UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

"Desarrollo e implementación en Django de una plataforma en la nube para la gestión y monitorización de máquinas de analítica clínica"

Titulación:	Grado en Ingeniería en Tecnologías de la Telecomunicación
Mención:	Telemática
Autor:	D. Luis Pellicer Collado
Tutor:	Dr. D. Alfonso Medina Escuela
Fecha:	Julio 2016

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

"Desarrollo e implementación en Django de una plataforma en la nube para la gestión y monitorización de máquinas de analítica clínica"

HOJA DE FIRMAS

Alumno/a

Fdo.: D. Luis Pellicer Collado

Tutor/a

Fdo.: Dr. D. Alfonso Medina Escuela

Fecha: Julio 2016

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

"Desarrollo e implementación en Django de una plataforma en la nube para la gestión y monitorización de máquinas de analítica clínica"

HOJA DE EVALUACIÓN

Calificación: _____

Presidente	**Vocal**	**Secretario/a**
Fdo.:	Fdo.:	Fdo.:

Fecha: Junio 2016

Índice

Capítulo 1. Introducción **1**

1.1 Antecedentes 1

1.2 Motivación y Objetivos 3

1.3 Biosensores 5

1.3.1 Descripción general 5

1.3.2 Biosensores SPR 7

1.3.2.1 Sensorgrama 8

1.4 Contexto del TFG 9

1.5 Solución adoptada 11

Capítulo 2. Infraestructura en la nube **15**

2.1 Introducción 15

2.1.1 Paradigmas y servicios comerciales 17

2.1.1.1 Infraestructura como servicio 19

2.1.1.2 Plataforma como servicio 19

2.1.1.3 Software como servicio 20

2.1.2 Ventajas e inconvenientes de la computación en la nube 21

2.1.3 IaaS vs PaaS 23

2.2 AWS 25

2.2.1 Introducción 25

2.2.1.1 Regiones y zonas de disponibilidad 28

2.2.2 Diseño de la infraestructura 29

2.2.3 Elastic Compute Cloud (EC2) 31

2.2.3.1 Instancias 32

2.2.3.2 Security Group 34

2.2.3.3 Elastic IPs 35
2.2.3.4 Elastic Load Balancing 36
2.2.3.5 Auto scaling 38
2.2.4 Relational Database Service (RDS) 41
2.2.5 Simple Storage Service (S3) 43
2.2.6 CloudFront 44
2.2.7 Identity and Access Management (IAM) 46
2.2.8 Route 53 (Dominio y DNS) 46
2.3 Conjunto de soluciones 48
2.3.1 Introducción 48
2.3.2 Diseño del conjunto sobre EC2 48
2.3.3 Ubuntu 49
2.3.3.1 SSH y Fichero de claves 50
2.3.4 Nginx 53
2.3.5 Python 55
2.3.5.1 uWSGI 56
Capítulo 3. Django 59
3.1 Introducción 59
3.1.1 Instalación 63
3.1.2 Configuración 64
3.1.3 Integración con S3 y CloudFront 68
3.1.4 Integración con RDS 69
3.2 API REST 70
3.2.1 Introducción 70
3.2.1.1 REST 70
3.2.1.2 JSON 71
3.2.1.3 Django REST framework 72

3.2.2 Diseño 73
3.2.3 Desarrollo de la aplicación en Django 74
3.2.3.1 Modelo 75
3.2.3.2 Vista 76
3.3 Aplicación Web 79
3.3.1 Introducción 79
3.3.1.1 Bootstrap 79
3.3.1.2 Google Charts 81
3.3.2 Diseño 83
3.3.3 Desarrollo de la aplicación en Django 85
Capítulo 4. Resultados y Conclusiones 87
4.1 Resultados 87
4.1.1 Resultados de la API REST 87
4.1.2 Resultado de la aplicación web. 96
4.2 Conclusiones 100
4.3 Líneas futuras 101
Presupuesto 103
Bibliografía 113
Anexo I. Código API 121
Anexo II. Código Aplicación Web 125

Índice de figuras

Figura 1. 1 Configuración básica de un biosensor 5
Figura 1. 2 Sensorgrama SPR. 8
Figura 1. 3 Puestos de trabajo con ordenador 10
Figura 1. 4 Distribución en rack de los biosensores. 11
Figura 1. 5 Esquema de la solución adoptada 12

Figura 2. 1 Esquema biosensor SPR. 8
Figura 2. 2 Tráfico de la Web del Servicio Geológico de los Estados Unidos tras producirse un terremoto. 17
Figura 2. 3 Paradigmas de computación en la nube 18
Figura 2. 4 Lista de fallos críticos sin resolver de la plataforma App Engine. 24
Figura 2. 5 Panel de control de AWS con los elementos utilizados para este TFG. 26
Figura 2. 6 Capa gratuita de AWS 27
Figura 2. 7 Logo de AWS Educate. 27
Figura 2. 8 Menú de la consola de AWS que permite seleccionar la Zona de Disponibilidad. 28
Figura 2. 9 Esquema de la plataforma que interviene en el uso de la API por parte de los biosensores SPR. 30
Figura 2. 10 Esquema de la plataforma que interviene en el uso de la aplicación web por parte del personal del laboratorio 30
Figura 2. 11 Instancia seleccionada. 32
Figura 2. 12 Instancia operativa en los servidores de EC2. 34
Figura 2. 13 Balanceador de carga operativo en los servidores de EC2 36

Figura 2. 14 Configuración de la comprobación del estado de las instancias. 37
Figura 2. 15 Tres instancias ejecutándose en paralelo. 39
Figura 2. 16 Creando una imagen (AMI) propia a partir de una instancia..... 39
Figura 2. 17 Grupo de escalado creado satisfactoriamente. 40
Figura 2. 18 Fragmento de las opciones avanzadas de autoscaling. 40
Figura 2. 19 Comprobación del correcto funcionamiento de la base de datos. 42
Figura 2. 20 Creación de un "*bucket*". 43
Figura 2. 21 Distribución generada de CloudFront 45
Figura 2. 22 Creación de IAM 46
Figura 2. 23 Configuración de Alias en Route 53. 47
Figura 2. 24 Diseño del conjunto sobre EC2..... 49
Figura 2. 25 Panel de datos de acceso a la instancia de EC2. 50
Figura 2. 26 Cuota de mercado de los distintos servidores web. 53
Figura 2. 27 Esquema de uso de uWSGI. 56

Figura 3. 1 Patrón de arquitectura Modelo-Plantilla-Vista 60
Figura 3. 2 Ciclo solicitud-respuesta de Django (ilustración de Ryan Nevius)..... 62
Figura 3. 3 Pantalla de bienvenida de Django. 67
Figura 3. 4 Captura de la política de acceso al Bucket de S3..... 68
Figura 3. 5 Esquema de comportamiento de la API. 73
Figura 3. 6 Aplicación web responsive. 80
Figura 3. 7 Proceso de generación de gráficas Google Charts..... 82
Figura 3. 8 Diseño de la pantalla de login para poder ingresar a la plataforma... 84
Figura 3. 9 Diseño del panel de información con la lista de últimos experimentos realizados y las gráficas y configuración del experimento seleccionado. 84
Figura 3. 10 Modelo de comportamiento de la aplicación web en Django..... 85

Figura 4. 1 Distribución de biosensores en el laboratorio. 87
Figura 4. 2 Gráfico de la curva angular obtenido con Gnuplot a partir del fichero Jul31-c01.dat. 88
Figura 4. 3 Gráfico de la curva cinética química obtenido con Gnuplot a partir del fichero Jul25-m01.dat. 89
Figura 4. 4 Uso de Postman, credenciales. 90
Figura 4. 5 Uso de Postman, cuerpo del mensaje. 91
Figura 4. 6 Configuración subida con éxito por el usuario biosensor. 92
Figura 4. 7 Salida por consola del script angular.sh mostrando las respuestas del servidor los las muestras de la curva angular añadidas correctamente a la base de datos por parte del usuario biosensor 95
Figura 4. 8 Salida por consola del script angular.sh mostrando las respuestas del servidor los las muestras de la curva angular añadidas correctamente a la base de datos por parte del usuario biosensor 96
Figura 4. 9 Pantalla de login desde un ordenador sobremesa 97
Figura 4. 10 Pantalla de login desde un terminal móvil. 97
Figura 4. 11 Panel de resultado de los experimentos en un monitor amplio. 98
Figura 4. 12 Panel visto desde un terminal móvil. 99
Figura 4. 13 Los ficheros estáticos tienen su origen en la distribución de CloudFront creada mientras que el contenido dinámico lo hace desde EC2. 100

Índice de tablas

Tabla 2. 1 Ubicaciones de borde de la red global de Amazon CloudFront..... 45

Tabla 5. 1 Costes de los recursos hardware. 104

Tabla 5. 2 Costes de los recursos software..... 105

Tabla 5. 3 Costes de los recursos en la nube. 106

Tabla 5. 4 Factor de corrección en función del número de horas trabajadas. 107

Tabla 5. 5 Presupuesto exento de impuestos. 108

Tabla 5. 6 Costes del material fungible. 109

Tabla 5. 7 Presupuesto para el derecho de visado del COITT. 110

Tabla 5. 8 Aplicación de impuestos. 111

Índice de códigos

Código 2. 1 Error OpenSSH (permisos del fichero de claves incorrectos)..... 51
Código 2. 2 Cambio de permisos al fichero de claves..... 51
Código 2. 3 Comando SSH para conectarse al servidor. 51
Código 2. 4 Mensaje de bienvenida de Ubuntu Server..... 52
Código 2. 5 Fichero de configuración Nginx. 54
Código 2. 6 Fichero de configuración de uWSGI. 57
Código 2. 7 Orden de ejecución de uWSGI..... 57

Código 3. 1 Comando para la creación del entorno virtual de Python. 63
Código 3. 2 Comandos para activar el entorno virtual de Python. 63
Código 3. 3 Comando para la instalación de django mediante el administrador de paquetes pip. 63
Código 3. 4 Comando para la creación del proyecto biosensor en Django..... 64
Código 3. 5 Estructura de archivos del proyecto generado. 64
Código 3. 6 Comandos para la creación de las aplicaciones biosensor_api y biosensor_app. 65
Código 3. 7 Lista de aplicaciones instaladas en el fichero settings.py. 66
Código 3. 8 Comando para lanzar el servidor. 66
Código 3. 9 Comando para actualizar el esquema de la base de datos. 67
Código 3. 10 Comando para crear un usuario administrador en Django. 67
Código 3. 11 Configuración añadida al archivo settings.py. 67
Código 3. 12 Comando para la instalación de boto y django-storages mediante el administrador de paquetes pip. 68

Código 3. 13 Contenido del fichero aws_storage_classes.py. 69
Código 3. 14 Variables de configuración de S3 y CloudFront en el fichero settings.py. 69
Código 3. 15 comando para la instalación de MySQL-python mediante el administrador de paquetes pip. 69
Código 3. 16 Variables de configuración de RDS en el fichero settings.py. 70
Código 3. 17 Comando para la instalación de Django REST framework mediante el administrador de paquetes pip. 74
Código 3. 18 Lista de aplicaciones instaladas en el fichero settings.py 74
Código 3. 19 Configuración de Django REST framework en el fichero settings.py. 74
Código 3. 20 Entrada añadida al fichero urls.py 75
Código 3. 21 Modelo de las aplicaciones biosensor_app y biosensor_api. 76
Código 3. 22 Clases principales del fichero views.py 77
Código 3. 23 Clases relevantes del fichero serilizers.py 78
Código 3. 24 Directorio Bootstrap. 80
Código 3. 25 Código para cargar las librerías de Google Charts. 82
Código 3. 26 Variables para la creación de gráficas con Google Charts. 83
Código 3. 27 Código HTML donde se incluye la gráfica de la curva cinética química. 83
Código 3. 28 Configuración del fichero urls.py de la aplicación biosenosr_app. .. 85
Código 3. 29 Configuración del archivo views.py de la aplicación biosensor_app. 86

Código 4. 1 Configuración del primer experimento enviado. 89
Código 4. 2 Ejemplo de la primera muestra de la curva angular perteneciente al primer experimento realizado. 93

Código 4. 3 Código del script cinetica.sh que genera la curva cinética a partir de los valores del fichero Jul25-m01.dat. 93
Código 4. 4 Código del script angular.sh que genera la curva cinética a partir de los valores del fichero Jul31-c01.dat. 94

Acrónimos

Acrónimos	Descripción
AMI	Amazon Machine Image
API	Application Programming Interface
ARN	Amazon Resource Names
AWS	Amazon Web Services
CPD	Centro de Procesamiento de datos
DDoS	Distributed Denial of Service
DNS	Domain Name System
EC2	Elastic Compute Cloud
ELB	Elastic Load Balancing
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a service
IETF	Internet Engineering Task Force
IP	Internet Protocol
IUMA	Instituto Universitario de Microelectrónica Aplicada
JSON	JavaScript Object Notation
LAMP	Linux, Apache, MySQL/MariaDB, Perl/PHP/Python
MEAN	MongoDB, Express, AngularJS, Node.js

MOTD	Message of the Day
MVT	Model View Template
ORM	Object-Relational mapping
PaaS	Platform as a service
POC	Point of care
RDS	Relational Database Service
REST	Representational State Transfer
SaaS	Software as a service
SO	Sistema Operativo
SPA	Single Page Application
SPR	Surface Plasmon Resonance
SSL	Secure Sockets Layer
TLS	Transport Layer Security
VPC	Virtual Private Cloud
WSGI	Web Server Gateway Interface
XML	Extensible Markup Language

Capítulo 1. Introducción

1.1 Antecedentes

En los últimos años, se ha generado un gran interés en el ámbito de los biosensores debido al avance de la tecnología. Los biosensores son dispositivos que presentan un amplio rango de posibilidades que van desde la diagnosis clínica a la monitorización ambiental o sectores farmacéuticos o alimenticios [1]. Es por ello que se busca mejorar los aspectos técnicos de los biosensores intentando disminuir el tiempo de detección, mejorar la simplicidad y ligereza de los dispositivos y reducir el coste de los mismos. Por otro lado, las aplicaciones de los biosensores no sólo están destinadas al ámbito de la investigación científica, sino que también son útiles en diversos sectores de la sociedad moderna y, por tanto, teniendo un gran impacto socioeconómico en esta [2].

Otra de las cuestiones por la que los biosensores han experimentado un gran desarrollo es debido al interés por la descentralización del diagnóstico clínico hacia diferentes lugares de atención médica. Este fenómeno se denomina POC y entre estos puntos nos podemos encontrar localizaciones como centros de atención primaria, clínicas, unidades hospitalarias de urgencias y cuidados intensivos, quirófanos, ambulancias, lugares de trabajo, en nuestros propios hogares e incluso en lugares remotos como en operaciones militares o en misiones espaciales. Por otro lado, los POC permiten desarrollar sistemas para llevar a cabo cribas de pacientes de riesgo, desde el diagnóstico y el seguimiento del tratamiento a la vigilancia de recurrencia de ciertas enfermedades en este tipo de emplazamientos, de manera más personalizada y eficaz[3].

Con el fin de satisfacer las exigencias del mercado, es necesario que estos sistemas posean ciertas características que cumplan con la calidad clínica exigida. Algunos de estos aspectos son la presión, sensibilidad y selectividad de la analítica clínica. También se pretende reducir los tiempos de análisis, hacer el manejo más sencillo, reducir el mantenimiento, calibración, automatización y mejorar la conectividad de los dispositivos. La ingeniería asociada a estas técnicas todavía está en busca de un dispositivo que sea capaz de cumplir todos estos requisitos, pero no es trivial. A día de hoy, se continúa investigando para mejorar los avances en este campo [4].

La innovación en el campo de los biosensores así como la necesidad de descentralización de los dispositivos hacen que, día a día, se busquen nuevas líneas de trabajo que mejoren la eficacia del personal de laboratorio y que permitan llevar a cabo un mayor número de experimentos de forma simultánea. Además, el almacenamiento de la información en un único lugar permite el análisis del "big data". De este modo, se pueden obtener resultados más concluyentes mediante la detección de patrones en los datos almacenados a gran escala. Esto sería posible incorporando la computación en la nube en los servicios ofrecidos por los biosensores. De esta forma, se eliminan paulatinamente la presencia de los ordenadores dedicados a estas tareas.

El concepto de computación en la nube hace referencia a un paradigma que permite ofrecer servicios de computación a través de la red sin la necesidad de que el usuario final posea conocimientos técnicos de la gestión de los recursos utilizados. La sociedad ha mostrado un gran interés en este ámbito y, por tanto, el desarrollo de este tipo de servicios ha aumentado de manera exponencial. La sinergia surgida por las mejoras en la capacidad de cómputo, almacenamiento y velocidad de conexión a Internet ha creado un auténtico ecosistema de servicios en la nube.

Uno de los mayores atractivos de este servicio es la posibilidad de acceder a la información desde cualquier dispositivo que posea conexión a Internet, como por ejemplo, teléfonos móviles, tablets u ordenadores personales ubicados en cualquier lugar geográfico. Además, se permite que los usuarios puedan conectarse de manera simultánea para acceder a los recursos. Por otro lado, la computación en la nube presenta un nivel de inversión y reducción de los costes operacionales atractivo, jugando un papel importantísimo a la hora de planificar el despliegue de servicios de esta índole. No solo se optimizan los recursos, sino que es posible escalar la utilización de la infraestructura de forma fácil, rápida y eficiente.

En cuanto a cuestiones técnicas, la virtualización de los servicios también permite la compartición de servidores y dispositivos de almacenamiento posibilitando que las aplicaciones puedan ser fácilmente migradas de un servidor a otro. Por otro lado, el mantenimiento es menos costoso gracias a que no necesita ser instalado en los dispositivos de cada usuario sino que es accesible a través del navegador web. Otro valor añadido, no por ello menos importante, es el tipo de software a instalar en estos servidores. La mayoría de ellos son tipo código abierto, así como los frameworks, lenguajes de programación y entornos de desarrollo.

1.2 Motivación y Objetivos

El objetivo fundamental de este TFG es desarrollar una plataforma de gestión y monitorización remota de máquinas de análisis clínico. Dicha plataforma consistirá principalmente en la elaboración de una infraestructura en la nube que de soporte al almacenamiento de los experimentos realizados en dichas máquinas a través de una API que se comunique con las aplicaciones que administran los equipos, así como el desarrollo de una aplicación web que permita un fácil acceso a

los datos experimentales generados por estos dispositivos. Por lo tanto, los objetivos que se llevan a cabo son los siguientes:

- Conocer el funcionamiento de los biosensores, así como, los datos que éstos generan con el fin de poder contextualizar este Trabajo de Fin de Grado y poder gestionar la información proporcionada por los mismos.

- Estudiar los servicios ofrecidos en la nube por AWS, siendo necesario la recopilación de información sobre la instalación y configuración de los servicios, prestando especial atención a su capa gratuita.

- Conocer el funcionamiento del servidor web Nginx, así como, su configuración e integración con distintas plataformas web.

- Estudiar el entorno de desarrollo del framework Django, así como la creación de APIs RESTful y el diseño e implementación de aplicaciones web.

- Desarrollar e implementar la plataforma que estará hospedada en los servicios de computación en la nube que ofrece AWS. En ellos se creará y configurará una base de datos donde almacenar los resultados obtenidos a partir de los experimentos.

- Desarrollar una API siguiendo en el estilo de arquitectura de software REST que permita la correcta comunicación entre la aplicación cliente y el servidor en la nube.

- Diseñar e implementar una aplicación web mediante Django que posibilite al usuario final acceder cómodamente a la plataforma a través de su navegador.

1.3 Biosensores

1.3.1 Descripción general

Los biosensores son dispositivos capaces de detectar la presencia de analitos en una nuestra biológico debido a medición de ciertas variaciones de parámetros físicos o químicos. Esta detección se produce en el entorno de la superficie sensora gracias a una capa de reconocimiento biológico que es capaz de reconocer la molécula que se pretende analizar. Este elemento puede estar compuesto por anticuerpos, proteínas, enzimas, receptores celulares o ADN entre otros. El otro componente esencial de los biosensores es el transductor, el cual es capaz de generar una señal eléctrica proporcional a la concentración de analito que interacciona con el elemento biológico. Los transductores pueden ser de diferentes tipos, como por ejemplo, óptico, electro-químico, piezoeléctrico o magnético.

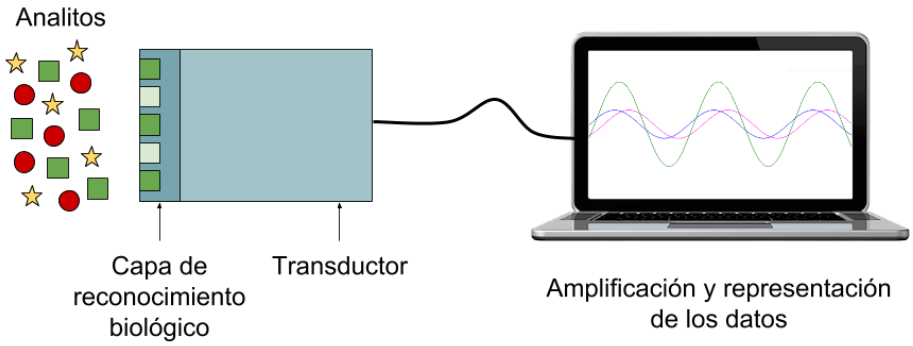


Figura 1. 1 Configuración básica de un biosensor

En la Figura 1. 1 se muestra el esquema básico de funcionamiento de un biosensor. Las principales características de los biosensores son aportadas por sus dos componentes principales, es decir, la capa de reconocimiento biológico y el transductor. La capa de reconocimiento biológico aporta selectividad ante los

diferentes tipos de analito. Por otro lado, el transductor contribuye con la sensibilidad conseguida mediante un mecanismo de traducción adecuado. Una vez el transductor genere la señal correspondiente a la cantidad de analito detectada, se procede a amplificar la señal para que pueda ser representada e interpretada por un sistema electrónico. [4].

A parte de las características mencionadas, los biosensores presentan otras características esenciales como pueden ser una elevada fiabilidad, de gran importancia para la integración en máquinas de analítica clínica. Además se tiene que tener en cuenta la estabilidad con el fin de que otorgue un tiempo de vida suficientemente largo para su amortización económica. También es importante la capacidad de miniaturización para que los dispositivos puedan ser portátiles. Por otro lado, es interesante reducir el pretratamiento de la muestra para simplificar los procedimientos y protocolos en el uso del biosensor, así como, el reducido volumen de muestra. Asimismo, es importante que el manejo del dispositivo sea sencillo y posea capacidad de automatización, para que puedan ser integrados en procesos industriales o en estaciones de medida.

Sin embargo, conseguir dispositivos que cumplan con todas estas características es muy difícil, pero algunos de los biosensores desarrollados hasta el momento reúnen muchas de ellas, por lo que presentan numerosas ventajas respecto a las técnicas analíticas convencionales [5].

La naturaleza multidisciplinar de estos tipos de dispositivos hace que estén presentes en diferentes áreas del conocimiento científico y tecnológico. Debido a esto, el desarrollo de biosensores basados en diferentes tipos de transductores y elementos de reconocimiento biológicos, se ha visto impulsado en los últimos años por los avances en diversos campos. Estos campos abordan ámbitos medioambientales, la industria agroalimentaria y farmacéutica e incluso la seguridad y defensa nacional.

1.3.2 Biosensores SPR

Los biosensores de resonancia de plasmón superficial pertenecen a la categoría de los dispositivos ópticos refractométricos, ya que miden los cambios de índice de refracción producidos en el campo de una onda electromagnética contenida en la estructura del sensor [6].

Los dispositivos de SPR detectan interacciones entre un elemento de reconocimiento inmovilizado en una superficie de oro y el analito en disolución como se muestra en la Figura 2. 1 Esquema biosensor SPR. Si un haz de luz monocromática polarizada incide sobre esta superficie en un determinado ángulo, se produce el fenómeno de la resonancia de plasmón superficial. Este fenómeno absorbe la luz a un ángulo que depende del índice de refracción del medio adyacente a la superficie metálica, ya que lo atraviesa el campo evanescente. Cualquier actividad en la interacción molecular producirá un cambio en el ángulo de absorción de la luz. Es decir, las moléculas de analito que entran en contacto con el biosensor, se unen al elemento de reconocimiento, produciéndose un incremento del índice de refracción local en la superficie. Los cambios en el índice de refracción de este medio producen cambios en la constante de propagación del plasmón superficial, que a su vez altera características de la luz acoplada a éste como el ángulo y la longitud de onda de acoplamiento, la intensidad o la fase. Al registrar de forma continua la luz reflejada, se obtiene un perfil de los cambios del índice de refracción en el medio adyacente a la superficie metálica, obteniendo información en tiempo real de las interacciones que ocurren en él. En función de cuáles de estas características de la luz sean moduladas por el plasmón de superficie.

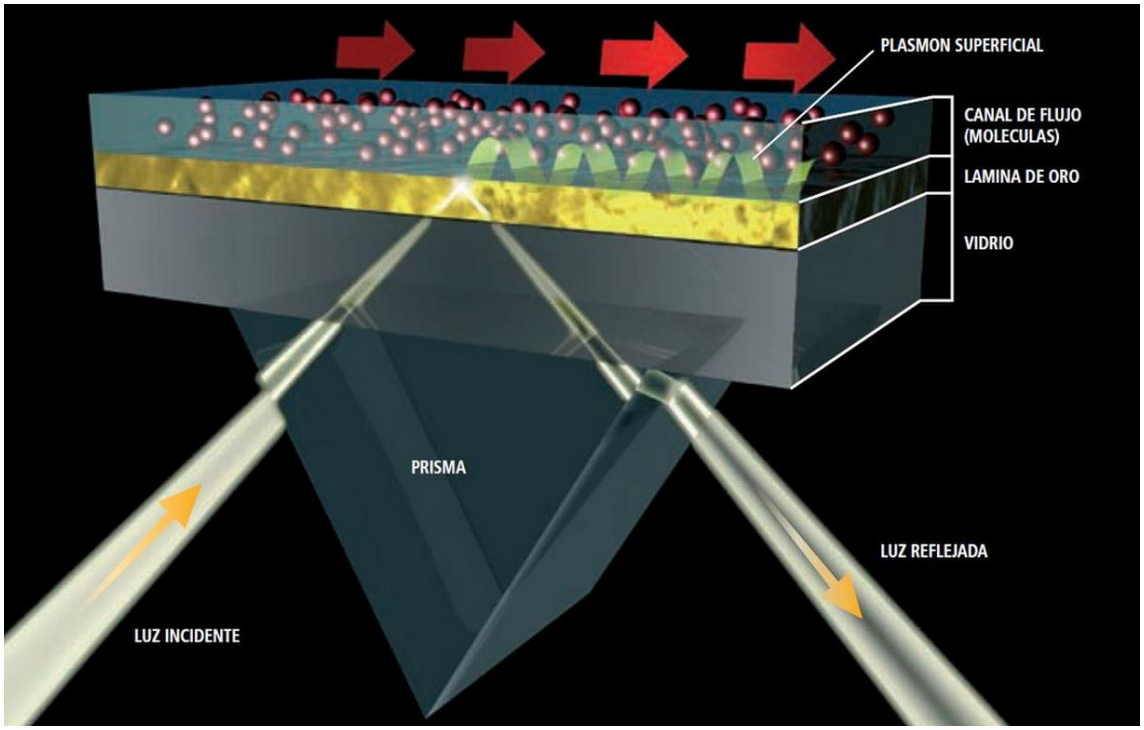


Figura 2. 1 Esquema biosensor SPR.

1.3.2.1 Sensorgrama

El sensorgrama es la representación gráfica de la evolución de la señal SPR en el tiempo. Esta curva representa la cinética de las interacciones químicas que están ocurriendo en la superficie sensora, dado que, la magnitud de la reflectancia medida ofrece la información necesaria para estimar la concentración del analito.

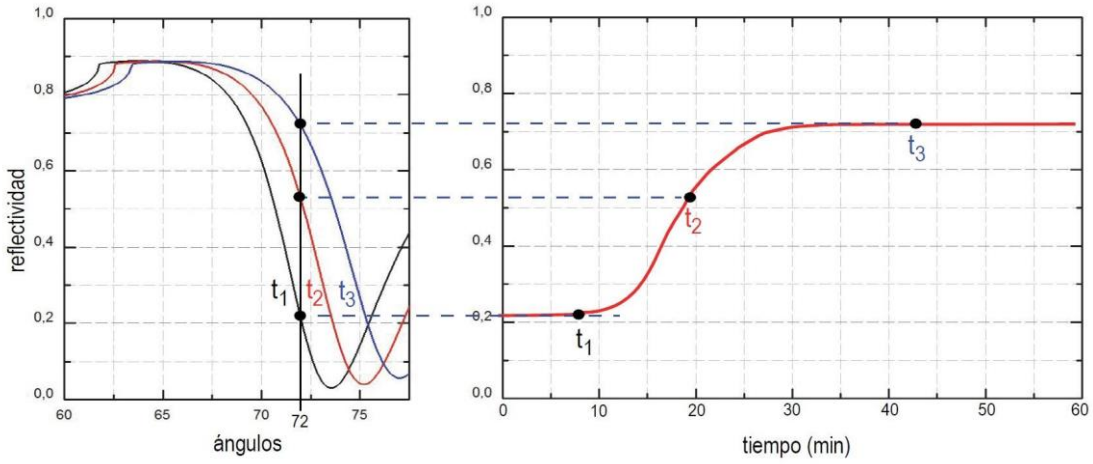


Figura 1. 2 Sensorgrama SPR.

1.4 Contexto del TFG

Este Trabajo de Fin de Grado, se ha desarrollado dentro de las líneas de investigación de la división de Equipos y Sistemas de Comunicación del IUMA, donde se trabaja en la optimización de la automatización de los biosensores, así como en facilitar el manejo de los mismos.

Mediante la incorporación de servicios de computación en la nube se ofrece la posibilidad de almacenar toda la información de los distintos experimentos realizados desde los biosensores a un servidor remoto, permitiendo el acceso a estos datos en cualquier momento y en cualquier lugar a través de una WebApp. La recolección de datos se realiza a través de la implementación de una API que permite comunicar el biosensor con el servidor. Por otro lado, gracias a la WebApp, se obtiene una interfaz gráfica que permite visualizar los resultados de los experimentos desde cualquier dispositivo.

La comunicación con el biosensor se ha desarrollado dentro del grupo de trabajo donde se ha implementado una aplicación Android con el fin de dotar a las máquinas de analítica clínica de una pantalla táctil que permita eliminar los ordenadores que han ido acompañando a estos dispositivos en todo momento tal como se muestra en la Figura 1. 3. De esta manera, el espacio en el laboratorio donde se encuentran los biosensores, y se facilita el manejo de estos dispositivos. Por otro lado, debido al uso de tablets Android se dota de conexión a Internet a las máquinas de analítica clínica, permitiendo el envío de los datos obtenidos al servidor.

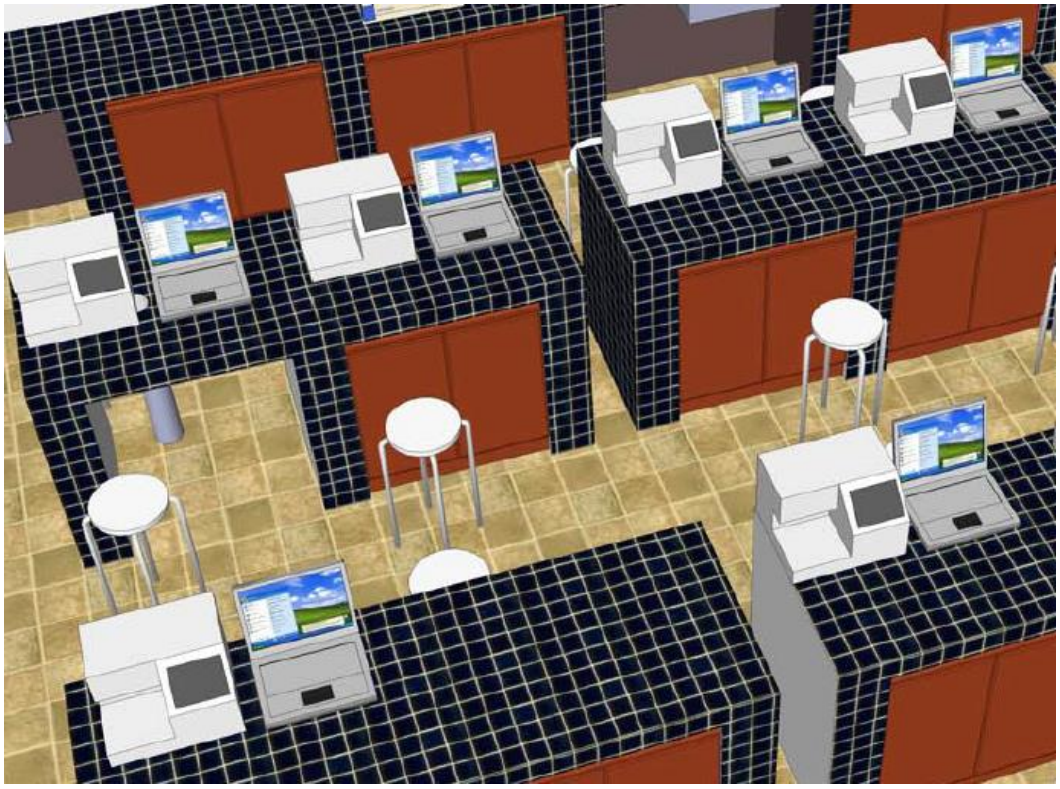


Figura 1. 3 Puestos de trabajo con ordenador

Al eliminar la dependencia de disponer de un ordenador junto a cada biosensor para consultar los resultados experimentales que estos generan, se consigue realizar una mejor gestión de estos dispositivos. Además, se abre la posibilidad de nuevas formas de ubicar dentro de los laboratorios este tipo de máquinas de analítica clínica. En este Trabajo de Fin de Grado se propone una solución como la que se muestra en la Figura 1. 4.

