32] Pandoc. Disponible en: <https://pandoc.org/>. [Último acceso: 14/09/2017]
- [33] YUIDoc. *YUIDoc Syntax Reference*. Disponible en: <http://yui.github.io/yuidoc/syntax/>. [Último acceso: 14/09/2017]
- [34] Doxygen. Disponible en: <http://www.stack.nl/~dimitri/doxygen/manual/index.html>. [Último acceso: 14/09/2017]
- [35] Semver. *Versionado Semántico 2.0.0-rc.2*. Disponible en: <http://semver.org/lang/es/>. [Último acceso: 14/09/2017]
- [36] JSDoc. Disponible en: <http://usejsdoc.org/>. [Último acceso: 14/09/2017]
- [37] Disponible en: <http://140.120.7.21/LinuxRef/DIYBigData/UbootReference/UbootProgramming.html>. [Último acceso: 14/09/2017]
- [38] *U-Boot programming: A tutorial*. Disponible en: <http://140.120.7.21/LinuxRef/DIYBigData/UbootReference/UbootProgramming.html>. [Último acceso: 14/09/2017]