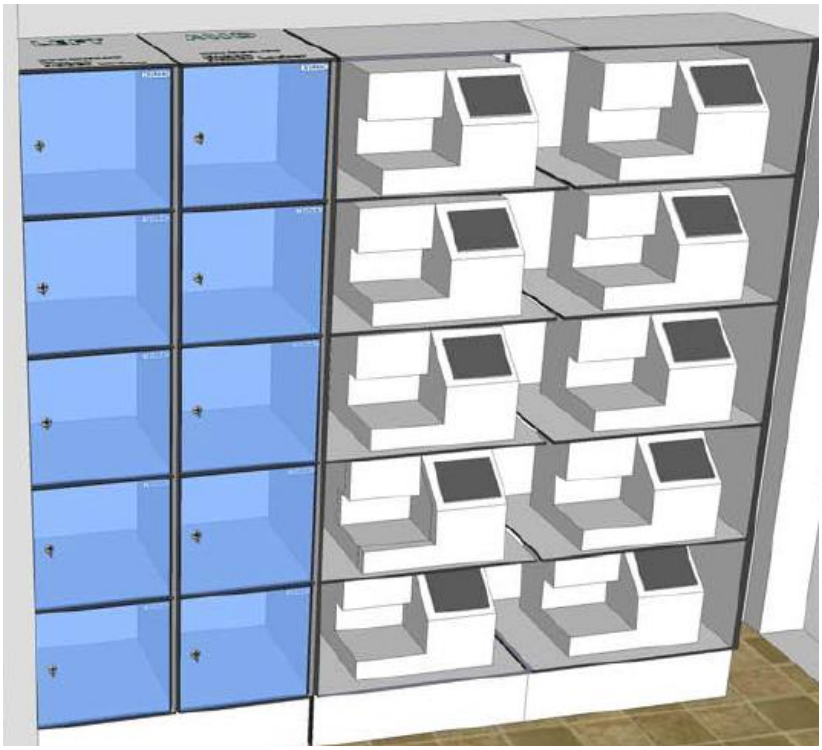


Figura 1. 4 Distribución en rack de los biosensores.

1.5 Solución adoptada

Se pretende desarrollar una infraestructura escalable en la nube que sirva para proporcionar una plataforma donde un número indeterminado de biosensores SPR envíen mediante una API la información de los experimentos realizados y que estos puedan ser consultados por el personal de laboratorio remotamente a través de una Aplicación Web.

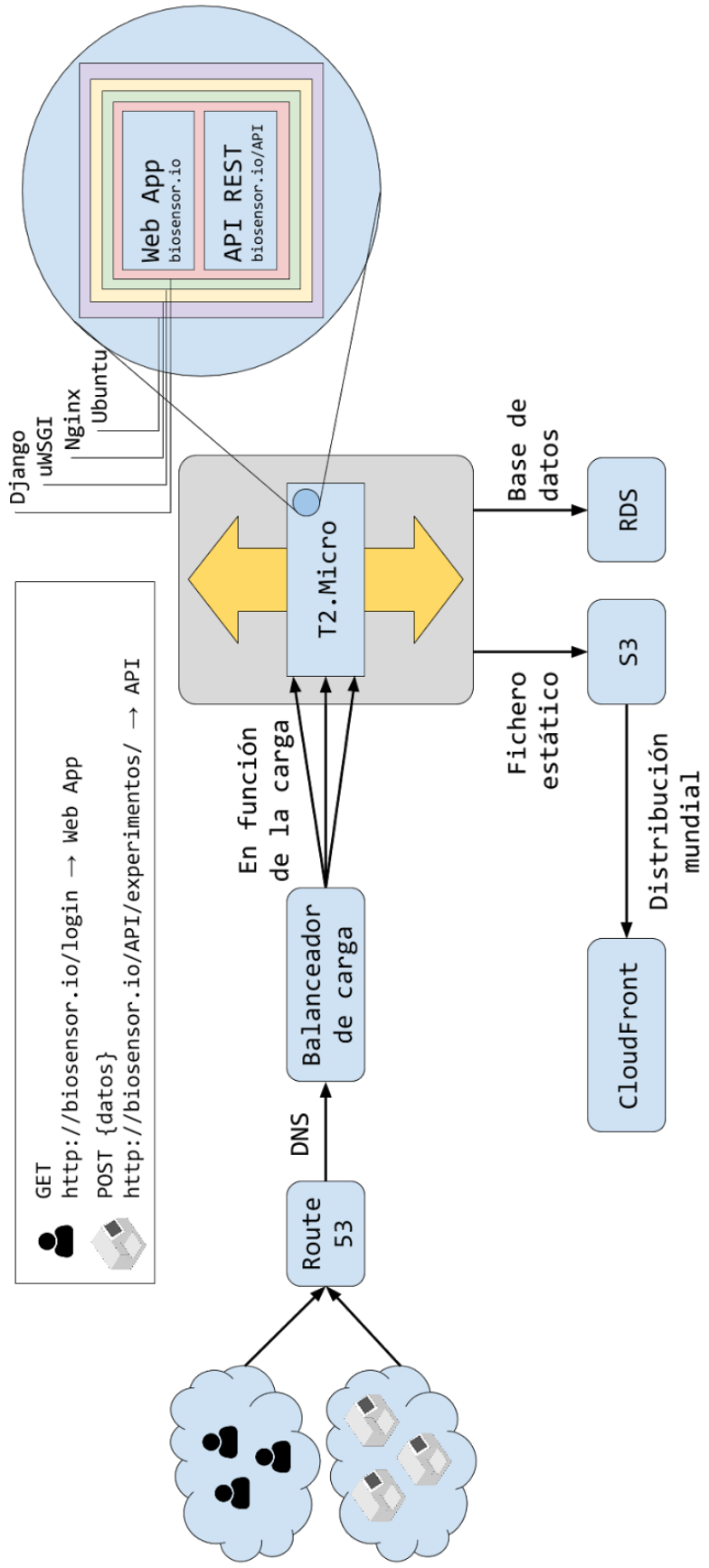


Figura 1. 5 Esquema de la solución adoptada.

Por tanto, se propone una arquitectura como la expuesta en la Figura 1. 5. En este esquema se pueden observar los diferentes elementos que intervienen en la infraestructura para llevar a cabo este proyecto. Se pueden apreciar dos grandes bloques, el primero de ellos correspondiente a la infraestructura en sí. El siguiente bloque hace referencia a la plataforma que soporta tanto a la API como a la aplicación web.

Podemos dividir en dos grupos los tipos de clientes que harán uso de la misma. Por un lado, el personal de laboratorio podrá hacer uso de la aplicación web con el fin de consultar los experimentos que el otro grupo de clientes, en este caso biosensores SPR, habrán ido generando y subiendo a la plataforma mediante la API REST.

A pesar de tratarse en todo momento de tráfico HTTP, el comportamiento de la plataforma en ambos casos de uso no es exactamente el mismo, puesto que el biosensor sólo manejará datos en JSON, mientras que el personal de laboratorio demandará páginas web HTML, cuyos ficheros estáticos estarán repartidos por todo el mundo, mediante el uso de la red de distribución de contenidos de Amazon.

Para hacer posible la labor de esta plataforma ha de usarse un conjunto de herramientas que integren las diferentes partes en un todo, preferiblemente haciendo uso de tecnologías de software libre y gratuito. El framework Django sin duda encaja en esta descripción y, por tanto, será ese elemento de unión que vuelva al conjunto funcional y le dote de herramientas con las que poder desarrollar tanto la API como la aplicación web y cualquier otra línea futura sobre la que se quiera poder trabajar.

Capítulo 2. Infraestructura en la nube

2.1 Introducción

El cloud computing o computación en la nube ha pasado de ser una mera perspectiva de futuro a una realidad cotidiana. Herramientas como Dropbox, Google Drive o servicios como Netflix ya forman parte de nuestro día a día gracias a la sinergia surgida por las mejoras en la capacidad de cómputo, almacenamiento y velocidad de conexión a Internet que se han venido experimentando en los últimos años. Dichas mejoras están propiciando una auténtica eclosión de servicios en la nube puestos a disposición de un público diverso, desde servicios orientados a usuarios comunes como los tres ejemplos anteriormente citados, hasta soluciones focalizadas en profesionales del sector TIC como las que se proponen en este TFG.

El concepto de computación en la nube hace referencia a un paradigma que permite ofrecer servicios de computación a través de la red sin la necesidad de que el usuario final posea conocimientos técnicos de la gestión de los recursos utilizados. Un símil muy recurrido y que ejemplifica muy bien este fenómeno es el de la producción de energía eléctrica. A finales del siglo XIX y principios del XX, cada fábrica poseía un sistema de generación propio, pero a medida que la red de distribución fue mejorando y las centrales de las compañías eléctricas se fueron volviendo cada vez más confiables, el número de fábricas que dejó de producir su propia electricidad disminuyó drásticamente, delegando esta tarea a las compañías eléctricas. Esta es la analogía con la situación actual, donde muchas empresas están externalizando sus recursos informáticos llevándolos a la nube.

Hace solo unos años los administradores de sistemas a menudo tenían que hacer frente a la difícil decisión de elegir el costoso equipamiento con el que dotar sus CPDs. Una tarea nada sencilla, si además de los aspectos puramente técnicos como los requisitos necesarios o los problemas que puedan surgir a la hora de migrar a un nuevo hardware, se tienen en cuenta aspectos económicos como el periodo de amortización o el consumo eléctrico.

Un ejemplo remarcable de lo rápido que se devalúan los equipos es el caso del mainframe de IBM z890 modelo 2086-320 que, en el año 2004, cuando saliór al mercado, su precio ascendía a los 340.000$. Tan solo once años después, a finales del 2015, un adolescente llamado Connor Krukosky instaló uno en el sótano de su casa tras comprarlo de segunda mano por tan solo 237$ [7].

Del mismo modo, hace unos años, los desarrolladores web debían preocuparse en gran medida por los límites de sus plataformas ya que, en función de las necesidades de escalado, podían verse forzados a tener que migrar a otra de mayor capacidad y, por norma general, más cara. Incluso haciendo esto último, era complicado que las plataformas manejaran correctamente situaciones puntuales donde la demanda de tráfico se pudiera incrementar drásticamente.

Un ejemplo de estas situaciones es el conocido como Slashdot Effect, Reddit Effect o Efecto Menéame [8] en España, que sucede cuando un sitio web popular y, por tanto, hospedado por una gran plataforma capaz de gestionar mucho tráfico, enlaza a una página pequeña provocándole de forma involuntaria un ataque DDoS al verse incapaz de atender el incremento masivo del número de usuarios que acaba por colapsar sus servidores temporalmente. En la Figura 2. 2 se puede observar este mismo fenómeno producido tras ocurrir un terremoto en Estados Unidos [9].

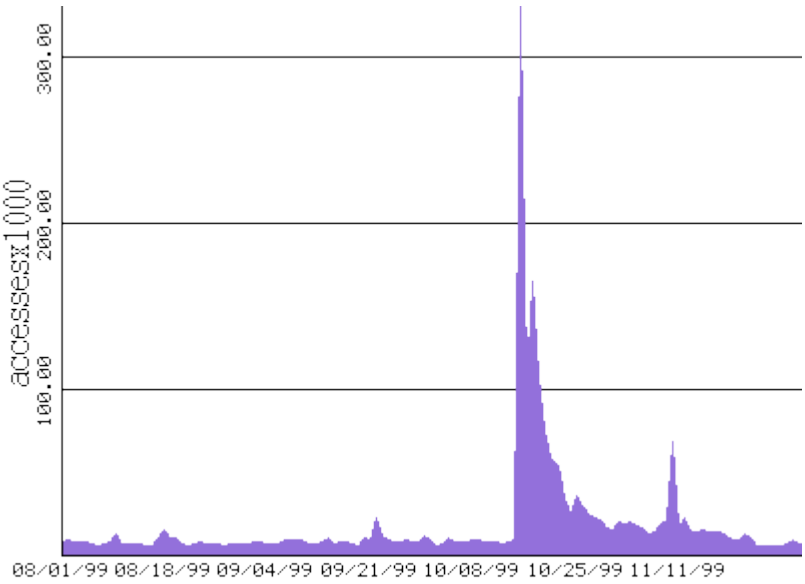


Figura 2. 2 Tráfico de la Web del Servicio Geológico de los Estados Unidos tras producirse un terremoto.

Hoy en día, mediante cloud computing, un administrador de sistemas puede dedicarse a realizar sus tareas sin importar qué equipos comprar y un desarrollador web puede dedicarse a programar sin necesidad de depender de los administradores de sistemas. Incluso un usuario final puede utilizar una aplicación sin necesidad de descargar e instalar un ejecutable. Como se puede observar en estos ejemplos expuestos, la nube se ha vuelto transversal al mercado tecnológico, es por ello que en este capítulo se describirán diferentes servicios que se ofertan y qué criterios se han tomado para seleccionar entre unas u otras soluciones.

2.1.1 Paradigmas y servicios comerciales

En la actualidad existe una enorme variedad de soluciones Cloud disponibles en el mercado. Tanto es así que hoy día el reto para el administrador de sistemas ya no reside en saber discernir qué equipamiento comprar, sino en elegir correctamente entre el amplísimo ecosistema de servicios en la nube presentes en el mercado y enfocados a

su labor. Lo mismo sucede con los desarrolladores que disponen de herramientas para programar aplicaciones. Incluso los usuarios finales ven como, a cada día que pasa, el abanico de soluciones en la nube a su medida es más y más amplio (Dropbox, Google Docs, Office 365, Netflix, etc.).

Atendiendo al servicio ofrecido, el fenómeno de la computación en la nube puede dividirse en tres grandes grupos (Figura 2. 3): Infraestructura como servicio (IaaS), plataforma como servicio (PaaS) y software como servicio (SaaS). Cada uno de ellos puede interpretarse como capas de una pirámide donde la "infraestructura como servicio" es la más próxima al hardware existente en las granjas de servidores mientras que el "software como servicio" es la capa más cercana al usuario final. A continuación, se describirán en estas categorías explicando cómo su existencia ha facilitado el desarrollo de los objetivos de este TFG.

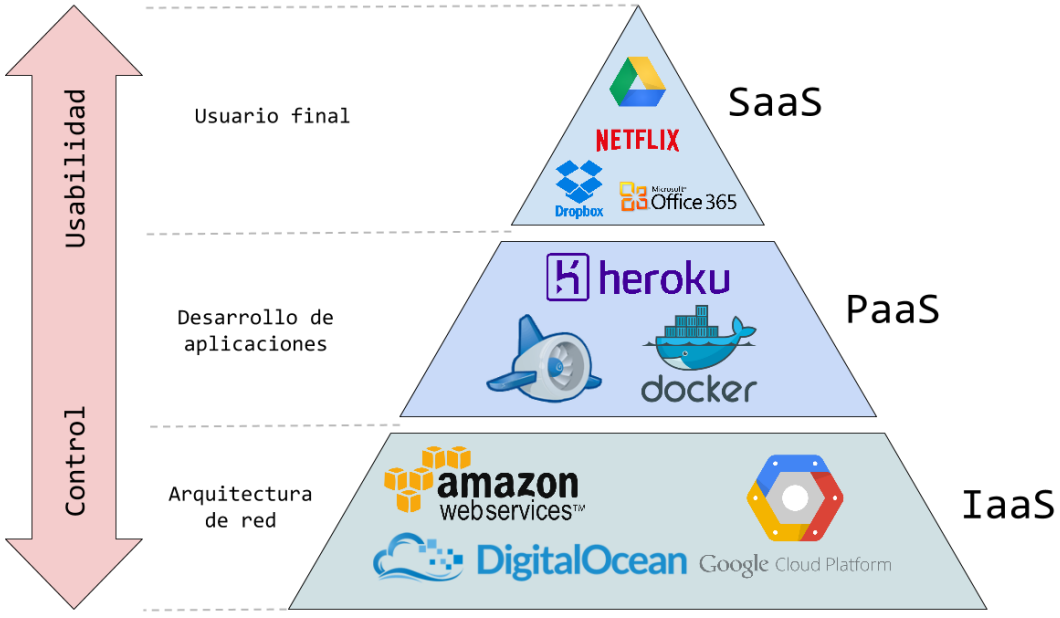


Figura 2. 3 Paradigmas de computación en la nube

2.1.1.1 Infraestructura como servicio

De acuerdo con el Internet Engineering Task Force (IETF) [10], el modelo más básico de servicio en la nube es aquel donde los proveedores ofrecen una infraestructura de cómputo en la que los usuarios pueden abstraerse de los detalles físicos de la misma. A este modelo se le denomina infraestructura como servicio aunque en ocasiones también se le llama hardware como servicio, es decir, un medio para entregar almacenamiento, capacidades de cómputo y transporte de datos a modo de servicios estandarizados en la red.

La principal característica de este tipo de soluciones comerciales es el uso de servidores virtuales. Amazon Web Services (AWS) es el ejemplo comercial más conocido de este paradigma y será el utilizado para desarrollar el presente TFG donde se utilizarán, entre otros, los servicios EC2 y S3 que ofrecen capacidad de cómputo y de almacenamiento respectivamente. Sin embargo, otras empresas que ofertan este tipo de servicios son: Google Compute Engine, Microsoft Azure, DigitalOcean, Linode, OpenStack o Joyent entre otros.

2.1.1.2 Plataforma como servicio

Esta categoría de servicio de computación en la nube ofrece a los clientes poder desarrollar, ejecutar y administrar aplicaciones sin la complejidad de construir y mantener la infraestructura típicamente asociada con el desarrollo y el lanzamiento de una aplicación. Por ejemplo, un entorno de desarrollo (Django) junto a un servidor web, ambos integrados sobre un sistema Linux. Esta es la plataforma que se ha desarrollado desde cero para este TFG gracias a la Infraestructura como servicio proporcionada por Amazon. En el sección 2.1.2, se explicarán los motivos por los que se ha creado una plataforma propia en lugar de elegir una entre la enorme variedad de ellas presentes en el mercado.

Las ofertas de PaaS pueden dar servicio a todas las fases del ciclo de desarrollo y pruebas del software, o pueden estar especializadas en cualquier área en particular, tal como la administración del contenido. Ejemplos comerciales son: Heroku, Google App Engine, OpenShift, Microsoft Azure (que también provee IaaS), etc. El abanico es tan amplio que incluso existen varios sitios dedicados exclusivamente a ayudar a escoger plataforma. Entre ellos destaca PaaSify un portal que busca y compara entre más de 60 opciones.

2.1.1.3 Software como servicio

En este grupo el proveedor del servicio ofrece un modelo de distribución de software donde el soporte lógico y los datos que se manejan están alojados en servidores externos a los que normalmente se accede mediante una aplicación web, por lo general multiplataforma. Ejemplos de este paradigma son: Dropbox, Google Apps (Gmail, Drive, etc.), Salesforce CRM, GoToMeeting, etc. Dichos servidores pueden pertenecer al propio proveedor o puede que a su vez éste los tenga externalizados, como sucede por ejemplo con Dropbox o Netflix, ya que ambos utilizan la infraestructura de Amazon Web Service.

Uno de los principales puntos fuertes del uso de este servicio es la reducción potencial de los costes del soporte técnico ya que por norma general si el dispositivo tiene acceso a Internet la plataforma estará operativa.

La WebApp que se ha desarrollado en este TFG pertenece a esta categoría ya que, para el personal de laboratorio, no se vuelve necesario la instalación de ningún software específico para poder acceder a los resultados de sus experimentos.

2.1.2 Ventajas e inconvenientes de la computación en la nube

La computación en la nube nos permite acceder a todas sus aplicaciones y documentos desde cualquier lugar del mundo. Este paradigma representa un gran cambio en la forma en la que se almacena la información y se ejecutan aplicaciones. En términos generales, la computación en nube tiende a beneficiar al usuario final. Sin embargo, se deben valorar cuáles son las ventajas y desventajas que ésta nos ofrece [11].

Ventajas

- No se necesitan dispositivos de alta potencia y capacidad de procesamiento debido a que las aplicaciones se ejecutan en la nube, es decir, no es necesario instalarlas en un disco duro. En el caso de las empresas, en lugar de tener que realizar una cuantiosa inversión en centros de datos y servidores antes de saber qué uso les va a dar, puede utilizar la computación en la nube y pagar únicamente en función del consumo realizado.
- Actualizaciones de software instantáneas. Desde el punto de vista del usuario final, las aplicaciones basadas en la web se actualizan de forma automática, estando disponibles la próxima vez que se inicie sesión en dicha aplicación.
- Capacidad de almacenamiento casi ilimitada. La computación en la nube nos ofrece un espacio de almacenamiento en función de nuestras necesidades, pudiendo redimensionar el almacenamiento dinámicamente.
- Mayor fiabilidad de los datos. Toda la información almacenada en la nube está menos expuesta a que se puedan perder los datos por un fallo en el disco duro.
- Acceso universal a los documentos. Se puede acceder a los datos en cualquier momento y desde cualquier dispositivo que posea conexión a internet.
- Independencia del dispositivo. Los datos son los mismos sin importar qué ordenador o dispositivo se esté utilizando para acceder a ellos, ya pueda ser desde

un ordenador, smartphone o tablet y sin importar el sistema operativo del dispositivo.

- Facilitar la colaboración en grupo. La computación en la nube permite que múltiples usuarios puedan colaborar fácilmente en documentos y proyectos.

Desventajas

Hay una serie de razones por las que es posible que haya usuarios que no quieran adoptar la computación en la nube para sus necesidades particulares. Vamos a examinar algunos de los riesgos relacionados con este tipo de computación.

- Se requiere una conexión permanente a Internet. Dado que se utiliza Internet para conectarse a sus aplicaciones y, por lo tanto, a sus datos y documentos, si no se dispone de conexión a Internet no se podrá acceder a ningún servicio alojado en la nube.
- No funciona bien con conexiones de baja velocidad. Del mismo modo, una conexión a Internet de baja velocidad hace que la computación en la nube sea en muchos casos imposible. Las aplicaciones web requieren una gran cantidad de ancho de banda para descargarse, al igual que documentos de gran tamaño.
- Algunas veces puede ser demasiado lento. Incluso con una conexión rápida, las aplicaciones basadas en web seguramente serán más lentas que aplicaciones similares instaladas en un ordenador de escritorio. Esto se basa en muchas variables de las que depende el procesamiento en la nube.
- Los datos almacenados pueden no estar seguros.
- Teóricamente siempre existirá la posibilidad de que los datos almacenados se puedan perder. La mayoría de las empresas que brindan servicios de computación en la nube toman las precauciones suficientes para que eso no ocurra.

2.1.3 IaaS vs PaaS

A la hora de desarrollar servicios en la nube es frecuente encontrarse en la tesitura de tener que decidir entre crear una plataforma propia, empleando IaaS, o decantarse por escoger una PaaS ya existente. Cada una de estas opciones tiene sus ventajas y sus inconvenientes. Por ejemplo, si se emplea IaaS, se tiene la oportunidad de depurar a nivel de SO con todas las posibilidades que esto ofrece pero, por otro lado, se hace necesario disponer de un nivel de conocimiento aún más detallado de las tecnologías implicadas. Para elaborar este TFG se decidió optar por tener el máximo control posible sobre el desarrollo del proyecto creando una infraestructura que pudiese ser muy versátil y adaptable ante cualquier necesidad futura, primando el deseo de experimentar hasta el más bajo nivel las posibilidades que la computación en la nube puede ofrecer.

Esos fueron los motivos determinantes por los que se escogió IaaS en lugar de PaaS porque, aunque sea una labor más ardua, elaborar una plataforma propia permite decidir libremente qué camino seguir sin estar sujeto a las restricciones impuestas por el proveedor. Esto es algo a tener en cuenta ya que migrar entre distintas plataformas no es trivial, puede llegar a ser algo muy problemático y, en el peor de los casos, habría que replantear y rediseñar el proyecto desde sus orígenes. Esta dependencia adquirida con la plataforma puede obstaculizar enormemente, e incluso llegar a paralizar, pequeños proyectos inciertos que necesitan de una gran flexibilidad a la hora de escoger nuevas herramientas de trabajo que pueden no estar disponibles en la plataforma elegida. Aunque también se pueden ver afectados grandes proyectos complejos. Por ejemplo, si la plataforma presenta algún fallo en algún módulo, carece de alguna biblioteca específica o no está actualizada a una versión determinada que permita los resultados deseados., etc. Un ejemplo de esto último es la lista de incidencias de la plataforma App Engine de Google. En la Figura 2. 4 se muestran 11 fallos de la misma reconocidos por Google y de prioridad crítica que impiden avanzar a los proyectos afectados.

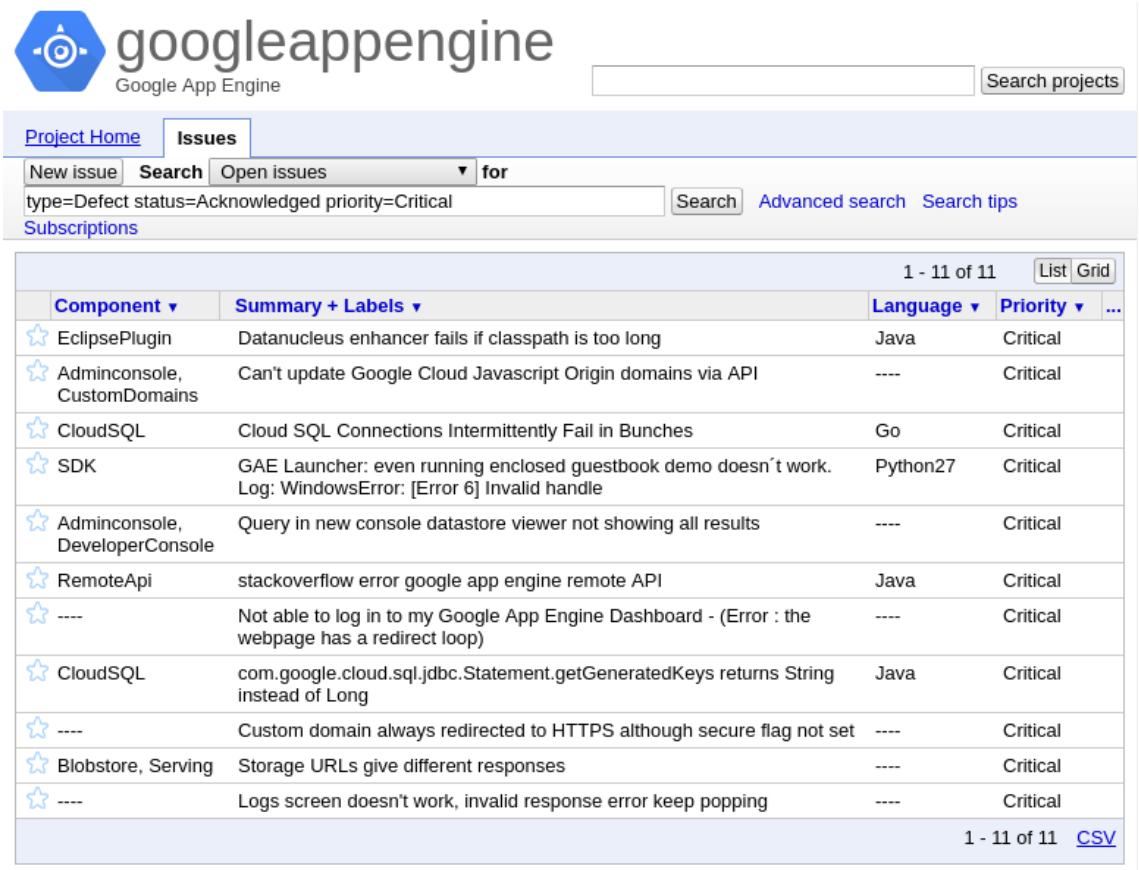


Figura 2. 4 Lista de fallos críticos sin resolver de la plataforma App Engine.

Teniendo esto presente, a continuación se enumeran algunos otros aspectos tanto positivos como negativos que a tener en cuenta para tomar la decisión sobre qué paradigma de computación utilizar [12]. Respecto a la plataforma como servicio tales como Heroku o App Engine los aspectos a valorar son los siguientes:

- Permite externalizar el mantenimiento y la gestión de la infraestructura.
- No se puede trabajar a nivel de Sistema Operativo para diagnosticar problemas si algo falla u optimizar la plataforma.
- Se deben usar las restricciones que imponga la plataforma. Por ejemplo, en el caso que sea necesario el uso de bases de datos, sólo se podrán usar aquellas a las que la plataforma brinde soporte.
- Se asume el riesgo de quedar anclado a una plataforma y no poder migrar.

Por otro lado, en el caso de la infraestructura como servicio, donde nos encontramos plataformas como AWS o Digital Ocean, tenemos que prestar atención a las siguientes consideraciones:

- Permite elegir la arquitectura que se desee dando una mayor flexibilidad.
- Permite migrar más fácilmente. Por ejemplo: Ubuntu es Ubuntu, da igual dónde esté alojado porque la diferencia será mínima.
- Hacerse cargo de gestionar la infraestructura requiere un mayor nivel de conocimientos y un mayor grado de decisiones que adoptar.

2.2 AWS

2.2.1 Introducción

A la hora de comenzar un proyecto de este tipo, resulta difícil decidir entre tanta oferta, corriendo el riesgo de acabar por escoger una mala opción, algo que a largo plazo puede llegar a tener un tremendo impacto negativo.

Amazon Web Services (AWS) es una compañía pionera en el sector y orientada principalmente a la IaaS. Provee todo un conjunto de herramientas que incluyen balanceadores de carga, servidores, sistemas operativos, almacenamiento, redes de distribución de contenido y todo lo que se pueda necesitar a nivel de infraestructura, enfatizando la escalabilidad, la seguridad y la sencillez de uso. En la Figura 2. 5 se observa el panel de control de esta infraestructura.

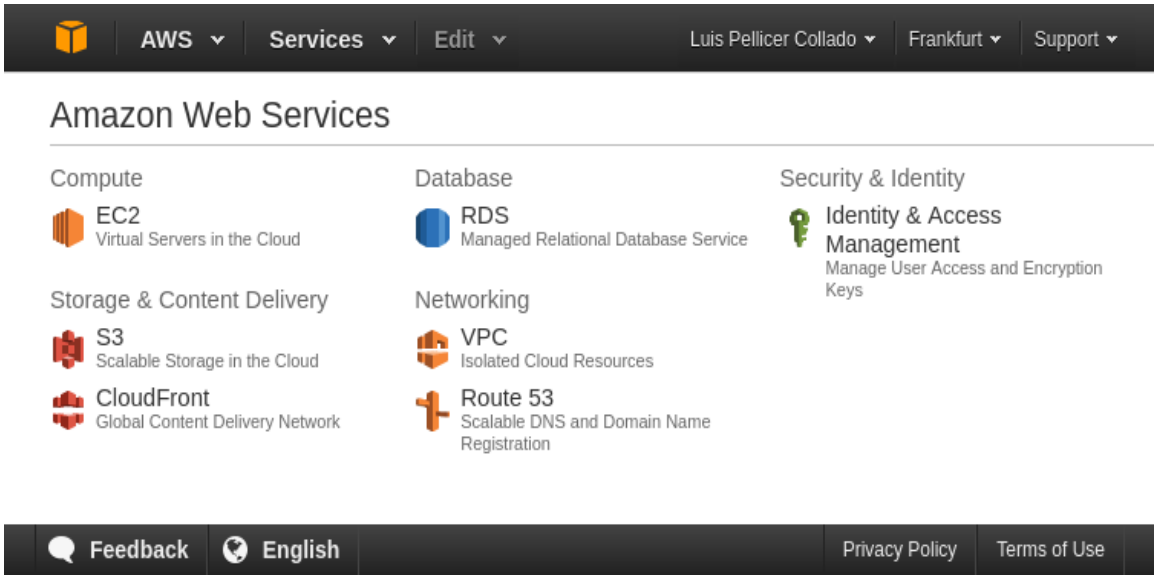


Figura 2. 5 Panel de control de AWS con los elementos utilizados para este TFG.

Sin embargo, existen diversas empresas con mucha experiencia como Dropbox o Netflix han utilizado sus servicios de infraestructura para desarrollar sus plataformas en ellos. Este hecho, unido a la existencia de una capa gratuita (“*free tier*”), cuyas posibilidades pueden ser observadas en la Figura 2. 6, permite usar gran cantidad de sus servicios de forma libre de cargos durante un año y una amplísima documentación muy bien estructurada, han sido los factores determinantes para optar por su elección en este trabajo. También, se barajó seriamente la opción de DigitalOcean, donde la documentación y el soporte son excelentes, pero en este caso el usuario carece del periodo de prueba gratuito y, por tanto, debe pagar desde el primer momento aunque siempre en función del coste de utilización.

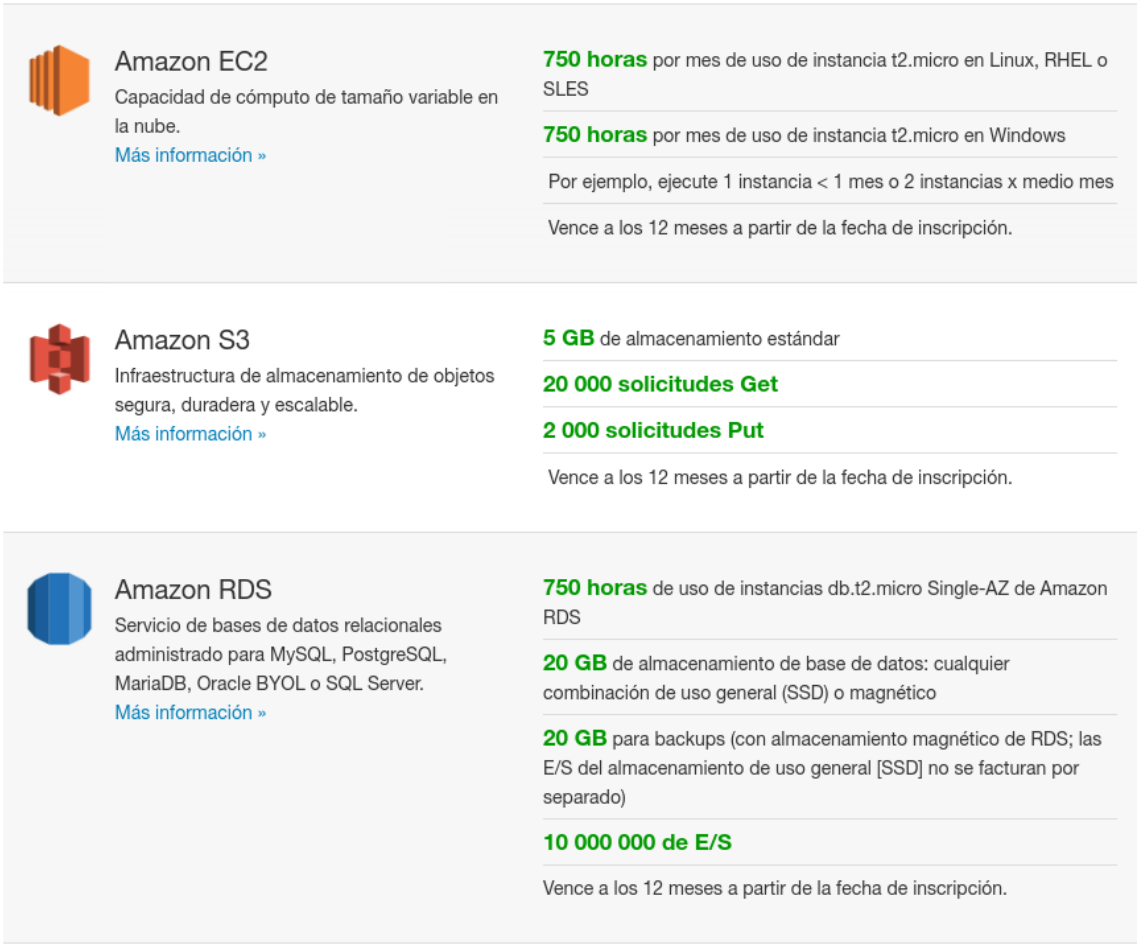


Figura 2. 6 Capa gratuita de AWS

AWS también dispone de una vía de aprendizaje llamada "*AWS Educate*", cuyo logo se muestra en la Figura 2. 7. Este servicio proporciona a alumnos y profesores los recursos necesarios para llevar a cabo su aprendizaje relacionado con la nube. No obstante, sólo con la existencia de la citada capa gratuita fue más que suficiente para poder realizar este trabajo sin tener que afrontar costes excesivos.

Figura 2. 7 Logo de AWS Educate.

2.2.1.1 Regiones y zonas de disponibilidad

Las granjas de servidores de AWS están presentes en varias regiones del globo como Estados Unidos, Europa, Asia-Pacífico y Sudamérica. Dentro de cada una de estas regiones AWS dispone de una o varias "Zonas de Disponibilidad" o "*Availability Zones*", donde se encuentran en última instancia los equipos físicamente. Las Zonas de Disponibilidad pertenecientes a una misma región están conectadas entre sí mediante enlaces de baja latencia.

Si el proyecto lo requiere, es perfectamente posible trasladar toda la infraestructura de una región a otra o incluso tener presencia simultánea en todas ellas a la vez, aunque esto excede las condiciones de uso de la capa gratuita e incrementa notablemente los costes. Para este trabajo se ha escogido la zona de Frankfurt dentro de la región de Europa, tal como se muestra en la Figura 2. 8, porque de entre todas las disponibles es la que menor latencia presenta desde Canarias, es aproximadamente de unos 75 ms de media frente a los 80 ms de Irlanda.

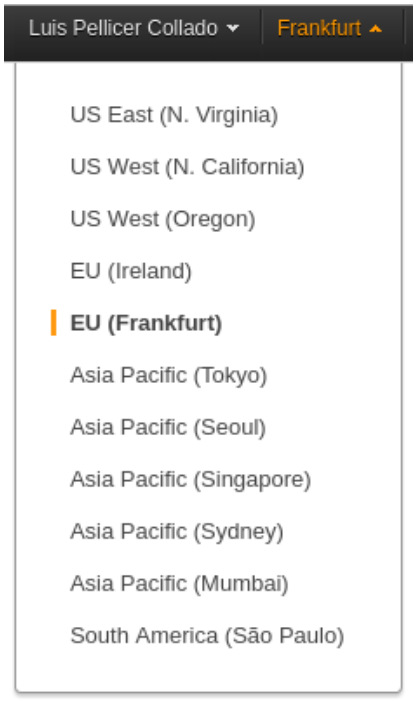


Figura 2. 8 Menú de la consola de AWS que permite seleccionar la Zona de Disponibilidad.

Existen diversas herramientas para poder medir de la forma más real posible el comportamiento de las granjas tales como http://www.cloudping.info/, que es una herramienta de HTTP ping que se ha usado para medir la latencia de las zonas de AWS o http://cloudharmony.com/, una herramienta más genérica pero más detallada para el análisis del rendimiento de la nube.

2.2.2 Diseño de la infraestructura

Haciendo uso de los servicios que ofrece AWS se ha diseñado una plataforma sobre la que montar Django de forma auto escalable. Para ello, ha sido necesario el uso de un balanceador de carga, un servidor dedicado de base de datos y múltiples máquinas virtuales (instancias) cuyo número se incrementa o disminuye en función de la carga computacional a la que estén sometidas. Esta infraestructura es compartida entre los biosensores SPR, que mediante una API REST envían sus resultados para ser almacenados, y el personal de laboratorio, que mediante una aplicación web pueden consultarlos en cualquier dispositivo.

La única diferencia entre ambos escenarios es la red de distribución de contenidos (CloudFront), que sólo interviene cuando sirve los archivos estáticos pertenecientes a la aplicación web. Tal como se muestra en la Figura 2. 9 y la Figura 2. 10.

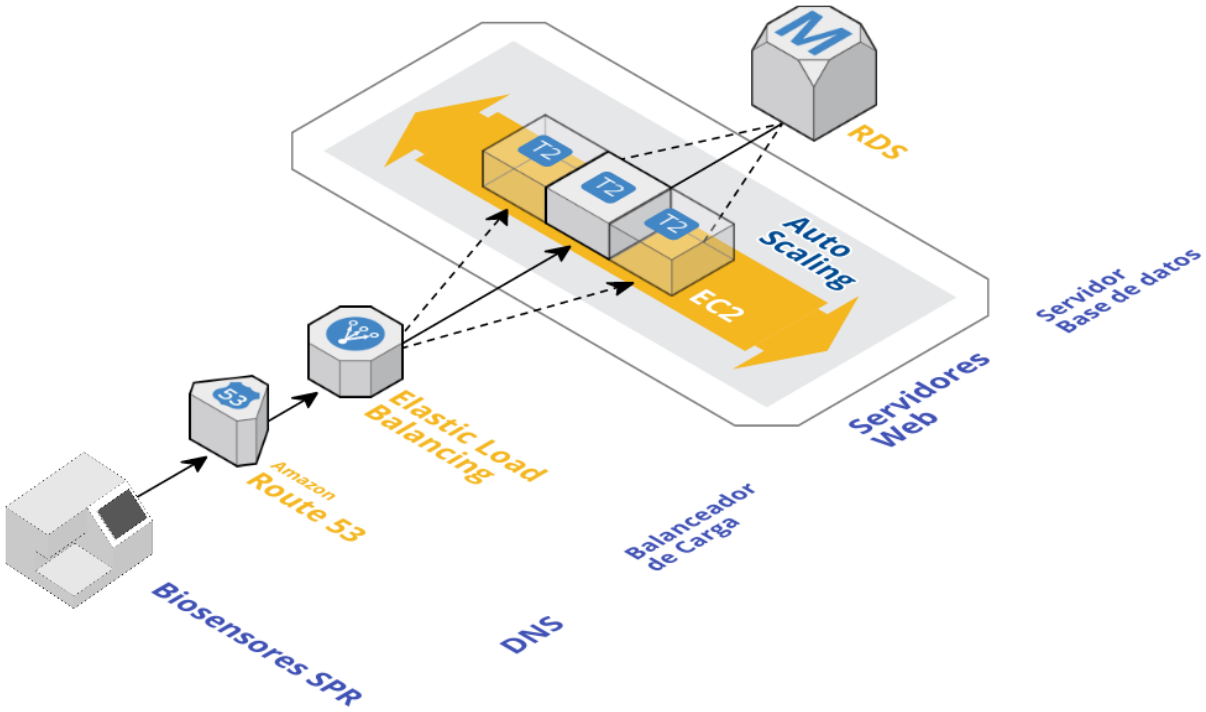


Figura 2. 9 Esquema de la plataforma que interviene en el uso de la API por parte de los biosensores SPR.

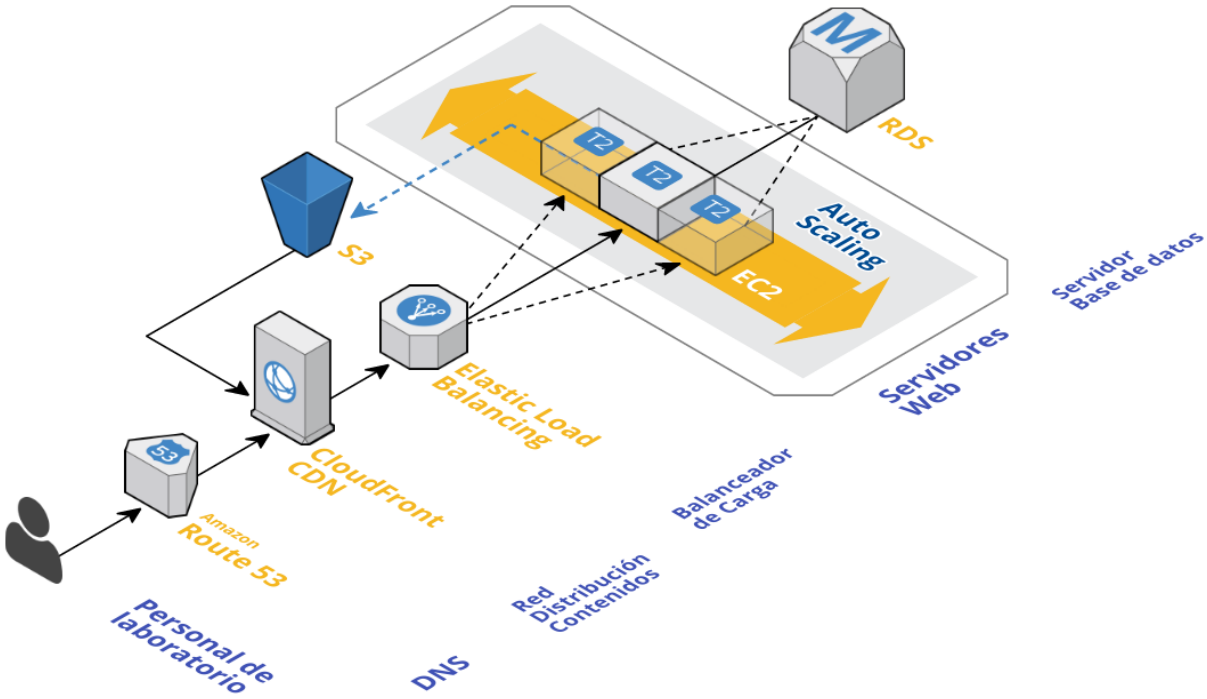
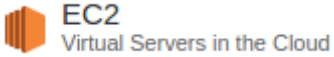


Figura 2. 10 Esquema de la plataforma que interviene en el uso de la aplicación web por parte del personal del laboratorio.

En las siguientes secciones de este capítulo se detallarán cada uno de los elementos que conforman la plataforma y qué uso se le ha dado a los mismos para poder desarrollar este TFG.

2.2.3 Elastic Compute Cloud (EC2)



Según la propia Amazon define, EC2 es: *"Un servicio web que proporciona capacidad de cómputo con tamaño modificable en la cloud. Está diseñado para facilitar a los desarrolladores el cloud computing escalable basado en web"* [13].

Es decir, EC2 es la plataforma de AWS que permite crear y gestionar servidores virtuales en función de las necesidades de cómputo existente. Amazon denomina a estos servidores virtuales "Instancias de Servidor". Hay de varios tipos y su elección dependerá en gran medida de la política que se adopte en cuanto a la reserva de recursos y el presupuesto de cada proyecto en concreto.

Para este proyecto se ha decidido optar por las instancias de tipo t2.micro (1 vCPU, 1GB RAM), que enfocadas a un desempeño a ráfagas, son las únicas disponibles dentro de la capa gratuita. Es importante remarcar que posteriormente, se pueden escalar de forma vertical (asignarles más recursos) muy fácilmente sin la necesidad de crear instancias nuevas, pero saliéndose de la capa gratuita.

Estas instancias están orientadas a un uso general y con un desempeño por ráfagas según la demanda de recursos a las que se vean sometidas.

Según Amazon: *"El desempeño de referencia y la capacidad de alcanzar ráfagas se rigen por los créditos de la CPU. Cada instancia T2 recibe créditos de CPU continuamente a un nivel establecido dependiendo del tamaño de la instancia. Las instancias T2 acumulan créditos de la CPU cuando están inactivas y los utilizan cuando*

están activas. Las instancias T2 son una buena opción para cargas de trabajo que no usan la CPU por completo, a menudo o de manera constante, pero que de vez en cuando tienen que alcanzar ráfagas (por ejemplo, servidores web, entornos para desarrolladores y pequeñas bases de datos)" [14].

Dentro de la capa gratuita hay disponible 750 horas (31 días) por mes de uso de instancias t2.micro. También se puede disponer de dos instancias t2.micro funcionando a la vez pero de este modo se consumirá el tiempo disponible el doble de rápido, o tres instancias concurrentes consumiendo el tiempo disponible el triple de rápido y así sucesivamente [15].

El precio actual por horas de este tipo de instancias en Frankfurt es de 0,015$. Si se multiplica por 750 horas se puede estimar un gasto mensual de 11,25$ [16].

2.2.3.1 Instancias

Después del paso previo de la debida documentación, la primera tarea que se realizó en este proyecto fue el lanzamiento de la instancia que daría soporte a los requerimientos de cómputo seleccionando la instancia indicada en la Figura 2. 11. AWS permite subir a su nube (importar) máquinas virtuales creadas con anterioridad en entornos particulares ajenos a su infraestructura.

Figura 2. 11 Instancia seleccionada.

Del mismo modo, Amazon también permite exportar las máquinas virtuales que se encuentren en su nube [17] y poder disponer de ellas de forma independiente. Estas

tareas deben ser realizadas mediante la API que AWS dispone para automatizar la gestión de sus sistemas pero, en este proyecto, no fue requerido el uso de estas características, a pesar de su enorme utilidad, porque desde el comienzo la instancia fue configurada y lanzada directamente desde los servidores de Amazon EC2 a través del panel web.

EC2 también dispone de un completo catálogo de máquinas virtuales preconfiguradas, que denomina Amazon Machine Images (AMI), con diferentes distribuciones de Linux y varias versiones de Windows. Cabe destacar el hecho de que en EC2 existe todo un mercado de AMIs donde diversas empresas ofrecen de forma gratuita o de pago multitud de distribuciones con todo tipo de servicios integrados. La opción elegida finalmente fue la AMI de Ubuntu Server 14.04 LTS Hardware Virtual Machine tal como se muestra en la siguiente figura.

Más adelante, en la sección dedicada al Sistema Operativo, se nombrará una de estas AMIs preconfiguradas perteneciente a la empresa Bitnami y se explica la creación de una AMI propia como parte necesaria del proceso para realizar el escalado horizontal.

Una vez lanzada la instancia se puede consultar su estado y de cuantas se hayan creado tal y como se muestra en la Figura 2. 12. Además, se puede seguir lanzando instancias desde el panel de EC2.

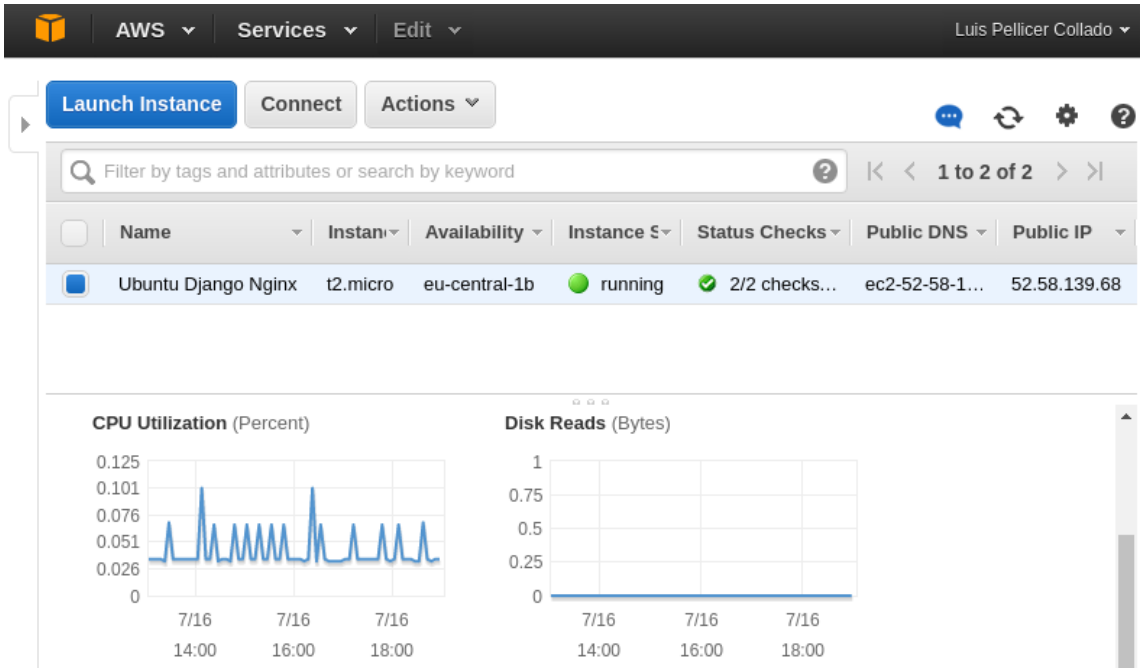


Figura 2. 12 Instancia operativa en los servidores de EC2.

Sobre el estado de una instancia se pueden efectuar cuatro acciones:

- Start, que la enciende si está apagada.
- Stop, que apaga la máquina virtual y pierde la IP pública.
- Reboot, que la reinicia sin perder la IP.
- Terminate, que la apaga y borra del sistema.

Para poder realizar ciertas tareas, como actualizar el tipo de instancia a una superior, es necesario que la instancia esté apagada.

2.2.3.2 Security Group

Los grupos de seguridad actúan como un firewall virtual que controla el tráfico de una o más instancias. Cuando se lanza una instancia esta puede ir asociada a uno o varios grupos de seguridad donde previamente se han definido las reglas de tráfico que se consideren oportunas. Dichas reglas pueden ser modificadas en cualquier momento y

son de aplicación inmediata a todas las instancias asociadas al grupo alterado. Este servicio añade una capa de seguridad extra al propio "*IP Tables*" de Ubuntu que puede resultar muy útil para entornos de producción.

Para las instancias de EC2 se creó un grupo para el tráfico de llegada con todos los puertos TCP cerrados salvo el 22 para poder administrar el servidor, y el 80, para poder acceder al servidor web. Así mismo, se cerraron todos los puertos de salida salvo el 3306 para que la instancia pudiera acceder a la base de datos del servidor dedicado a tal fin, del que se hablará más adelante.

2.2.3.3 Elastic IPs

Cuando una instancia se apaga se pierde su IP pública. Elastic IP Address [18] es un servicio ofrecido por AWS que nos permite asociar una dirección IP pública a nuestra cuenta de AWS de modo que, si la instancia termina, no perderemos la IP. Este servicio es de una gran utilidad, ya que se puede enmascarar rápidamente un fallo en una instancia remapeando la IP a otra instancia que esté operativa en nuestra cuenta. Pero tiene el inconveniente de que AWS, para prevenir un abuso o garantizar un uso eficiente de Elastic IP Address, impone un pequeño precio por hora a aquellas IPs que no estén asociadas a una instancia en funcionamiento o que estén asociadas a tarjetas de red no operativas. En caso de que la instancia esté en funcionamiento y haciendo uso de la IP no existirá cargo alguno.

Con Elastic IP address se puede asignar una IP pública a una instancia determinada, a una interfaz virtual, a una VPC, a un balanceador de carga, etc. Aunque AWS recomienda usar ambos en conjunción, este servicio no es necesario si se dispone de un dominio en Route 53.

2.2.3.4 Elastic Load Balancing

Elastic Load Balancing (ELB) es el servicio de balanceo de carga que EC2 proporciona si se desea repartir el tráfico entre varias instancias funcionando en paralelo. De este modo, el tráfico dirigido al servidor no va directamente contra una sola instancia, sino que ELB hace de intermediario repartiéndolo entre cuantas haya disponibles.

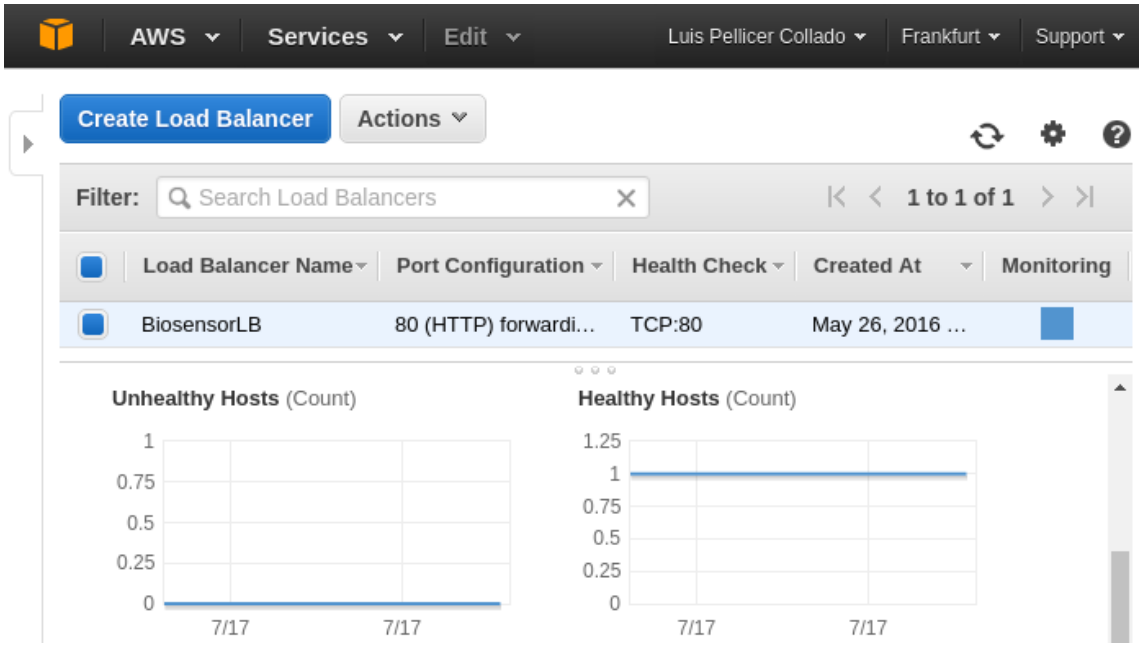


Figura 2. 13 Balanceador de carga operativo en los servidores de EC2.

Aunque para realizar el balanceo de carga también se barajó la opción de dedicar una instancia específica con Nginx configurado para tal fin y emplear la API de AWS para las métricas del autoescalado, esta solución requería de un mínimo de dos instancias para ser funcional, con el consiguiente decremento en las horas de instancias gratuitas. Pero dado que AWS proporciona ya una solución integrada y dentro de su capa gratuita, se optó por usarla, tal y como se muestra Figura 2. 13. Dicha capa incluye 750 horas de Elastic Load Balancing cada mes durante un año y 15 GB de procesamiento de datos. Fuera de ella se paga por cada hora u hora parcial en la que Elastic Load Balancer esté

activo y por cada GB de datos transferido a través de ELB. Los precios para la región de Frankfurt son de 0,030$ por hora (u hora parcial) y 0,008$ por GB de datos procesados [19].

El balanceador de carga puede comprobar la salud de nuestras instancias haciendo comprobaciones regulares sobre ellas para así poder repartir el tráfico sólo a quienes estén activas y con buena salud. Esta configuración se muestra en la Figura 2. 14.

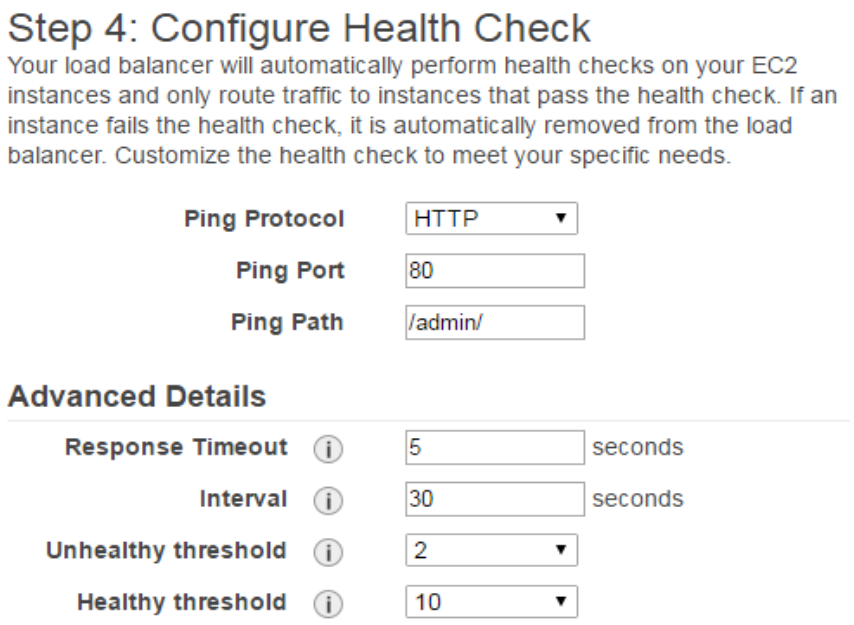


Figura 2. 14 Configuración de la comprobación del estado de las instancias.

Una vez creado el balanceador, si se dispone de varias instancias operativas, irá repartiendo el tráfico entre las que se le hayan asignado mientras va haciendo comprobaciones para saber a qué instancias debe, o no, tener en cuenta para realizar esta labor. Pero, sin lugar a dudas, el verdadero potencial del balanceador de carga es utilizarlo en conjunción con el servicio de autoescalado, que se explica en la siguiente sección.

2.2.3.5 Auto scaling

La cuestión de la escalabilidad de recursos se puede abordar por medio de dos aproximaciones. Mediante el escalado vertical o mediante el escalado horizontal. Ante una demanda creciente en la capacidad de cómputo, el escalado vertical consiste en asignar más recursos a una instancia incrementando el número de vCores, RAM, SSD, etc. EC2 puede lanzar la instancia t2.micro creada para este proyecto como una d2.8xlarge otorgándole 48 TB de almacenamiento o lanzarla de tipo x1.32xlarge y disponer de 1952 GiB de memoria RAM DDR4, o del tipo m4.10xlarge que posee 40 núcleos del procesador Intel Xeon E5-2676 v3.

Si toda la plataforma (servidor web, bases de datos, etc.) está integrada dentro de una sola instancia, esta solución es sin duda la menos compleja de llevar a cabo, ya que bastan unos pocos clicks para transformar una instancia en un superordenador y salirse por completo de la capa gratuita.

Sin embargo, este tipo de escalado tiene un límite muy definido dado que, si en cualquier momento se pretende ir un paso allá, simplemente no será posible. Además, a medida que el techo tecnológico se aproxima, el incremento de los costes no es lineal sino exponencial.

El escalado horizontal, en cambio, consiste en generar más instancias, clones unas de otras tal y como se en la Figura 2. 15, y repartir entre ellas la carga de trabajo. Esto implica que el incremento de costes se mantiene lineal, pero en contrapartida se deben tener en cuenta unas series de consideraciones importantes en el diseño de la infraestructura. Por ejemplo, si la base de datos está ubicada dentro de la propia instancia, al duplicar la instancia también se duplica la base de datos y, por tanto, se rompe la consistencia de la misma. Esto fuerza a desligar el almacenamiento de los datos del procesamiento de los mismos.

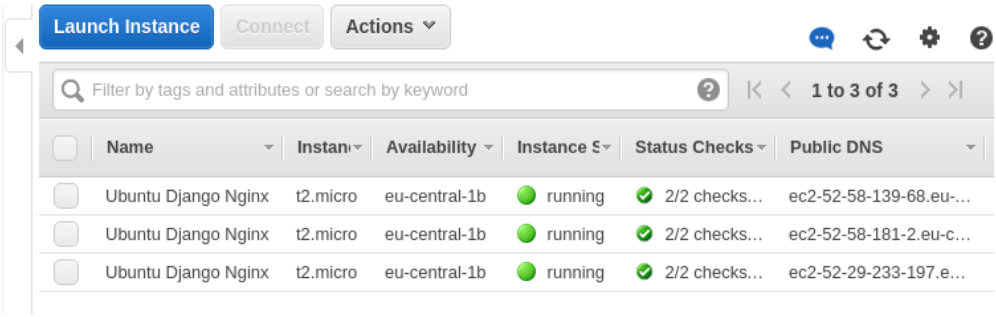


Figura 2. 15 Tres instancias ejecutándose en paralelo.

AWS ofrece soluciones muy interesantes a estos problemas permitiendo un diseño modular donde se puede separar la capacidad de cómputo de la capacidad de almacenamiento y de la base de datos.

Una vez plateado el diseño modular, el primer paso necesario para efectuar el escalado horizontal de las instancias en EC2, es crear una imagen (AMI) de arranque a partir de una de ellas. Pero antes ha de asegurarse que la instancia no requiere de ninguna intervención humana para estar completamente operativa después del arranque tras un lanzamiento. Es decir, que al pasar de estar apagada, a encendida, todos sus servicios internos han de estar configurados correctamente para levantarse por sí solos y estar operativos sin intervención externa. Teniendo esto presente se pudo proceder a la creación de la AMI seleccionando la instancia y pulsando sobre el menú que aparece en la Figura 2. 16.

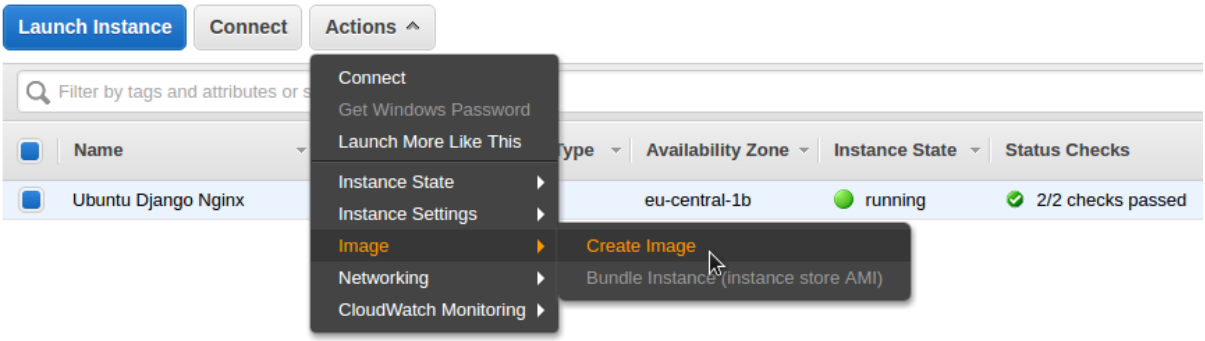


Figura 2. 16 Creando una imagen (AMI) propia a partir de una instancia

El paso siguiente, una vez creada la AMI que sirve de base para el escalado horizontal, fue generar una configuración de lanzamiento o "*launch configuration*", es decir, una plantilla que indica a EC2 cómo debe ser la configuración de las AMIs que va a lanzar.

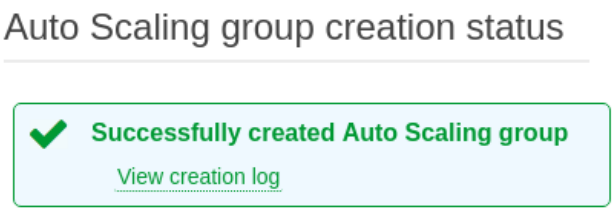


Figura 2. 17 Grupo de escalado creado satisfactoriamente.

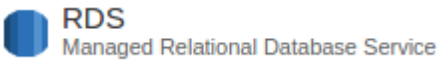
Posteriormente, se construyó un grupo de autoescalado, o Auto scaling Group, con una instancia mínima de inicio y un máximo limitado a 3. En las opciones avanzadas se indicó que el grupo recibiría el tráfico desde el balanceador de carga y que utilizaría sus métricas para determinar el estado del grupo tal y como se muestra en la Figura 2. 18.

Figura 2. 18 Fragmento de las opciones avanzadas de autoscaling.

A continuación, se crearon una serie de alertas para incrementar y disminuir el grupo en función de la carga media de los procesadores en un periodo de 5 minutos. Configurar las políticas de autoescalado es una labor compleja que depende mucho de cada caso en particular. Lograr un ajuste fino de las mismas puede llegar a requerir años

de pruebas a base de ensayo y error ya que, dependiendo del tipo de servicio ofrecido, el tráfico puede llegar a ser realmente difícil de modelar. Por otro lado, dado que en EC2 se paga por el uso, existe el aliciente de que conseguir tener las instancias a una carga óptima puede ayudar a reducir enormemente los costes.

2.2.4 Relational Database Service (RDS)



Amazon Relational Database Service o Amazon RDS [20] es un servicio que ofrece AWS para crear, operar y escalar bases de datos relacionales en la nube.

Como se comentó en capítulos anteriores, tener un servidor dedicado de base de datos posibilita un diseño modular donde poder escalar horizontalmente la capacidad de cómputo. En caso de que la carga de la base de datos se vea muy afectada, siempre se podrá optar por un escalado vertical de la misma sin que esto repercuta en la consistencia del sistema. Este tándem de escalado horizontal de EC2 y vertical de RDS ha sido el diseño de base escogido para este proyecto.

RDS permite elegir entre varios motores de bases de datos relacionales tales como Oracle, Microsoft SQL Server, PostgreSQL, MySQL y MariaDB [21], tratándose este último del elegido. A la hora de optar por esta solución se han tenido en cuenta los siguientes factores [22]:

- Se posee experiencia previa en este sistema de gestión de bases de datos.
- Se trata de un fork de MySQL con licencia GPL creado por su propio fundador tras la compra de Sun Microsystems por parte de Oracle.
- Tiene una alta compatibilidad con MySQL ya que posee las mismas órdenes, interfaces, APIs y bibliotecas, siendo su objetivo poder cambiar un servidor por otro directamente.

- Django permite abstraerse del motor de bases de datos seleccionado, con lo que una hipotética migración a otro motor no sería, en principio, excesivamente compleja.

La capa gratuita de AWS para Amazon RDS permite el uso de instancias micro Single-AZ. La capa de uso gratuita cuenta con 750 horas de instancias al mes y 20 GB de almacenamiento de base de datos, 10 millones de E/S y 20 GB de almacenamiento de backup.

A la hora de lanzar el servidor de bases de datos hubieron diversos problemas, entre ellos desconocer el hecho de que la red Wi-Fi de la ULPGC filtra el tráfico del puerto 3306 y que el grupo de seguridad al que pertenecía la instancia RDS no aceptaba a priori conexiones entrantes desde fuera de la VPC.

Para verificar la funcionalidad de la base (ver Figura 2. 19) de datos se instaló la herramienta multiplataforma MySQL Workbench con la que se pudo cerciorar la operatividad de la misma para que Django pudiese crear sus tablas.

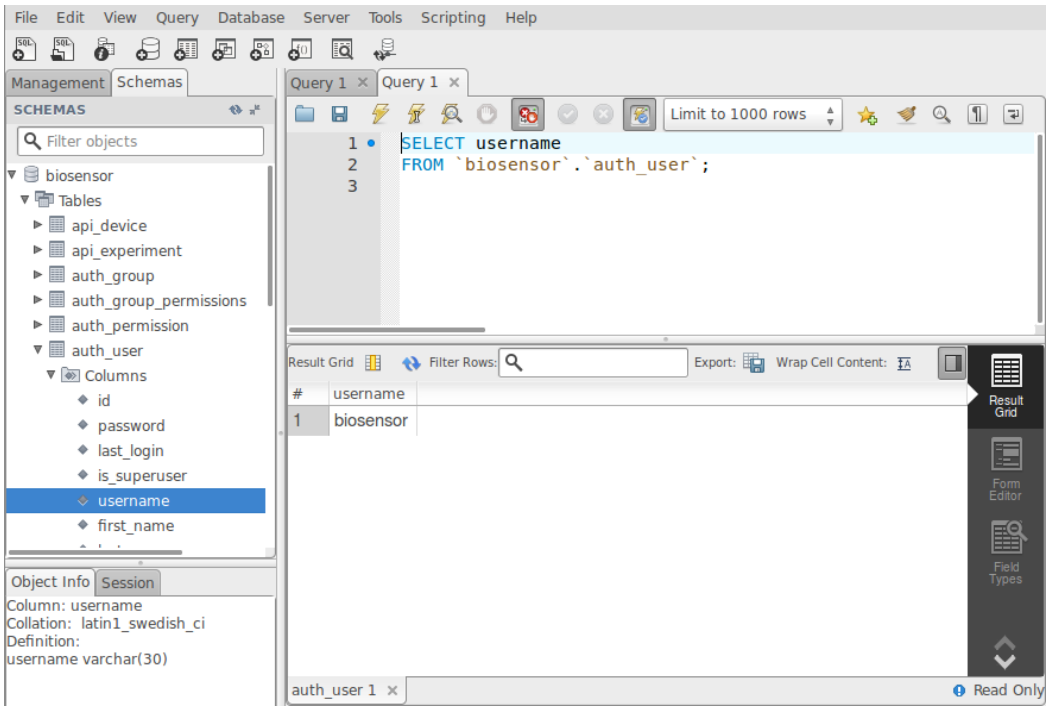


Figura 2. 19 Comprobación del correcto funcionamiento de la base de datos.

2.2.5 Simple Storage Service (S3)

Amazon S3 [23] provee un servicio para almacenar datos de forma escalable en la nube. Esto permite, siguiendo con el diseño modular, separar la capacidad de almacenamiento. De modo que, en caso de darse la necesidad de que el personal de laboratorio quiera almacenar contenido en el servidor web, éste no se viese duplicado cuando entrase en juego el escalado horizontal. De esta manera, se logra separar de la capacidad de cómputo la capacidad de almacenamiento de archivos, tal como se realizó con el servicio de base de datos.

Del mismo modo que el elemento principal de EC2 son las instancias, el de S3 son las cubetas o "*buckets*", que no son más que directorios donde guardar los ficheros y que tienen una serie de políticas asociadas. La manera de crear un "*bucket*" se muestra en la Figura 2. 20.

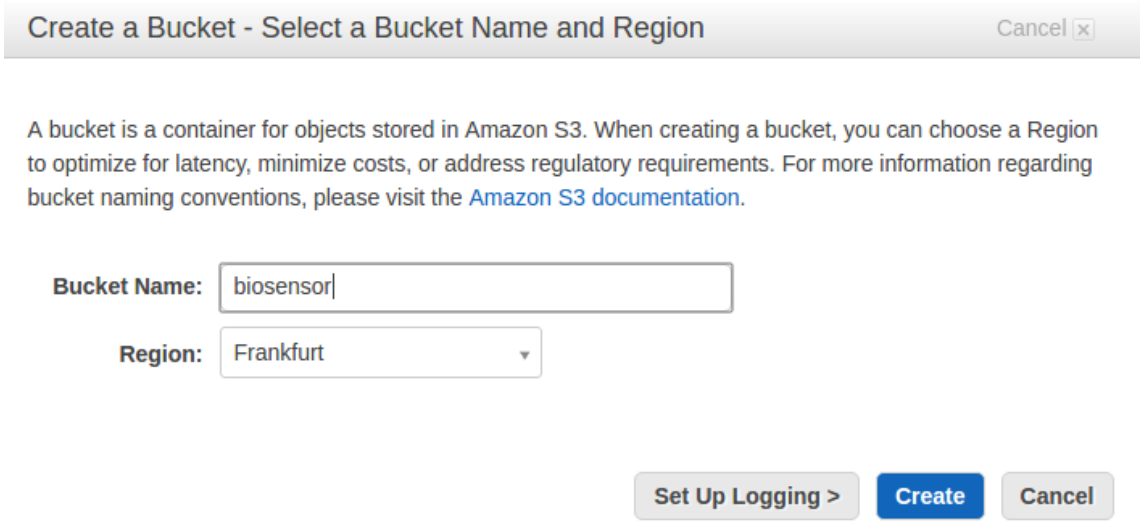


Figura 2. 20 Creación de un "*bucket*".

A efectos de este TFG este servicio sólo será usado para almacenar los ficheros estáticos de la aplicación web, tales como imágenes, ficheros JavaScript y hojas de estilo CSS. Estos archivos, serán difundidos por todo el globo a través de la red de distribución de contenidos que AWS ofrece.

Cabe citar que hubo un error al realizar esta tarea debido a un fallo en la granja de Frankfurt. Dicho error se subsanó replicando el Bucket en la de Irlanda. Dado que la función principal del Bucket creado es la de dar soporte a los archivos estáticos para que estos sean posteriormente por el CDN de Amazon, tal como se explica más adelante, esta eventualidad no supone perjuicio alguno en el rendimiento general de la infraestructura.

Dentro de la capa gratuita Amazon S3 ofrece 5 GB de almacenamiento, 20.000 solicitudes GET, 2.000 solicitudes PUT y 15 GB de transferencia de datos saliente al mes.

2.2.6 CloudFront

AWS dispone de su propia red de distribución de contenidos (CDN) [24] que pone a disposición de sus usuarios mediante el servicio llamado CloudFront. En este proyecto se ha hecho uso de él (como se indica en la Figura 2. 21) para que replique el contenido alojado en el Bucket de S3 en cada uno de los extremos de su red. Con esto se logra reducir drásticamente la latencia de la WebApp dado que los ficheros estáticos contenidos en el Bucket pasan a estar disponibles mucho más cerca de los clientes finales.

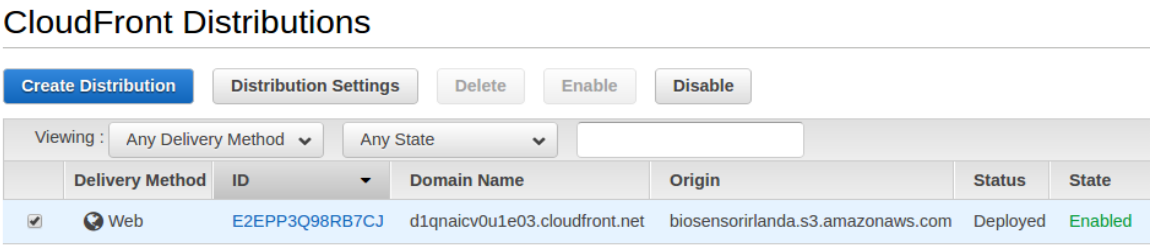


Figura 2. 21 Distribución generada de CloudFront

Las ubicaciones de borde de la red global de Amazon CloudFront se sitúan en los lugares recogidos en la Tabla 2. 1:

Estados Unidos	Europa	Asia
Atlanta, GA Ashburn, VA (3) Chicago, IL Dallas/Fort Worth, TX (2) Hayward, CA Jacksonville, FL Los Ángeles, CA (2) Miami, FL Nueva York, NY (3) Newark, NJ Palo Alto, CA San José, California Seattle, WA South Bend, IN St. Louis, MO	Ámsterdam, Países Bajos (2) Dublín, Irlanda Fráncfort, Alemania (3) Londres, Inglaterra (3) Madrid, España Marsella, Francia Milán, Italia París, Francia (2) Estocolmo, Suecia Varsovia, Polonia	Chennai, India Hong Kong (2) Mumbai, India Manila, Filipinas Osaka, Japón Seúl, Corea del Sur (3) Singapur (2) Taipei, Taiwán Tokio, Japón (2)
	Australia	América del Sur
	Melbourne, Australia Sídney, Australia	São Paulo, Brasil Río de Janeiro, Brasil

Tabla 2. 1 Ubicaciones de borde de la red global de Amazon CloudFront.

Amazon CloudFront oferta de forma gratuita 50 GB de transferencia de datos salientes y 2 000 000 de solicitudes HTTP y HTTPS al mes durante un año [25].

2.2.7 Identity and Access Management (IAM)

Identity and Access Management (IAM) [26] permite controlar de forma segura el acceso de los usuarios a servicios y recursos de AWS. Con IAM se puede crear y administrar usuarios y grupos de AWS, así como utilizar permisos para permitir o denegar el acceso de estos a los recursos de AWS. Dichos recursos poseen un identificador único llamado Amazon Resource Name (ARN).

Para poder integrar S3 con Django, tal como se muestra en el capítulo 3.1.3, primero fue necesario crear un usuario mediante IAM (Figura 2. 22) y permitir así que Django haga uso de él. Para ello se hizo al nuevo usuario biosensor_iam propietario del bucket desde las políticas del bucket utilizando el ARN [27] del usuario recién creado.

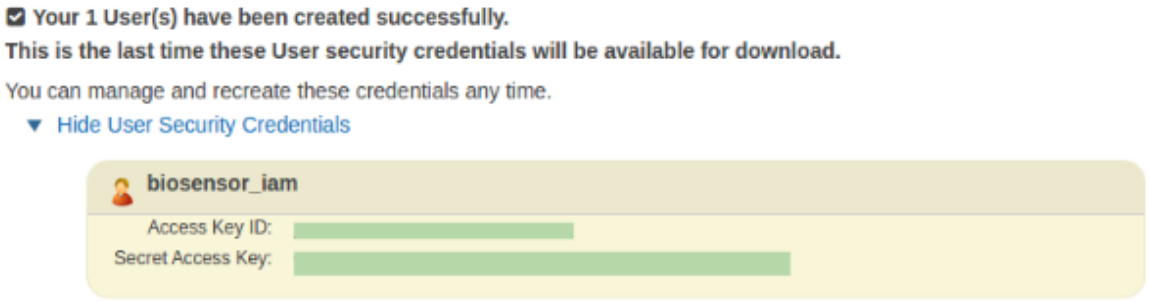
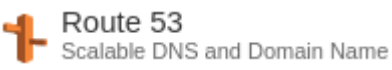


Figura 2. 22 Creación de IAM

2.2.8 Route 53 (Dominio y DNS)



Amazon Route 53 [28] es un servicio web DNS (Sistema de nombres de dominio) escalable y de alta disponibilidad en la nube que conecta de forma efectiva las solicitudes

del usuario con la infraestructura en ejecución como instancias de Amazon EC2, balanceadores de carga de Elastic Load Balancing o buckets de Amazon S3. Este servicio también puede utilizarse para redireccionar usuarios a infraestructuras externas a AWS. Además, Route 53 ofrece el registro de nombre de dominio en los que establece de forma automática la configuración DNS.

Para el correcto desarrollo de la plataforma se decidió comprar el dominio "biosensor.io". Este es el único de los servicios de AWS usados en el presente trabajo que no se encuentra disponible dentro de la capa gratuita. Para supervisar la propagación del dominio se usó la herramienta https://www.whatsmydns.net/.

Se da la situación de que el balanceador de carga no posee una IP estática aun siendo la puerta de acceso a la infraestructura desarrollada. Sin embargo, los registros DNS de tipo A sólo admiten apuntar a direcciones IP y no a otros nombres DNS como el que posee el balanceador de carga. De hacerlo así ya no sería un registro de tipo A sino un CNAME.

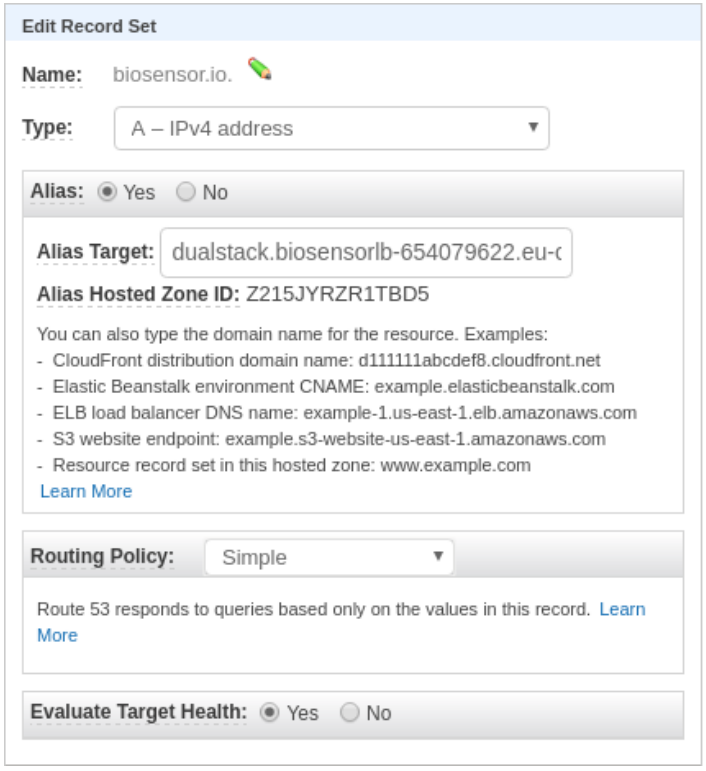


Figura 2. 23 Configuración de Alias en Route 53.

Para solucionar este problema Route 53 dispone de una herramienta "Alias" que permite crear un registro de tipo A usando el nombre DNS de cualquier recurso dentro de su infraestructura tal y como se muestra en la Figura 2. 23.

2.3 Conjunto de soluciones

2.3.1 Introducción

Un conjunto de soluciones (del inglés "solution stack"), es un juego de tecnologías informáticas que se combinan para dar una solución funcional y robusta a un problema dado. Ejemplos de estos "Stacks" son LAMP (Linux, Apache, MySQL, PHP) o MEAN (MongoDb, ExpressJS, AngularJS, NodeJS). En este TFG se ha hecho uso del conjunto LNMP (**L**inux, **N**ginx, **M**ariaDB, **P**ython). En este capítulo se describirá brevemente la implementación de cada uno de ellos.

2.3.2 Diseño del conjunto sobre EC2

El diseño del conjunto de soluciones que se ha integrado sobre la plataforma de Amazon EC2 es el que aparece en la Figura 2. 24. MariaDB no se ha tenido en cuenta ya que la base de datos posee un servidor dedicado mediante Amazon RDS. Como distribución de GNU/Linux se ha empleado Ubuntu, para el servidor web se ha elegido Nginx, y como lenguaje de desarrollo se ha optado por Python mediante el framework Django. Integrar desde cero todos estos componentes fue una tarea ardua y compleja. En las siguientes secciones se comenta con detalle cada uno de estos elementos y las configuraciones necesarias para poderlos integrar.

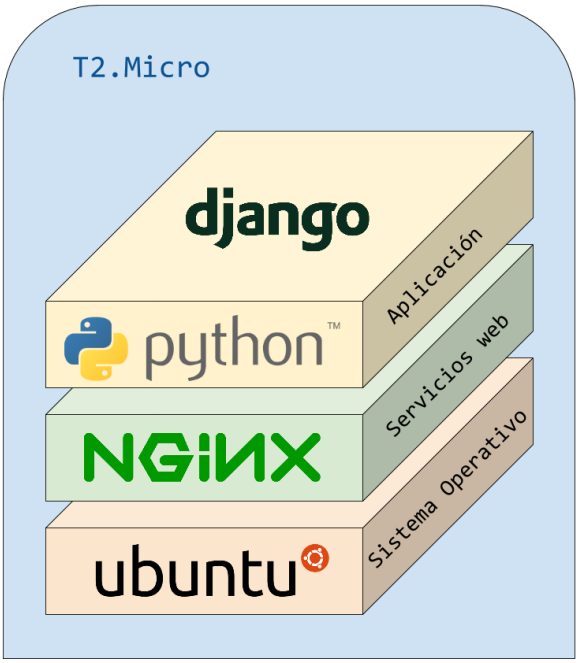


Figura 2. 24 Diseño del conjunto sobre EC2.

2.3.3 Ubuntu

AWS posee un extenso mercado de AMIs donde poder encontrar casi cualquier imagen, ya preconfigurada, con las pilas tecnológicas existentes más demandadas. LAMP, MEAN, etc. También están presentes multitud de distribuciones Linux incluyendo algunas orientadas específicamente a servir de contenedores y realizar despliegues horizontales como CoreOS.

Se dio el caso de que la empresa Bitnami ya ofrecía de forma gratuita una AMI con Ubuntu preinstalado, Apache como servidor Web y MySQL como base de datos pero se decidió crear una instalación desde cero propia porque la ofertada por Bitnami no soportaba el escalamiento horizontal [29].

Finalmente se decidió por escoger una distribución original de Ubuntu 14.04.4 LTS y construir sobre ella todo lo necesario para dar soporte a Django, el entorno de desarrollo para este proyecto.

2.3.3.1 SSH y Fichero de claves

Después de lanzar la instancia de Ubuntu, Amazon proporciona una dirección DNS y un fichero de claves, tal como se muestra en la Figura 2. 25. Este fichero es borrado de AWS una vez generado, por tanto es de vital importancia almacenarlo a buen recaudo o no se podrá acceder a la instancia. Para poder conectarse al Ubuntu de la instancia se debe usar un cliente SSH que dependerá de la plataforma desde la que se quiera acceder [30].

Amazon también proporciona un cliente web multiplataforma basado en Java en caso de no disponer de uno propio. Dado que la estación de trabajo utilizada para este proyecto contó con una distribución GNU/Linux se optó por usar OpenSSH [31].

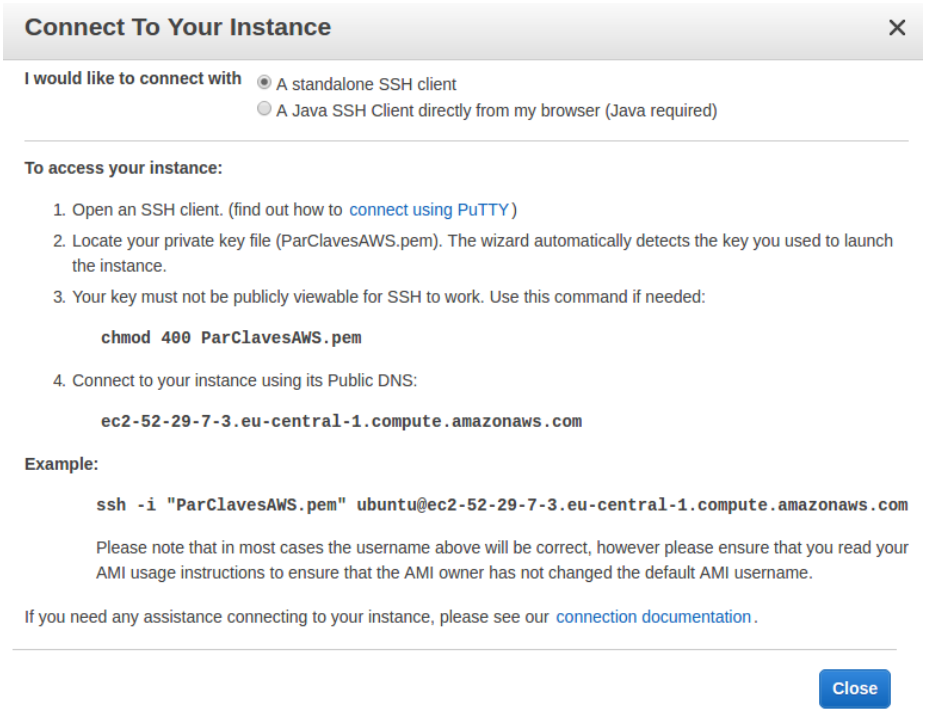


Figura 2. 25 Panel de datos de acceso a la instancia de EC2.

Un dato a tener en cuenta si se desea utilizar este cliente es el de otorgar los permisos adecuados al fichero de claves descargado o de lo contrario OpenSSH mostrará el error presentado en el Código 2. 1.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0664 for 'ParClavesAWS.pem' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
bad permissions: ignore key: ParClavesAWS.pem
Permission denied (publickey).
```

Código 2. 1 Error OpenSSH (permisos del fichero de claves incorrectos).

Para que esto no suceda se deben otorgar al fichero de claves permisos de sólo lectura por el propietario mediante la orden mostrada en el Código 2. 2.

```
chmod 400 ParClavesAWS.pem
```

Código 2. 2 Cambio de permisos al fichero de claves.

Una vez realizadas estas acciones se pudo acceder a la instancia a través de su servidor SSH y el fichero de claves proporcionado por Amazon. Para ello, bastó con copiar y pegar el comando de ejemplo que aparece en la Figura 2. 25, donde viene reflejado el usuario y la DNS que AWS ha asignado a la instancia de Ubuntu.

```
ssh -i "ParClavesAWS.pem" ubuntu@ec2-52-29-7-3.eu-central-1.compute.amazonaws.com
```

Código 2. 3 Comando SSH para conectarse al servidor.

Posteriormente, se ha ejecutado la orden del Código 2. 3, se habrá accedido con éxito al servidor remoto, debiendo aparecer en pantalla el MOTD personalizado por Amazon y la consola estará lista para poner a punto el sistema tal como se muestra en el Código 2. 4.

```
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-74-generic x86_64)
 * Documentation:  https://help.ubuntu.com/
  System information as of Thu Apr 28 21:31:28 UTC 2016
  System load: 0.08              Memory usage: 5%   Processes:       82
  Usage of /:  9.9% of 7.74GB   Swap usage:   0%   Users logged in: 0
  Graph this data and manage this system at:
    https://landscape.canonical.com/
  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud
0 packages can be updated.
0 updates are security updates.


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.


Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.


ubuntu@ip-172-31-17-201:~$
```

Código 2. 4 Mensaje de bienvenida de Ubuntu Server.

A pesar de que Django tiene un buen historial de seguridad [32], las aplicaciones web pueden llegar a verse comprometidas. Si la aplicación tiene un acceso limitado a los recursos del servidor, el daño potencial que se pueda producir también será limitado. Por

ello, la aplicación web debería funcionar como un usuario con privilegios limitados y en un entorno lo más independiente posible del resto del sistema. Para lograrlo se creó un usuario llamado `biosensor` que se añadió al grupo `www-data`.

2.3.4 Nginx

Nginx es el servidor web que se decidió usar para este TFG por ser una excelente alternativa a Apache [33]. Fue desarrollado en 2002 por Igor Sysoev para dar respuesta al problema C10K [34] (un problema de optimización de conexiones de red para gestionar un gran número de clientes al mismo tiempo) y, desde su lanzamiento en 2004, su popularidad no ha dejado de crecer, especialmente en los últimos años.

Nginx es el segundo servidor web más utilizado entre el top un millón de páginas web más concurridas de Internet [35]. Con un grado de utilización del 27.61% frente al 43.73% de Apache. Tal como se muestra en la Figura 2. 26.

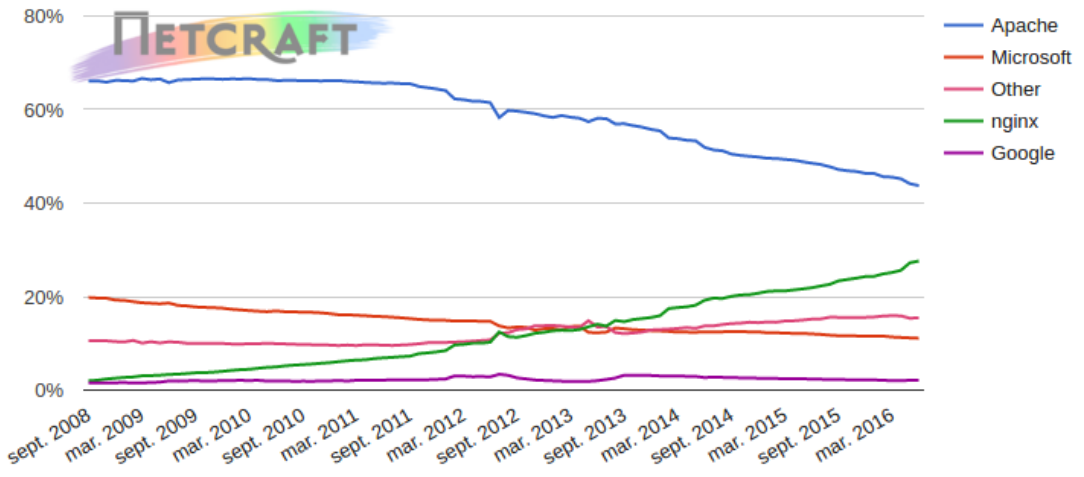


Figura 2. 26 Cuota de mercado de los distintos servidores web.

Este incremento en la popularidad de este servidor es en gran parte debido a los magníficos resultados que obtiene en los tests a los que se ha visto sometido frente a su inmediato competidor, Apache [36] [37].

Dado que los ficheros estáticos (como imágenes, .css o .js) son distribuidos directamente por CloudFront, el único propósito de Nginx en este TFG es el de atender las peticiones de HTTP dinámico que, tal como se verá más adelante, Django se encargará de procesar. Para alcanzar dicho objetivo se conectó mediante un socket de tipo UNIX el servidor Nginx con uWSGI, aplicación que hace de intermediaria entre el servidor y Django.

A continuación se muestra el fichero de configuración específico de este proyecto, el cual se encuentra en la ruta "/etc/nginx/sites-enabled/biosensor_nginx.conf". En el Código 2. 5 se muestra el contenido de este fichero.

```
upstream django {
    server unix:///biosensor/biosensor_env/biosensor/biosensor.sock;
}
server {
    listen      80;
    server_name .biosensor.io;
    charset     utf-8;
    client_max_body_size 75M;
    location / {
        uwsgi_pass  django;
        include     /biosensor/biosensor_env/biosensor/uwsgi_params;
    }
}
```

Código 2. 5 Fichero de configuración Nginx.

Cabe destacar que la versión de Nginx existente en los repositorios de Ubuntu 14.04.4 se encontraba algo obsoleta (nginx 1.4.6). Se decidió añadir la versión existente

en los repositorios de PPA, tal como los desarrolladores de nginx recomiendan, que en el momento de la redacción de este TFG es nginx 1.8.1 [38].

2.3.5 Python

Python es un lenguaje de programación de alto nivel, de propósito general, interpretado, dinámico y cuya filosofía hace hincapié en una sintaxis que favorezca un código legible [39].

Se trata de un lenguaje multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional y está siendo el lenguaje de programación que más popularidad ha ganado en los últimos años.

Según el índice TIOBE [40], que mide la popularidad de los lenguajes de programación en función de las búsquedas en diversos buscadores, Python se sitúa en cuarta posición. Por otro lado, según el índice PYPL, que sólo tiene en cuenta las búsquedas en Google, Pyhton ocuparía la segunda posición sólo por detrás de Java [41]. Tal popularidad es una gran ventaja frente a Ruby o Perl, ya que la comunidad que lo soporta es mucho más amplia. Esa fue una de las razones más determinantes a la hora de elegir Python y Django frente a Ruby y Rails. Otras fueron la existencia del administrador de paquetes "pip" y virtualenv. Una herramienta que permite crear entornos aislados para Python en los que es posible instalar paquetes sin interferir con otros virtualenvs ni con los paquetes de Python del sistema. Para poder comunicar las aplicaciones escritas en Python con el servidor web es necesario hacer uso de la especificación Web Server Gateway Interface (WSGI) [42].

A tal fin se instaló mediante pip una aplicación que implementa dicha especificación, permitiéndole con ello actuar de intermediaria entre las aplicaciones de

Django, escritas en Python, y el servidor web Nginx, que atiende las peticiones HTTP. La aplicación elegida para realizar este cometido fue uWSGI.

2.3.5.1 uWSGI

Se ha utilizado uWSGI porque es un contenedor de servidor de aplicaciones que implementa la especificación WSGI [43]. Esta herramienta permite la comunicación, mediante un sockect Unix, entre el servidor Nginx y las aplicaciones en Python desarrolladas con Django tal y como se muestra en la Figura 2. 27 .

Figura 2. 27 Esquema de uso de uWSGI.

La instalación de este programa se puede llevar a cabo mediante pip con el comando: `pip install uwsgi`. A continuación, se muestra el principal archivo de configuración situado en /biosensor/biosensor_env/biosensor/biosensor_uwsgi.ini y cuyo contenido se representa en Código 2. 6:

```
[uwsgi]
home            = /biosensor/biosensor_env/
chdir           = /biosensor/biosensor_env/biosensor/
module          = biosensor.wsgi
master          = true
processes       = 2
socket  = /biosensor/biosensor_env/biosensor/biosensor.sock
chmod-socket    = 666
vacuum          = true
```

Código 2. 6 Fichero de configuración de uWSGI.

Para que la aplicación se lance en cada arranque se añade el comando expuesto en el Código 2. 7 al rc.local.

```
uwsgi --ini /biosensor/biosensor_env/biosensor/biosensor_uwsgi.ini --uid biosensor -
-gid www-data
```

Código 2. 7 Orden de ejecución de uWSGI.

Capítulo 3. Django

3.1 Introducción

Django es un framework (entorno de trabajo) de desarrollo web de alto nivel, gratuito, de código abierto y escrito en Python, que fomenta el desarrollo rápido y el diseño limpio y pragmático siguiendo el patrón modelo-plantilla-vista (MTV) [44].

Este patrón pretende separar la lógica de acceso a la base de datos, la lógica de negocios y la lógica de presentación, tal como hace hace el famoso patrón de arquitectura de software Modelo-Vista-Controlador (MVC). Dicho de otro modo, el "Modelo" hace referencia al acceso a la capa de datos, la "Vista" se refiere a la parte del sistema que selecciona qué mostrar y cómo mostrarlo, y el "Controlador" implica la parte del sistema que decide qué vista usar, dependiendo de la entrada del usuario, accediendo al modelo si es necesario [45].

Django sigue la esencia del patrón MVC aunque posee pequeñas diferencias al respecto, sobretodo de nomenclatura:

- M, la porción de acceso a la base de datos, es manejada por la capa de la base de datos de Django.
- V, la porción que selecciona qué datos mostrar y cómo mostrarlos, es manejada por la vista y las plantillas.
- C, la porción que delega a la vista dependiendo de la entrada del usuario, es manejada por el framework siguiendo lo establecido en el archivo de configuración de las URLs y llamando a la función apropiada de Python para la URL obtenida.

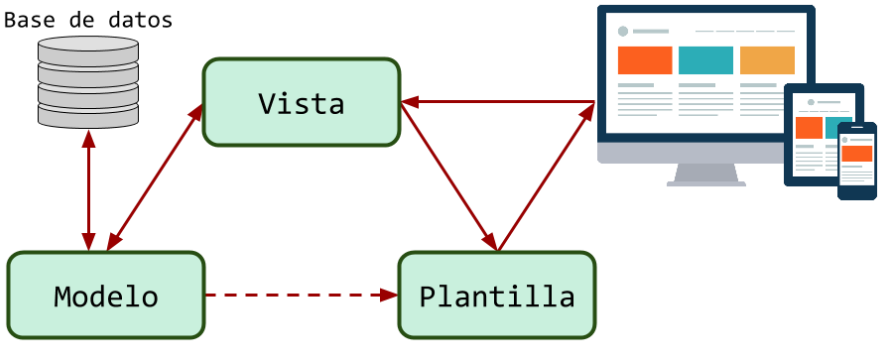


Figura 3. 1 Patrón de arquitectura Modelo-Plantilla-Vista

En el caso de Django el controlador es manejado por el propio framework y la parte más importante se produce en los modelos, las plantillas y las vistas. Por ello a su patrón de diseño se le conoce como MTV tal y como se muestra en la Figura 3. 1. En este caso:

- M significa "Model" (Modelo), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a ellos, cómo validarlos, cuál es su comportamiento y sus relaciones entre sí.
- T significa "Template" (Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación y son fundamentalmente plantillas de HTML.
- V significa "View" (Vista), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada.

Otra particularidad de Django es que utiliza la técnica conocida como mapeo objeto-relacional, o por sus siglas en inglés ORM. Esto viene a significar que una sentencia SQL como la siguiente:

```
SELECT * FROM "autores_autor" INNER JOIN "autor_libro" ON ("autores_autor"."id"
= "autores_libro"."autor_id")
INNER JOIN "autores_libro_librerias" ON ("autores_libro"."id" =
"autores_libro_librerias"."libro_id")
INNER JOIN "autores_libreria" ON ("autores_libro_librerias"."libreria_id" =
"autores_libreria"."id") WHERE "autores_libreria"."nombre" = "La Cultura"
```

En Django quedaría descrita de esta otra manera [46]:

```
Autor.objects.filter(libros__libreria__nombre = "La Cultura")
```

Otra de las principales características de Django es su servidor web integrado. Con él se pueden lanzar y depurar las aplicaciones muy rápidamente, algo muy útil en la etapa de desarrollo de un proyecto, pero nunca debe ser usado en un entorno de producción real pues este servidor integrado carece prácticamente de medidas de seguridad. Django además posee una consola de administración web que se instala por defecto, permitiendo un desarrollo muy ágil, y teniendo el control de la plataforma desde el primer momento de su lanzamiento.

El funcionamiento básico de Django, tal como se ve descrito en la Figura 3. 2, es el siguiente: Cuando un usuario accede a una URL como http://biosensor.io/login, Django recibe la solicitud del recurso /login a través del servidor web y consulta las reglas que se hayan descrito en el fichero urls.py. En función de ellas llama a la función del fichero views.py que será la encargada de atender dicha petición. Esta función puede acceder a la base de datos mediante llamadas a las funciones contenidas en el fichero models.py y, una vez tenga los datos, podrá componer una respuesta empleando las plantillas que sean necesarias.

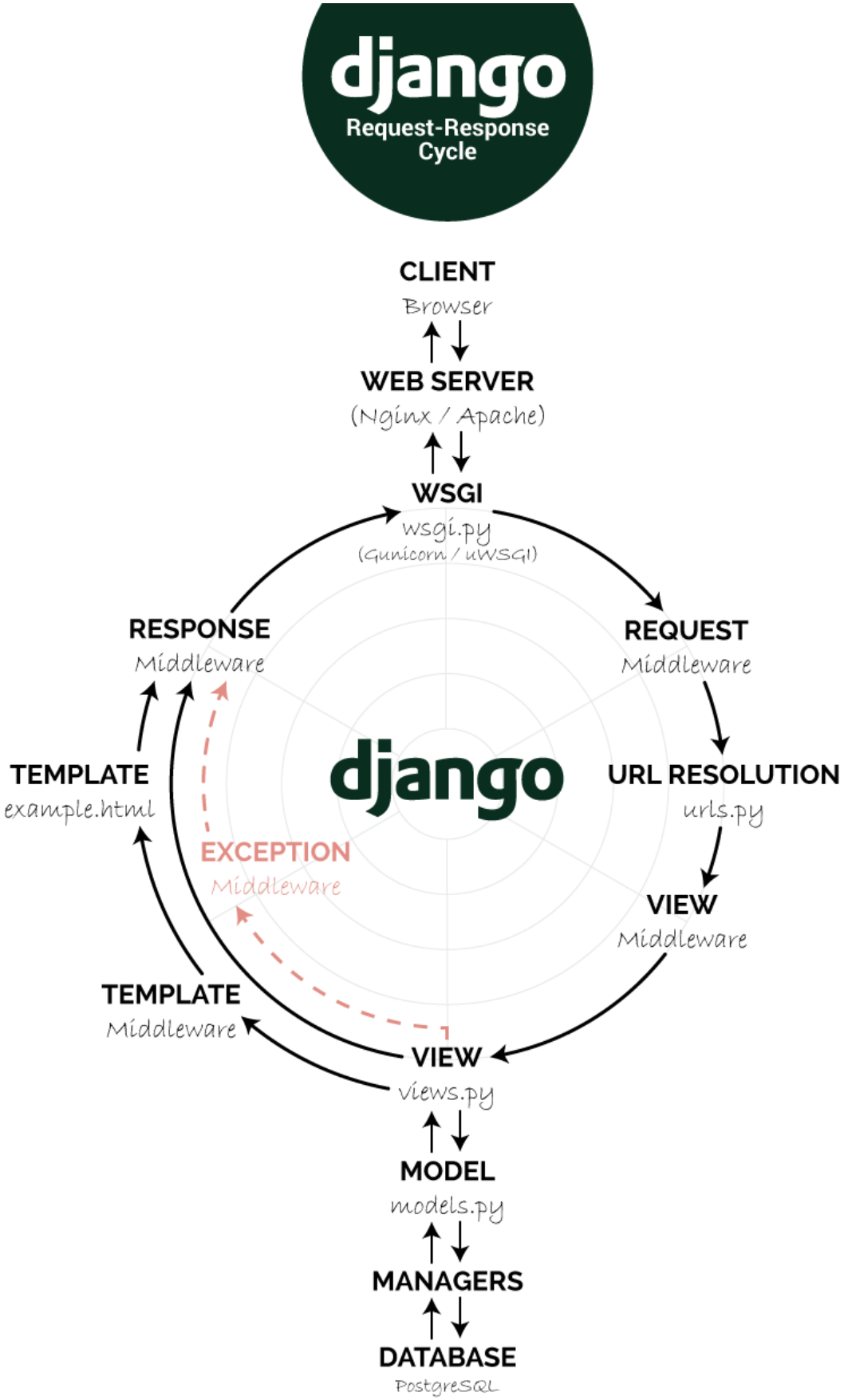


Figura 3. 2 Ciclo solicitud-respuesta de Django (ilustración de Ryan Nevius).

3.1.1 Instalación

Para la instalación de Django en la instancia de EC2 se hizo uso del usuario `biosensor` perteneciente al grupo `www-data` y cuya carpeta raíz se encuentra en `/biosensor.` Posteriormente se construyó el entorno virtual necesario para aislar el proyecto del resto del sistema mediante el comando mostrando en el Código 3. 1.

```
virtualenv /biosensor/biosensor_env/
```

Código 3. 1 Comando para la creación del entorno virtual de Python.

Dicho entorno permite generar una copia de todos los componentes necesarios para desarrollar una aplicación en Python con el fin de que estos permanezcan inalterados y separados del resto del sistema. Gracias a ello se podrán seguir desarrollando otras aplicaciones que requieran de otras dependencias sin que estas se afecten entre sí.

Para activar el entorno basta con acceder al directorio donde fue creado y cargarlo mediante los comandos mostrados en el Código 3. 2:

```
cd /biosensor/biosensor_env/
source bin/activate
```

Código 3. 2 Comandos para activar el entorno virtual de Python.

Finalmente, una vez listo el entorno, se procedió a la instalación de Django a través del administrador de paquetes pip mediante la orden del Código 3. 3:

```
pip install django
```

Código 3. 3 Comando para la instalación de django mediante el administrador de paquetes pip.

3.1.2 Configuración

Para poder acceder a Django vía web primero es necesario realizar la creación de un proyecto, es decir, un conjunto de ajustes para una instancia de Django que incluyen la configuración de la base de datos, opciones específicas de Django y configuraciones específicas de la aplicación.

Para el desarrollo de este TFG se procedió a crear un proyecto llamado "biosensor" que contuviese dos aplicaciones, de las que se hablará más adelante, llamas "biosensor_api" y "biosensor_app".

Para crear el proyecto se empleó el comando del Código 3. 4:

```
django-admin.py startproject biosensor
```

Código 3. 4 Comando para la creación del proyecto biosensor en Django.

Al hacerlo se crearon dentro del directorio biosensor los ficheros básicos de configuración de Django mostrados en Código 3. 5.

```
biosensor
├── biosensor
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

Código 3. 5 Estructura de archivos del proyecto generado.

Estos archivos son:

- **biosensor**: El directorio raíz biosensor es el contenedor del proyecto y su nombre no es relevante para Django.

- **manage.py**: Una utilidad de la línea de comandos que permite interactuar con Django de varias maneras como se verá más adelante.
- **biosensor**: El directorio interno biosensor. Su nombre es el nombre del paquete de Python que se tendrá que utilizar para importar cualquier módulo en su interior (por ejemplo biosensor.urls).
- **___init___.py**: Es un archivo vacío que le indica a Python que el directorio debe ser considerado como un paquete de Python.
- **settings.py**: Fichero de configuración principal del proyecto donde viene detallado desde las aplicaciones que hay instaladas hasta la configuración necesaria para conectarse a la base de datos.
- **urls.py**: En este fichero se especifica cómo se debe comportar Django en función de la URL que se le soliciten. Para Django podría considerarse como una "tabla de contenidos" del sitio web.
- **wsgi.py**: Un punto de acceso para los servidores web compatibles con WSGI para servir el proyecto.

Dentro de este proyecto se construyeron dos aplicaciones, `biosensor_api` y `biosensor_app`, explicadas en detalle en capítulos posteriores. Para ello se accedió a la carpeta raíz del proyecto creado y se ejecutaron los siguientes comandos del Código 3. 6:

```
cd biosensor
django-admin.py startapp biosensor_api
django-admin.py startapp biosensor_app
```

Código 3. 6 Comandos para la creación de las aplicaciones biosensor_api y biosensor_app.

La aplicación `biosensor_api` es necesaria para desarrollar la API REST haciendo uso del complemento Django Rest Framework. Mientras que en la aplicación web para consultar los experimentos fue desarrollada dentro de la aplicación

`biosensor_app`. Una vez creadas estas aplicaciones deben ser añadidas al fichero settings.py tal como se muestra en el Código 3. 7, o de otra forma, no tendrán efecto:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'biosensor_app',
    'biosensor_api',
]
```

Código 3. 7 Lista de aplicaciones instaladas en el fichero settings.py.

Para comprobar que el proyecto y las aplicaciones se crearon de forma correcta se hizo uso del servidor web integrado que incorpora Django. Para ello, primero fue preciso abrir el puerto 8000 en el grupo de seguridad de EC2, dado que este es el empleado por el servidor web de pruebas que utiliza Django. Una vez realizada la operación, para lanzar el servidor bastó con ejecutar la orden del Código 3. 8:

```
python manage.py runserver
```

Código 3. 8 Comando para lanzar el servidor.

Hecho esto, se pudo acceder vía web a la URL que mostró la salida del comando y, de este modo, apareció en el navegador la pantalla de bienvenida a Django, tal como se muestra en la Figura 3. 3, comprobando así que la instalación se efectuó correctamente.

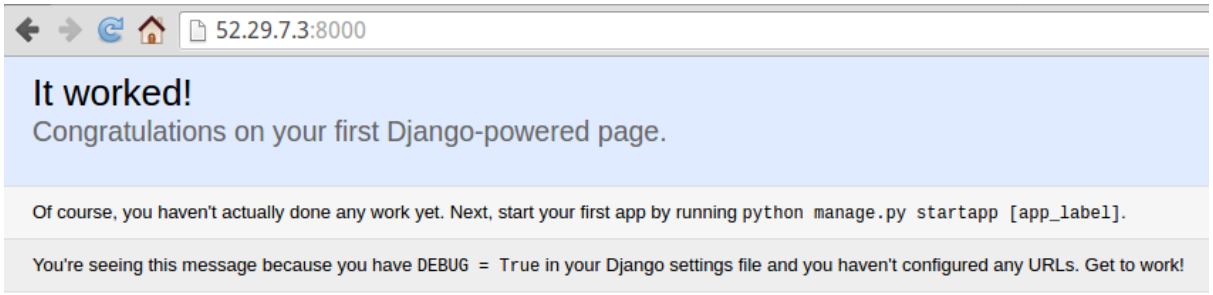


Figura 3. 3 Pantalla de bienvenida de Django.

En Dango se denomina "migrations" a la inclusión de los cambios realizados en el modelo dentro de la base de datos. Cuando se crea un nuevo proyecto Django necesita añadir a la base de datos una serie de información relativa al proyecto. Para aplicar dichos cambios, basta ejecutar el comando del Código 3. 9:

```
python manage.py migrate
```

Código 3. 9 Comando para actualizar el esquema de la base de datos.

Esto permite poder crear un usuario administrador con el que poder entrar a la consola de administración web. Para ello, basta ejecutar el comando del Código 3. 10:

```
python manage.py createsuperuser
```

Código 3. 10 Comando para crear un usuario administrador en Django.

Por último se modifica el fichero biosensor/settings.py añadiendo las configuraciones mostradas en el Código 3. 11:

```
ROOT_URLCONF = 'biosensor.urls'
LANGUAGE_CODE = 'es-es'
```

Código 3. 11 Configuración añadida al archivo settings.py.

3.1.3 Integración con S3 y CloudFront

Para poder integrar Django con S3 se creó un usuario específico para tal fin con IAM. Utilizando su ARN, junto con la herramienta AWS Policy Generator [47], se creó una política de uso del bucket que le permitía ser usado por este usuario, tal como se muestra en la Figura 3. 4. Posteriormente, se emplearon las credenciales del mismo para ser usadas en python mediante "boto", el SDK para Python de Amazon [48], junto a "django-storages" para integrarlas en Django [49].

Principal(s)	Effect	Action	Resource	Conditions
• arn:aws:iam::492623713284:user/biosensor_iam	Allow	s3:*	• arn:aws:s3:::biosensor/* • arn:aws:s3:::biosensor	*None*
• *	Allow	• s3:GetObject	arn:aws:s3:::biosensor/*	*None*

Figura 3. 4 Captura de la política de acceso al Bucket de S3.

Una vez creada la política del bucket se procedió a instalar las bibliotecas mediante los comandos del Código 3. 12.

```
pip install boto
pip install django-storages
```

Código 3. 12 Comando para la instalación de boto y django-storages mediante el administrador de paquetes pip.

Posteriormente, se creó en la carpeta raíz del proyecto un archivo con las clases necesarias para que Django pudiera hacer uso de boto llamado `aws_storage_classes.py` y cuyo contenido se muestra en el Código 3. 13.

```
from storages.backends.s3boto import S3BotoStorage

class StaticStorage(S3BotoStorage):
   location = 'static'

class MediaStorage(S3BotoStorage):
   location = 'media'
```

Código 3. 13 Contenido del fichero aws_storage_classes.py.

Por último, se editó el fichero settings.py del proyecto añadiendo las variables indicadas en el Código 3. 14 para poder hacer uso de las bibliotecas antes mencionadas:

```
DEFAULT_FILE_STORAGE = 'aws_storage_classes.MediaStorage'
STATICFILES_STORAGE = 'aws_storage_classes.StaticStorage'

AWS_ACCESS_KEY_ID = 'AKIAIFUOUZWYY4HUTTNA'
AWS_SECRET_ACCESS_KEY = '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'
AWS_STORAGE_BUCKET_NAME = 'biosensorirlanda'
AWS_S3_CUSTOM_DOMAIN = "d1qnaicv0u1e03.cloudfront.net"

STATIC_URL = "http://%s/static/" % AWS_S3_CUSTOM_DOMAIN
MEDIA_URL = "http://%s/media/" % AWS_S3_CUSTOM_DOMAIN
```

Código 3. 14 Variables de configuración de S3 y CloudFront en el fichero settings.py.

3.1.4 Integración con RDS

Para poder emplear la base de datos MariaDB proporcionada por Amazon RDS basta con instalar la biblioteca MySQL-python mediante el comando del Código 3. 15:

```
pip install MySQL-python
```

Código 3. 15 comando para la instalación de MySQL-python mediante el administrador de paquetes pip.

Una vez instaladas las dependencias necesarias, integrar Django con Amazon RDS fue posible sustituyendo la configuración existente de la entrada DATABASES del fichero settings.py del proyecto por la mostrada en el Código 3. 16:

```
DATABASES = {
 'default': {
   'ENGINE': 'django.db.backends.mysql',
   'NAME': 'biosensor',
   'USER': 'biosensor',
   'PASSWORD': 'passwordfalsa',
   'HOST': 'biosensor-db.cqhaazja3xms.eu-central-1.rds.amazonaws.com',
   'PORT': '3306',
 }
}
```

Código 3. 16 Variables de configuración de RDS en el fichero settings.py.

3.2 API REST

3.2.1 Introducción

A la hora de llevar a cabo la recolección de datos generados por los biosensores, se decidió diseñar una API, por medio de la cual éstos pudieran enviar sus resultados a un servidor en la nube con capacidad para poder almacenarlos. A la vez que en un futuro, permitiese la descarga de esquemas de configuraciones previas para este tipo de dispositivos.

3.2.1.1 REST

REST (REpresentational State Transfer) es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es más sencillo implementar software en la nube basado en esta

arquitectura que emplear protocolos tradicionales que se han usado en la última década como SOAP y XML-RPC.

Para el desarrollo de este TFG se ha hecho uso de esta arquitectura de desarrollo web para poder implementar una API que permita el intercambio de datos entre las máquinas de analítica clínica y el servidor en la nube.

REST se definió en el 2000 por Roy Fielding, coautor principal también de la especificación HTTP y pretende extrapolar el mismo modelo de comportamiento para desarrollar cualquier otro software. Por lo tanto, REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet. Además, una de sus características más destacables es que nunca se debe guardar el estado en el servidor, ya que todo lo que se requiere para mostrar la información solicitada debe estar en la consulta por parte del cliente. Este hecho de no guardar el estado, posibilita un mejor escalado ya que no se destinan recursos al almacenamiento de variables de sesión [50].

3.2.1.2 JSON

JSON (JavaScript Object Notation), es un formato abierto que usa texto ligero para el intercambio de datos basado en la notación literal de objetos de JavaScript. Una de sus ventajas frente a alternativas como XML es la facilidad a la hora de escribir un analizador sintáctico de JSON, ya que en JavaScript un texto JSON se puede analizar sin mucha complejidad utilizando la función eval().

En este proyecto se ha decidido que el intercambio de mensajes entre los biosensores y el servidor se realice empleando este formato de tipado de datos debido a la comodidad de su uso, empleando las herramientas que más adelante se nombrarán.

Actualmente existen intentos por estandarizar el uso de este formato para emplearlo en el desarrollo de APIs REST [51]. Sin embargo, aún se encuentran en una

fase muy temprana de adopción. Es por esto que no se han tenido en cuenta a la hora de elaborar este TFG.

Lo que sí se ha tenido en cuenta es respetar los estándares existentes sobre el uso del propio lenguaje en sí como soporte de datos. Para cerciorarse de ello, se ha hecho uso de diversos validadores online existentes tales como:

- https://jsonformatter.curiousconcept.com/
- http://jsonlint.com/

3.2.1.3 Django REST framework

Django REST framework es un módulo de Django que permite el desarrollo ágil y rápido de APIs REST. Además, provee una interfaz gráfica para poder manipular la API desde el propio navegador sin tener que depender de aplicaciones externas como Postman, la cual se usará más adelante con el fin de emular el comportamiento real de los biosensores [52].

Este complemento es capaz de detectar en función del user-agent si quien hace la consulta contra la API se trata de un navegador u otro tipo de aplicación que haga uso del protocolo HTTP. De modo que, si esta es accedida mediante el navegador, se mostrará una interfaz gráfica con la que se podrá hacer uso directo de la API, mientras que, si se trata de otro tipo de aplicación, mostrará en su lugar los datos en formato JSON.

3.2.2 Diseño

El modelo de comportamiento propuesto que debe cumplir la API es el mostrado en la Figura 3. 5, donde el biosensor que realizará un experimento envía, mediante un método POST, la configuración del mismo en formato JSON. A lo que, si la solicitud es correcta, el servidor ha de contestar reenviando los datos de la configuración más un campo con el identificador único del experimento.

Posteriormente, para cada muestra correspondiente a dicho experimento que el biosensor genera, este enviará sus valores al servidor especificando el identificador del experimento al que corresponde, a lo que el servidor deberá contestar replicando los mismos datos más un identificador único de la muestra recibida.

En la cabecera de mensajes HTTP debe estar presente usuario y su contraseña siguiendo el método de autenticación de acceso básica descrito en el RFC 1945 [53].

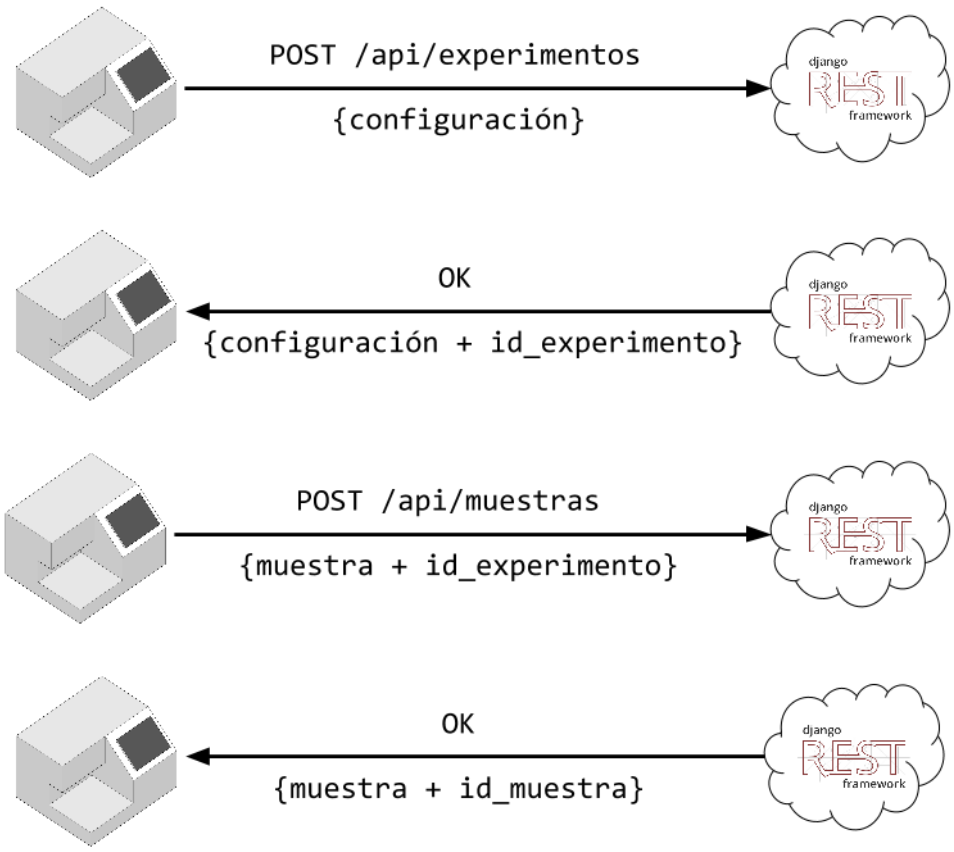


Figura 3. 5 Esquema de comportamiento de la API.

3.2.3 Desarrollo de la aplicación en Django

Para poder desarrollar la API REST se decidió utilizar el módulo Django REST Framework y para ello fue preciso instalarlo a través de pip tal como se muestra en el comando del Código 3. 17.

```
pip install djangorestframework
```

Código 3. 17 Comando para la instalación de Django REST framework mediante el administrador de paquetes pip.

Una vez instalado el complemento, el Código 3. 18 debe ser añadido a la lista de aplicaciones instaladas en el fichero principal de configuración del proyecto settings.py. Además, también se incluyó una regla específica de paginación para evitar una posible consulta masiva de la base de datos mostrada en el Código 3. 19.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'biosensor_app',
    'biosensor_api',
    'rest_framework',
]
```

Código 3. 18 Lista de aplicaciones instaladas en el fichero settings.py.

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10
}
```

Código 3. 19 Configuración de Django REST framework en el fichero settings.py.

También fue añadida la entrada del Código 3. 20 al fichero urls.py para poder autenticarse en la API desde el propio navegador web.

```
urlpatterns = [
  url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

Código 3. 20 Entrada añadida al fichero urls.py.

3.2.3.1 Modelo

El modelo aquí descrito fue importado desde la aplicación en Django biosensor_app, que se verá más adelante, para que de este modo ambas compartan el mismo [54].

El modelo consta de tres clases (Experimento, MuestraCurvaAngular y MuestraCineticaQuimica). La primera de ellas, se corresponde con la configuración del experimento. Por otro lado, las dos siguientes hacen referencia a los valores de las muestras obtenidas en el experimento con dicha configuración. Por tanto, existe una relación de dependencia entre los dos tipos de muestras y la clase Experimento tal como se muestra en el Código 3. 21.

```
class Experimento(models.Model):
   usuario = models.ForeignKey('auth.User', related_name='experimento')
   nombre = models.CharField(max_length=50)
   descripcion = models.CharField(blank=True,max_length=300)
   laser_encendido = models.BooleanField(default=False)
   bomba_peristaltica_sentido = models.CharField(max_length=10)
   bomba_peristaltica_velocidad = models.IntegerField()
   bomba_impulsional_ch1 = models.IntegerField()
   bomba_impulsional_ch2 = models.IntegerField()
   selector_viales_ch1 = models.IntegerField()
   selector_viales_ch2 = models.IntegerField()
   valvula_inyeccion_ch1 = models.IntegerField()
   valvula_inyeccion_ch2 = models.IntegerField()
   timestamp = models.DateTimeField(auto_now_add=True,auto_now=False)

class MuestraCurvaAngular(models.Model):
   usuario = models.ForeignKey('auth.User', related_name='muestra_ca')
   experimento = models.ForeignKey('Experimento', on_delete=models.CASCADE,
related_name='muestra_ca')
   numero = models.IntegerField()
   angulo = models.DecimalField(max_digits=8, decimal_places=6)
   ch1 = models.DecimalField(max_digits=8, decimal_places=6)
   ch2 = models.DecimalField(max_digits=8, decimal_places=6)

class MuestraCineticaQuimica(models.Model):
   usuario = models.ForeignKey('auth.User', related_name='muestra_ccq')
   experimento = models.ForeignKey('Experimento', on_delete=models.CASCADE,
related_name='muestra_ccq')
   numero = models.IntegerField()
   tiempo = models.DecimalField(max_digits=10, decimal_places=6)
   ch1 = models.DecimalField(max_digits=8, decimal_places=6)
   ch2 = models.DecimalField(max_digits=8, decimal_places=6)
```

Código 3. 21 Modelo de las aplicaciones biosensor_app y biosensor_api.

3.2.3.2 Vista

Los datos descritos en el Modelo pueden ser accedidos mediante la API gracias al uso de los métodos implementados en la Vista. Estos métodos, a su vez, hacen uso de la clase *Serializer*, que se encarga de formatear la información obtenida de la base de datos en formato JSON y viceversa. Estas clases se muestran en el Código 3. 22.

```
class MuestraCAViewSet(viewsets.ModelViewSet):
      queryset = MuestraCurvaAngular.objects.all()
      serializer_class = MuestraCASerializer
      permission_classes = (permissions.IsAuthenticatedOrReadOnly,
IsOwnerOrReadOnly)


class MuestraCCQViewSet(viewsets.ModelViewSet):
      queryset = MuestraCineticaQuimica.objects.all()
      serializer_class = MuestraCCQSerializer
      permission_classes = (permissions.IsAuthenticatedOrReadOnly,
IsOwnerOrReadOnly)


class ExperimentoViewSet(viewsets.ModelViewSet):
      queryset = Experimento.objects.all()
      serializer_class = ExperimentoSerializer
      permission_classes = (permissions.IsAuthenticatedOrReadOnly,
IsOwnerOrReadOnly)
```

Código 3. 22 Clases principales del fichero views.py.

Para poder serializar correctamente los datos ha sido necesario crear un archivo llamado serializers.py mostrado en el Código 3. 23 donde podemos encontrar las siguientes clases. En ellas, se indica qué datos del modelo se van a serializar y si deben hacerlo o no en función de los permisos requeridos. Aquellos campos que no estén presentes dentro de las clases no podrán ser serializados. De este modo, se puede restringir el acceso de la API a la base de datos.

```
class MuestraCASerializer(serializers.ModelSerializer):
	usuario = serializers.ReadOnlyField(source='usuario.username')
	class Meta:
		model = MuestraCurvaAngular
		fields = (
			'id',
			'numero',
			'experimento',
			'usuario',
			'numero',
			'angulo',
			'ch1',
			'ch2',)

class MuestraCCQSerializer(serializers.ModelSerializer):
	usuario = serializers.ReadOnlyField(source='usuario.username')
	class Meta:
		model = MuestraCineticaQuimica
		fields = (
			'id',
			'numero',
			'experimento',
			'usuario',
			'numero',
			'tiempo',
			'ch1',
			'ch2',)

class ExperimentoSerializer(serializers.ModelSerializer):
	usuario = serializers.ReadOnlyField(source='usuario.username')
	class Meta:
		model = Experimento
		fields = (
			'id',
			'nombre',
			'usuario',
			'descripcion',
			'laser_encendido',
			'bomba_peristaltica_sentido',
			'bomba_peristaltica_velocidad',
			'bomba_impulsional_ch1',
			'bomba_impulsional_ch2',
			'selector_viales_ch1',
			'selector_viales_ch2',
			'valvula_inyeccion_ch1',
			'valvula_inyeccion_ch2',)
```

Código 3. 23 Clases relevantes del fichero serilizers.py.

3.3 Aplicación Web

3.3.1 Introducción

El desarrollo de la aplicación web permite la eliminación de los procesos de descarga e instalación de software, así como la posibilidad de obtener estadísticas sobre el uso de la aplicación y, a consecuencia de estos dos factores, también se reduce la necesidad por parte de los usuarios de necesitar soporte técnico.

Como parte de la solución al problema abordado en este TFG, se ha decidido desarrollar una aplicación web, dirigida al personal de los laboratorios para que puedan consultar el estado los experimentos que se están llevando a cabo, así como los parámetros de configuración de los mismos y también el resultado de aquellos ya concluidos. Además, esta información, ha de poder ser consultada desde cualquier tipo de dispositivo ya sea móvil, tablets, sobremesas, etc. Por tanto la aplicación web debe tener un diseño adaptable que permita mostrar las gráficas de forma correcta en pantallas de diversos tamaños. Es por ello que se ha hecho uso de tecnologías como Bootstrap o las bibliotecas de gráficos de Google, que permiten lograr esta meta cumpliendo los estándares existentes en cuanto a desarrollo web.

3.3.1.1 Bootstrap

Bootstrap es uno de los framework de código abierto más populares de HTML, CSS y JS para el desarrollo de páginas “web responsive”, es decir, que el formato de la página web es adaptable a cualquier navegador o dispositivo como se muestra en la Figura 3. 1. Gracias a Bootstrap, se puede escalar las aplicaciones web mediante el desarrollo de un único código para tablets, móviles u ordenadores. Además, presenta múltiples elementos HTML, componentes CSS y estupendos plugins de jQuery. Con

Bootstrap se puede usar y personalizar de manera sencilla todos sus elementos (como las Barras de Navegación, Formularios, Tablas, Botones, Glyphicons, etc). Las interfaces creadas con Bootstrap son de gran usabilidad, y lo que es mejor, se dispone de un sistema grid de 12 columnas que trae por defecto para distribuir todos los elementos que deseen colocar en la página web [55].

Figura 3. 6 Aplicación web responsive.

Una vez descargado y descomprimido el archivo de Bbootstrap nos encontramos con la estructura presente en el Código 3. 1 Comando para la creación del entorno virtual de Python., en donde nos encontramos con los archivos css, js y fonts ya precompilados.

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   ├── bootstrap-theme.min.css
│   └── bootstrap-theme.min.css.map
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

Código 3. 24 Directorio Bootstrap.

3.3.1.2 Google Charts

Google Charts es la herramienta con la que se van a diseñar las gráficas de la curva angular SPR y de la curva cinética química generadas por el biosensor. Google Charts son unas bibliotecas gratuitas y muy potentes que nos permiten generar cualquier gráfica mediante un código Javascript embebido en la página web. De esta manera se facilita el trabajo al diseñador para que pueda plasmar las ideas deseadas ya que las gráficas son completamente personalizables gracias a la gran cantidad de opciones de configuración que presenta. Además, otra característica destacable de Google Charts es la interactividad que se le puede incorporar a las gráficas permitiendo realizar pan, zoom y scroll entre otras opciones. Por tanto, algunas representaciones que se pueden crear con este framework son:

- Gráficos de barras
- Gráficos de cajas
- Candlestick
- Gráficos compuestos
- Iconos dinámicos
- Gráficos de Línea
- Mapas
- Diagramas circulares
- Etc…

Por otro lado, con el fin de garantizar la compatibilidad entre navegadores, los gráficos se representan en HTML5/SVG. Además, garantiza la portabilidad entre plataformas para que los usuarios finales no tengan que instalar plugins o software adicional. Simplemente será necesario disponer de un navegador web [56].

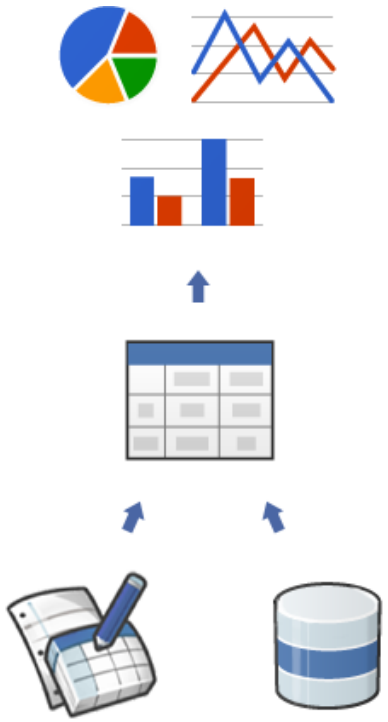


Figura 3. 7 Proceso de generación de gráficas Google Charts.

Para la implementación de las gráficas debemos seguir los siguientes pasos, los cuales además se muestran en la Figura 3. 7.

- Cargar las librerías de Google Charts que previamente se han descargado y que se encuentran en la carpeta de archivos estáticos del proyecto. Se añaden con el Código 3. 25.

```
<script src="{% static 'js/loader.js' %}"></script>
```

Código 3. 25 Código para cargar las librerías de Google Charts.

- Listar los datos que queremos representar y configurar las opciones de personalización de las gráficas. El código se puede observar en el Anexo II. Dentro del anexo, en el archivo `graficas.html` podemos encontrar en JavaScript dos funciones `drawCurveA()` y `drawCurveCQ()`, estas funciones generan la curva

angular y la curva cinética química respectivamente. Dentro de cada una de estas funciones nos encontramos con las variables presentes en el Código 3. 26.

```
var data = new google.visualization.DataTable();
var chart = new google.visualization.ChartWrapper(...);
var control = new google.visualization.ControlWrapper(...);
var dashboard =
new     google.visualization.Dashboard(document.querySelector('#dashboard_div'));
```

Código 3. 26 Variables para la creación de gráficas con Google Charts.

La variable *data* se encargará de generar una clase *DataTable,* esta clase hace adaptable los datos a cualquier tipo de gráfica que queramos generar. La variable `data` es donde se almacenan los datos extraídos de la base de datos. Por otro lado, la variable *chart* configura las distintas opciones para personalizar la gráfica. En cuanto a la variable `control` se configura la interactividad de la gráfica. Finalmente, en la variable dashboard se pasan los parámetros para que puedan ser representados en la aplicación web mediante un identificador.

- Finalmente, se crea un con el identificador del objeto mencionado anteriormente. Por ejemplo, en el caso de la curva cinética química se implementa el Código 3. 27.

```
<div id="chart_div_ccq" style="width=100%; height=100%;"></div>
```

Código 3. 27 Código HTML donde se incluye la gráfica de la curva cinética química.

3.3.2 Diseño

La interfaz gráfica de la aplicación web debe contar con una pantalla de login, tal como se muestra en la Figura 3. 8, donde los usuarios (registrados directamente por el

administrador) puedan ingresar en la plataforma usando para ello las credenciales proporcionadas por el administrador. Si el proceso de login se realiza con éxito, deberán acceder a un panel de información donde poder observar los resultados de los últimos experimentos realizados de forma gráfica y la configuración de los biosensores en el momento de generar los datos. El diseño de esta interfaz se muestra en la Figura 3. 9.

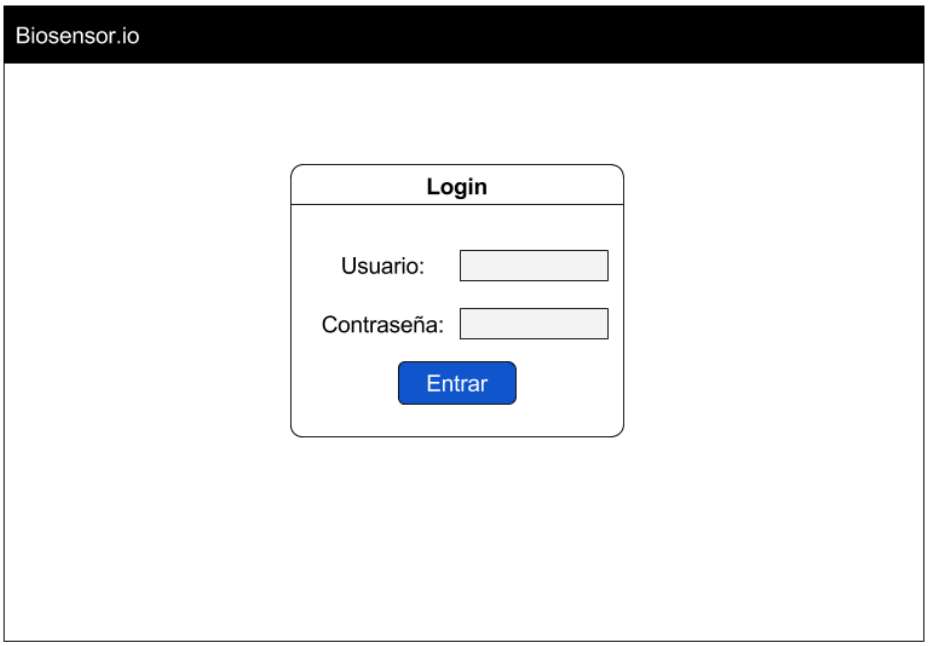


Figura 3. 8 Diseño de la pantalla de login para poder ingresar a la plataforma.

Figura 3. 9 Diseño del panel de información con la lista de últimos experimentos realizados y las gráficas y configuración del experimento seleccionado.

3.3.3 Desarrollo de la aplicación en Django

Una vez creada la aplicación biosensor_app, tal como se mostró al principio de este capítulo, se procede a describir el desarrollo de los principales componentes de este módulo por el orden descrito en su modelo de comportamiento presente en la Figura 3. 10, con la salvedad del archivo models.py, que ya fue descrito en apartados anteriores.

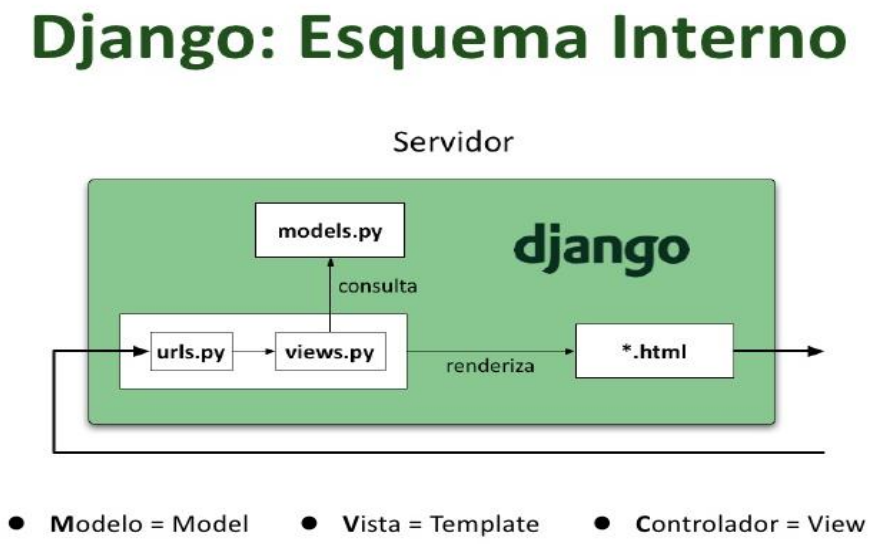


Figura 3. 10 Modelo de comportamiento de la aplicación web en Django.

Una vez recibida la petición del recurso Django determina, en función de su URL, qué vista se encargará de resolverla atendiendo a su expresión regular, tal como se muestra en el Código 3. 28. Si se trata simplemente de la URL http://biosensor.io, Django ejecutará el método "home". Si en lugar de esa URL la solicitud va dirigida hacia http://biosensor.io/experimento/1, entonces Django ejecutará el método "experimento". Ambos métodos se encuentran definidos dentro del archivo views.py.

```
urlpatterns = [
   url(r'^$', views.home, name='home'),
   url(r'^experimento/(?P<id_experimento>[0-9]+)/$', views.experimento,
name="experimento"),
]
```

Código 3. 28 Configuración del fichero urls.py de la aplicación biosenosr_app.

En el fichero views.py cuyo contenido se muestra en Código 3. 29, tanto la función "home", como la función "experimento", requieren que el usuario haya ingresado en el sistema. De no ser así, se le redirecciona a la página de login. Por otro lado, si la URL que se solicitó fue http://biosensor.io, la vista renderiza el template dash.html pasándole la lista de los últimos 20 experimentos del usuario que haya realizado la petición y seleccionando el primero de todos ellos como predefinido. Si por el contrario la URL solicitada fue http://biosensor.io/experimento/1, será la función experimento la encargada de renderizar la página, pero esta vez pasando como seleccionado el experimento cuyo identificador coincida con el especificado en la URL.

```
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from biosensor_app.forms import LoginForm
from models import Experimento, MuestraCurvaAngular, MuestraCineticaQuimica

@login_required(login_url="login/")
def home(request):
   experimentos = Experimento.objects.filter(usuario=request.user).order_by('-
timestamp')[:20]
   contexto = { "experimentos": experimentos,}
   if experimentos:
       seleccionado = experimentos[0]
       muestras_ca = seleccionado.muestra_ca.all()
       muestras_ccq = seleccionado.muestra_ccq.all()
       contexto = {"experimentos": experimentos, "seleccionado": seleccionado,
"muestras_ca": muestras_ca, "muestras_ccq": muestras_ccq }
   return render(request, "dash.html", contexto)

@login_required(login_url="login/")
def experimento(request, id_experimento):
   experimentos = Experimento.objects.filter(usuario=request.user).order_by('-
timestamp')[:20]
   seleccionado = experimentos.get(id=id_experimento)
   muestras_ca = seleccionado.muestra_ca.all()
   muestras_ccq = seleccionado.muestra_ccq.all()
   contexto = {
       "experimentos": experimentos,
       "seleccionado": seleccionado,
       "muestras_ca": muestras_ca,
       "muestras_ccq": muestras_ccq }
   return render(request, "dash.html", contexto)
```

Código 3. 29 Configuración del archivo views.py de la aplicación biosensor_app.

Capítulo 4. Resultados y Conclusiones

4.1 Resultados

En este capítulo se muestran los resultados obtenidos tras la integración de los diferentes elementos que conforman la plataforma descrita en la Figura 1. 5. A tal fin, esta sección de resultados se ha dividido en dos subapartados, los correspondientes a la API que será usada por los biosensores SPR y los correspondientes a la aplicación web destinada al personal de los laboratorios. Figura 4. 1.

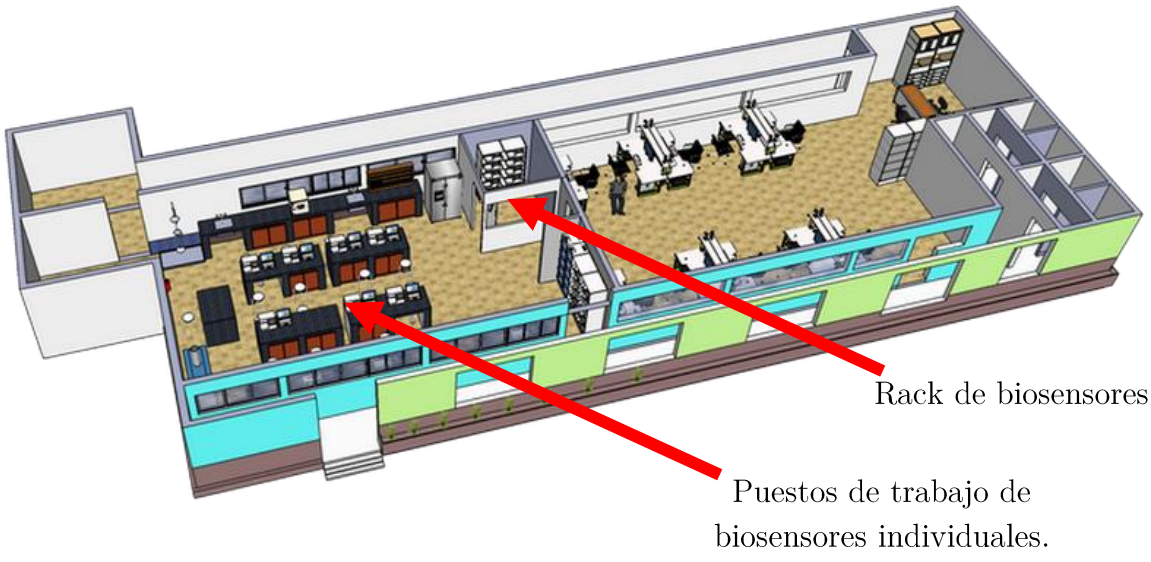


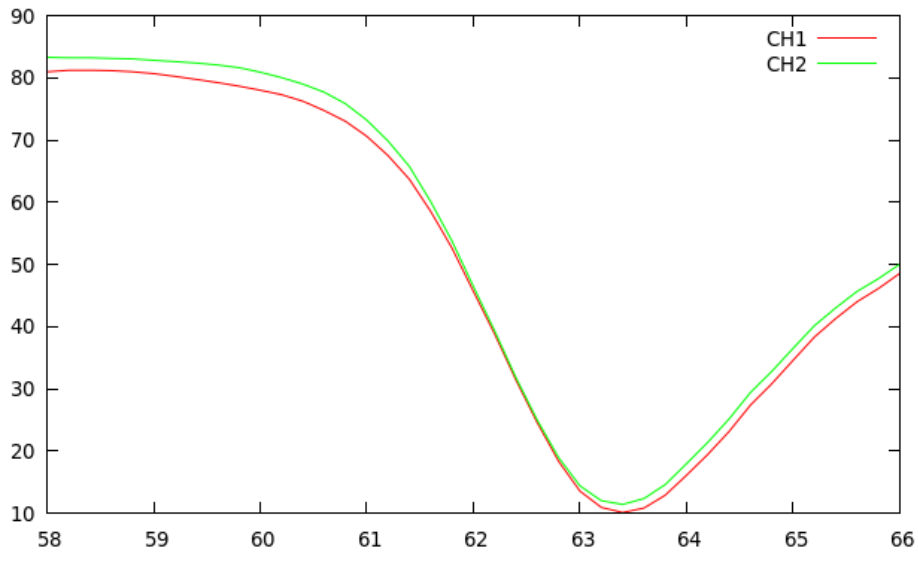
Figura 4. 1 Distribución de biosensores en el laboratorio.

4.1.1 Resultados de la API REST

Como no se dispone físicamente de ningún dispositivo biosensor con en el que poder testear la API en un entorno real, se ha optado por utilizar la herramienta Postman para enviar los datos correspondientes a la configuración de un nuevo experimento y, a su vez, desarrollar dos pequeños scripts en BASH que emulan el funcionamiento de este

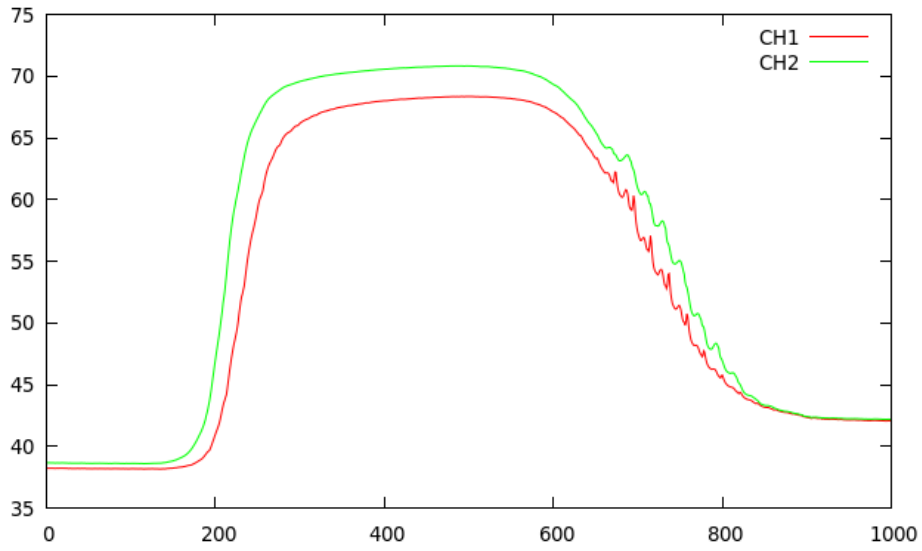
tipo de dispositivos a partir de los datos reales obtenidos de un experimento y proporcionados por el *Grupo de Investigación de Nanobiosensores y Aplicaciones de Bioanalítica Clínica del Instituto Catalán de Nanociencia y Nanotecnología* [57], que colabora con la división de *Equipos y Sistemas de Comunicación del IUMA*. Estos datos consisten en dos ficheros, adjuntos al soporte electrónico de este TFG, que contienen muestras de una curva angular y de una curva cinética química de lo medida de un biosensor real.

Para poder mostrar previamente y de forma gráfica el contenido de dichos ficheros, a fin de hacerse una idea del contenido en crudo de los mismos, se ha empleado la herramienta Gnuplot [58], tal como se muestra en la Figura 4. 2 y la Figura 4. 3.




```
gnuplot -p -e "set decimalsign locale ; plot for [col=2:3] 'Jul31-c01.dat'
using 1:col with lines title columnheader"
```

Figura 4. 2 Gráfico de la curva angular obtenido con Gnuplot a partir del fichero Jul31-c01.dat.




```
gnuplot -p -e "set decimalsign locale ; plot for [col=2:3] 'Jul25-m01.dat' using 1:col with lines title columnheader"
```

Figura 4. 3 Gráfico de la curva cinética química obtenido con Gnuplot a partir del fichero Jul25-m01.dat.

Teniendo esto en cuenta, se procede al envío de la configuración de un experimento ejemplo al que poder asociar las muestras de dichas gráficas. Para ello, empleando las credenciales de un usuario llamado biosensor, se hace un POST contra la ULR http://biosensor.io/api/experimentos, cuyo cuerpo del mensaje es el esquema en JSON mostrado en el Código 4. 1.

```
{
	"nombre": "Hormona de crecimiento",
	"descripcion": "Estudio del nivel hormonas del crecimiento en sangre",
	"laser_encendido": true,
	"bomba_peristaltica_sentido": "Derecha",
	"bomba_peristaltica_velocidad": 50,
	"bomba_impulsional_ch1": 44,
	"bomba_impulsional_ch2": 65,
	"selector_viales_ch1": 3,
	"selector_viales_ch2": 4,
	"valvula_inyeccion_ch1": 1,
	"valvula_inyeccion_ch2": 2
}
```

Código 4. 1 Configuración del primer experimento enviado.

A fin de poder realizar dicha tarea, se hizo uso de Postman, un plugin para el navegador Google Chrome que le permite actuar como una herramienta gráfica de testeo de APIs. Tal y como se muestra en la Figura 4. 4, en él se indicó el método HTTP a ejecutar, la URL contra la que realizar la solicitud y las credenciales del usuario biosensor. Además, se especificó en el cuerpo del mensaje los datos JSON de la configuración acordada, tal y como se muestra en la Figura 4. 5.

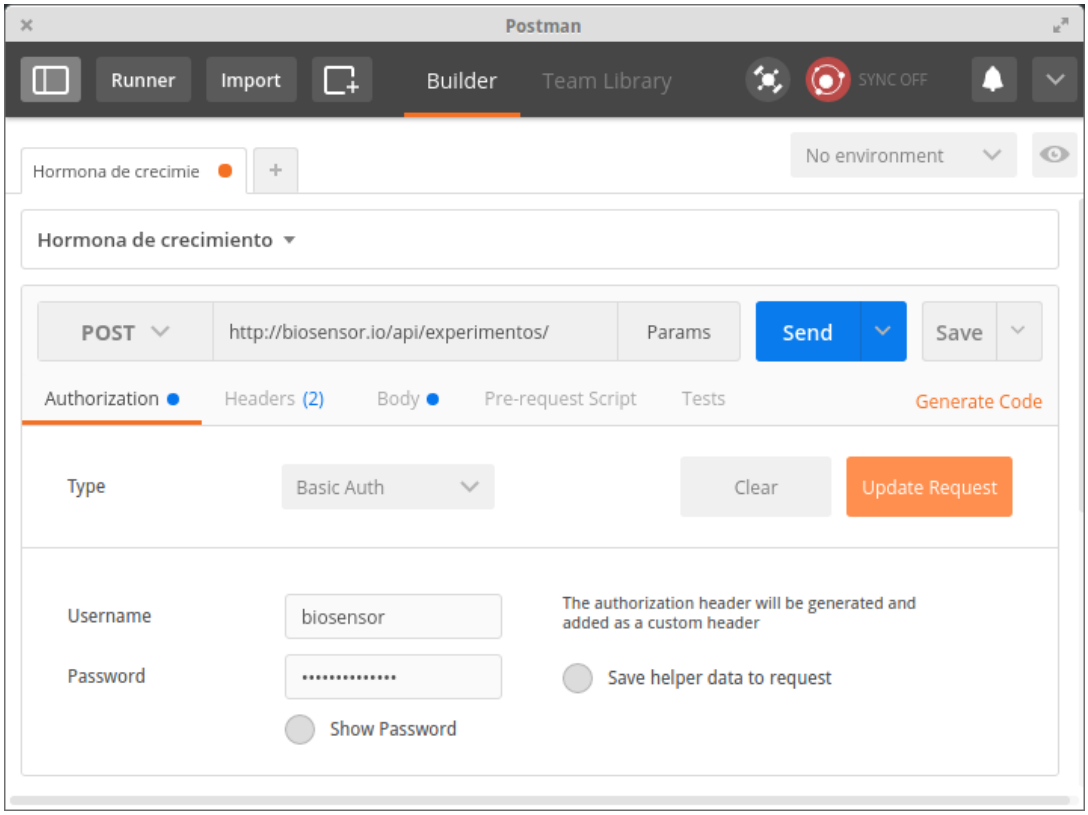


Figura 4. 4 Uso de Postman, credenciales.

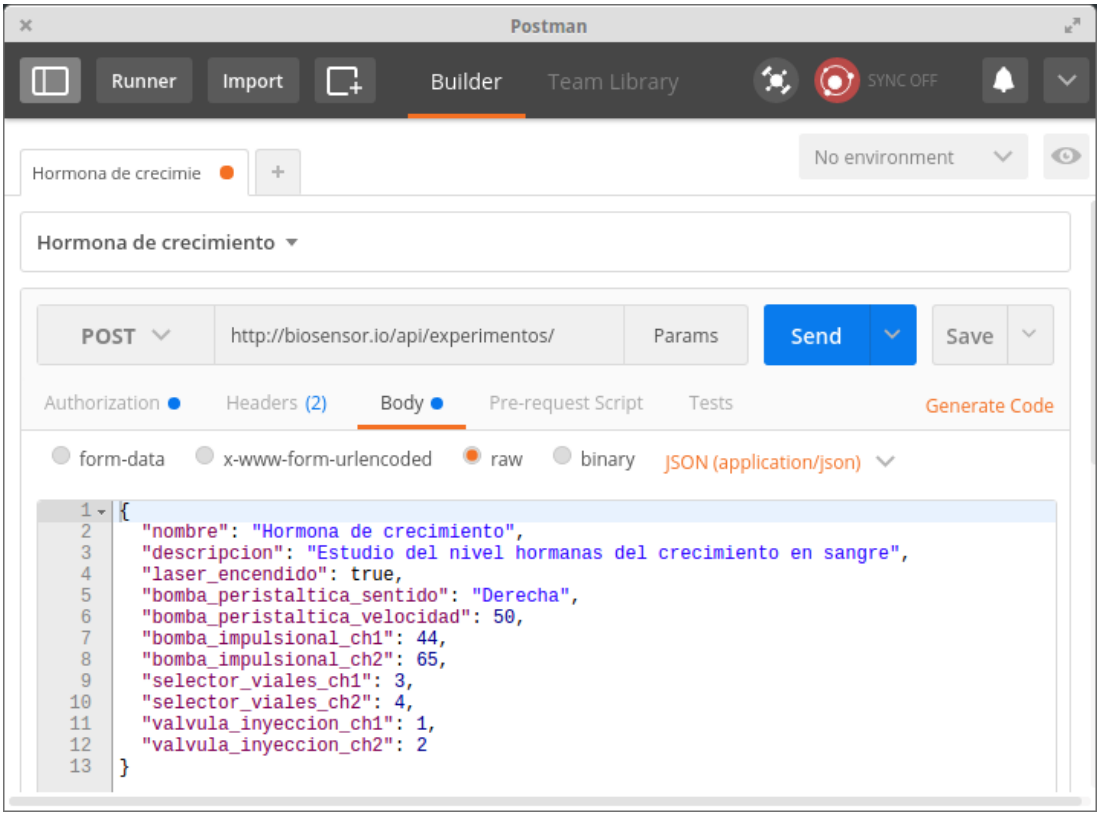


Figura 4. 5 Uso de Postman, cuerpo del mensaje.

Una vez realizada la petición de envío de los datos, se pudo comprobar gracias al código de respuesta 201 por parte del servidor, que la solicitud fue atendida de forma satisfactoria y, por tanto, los datos de la configuración se almacenaron correctamente en el servidor tal como muestra la Figura 4. 6. Además, en el cuerpo del mensaje se puede observar que se han añadido el campo del usuario que realizó la petición y el identificador único de la configuración del experimento, que podrá ser utilizado posteriormente para asignarle las muestras que se hayan obtenido empleando dicha configuración.

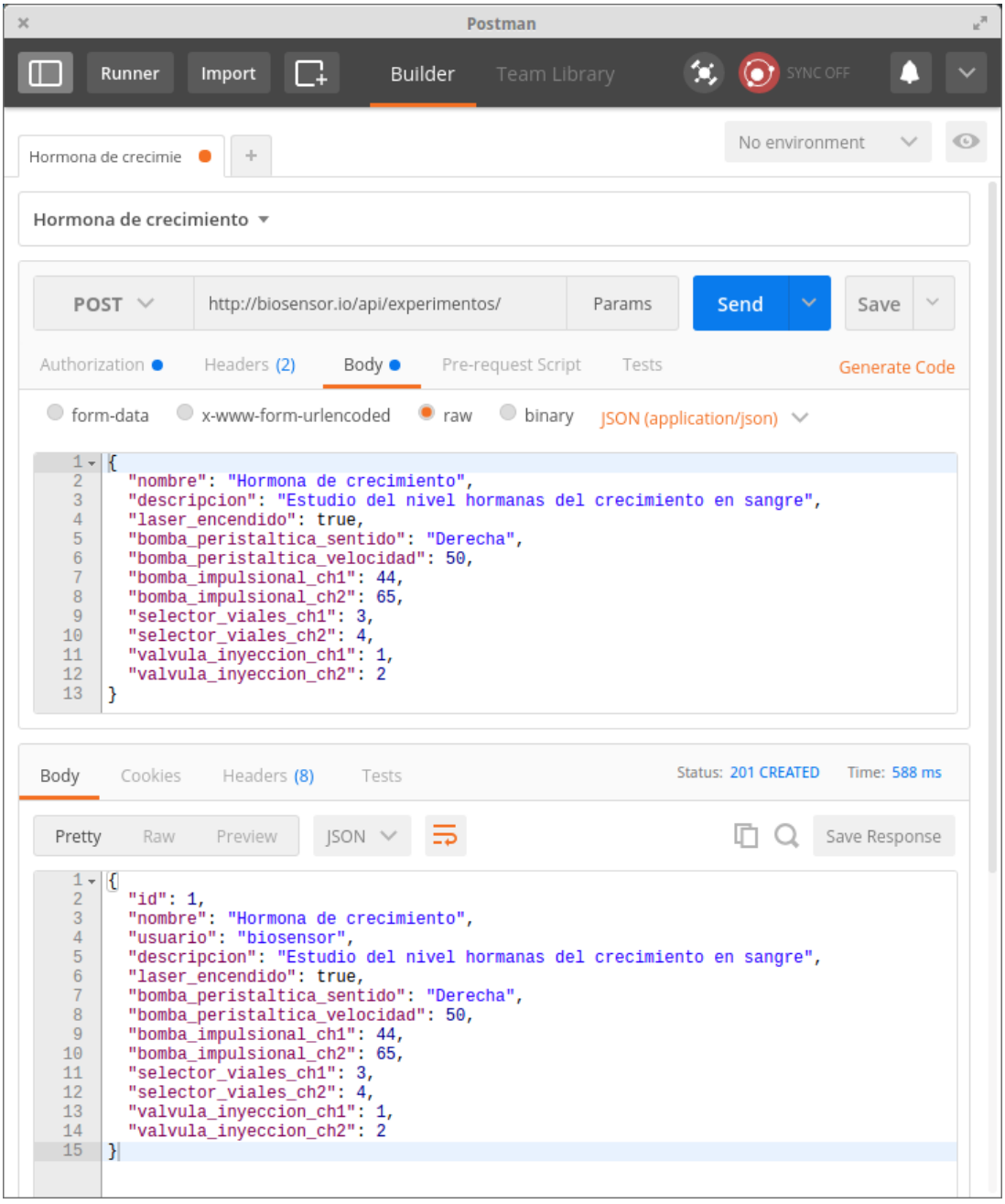


Figura 4. 6 Configuración subida con éxito por el usuario biosensor.

Con el fin de automatizar el proceso de envío de muestras y emular el comportamiento fidedigno de un biosensor, se ha desarrollado dos pequeños scripts en BASH, tal como se muestra en el Código 4. 3 y Código 4. 4. Uno para la curva angular y otro para la curva cinética química, que extraen las muestras de los ficheros y las van enviando una a una según lo establecido en el diseño de la API. Es decir, empleando el protocolo HTTP, realizan solicitudes de envío de datos con el método POST contra la

URL http://biosensor.io/api/muestras-ca/ si se trata de muestras de la curva angular, o http://biosensor.io/api/muestras-ccq/ si se trata de muestras de la curva cinética química. Usando en ambos las credenciales del usuario *biosensor* y en el cuerpo del mensaje el identificador del experimento al que pertenecen. Tal como se muestra en el ejemplo del Código 4. 2.

```
{
    "numero": 1,
    "experimento": 1,
    "angulo": "58.000000",
    "ch1": "80.965714",
    "ch2": "83.305714"
}
```

Código 4. 2 Ejemplo de la primera muestra de la curva angular perteneciente al primer experimento realizado.

```
#!/bin/bash
tail -n +6 Jul25-m01.dat | tr ',' '.' >> TIME.txt

USER="biosensor"
PASS="passfalsa"
EXPERIMENTO="1"

MUESTRA=1
while read TIEMPO CH1 CH2
do
curl --request POST \
    --url 'http://biosensor.io/api/muestras-ccq/' \
    --user $USER:$PASS \
    --header 'Content-Type: application/json' \
    --data
'{"numero":'$((MUESTRA++))',"experimento":'$EXPERIMENTO',"tiempo":"'$TIEMPO'","ch1":"'$C
H1'","ch2":"'${CH2//$'\r'}'"}'
done < TIME.txt
rm TIME.txt
```

Código 4. 3 Código del script cinetica.sh que genera la curva cinética a partir de los valores del fichero Jul25-m01.dat.

```
#!/bin/bash
tail -n +6 Jul31-c01.dat | tr ',' '.' >> ANG.txt

USER="biosensor"
PASS="passfalsa"
EXPERIMENTO="1"

MUESTRA=1
while read ANGULO CH1 CH2
do
curl --request POST \
    --url 'http://biosensor.io/api/muestras-ca/' \
    --user $USER:$PASS \
    --header 'Content-Type: application/json' \
    --data
'{"numero":'$((MUESTRA++))',"experimento":'$EXPERIMENTO',"angulo":"'$ANGULO'","ch1":"'$C
H1'","ch2":"'${CH2//$'\r'}'"}'
done < ANG.txt
rm ANG.txt
```

Código 4. 4 Código del script angular.sh que genera la curva cinética a partir de los valores del fichero Jul31-c01.dat.

Una vez ejecutados, la salida por consola de los scripts angular.sh y cinetica.sh muestra los valores que va devolviendo de las muestras enviadas junto con su identificador único y el usuario que las ha creado, dando testimonio de que estas han sido agregadas de forma correcta a la base de datos. Tal como se ve en la Figura 4. 7 y Figura 4. 8.

Posteriormente, lo único que debe hacer el personal de laboratorio que quiera consultarlos, es acceder a la aplicación web desde cualquier tipo de terminal empleando las credenciales del usuario biosensor. Esto se mostrará en la siguiente sección de este capítulo.

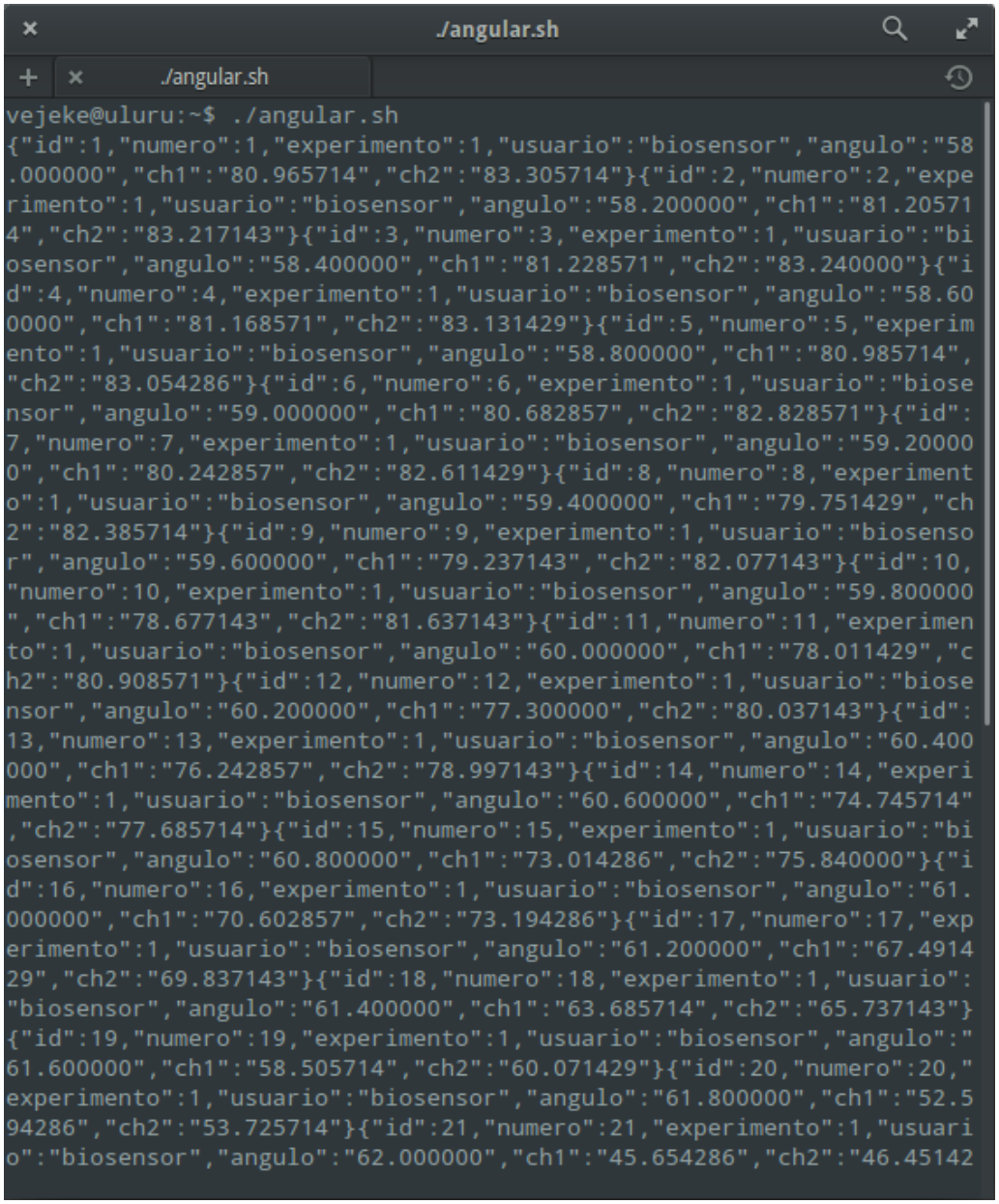


Figura 4. 7 Salida por consola del script angular.sh mostrando las respuestas del servidor los las muestras de la curva angular añadidas correctamente a la base de datos por parte del usuario biosensor.

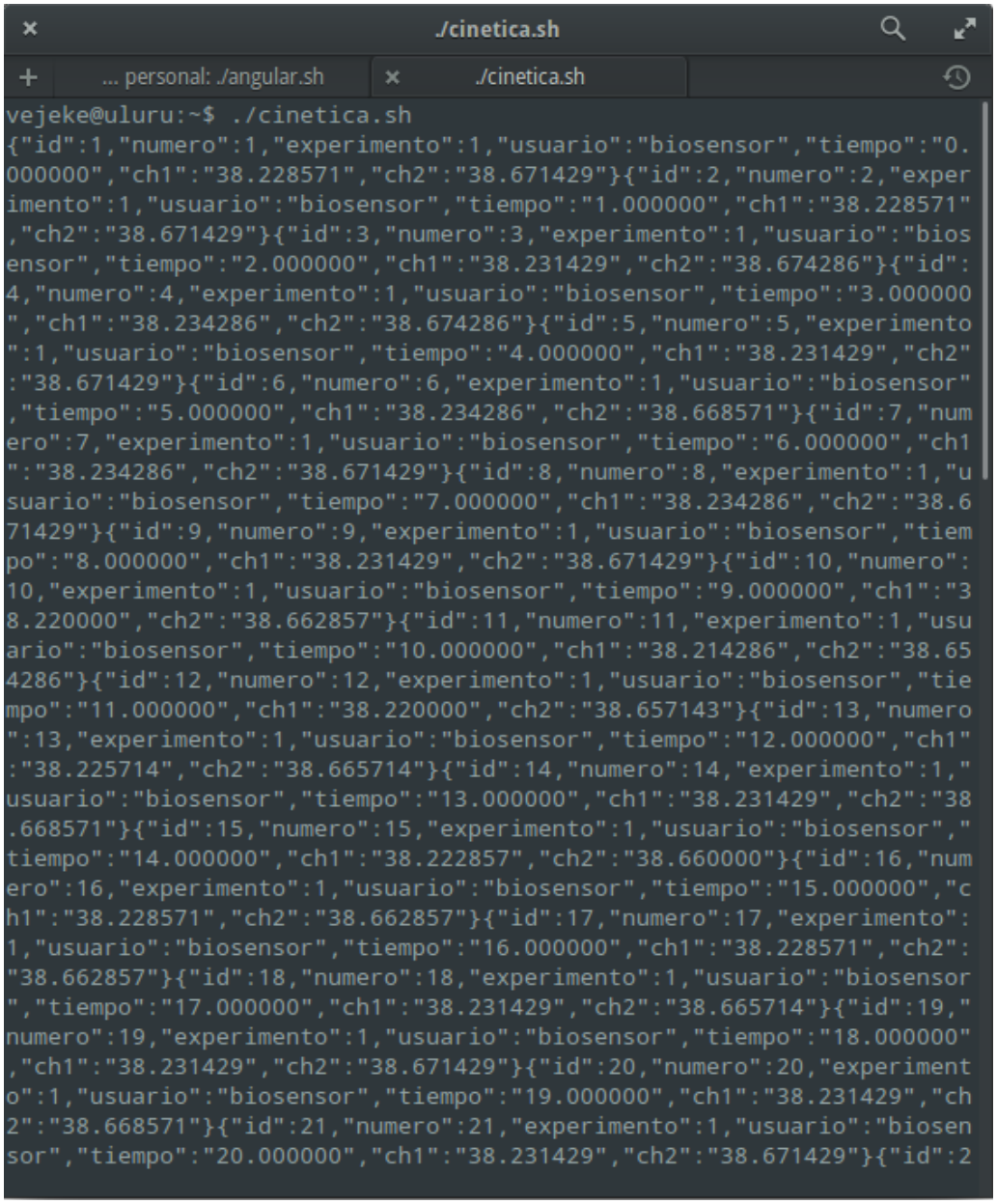


Figura 4. 8 Salida por consola del script angular.sh mostrando las respuestas del servidor los las muestras de la curva angular añadidas correctamente a la base de datos por parte del usuario biosensor.

4.1.2 Resultado de la aplicación web.

Una vez el personal de laboratorio haya realizado los experimentos oportunos, podrá acceder a los resultados a través de la aplicación web que se muestra en esta sección. Para ello, tan solo tiene que acceder al dominio biosensor.io donde podrá encontrar la siguiente pantalla de login, que en el caso de hacerlo mediante un ordenador

de sobremesa se muestra la página de la Figura 4. 9, pero si se accede desde un smartphone, esta pantalla se ve como se muestra en la Figura 4. 10.

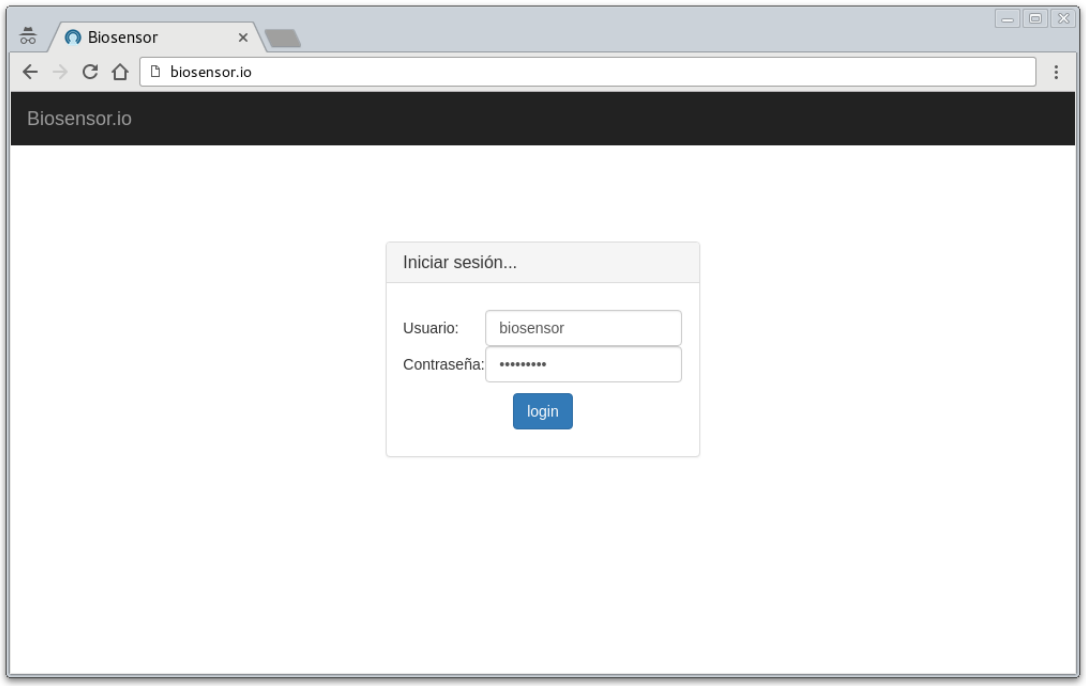


Figura 4. 9 Pantalla de login desde un ordenador sobremesa.

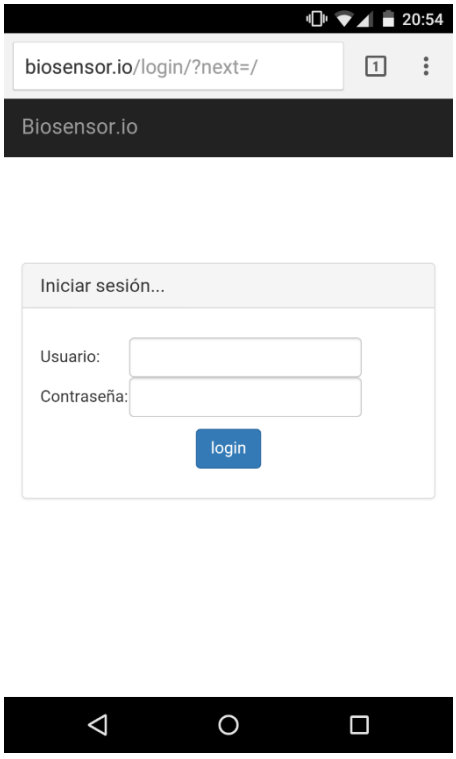


Figura 4. 10 Pantalla de login desde un terminal móvil.

Si el usuario no dispone de las credenciales necesarias, o las introduce de manera errónea, no podrá acceder a la plataforma.

Para el caso de que las credenciales introducidas sean correctas se mostrará un panel de información con las gráficas y la configuración del último experimento realizado, así como una lista interactiva de los que le precedieron ordenados de forma inversa a su fecha de creación. La apariencia de este panel variará en función del tamaño de la pantalla en la que se visualice. Por ejemplo, si se accede a la plataforma desde un ordenador de sobremesa con un gran monitor su apariencia será la mostrada en la Figura 4. 11.

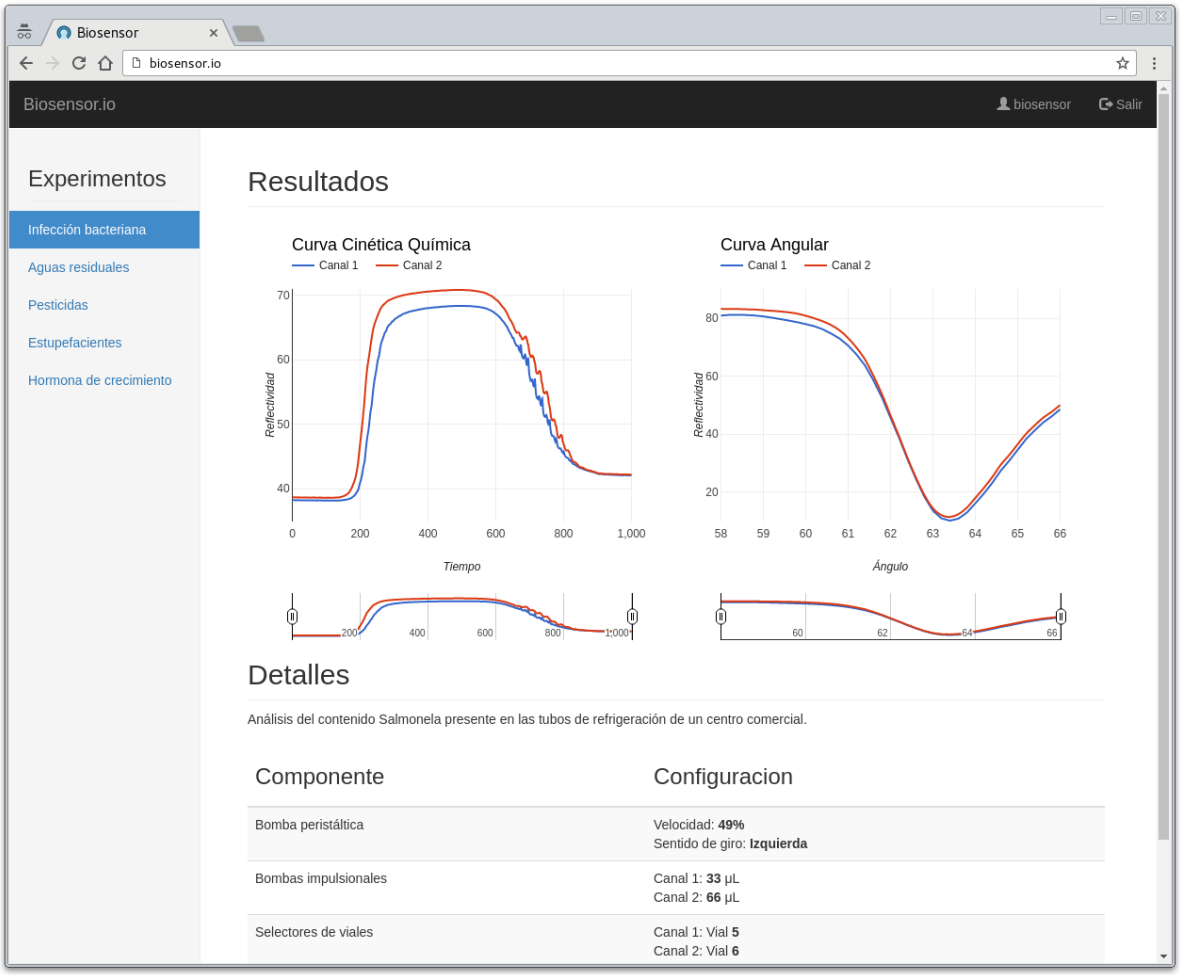


Figura 4. 11 Panel de resultado de los experimentos en un monitor amplio.

En cambio si se accede desde un teléfono móvil la barra de menú superior colapsará, y la lista de experimentos realizados se integrará dentro de esta a través de un botón de menú desplegable, dejando así un mayor espacio a las gráficas que, por otro lado, dejarán de mostrarse una junto a otra horizontalmente y pasarán a hacerlo de forma vertical, tal como se muestra en la Figura 4. 12.

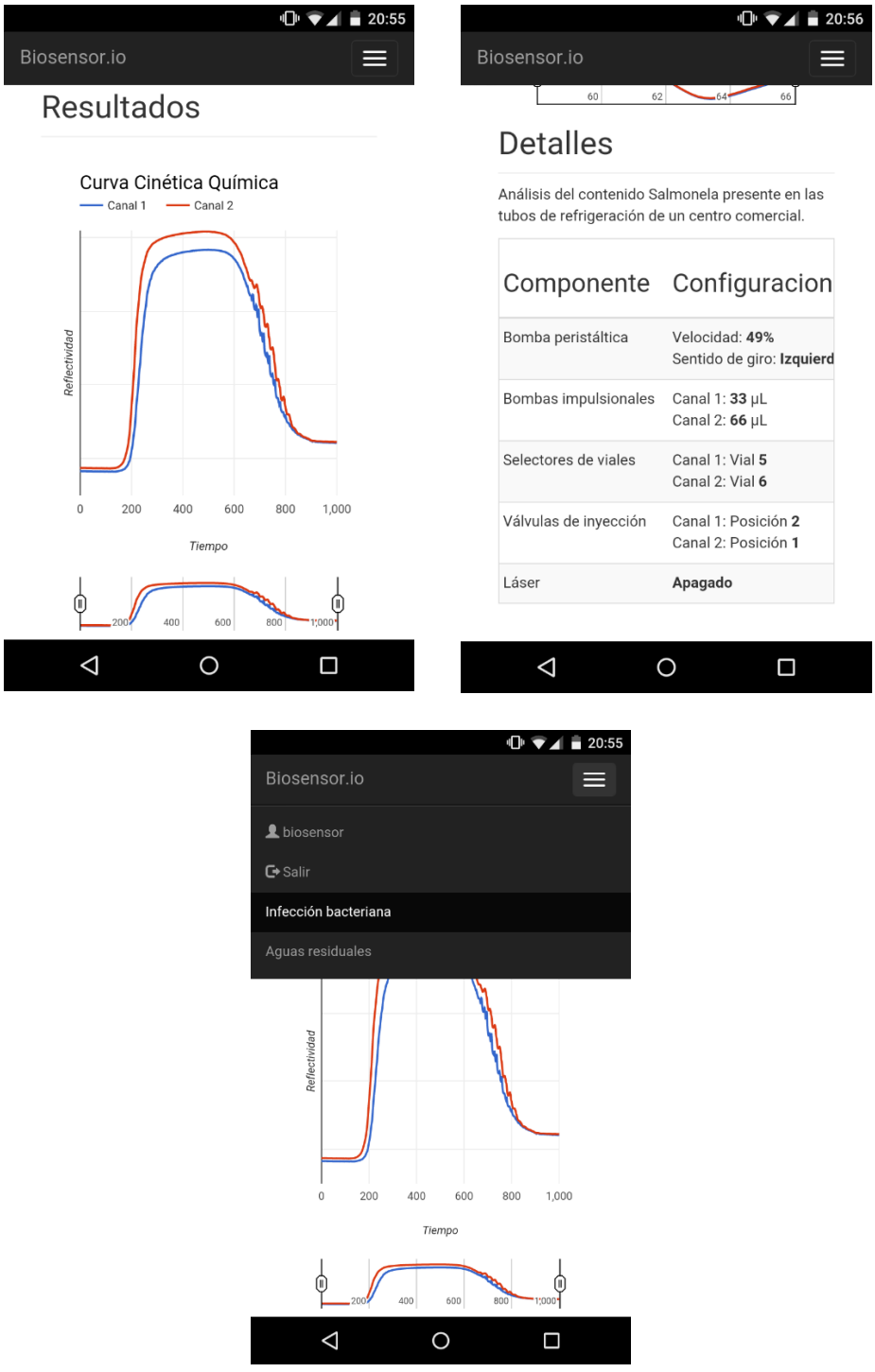


Figura 4. 12 Panel visto desde un terminal móvil.

Cabe destacar que los ficheros de estilos necesarios para mostrar correctamente la página se descargan directamente de la red de distribución de contenido en lugar de hacerlo desde EC2. Este hecho queda patente en la rapidez de la carga de la aplicación web mediante el uso del inspector de elementos del navegador Google Chrome que nos muestra los orígenes de los elementos necesarios para cargar la página como se puede comprobar en la Figura 4. 13.

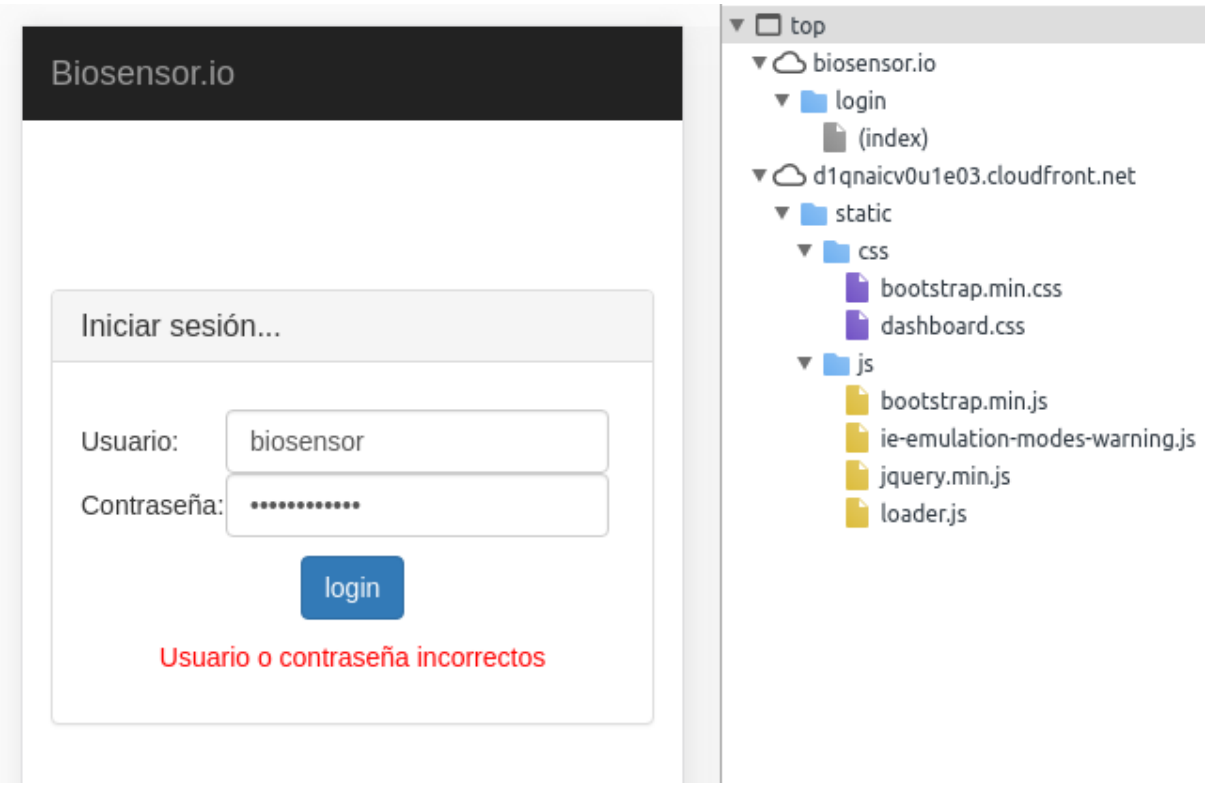


Figura 4. 13 Los ficheros estáticos tienen su origen en la distribución de CloudFront creada mientras que el contenido dinámico lo hace desde EC2.

4.2 Conclusiones

El objetivo fundamental de este Trabajo de Fin de Grado fue desarrollar una plataforma en la nube para la gestión y monitorización de máquinas de analítica clínica. Dicho objetivo se ha cumplido satisfactoriamente pero no sin antes haber desarrollado trabajo de documentación, apoyándose tanto en la información facilitada por Amazon

Web Services, como en la formación académica adquirida en el Grado de Ingeniería en Tecnologías de la Telecomunicación, y que ha permitido poder llevar a cabo todas las configuraciones necesarias de los diversos elementos que conforman la plataforma desarrollada.

En líneas generales, se desarrollado e implementado una API REST que permite a los dispositivos biosensores enviar los datos de resultados. También se ha realizado una aplicación web que permite al personal de laboratorio poder acceder a ellos en todo momento y lugar, a través de cualquier dispositivo con conexión a internet.

4.3 Líneas futuras

Debido a la naturaleza del trabajo, es difícil no imaginar mejoras y actualizaciones funcionales del mismo. Cabe destacar:

- Mejorar la seguridad del sistema adoptando HTTPS de forma gratuita gracias a los certificados ofrecidos por https://letsencrypt.org/
- Añadir más funcionalidades al panel de información como poder exportar los resultados de los experimentos en formato CSV o XLS.
- Realizar el front-end en el lado del cliente empleando para ello AngularJS tras realizar un rediseño en la API para tal finalidad.
- Permitir controlar el biosensor remotamente desde la propia aplicación web ofreciendo la posibilidad de configurar el experimento o bien desde la propia máquina de analítica clínica como sucede actualmente, o bien, desde la aplicación web.

Presupuesto

Siguiendo las recomendaciones del Colegio Oficial de Ingenieros Técnicos de Telecomunicación (COITT), se procede a desglosar en diferentes secciones los costes orientativos generados con el fin de llevar a cabo este TFG. Estos costes se dividen en:

- Recursos materiales.
    - Recursos hardware.
    - Recursos software.
    - Recursos en la nube.
- Trabajo tarifado por tiempo empleado.
- Costes de redacción del Trabajo de Fin de Grado.
- Material fungible.
- Derechos de visado del COITT.
- Costes de tramitación y envío.
- Aplicación de impuestos.

Recursos materiales

Para la elaboración de este TFG han sido necesarias diferentes recursos tanto hardware como software así como el uso de herramientas para la computación en la nube. Además, se ha usado el paquete Microsoft Office para la redacción de la memoria y la preparación de la presentación.

Recursos hardware

Para la ejecución de este estudio las herramientas hardware que se han utilizado son las siguientes:

- Ordenador portátil marca TOSHIBA modelo PORTÉGÉ Z30-A- 183 con 8GB de RAM, procesador Intel(R) Core(TM) i5-4210U 1.70GHz y una distribución GNU/Linux instalada.

Indicar que el período estimado de amortización de un equipo es de 24 meses. Para el desarrollo de este TFG se han empleado 4 meses, de esta forma, el importe corresponde a una sexta parte del precio de mercado.

Hardware	Precio mercado	Período de amortización	Tiempo de uso (meses)	Importe
TOSHIBA PORTÉGÉ Z30-A-183	806,00 €	24	4	134,33 €
TOTAL				**134,33 €**

Tabla 5. 1 Costes de los recursos hardware.

Recursos software

Las herramientas software utilizadas para el desarrollo de este TFG fueron:

- Django 1.9.2.
- Nginx 1.8.1.
- Python 3.5.1.
- JavaScript 1.8.5.
- Google Charts 1.0.
- Bootstrap 3.3.6.
- Microsoft Office 2013.
- Atom 1.8.0

- Gnuplot 5.0.2
- Postman 4.4.2
- Google Chrome 52.0

Al igual que los recursos hardware, el período estimado de amortización de un software es de 24 meses. Por lo tanto, para el desarrollo de este Trabajo Fin de Grado se emplean 4 meses, de esta forma, el importe corresponde a una sexta parte del precio de mercado.

Software	Precio mercado	Período de amortización	Tiempo de uso (meses)	Importe
Django 1.9.2	0,00 €	-	-	0,00 €
Nginx 1.8.1	0,00 €	-	-	0,00 €
Python 3.5.1	0,00 €	-	-	0,00 €
JavaScript 1.8.5	0,00 €	-	-	0,00 €
Google Charts 1.0	0,00 €	-	-	0,00 €
Bootstrap 3.3.6	0,00 €	-	-	0,00 €
Atom 1.8.0	0,00 €	-	-	0,00 €
Gnuplot 5.0.2	0,00 €	-	-	0,00 €
Postman 4.4.2	0,00 €	-	-	0,00 €
Google Chrome 52.0	0,00 €	-	-	0,00 €
Microsoft Office 2013	149,00 €	24	4	24,83 €
TOTAL				**24,83 €**

Tabla 5. 2 Costes de los recursos software.

Recursos en la nube

Para la implementación de este Trabajo Fin de Grado se utiliza la capa gratuita proporcionada por Amazon Web Service dentro del periodo de evaluación sin costes salvo el siguiente item.

- Dominio en Amazon Route 53.

Nube	Precio mercado	Período de amortización	Tiempo de uso (meses)	Importe
Amazon Route 53	34,40 €	-	-	34,40 €
TOTAL				**34,40 €**

Tabla 5. 3 Costes de los recursos en la nube.

Trabajo tarifado por tiempo empleado

Para la realización de este Trabajo de Fin de Grado se han empleado 300 horas, en las cuales se incluyen las tareas de preparación, desarrollo y documentación necesarias para la elaboración del mismo. Siguiendo las recomendaciones del COITT el importe de las horas de trabajo utilizadas para la realización del proyecto se calcula siguiendo la siguiente ecuación:

$$H = C_t \cdot 74{,}88 \cdot H_n + C_t \cdot 96{,}72 \cdot H_e$$

Donde:

- H son los honorarios totales por el tiempo dedicado.
- H_n son las horas normales trabajadas (dentro de la jornada laboral).
- H_e son las horas especiales.
- C_t es un factor de corrección en función del número de horas trabajadas.

Como se mencionó anteriormente, el número total de horas empleadas ha sido de 300 horas, llevando a cabo el trabajo durante 4 horas/día durante 5 días a la semana. Lo que hace un total de 20 horas semanales. Por tanto, el trabajo se ha desarrollado durante 15 semanas.

Según el COITT, el coeficiente C_t tiene un valor variable en función del número de horas trabajadas de acuerdo con la siguiente tabla:

Horas empleadas	Factor de corrección
Hasta 36 horas	1
Desde 36 horas a 72 horas	0,9
Desde 72 horas a 108 horas	0,8
Desde 108 horas a 144 horas	0,7
Desde 144 horas a 180 horas	0,65
Desde 180 horas a 360 horas	0,6
Desde 360 horas a 540 horas	0,55

Tabla 5. 4 Factor de corrección en función del número de horas trabajadas.

Como se puede observar el número de horas trabajadas está comprendido entre 180 horas y 360 horas, por lo que según la Tabla 5. 4. El factor de corrección es de 0,6. Con ello, la ecuación del importe de horas de trabajo es:

$$H = 0{,}6 \cdot 74{,}88 \cdot 300 + C_t \cdot 96{,}72 \cdot 0 = 13.478{,}40\ €$$

El trabajo tarifado por tiempo empleado, libre de impuestos, asciende a *trece mil cuatrocientos setenta y ocho euros con cuarenta céntimos* (**13.478,40 €**).

Coste de redacción del Trabajo de Fin de Grado

El importe de redacción del proyecto se calcula de acuerdo con la siguiente expresión:

$$R = 0{,}07 \cdot P \cdot C_n$$

Donde:

- P es el presupuesto del proyecto.
- C_n es el coeficiente de ponderación en función del presupuesto.

En la siguiente tabla se resume el presupuesto calculado hasta el momento, exento de impuestos:

Recursos	Coste
Recurso hardware	**134,33 €**
Recurso software	**24,83 €**
Recurso en la nube	**34,40 €**
Trabajo tarifado por tiempo empleado	**13.478,40 €**
TOTAL	**13.671,96 €**

Tabla 5. 5 Presupuesto exento de impuestos.

El presupuesto calculado hasta el momento asciende a 13.671,96 €. Como el coeficiente de ponderación para presupuestos inferiores a 30.500€ viene definido por el COITT, con un valor de 1,00 el coste derivado de la redacción del Trabajo Fin de Grado es de:

$$R = 0{,}07 \cdot 13.637{,}56 \cdot 1 = \ 957{,}04\,€$$

El importe por redacción del trabajo asciende a la cantidad de: *novecientos cincuenta y siete con cuatro céntimos* (**957,04 €**).

Material fungible

Además de los recursos hardware y software, en este proyecto se han empleado otros materiales como folios y gastos de impresión y encuadernación.

Materiales	Coste
Folios	12,00 €
Impresión	40,00 €
Encuadernación	9,00 €
Total	**61,00 €**

Tabla 5. 6 Costes del material fungible.

Derechos de visados del COITT

Los gastos de visado del COITT, se tarifan mediante la siguiente expresión:

$$V = 0{,}006 \cdot P \cdot C_v$$

Donde:

- P es el presupuesto del proyecto.
- C_v es el coeficiente reductor en función del presupuesto del proyecto.

El presupuesto P, calculado hasta el momento corresponde con la suma de los costes de los recursos materiales, de redacción y del material fungible siendo el total de:

Recursos	Coste
Recurso hardware	**134,33 €**
Recurso software	**24,83 €**
Recurso en la nube	**34,40 €**
Trabajo tarifado por tiempo empleado	**13.478,40 €**
Redacción del proyecto	**957,04 €**
Material fungible	**61,00 €**
TOTAL	**14.690,00 €**

Tabla 5. 7 Presupuesto para el derecho de visado del COITT.

Como el coeficiente de ponderación para presupuestos inferiores a 30.500€ viene definido por el COITT, con un valor de 1,00, el coste de los derechos de visado del proyecto es:

$$V = 0{,}006 \cdot 14.690{,}00 \cdot 1 = 88{,}14\,€$$

Por tanto, el coste de los derechos de visados del proyecto, asciende a la cantidad de: *ochenta y ocho euros con catorce céntimos* (**88,14 €**).

Gastos de tramitación y envío

Los gastos de tramitación y envío están fijados en 6,01 €.

Aplicación de impuestos

Para la actividad económica del presenta trabajo, el valor del Impuesto General Indirecto Canario (I.G.I.C) graba el presupuesto con un 7%. El coste total con el I.G.I.C incluido se desglosa en la siguiente tabla:

Descripción	Coste
Recursos hardware	134,33 €
Recursos software	24,83 €
Recursos en la nube	34,40 €
Trabajo tarifado por tiempo empleado	13.478,40 €
Redacción del TFG	957,04 €
Material fungible	61,00 €
Derechos de visado	88,14 €
Gastos de tramitación y envío	6,01 €
TOTAL (sin I.G.I.C)	14.749,75 €
Aplicación de impuestos	1.032,48 €
TOTAL (con I.G.I.C)	**15.782,23 €**

Tabla 5. 8 Aplicación de impuestos.

El presupuesto total del proyecto asciende a la cantidad de: ***quince mil setecientos ochenta y dos con veintitrés céntimos*** (15.779,42 €).

Las Palmas de Gran Canaria a 22 de Julio del 2016.

Fdo. D. Luis Pellicer Collado

Bibliografía

[1] Y. Liu, Q. Liu, S. Chen, F. Cheng, H. Wang, and W. Peng, "Surface Plasmon Resonance Biosensor Based on Smart Phone Platforms," *Sci. Rep.*, vol. 5, p. 12864, 2015.

[2] A. F. Medina Escuela, "Desarrollo e implementación electrónica de biosensores basados en la resonancia de plasmón superficial para medidas de alta sensibilidad," 2011.

[3] E. D. J. Franco, "Estrategias de inmovilización de anticuerpos para la detección directa de hormonas mediante Inmunosensores de Resonancia de Plasmón Superficial," p. 171, 2013.

[4] J. Treviño Castillo, "Desarrollo de un biosensor de resonancia de plasmón superficial para la determinación de hormonas pituitarias en muestras biológicas," 2009.

[5] V. Rumayor González, E. García Iglesias, O. Ruiz Galán, and L. Gago Cabezas, "Aplicaciones De Biosensores En La Industria Agroalimentaria," *CEIM,Dirección Gen. Universidades e Investig.*, 2005.

[6] C. T. Campbell, "Surface Plasmon Resonance (SPR) Biosensor Development," *Igarss 2014*, no. 1, pp. 1–5, 2014.

[7] G. Goldberg, "A Gen Z Mainframe Adventure," *IBM Systems Magazine*, 2016. [Online]. Available: http://www.ibmsystemsmag.com/mainframe/stoprun/Stop-Run/Connor-Krukosky/. [Accessed: 19-Jul-2016].

[8] E. Dans, "Anatomía del Efecto Menéame," 2008. [Online]. Available:

http://www.enriquedans.com/2008/08/anatomia-del-efecto-meneame.html. [Accessed: 19-Jul-2016].

[9] Stan Schwarz, "Web Servers, Earthquakes, and the Slashdot Effect," 2000. [Online]. Available: http://pasadena.wr.usgs.gov/office/stans/slashdot.html. [Accessed: 19-Jul-2016].

[10] P. Unbehagen, J. Chu, M. Hasan, S. Ma, B. Khasnabish, Y. Demchenko, M. Yu, N. So, and M. Morrow, "Cloud Reference Framework."

[11] I. Amazon Web Services, "¿Qué es la cloud computing?," 2016. [Online]. Available: https://aws.amazon.com/es/what-is-cloud-computing/. [Accessed: 19-Jul-2016].

[12] Jon Evans, "Whatever Happened To PaaS?," *Crunch Network*, 2015. [Online]. Available: https://techcrunch.com/2015/04/11/whatever-happened-to-paas/. [Accessed: 19-Jul-2016].

[13] I. Amazon Web Services, "Amazon EC2 – Hospedaje de servidores virtuales," 2016. [Online]. Available: https://aws.amazon.com/es/ec2/. [Accessed: 19-Jul-2016].

[14] I. Amazon Web Services, "Tipos de instancias de Amazon EC2," 2016. [Online]. Available: https://aws.amazon.com/es/ec2/instance-types/. [Accessed: 19-Jul-2016].

[15] I. Amazon Web Services, "Free Tier Limits," 2016. [Online]. Available: http://docs.aws.amazon.com/es_es/awsaccountbilling/latest/aboutv2/free-tier-limits.html. [Accessed: 19-Jul-2016].

[16] I. Amazon Web Services, "Precios de Amazon EC2," 2016. [Online]. Available: https://aws.amazon.com/es/ec2/pricing/. [Accessed: 19-Jul-2016].

[17] I. Amazon Web Services, "Importing a VM into Amazon EC2 as an Instance,"

2016. [Online]. Available: http://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/UsingVirtualMachinesinAmazonEC2.html. [Accessed: 19-Jul-2016].

[18] I. Amazon Web Services, "Elastic IP Addresses," 2016. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html. [Accessed: 19-Jul-2016].

[19] I. Amazon Web Services, "Precios de Elastic Load Balancing," 2016. [Online]. Available: https://aws.amazon.com/es/elasticloadbalancing/pricing/. [Accessed: 19-Jul-2016].

[20] I. Amazon Web Services, "Amazon RDS," 2016. [Online]. Available: https://aws.amazon.com/es/rds/. [Accessed: 19-Jul-2016].

[21] I. Amazon Web Services, "Amazon RDS para MariaDB," 2016. [Online]. Available: https://aws.amazon.com/es/rds/mariadb/. [Accessed: 19-Jul-2016].

[22] Santiago Borges, "MariaDB vs MySQL: ¿cuál debo elegir?," 2016. [Online]. Available: https://guiadev.com/mariadb-vs-mysql-cual-debo-elegir. [Accessed: 19-Jul-2016].

[23] I. Amazon Web Services, "Amazon S3," 2016. [Online]. Available: https://aws.amazon.com/es/s3/. [Accessed: 19-Jul-2016].

[24] I. Amazon Web Services, "Amazon CloudFront – Red de entrega de contenido (CDN)," 2016. [Online]. Available: https://aws.amazon.com/es/cloudfront/. [Accessed: 19-Jul-2016].

[25] I. Amazon Web Services, "Detalles del producto Amazon CloudFront," 2016. [Online]. Available: https://aws.amazon.com/es/cloudfront/details/. [Accessed: 19-Jul-2016].

[26] I. Amazon Web Services, “AWS Identity and Access Management (IAM),” 2016. [Online]. Available: https://aws.amazon.com/es/iam/. [Accessed: 19-Jul-2016].

[27] I. Amazon Web Services, “Amazon Resource Names (ARNs) and AWS Service Namespaces,” 2016. [Online]. Available: http://docs.aws.amazon.com/es_es/general/latest/gr/aws-arns-and-namespaces.html. [Accessed: 19-Jul-2016].

[28] I. Amazon Web Services, “Amazon Route 53,” 2016. [Online]. Available: https://aws.amazon.com/es/route53/. [Accessed: 19-Jul-2016].

[29] I. Amazon Web Services, “Django powered by Bitnami,” 2016. [Online]. Available: https://aws.amazon.com/marketplace/pp/B007I9Z8HG. [Accessed: 19-Jul-2016].

[30] I. Amazon Web Services, “Connect to Your Linux Instance,” 2016. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html. [Accessed: 19-Jul-2016].

[31] I. Amazon Web Services, “Connecting to Your Linux Instance Using SSH,” 2016. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html. [Accessed: 19-Jul-2016].

[32] “Archive of security issues.” [Online]. Available: http://django.readthedocs.io/en/latest/releases/security.html. [Accessed: 19-Jul-2016].

[33] DigitalOcean Inc., “Apache vs Nginx: Practical Considerations,” 2015. [Online]. Available:

https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations. [Accessed: 19-Jul-2016].

[34] Dan Kegel, "The C10K problem," 2014. [Online]. Available: http://www.kegel.com/c10k.html. [Accessed: 19-Jul-2016].

[35] Netcraft, "June 2016 Web Server Survey," 2016. [Online]. Available: http://news.netcraft.com/archives/2016/06/22/june-2016-web-server-survey.html. [Accessed: 19-Jul-2016].

[36] OrganicTeam, "Apache vs Nginx (Performance comparison)," 2013. [Online]. Available: https://www.theorganicagency.com/blog/apache-vs-nginx-performance-comparison/. [Accessed: 19-Jul-2016].

[37] Ruggero Tonelli, "Nginx Vs Apache in AWS," 2013. [Online]. Available: http://blog.celingest.com/en/2013/02/25/nginx-vs-apache-in-aws/. [Accessed: 19-Jul-2016].

[38] Nginx, "Install Nginx," 2016. [Online]. Available: https://www.nginx.com/resources/wiki/start/topics/tutorials/install/. [Accessed: 19-Jul-2016].

[39] RedMonk, "The RedMonk Programming Language Rankings: June 2015," 2015. [Online]. Available: http://redmonk.com/sogrady/2015/07/01/language-rankings-6-15/. [Accessed: 19-Jul-2016].

[40] TIOBE, "TIOBE Index for July 2016," 2016. [Online]. Available: http://www.tiobe.com/tiobe_index?page=index. [Accessed: 19-Jul-2016].

[41] Pierre Carbonnelle, "PYPL PopularitY of Programming Language," 2015. [Online]. Available: http://pypl.github.io/PYPL.html. [Accessed: 19-Jul-2016].

[42] Python, “PEP 3333 -- Python Web Server Gateway Interface v1.0.1,” 2010. [Online]. Available: https://www.python.org/dev/peps/pep-3333/. [Accessed: 19-Jul-2016].

[43] The uWSGI project, “uWSGI application server container,” 2016. [Online]. Available: https://github.com/unbit/uwsgi. [Accessed: 19-Jul-2016].

[44] Django Software Foundation, “Djanjo Project,” 2016. [Online]. Available: https://www.djangoproject.com/. [Accessed: 19-Jul-2016].

[45] J. K.-M. Adrian Holovaty, “El libro de Django 1.0,” 2015. [Online]. Available: http://librosweb.es/libro/django_1_0/. [Accessed: 19-Jul-2016].

[46] Jean Carlos Mariños Urquiaga, “¿Por qué usar Django?,” 2015. [Online]. Available: https://devcode.la/blog/por-que-usar-django/. [Accessed: 19-Jul-2016].

[47] Amazon Web Services LLC, “AWS Policy Generator,” 2010. [Online]. Available: http://awspolicygen.s3.amazonaws.com/policygen.html.

[48] I. Amazon Web Services, “Getting Started with the AWS SDK for Python (Boto),” 2016. [Online]. Available: https://aws.amazon.com/es/developers/getting-started/python/. [Accessed: 19-Jul-2016].

[49] David Larlet, “django-storages,” 2013. [Online]. Available: http://django-storages.readthedocs.io/en/latest/. [Accessed: 19-Jul-2016].

[50] Asier Marqués, “Conceptos sobre APIs REST,” 2016. [Online]. Available: http://asiermarques.com/2013/conceptos-sobre-apis-rest/. [Accessed: 19-Jul-2016].

[51] Steve Klabnik, Yehuda Katz, Dan Gebhardt, Tyler Kellen, and Ethan Resnick, “JSON API Documentation,” 2015. [Online]. Available:

http://jsonapi.org/. [Accessed: 19-Jul-2016].

[52] Tom Christie, "Django REST Framework," 2016. [Online]. Available: http://www.django-rest-framework.org/. [Accessed: 19-Jul-2016].

[53] R. T. Fielding, T. Berners-Lee, and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0."

[54] Django Software Foundation, "Django Documentation," 2016. [Online]. Available: https://docs.djangoproject.com/en/dev/ref/models/. [Accessed: 19-Jul-2016].

[55] Bootstrap, "Bootstrap Documentation," 2016. [Online]. Available: http://getbootstrap.com/. [Accessed: 19-Jul-2016].

[56] I. Google, "Using Google Charts," 2015. [Online]. Available: https://developers.google.com/chart/interactive/docs/. [Accessed: 19-Jul-2016].

[57] Institut Català de Nanociència i Nanotecnologi, "Nanobiosensor and bioanalytical applications group." [Online]. Available: http://icn2.cat/en/nanobiosensors-and-bioanalytical-applications-group.

[58] Gnuplot, "gnuplot homepage," 2016. [Online]. Available: http://www.gnuplot.info/. [Accessed: 19-Jul-2016].

Anexo I. Código API

Fichero urls.py

```
from biosensor_api import views
from biosensor_api.views import UserViewSet, ExperimentoViewSet, MuestraCAViewSet,
MuestraCCQViewSet
from django.conf.urls import url, include
from rest_framework.routers import DefaultRouter

router = DefaultRouter()
router.register(r'users', UserViewSet, base_name='users')
router.register(r'experimentos', ExperimentoViewSet)
router.register(r'muestras-ca', MuestraCAViewSet)
router.register(r'muestras-ccq', MuestraCCQViewSet)

urlpatterns = [
   url(r'^', include(router.urls)),
   url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

Fichero views.py

```
from django.contrib.auth.models import User
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework.reverse import reverse
from rest_framework import viewsets, permissions, generics
from biosensor_api.models import Experimento, MuestraCurvaAngular, MuestraCineticaQuimica
from biosensor_api.serializers import ExperimentoSerializer, UserSerializer,
MuestraCASerializer, MuestraCCQSerializer
from biosensor_api.permissions import IsOwnerOrReadOnly

class MuestraCAViewSet(viewsets.ModelViewSet):
       queryset = MuestraCurvaAngular.objects.all()
       serializer_class = MuestraCASerializer
       permission_classes = (permissions.IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly)
       def perform_create(self, serializer):
              serializer.save(usuario=self.request.user)

class MuestraCCQViewSet(viewsets.ModelViewSet):
       queryset = MuestraCineticaQuimica.objects.all()
       serializer_class = MuestraCCQSerializer
       permission_classes = (permissions.IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly)
       def perform_create(self, serializer):
              serializer.save(usuario=self.request.user)

class ExperimentoViewSet(viewsets.ModelViewSet):
       queryset = Experimento.objects.all()
       serializer_class = ExperimentoSerializer
       permission_classes = (permissions.IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly)
```

```
        def perform_create(self, serializer):
                serializer.save(usuario=self.request.user)

class UserViewSet(viewsets.ReadOnlyModelViewSet):
   queryset = User.objects.all()
   serializer_class = UserSerializer

@api_view(['GET'])
def api_root(request, format=None):
   return Response({
       'users': reverse('user-list', request=request, format=format)
   })
```

Fichero serializers.py

```
from rest_framework import serializers
from biosensor_api.models import Experimento, MuestraCurvaAngular,MuestraCineticaQuimica
from django.contrib.auth.models import User
from rest_framework.reverse import reverse

class MuestraCASerializer(serializers.ModelSerializer):
        usuario = serializers.ReadOnlyField(source='usuario.username')
        class Meta:
                model = MuestraCurvaAngular
                fields = (
                        'id',
                        'numero',
                        'experimento',
                        'usuario',
                        'numero',
                        'angulo',
                        'ch1',
                        'ch2',)

class MuestraCCQSerializer(serializers.ModelSerializer):
        usuario = serializers.ReadOnlyField(source='usuario.username')
        class Meta:
                model = MuestraCineticaQuimica
                fields = (
                   'id',
                        'numero',
                        'experimento',
                        'usuario',
                        'numero',
                        'tiempo',
                        'ch1',
                        'ch2',)

class ExperimentoSerializer(serializers.ModelSerializer):
        usuario = serializers.ReadOnlyField(source='usuario.username')
        class Meta:
                model = Experimento
                fields = (
                        'id',
                        'nombre',
                        'usuario',
                        'descripcion',
                        'laser_encendido',
                        'bomba_peristaltica_sentido',
                        'bomba_peristaltica_velocidad',
```

```
                    'bomba_impulsional_ch1',
                    'bomba_impulsional_ch2',
                    'selector_viales_ch1',
                    'selector_viales_ch2',
                    'valvula_inyeccion_ch1',
                    'valvula_inyeccion_ch2',)

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('id', 'username')
```

Fichero permissions.py

```
from rest_framework import permissions

class IsOwnerOrReadOnly(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return True
        return obj.usuario == request.user
```

Fichero models.py

```
from __future__ import unicode_literals
from django.db import models
from biosensor_app.models import Experimento, MuestraCurvaAngular, MuestraCineticaQuimica
```

Fichero apps.py

```
from __future__ import unicode_literals
from django.apps import AppConfig

class BiosensorApiConfig(AppConfig):
    name = 'biosensor_api'
```

Anexo II. Código Aplicación Web

Fichero urls.py

```
from django.conf.urls import url
from biosensor_app import views

urlpatterns = [
   url(r'^$', views.home, name='home'),
   url(r'^experimento/(?P<id_experimento>[0-9]+)/$', views.experimento,
name="experimento"),
]
```

Fichero views.py

```
# -*- coding: utf-8 -*-
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from biosensor_app.forms import LoginForm
from models import Experimento, MuestraCurvaAngular, MuestraCineticaQuimica

@login_required(login_url="login/")
def home(request):
   experimentos = Experimento.objects.filter(usuario=request.user).order_by('-
timestamp')[:20]
   contexto = { "experimentos": experimentos,     }
   if experimentos:
       seleccionado = experimentos[0]
       muestras_ca = seleccionado.muestra_ca.all()
       muestras_ccq = seleccionado.muestra_ccq.all()
       contexto = {"experimentos": experimentos, "seleccionado": seleccionado,
"muestras_ca": muestras_ca, "muestras_ccq": muestras_ccq }
   return render(request, "dash.html", contexto)

@login_required(login_url="login/")
def experimento(request, id_experimento):
   experimentos = Experimento.objects.filter(usuario=request.user).order_by('-
timestamp')[:20]
   seleccionado = experimentos.get(id=id_experimento)
   muestras_ca = seleccionado.muestra_ca.all()
   muestras_ccq = seleccionado.muestra_ccq.all()
   contexto = {
       "experimentos": experimentos,
       "seleccionado": seleccionado,
       "muestras_ca": muestras_ca,
       "muestras_ccq": muestras_ccq }
   return render(request, "dash.html", contexto)
```

Fichero models.py

```
from __future__ import unicode_literals
from django.db import models

class Experimento(models.Model):
   usuario = models.ForeignKey('auth.User', related_name='experimento')
   nombre = models.CharField(max_length=50)
   descripcion = models.CharField(blank=True,max_length=300)
   laser_encendido = models.BooleanField(default=False)
   bomba_peristaltica_sentido = models.CharField(max_length=10)
   bomba_peristaltica_velocidad = models.IntegerField()
   bomba_impulsional_ch1 = models.IntegerField()
   bomba_impulsional_ch2 = models.IntegerField()
   selector_viales_ch1 = models.IntegerField()
   selector_viales_ch2 = models.IntegerField()
   valvula_inyeccion_ch1 = models.IntegerField()
   valvula_inyeccion_ch2 = models.IntegerField()
   timestamp = models.DateTimeField(auto_now_add=True,auto_now=False)

   def __unicode__(self):
       return self.nombre

class MuestraCurvaAngular(models.Model):
   usuario = models.ForeignKey('auth.User', related_name='muestra_ca')
   experimento = models.ForeignKey('Experimento',
                                   on_delete=models.CASCADE,
                                   related_name='muestra_ca')
   numero = models.IntegerField()
   angulo = models.DecimalField(max_digits=8, decimal_places=6)
   ch1 = models.DecimalField(max_digits=8, decimal_places=6)
   ch2 = models.DecimalField(max_digits=8, decimal_places=6)

   def __unicode__(self):
       return str(self.numero)

class MuestraCineticaQuimica(models.Model):
   usuario = models.ForeignKey('auth.User', related_name='muestra_ccq')
   experimento = models.ForeignKey('Experimento',
                                   on_delete=models.CASCADE,
                                   related_name='muestra_ccq')
   numero = models.IntegerField()
   tiempo = models.DecimalField(max_digits=10, decimal_places=6)
   ch1 = models.DecimalField(max_digits=8, decimal_places=6)
   ch2 = models.DecimalField(max_digits=8, decimal_places=6)

   def __unicode__(self):
       return str(self.numero)
```

Fichero forms.py

```
from django.contrib.auth.forms import AuthenticationForm
from django import forms

class LoginForm(AuthenticationForm):
   username = forms.CharField(label="Username", max_length=30,
                              widget=forms.TextInput(attrs={'class': 'form-control',
```

```
                                                        'name': 'username'}))
    password = forms.CharField(label="Password", max_length=30,
                               widget=forms.TextInput(attrs={'type': 'password',
                                                        'class': 'form-control',
                                                        'name': 'password'}))
```

Fichero apps.py

```
from __future__ import unicode_literals
from django.apps import AppConfig

class BiosensorAppConfig(AppConfig):
    name = 'biosensor_app'
```

Fichero admin.py

```
from django.contrib import admin
from biosensor_api.models import Experimento, MuestraCurvaAngular, MuestraCineticaQuimica

admin.site.register(Experimento)
admin.site.register(MuestraCurvaAngular)
admin.site.register(MuestraCineticaQuimica)
```

Fichero base.html

```
{% load staticfiles %}
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <meta name="author" content="Luis Pellicer Collado">
   <title>Biosensor
   </title>
   <link rel="icon" type="image/png" href="{% static 'img/biosensor_logo.png' %}">
   <link href ="{% static 'css/bootstrap.min.css' %}" rel="stylesheet">
   <link href="{% static 'css/dashboard.css' %}" rel="stylesheet">
 </head>
 <body>
   {% include 'navbar.html' %}
   {% block content %}
   {% endblock %}
   <script src ="{% static 'js/jquery.min.js' %}">
   </script>
   <script src ="{% static 'js/bootstrap.min.js' %}">
   </script>
   <script src ="{% static 'js/ie-emulation-modes-warning.js' %}">
   </script>
   <script src ="{% static 'js/loader.js' %}">
   </script>
   {% block javascript %}
   {% endblock %}
```

```
 </body>
</html>
```

Fichero login.html

```
{% extends 'base.html' %}
{% block content %}

<div class="container">
<div class="row">
 <div class="col-md-4 col-md-offset-4" style="top: 90px">
  <div class="login-panel panel panel-default">
   <div class="panel-heading">
    <h3 class="panel-title">Iniciar sesión...</h3>
   </div>
   <div class="panel-body">
    <form method="post" action="{% url 'django.contrib.auth.views.login' %}">
     {% csrf_token %}
     <p class="bs-component">
      <table>
       <tr>
        <td>Usuario: </td>
        <td>{{ form.username }}</td>
       </tr>
       <tr>
        <td>Contraseña: </td>
        <td>{{ form.password }}</td>
       </tr>
      </table>
     </p>
     <p class="bs-component">
      <center>
       <input class="btn btn-primary" type="submit" value="login" />
      </center>
     </p>
     <input type="hidden" name="next" value="{{ next }}" />
    </form>
    {% if form.errors %}
    <center>
     <p style="color:red;">Usuario o contraseña incorrectos</p>
    </center>
    {% endif %}
    {% if next %}
    {% if user.is_authenticated %}
    <p>Tu usuario no está logueado. Por favor, vuelve a iniciar sesión.</p>
    {% else %}
    {% endif %}
    {% endif %}
   </div>
  </div>
 </div>
</div>
</div>
{% endblock %}
{% block javascript %}
<script>
 {% if not user.is_authenticated %}
 $("ul.nav.navbar-nav.navbar-right").css("display","none");
 {% endif %}
</script>
{% endblock %}
```

Fichero navbar.html

```
<nav class="navbar navbar-inverse navbar-fixed-top">
<div class="container-fluid">
 <div class="navbar-header">
 {% if request.user.is_authenticated %}
  <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
target="#navbar" aria-expanded="false" aria-controls="navbar">
   <span class="icon-bar"></span>
   <span class="icon-bar"></span>
   <span class="icon-bar"></span>
  </button>
  {% endif %}
  <a class="navbar-brand" href="#">Biosensor.io</a>
 </div>
 <div id="navbar" class="navbar-collapse collapse">
  <ul class="nav navbar-nav navbar-right">
   <li>
    <a href="#">
     <span class="glyphicon glyphicon-user"></span>
     {% if request.user.is_authenticated %} {{ user }} {% endif %}
    </a>
   </li>
   <li>
    <a href="/logout">
     <span class="glyphicon glyphicon-log-out"></span> Salir
    </a>
   </li>
   {%  for experimento in experimentos|slice:":5" %}
   <li class="nav hidden-sm hidden-md hidden-lg{% if seleccionado.id  == experimento.id
%} active{% endif %}">
    <a href="/experimento/{{experimento.id}}">{{ experimento.nombre }}</a>
   </li>
   {% endfor %}
  </ul>
 </ul>
</div>
</div>undefined</nav>
```

Fichero lista-experimentos.html

```
<ul class="nav nav-sidebar">
{% for experimento in experimentos %}
<li{% if seleccionado.id  == experimento.id %} class="active"{% endif %}>
 <a href="/experimento/{{experimento.id}}">{{ experimento.nombre }}</a>
</li>
{% endfor %}
</ul>
```

Fichero configuracion-experimento.html

```
<tbody>
<tr>
 <td>Bomba peristáltica</td>
 <td>Velocidad:
```

```
  <b>{{ seleccionado.bomba_peristaltica_velocidad }}%</b>
  <br>Sentido de giro:
  <b>{{ seleccionado.bomba_peristaltica_sentido }}</b>
 </td>
 </tr>
 <td>Bombas impulsionales</td>
 <td>Canal 1:
  <b>{{ seleccionado.bomba_impulsional_ch1 }}</b> µL
  <br> Canal 2:
  <b>{{ seleccionado.bomba_impulsional_ch2 }}</b> µL
 </td>
</tr>
<tr>
 <td>Selectores de viales</td>
 <td>Canal 1: Vial
  <b>{{ seleccionado.selector_viales_ch1 }}</b>
  <br> Canal 2: Vial
  <b>{{ seleccionado.selector_viales_ch2 }}</b>
 </td>
</tr>
<tr>
 <td>Válvulas de inyección</td>
 <td>Canal 1: Posición
  <b>{{ seleccionado.valvula_inyeccion_ch1 }}</b>
  <br>Canal 2: Posición
  <b>{{ seleccionado.valvula_inyeccion_ch2 }}</b>
 </td>
</tr>
<tr>
 <td>Láser</td>
 <td>
  <b>{{ seleccionado.laser|yesno:"Encendido,Apagado" }}
  <b>
 </td>
</tr>
</tbody>
```

Fichero dash.html

```
{% include 'base.html' %}

<!DOCTYPE html>
<html lang="en">
<body>
 {% include 'navbar.html' %}
 <div class="container-fluid">
  <div class="row-fluid">
   <div class="col-sm-3 col-md-2 sidebar" role="navigation">
    <h3 class="sub-header">Experimentos</h3>
    {% include 'lista-experimentos.html' %}
   </div>
   <div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
    <h2 class="sub-header">Resultados</h2>
    <div class="container-fluid">
    {% include 'graficas.html' %}
    </div>
    <h2 class="sub-header">Detalles</h2>
    <p>{{ seleccionado.descripcion }}</p>
    <div class="table-responsive">
     <table class="table table-striped">
```

```
      <thead>
       <tr>
        <th>
         <h3>Componente</h3>
        </th>
        <th>
         <h3>Configuracion</h3>
        </th>
       </tr>
      </thead>
      {% include 'configuracion-experimento.html' %}
     </table>
    </div>
   </div>
  </div>
 </div>
</body>
</html>
```

Fichero graficas.html

```
<div class="row">
<div class="col-sm-6">
 <div id="chart_div_ca" style="width=100%; height=100%;"></div>
 <div id="control_div_ca"></div>
</div>
<div class="col-sm-6">
 <div id="chart_div_ccq" style="width=100%; height=100%;"></div>
 <div id="control_div_ccq"></div>
</div>
</div>
<script type="text/javascript">

 function drawCurveA() {

   var data = new google.visualization.DataTable();
   data.addColumn('number', 'X');
   data.addColumn('number', 'Canal 1');
   data.addColumn('number', 'Canal 2');
   data.addRows([
     {% for muestra_ca in muestras_ca %}
     [{{ muestra_ca.angulo }}, {{ muestra_ca.ch1 }}, {{ muestra_ca.ch2 }}],
     {% endfor %}
   ]);

   var chart = new google.visualization.ChartWrapper({
       chartType: 'LineChart',
       containerId: 'chart_div_ca',
       options: {
         title: 'Curva Angular',
           titleTextStyle: {
             color: 'black',
             fontName: 'sans-serif',
             fontSize: '18',
             bold:'false'
           },
           height: 400,
           hAxis: {
             title: 'Ángulo',
             format: 'decimal',
             gridlines: {color: '#EEE', count: 10}
```

```
            },
            vAxis: {
              title: 'Reflectividad',
              format: 'decimal',
              gridlines: {color: '#EEE', count: 10}
            },
            series: {
              1: {curveType: 'function'}
            },
            legend: { position: 'top' },
            explorer: { actions: ['dragToZoom', 'rightClickToReset'] },
            chartArea: {
                width: '85%'
            }
        }
    });

    var control = new google.visualization.ControlWrapper({
        controlType: 'ChartRangeFilter',
        containerId: 'control_div_ca',
        options: {
            filterColumnIndex: 0,
            ui: {
                chartOptions: {
                    height: 50,
                    chartArea: {
                        width: '85%'
                    }
                }
            }
        }
    });

    var dashboard = new
google.visualization.Dashboard(document.querySelector('#dashboard_div'));
    dashboard.bind([control], [chart]);
    dashboard.draw(data);
 }

 function drawCurveCQ() {

    var data = new google.visualization.DataTable();
    data.addColumn('number', 'X');
    data.addColumn('number', 'Canal 1');
    data.addColumn('number', 'Canal 2');
    data.addRows([
      {% for muestra_ccq in muestras_ccq %}
         [{{ muestra_ccq.tiempo }}, {{ muestra_ccq.ch1 }}, {{ muestra_ccq.ch2 }}],
      {% endfor %}
    ]);

    var chart = new google.visualization.ChartWrapper({
        chartType: 'LineChart',
        containerId: 'chart_div_ccq',
        options: {
          title: 'Curva Cinética Química',
            titleTextStyle: {
              color: 'black',
              fontName: 'sans-serif',
              fontSize: '18',
              bold:'false'
            },
            height: 400,
            hAxis: {
              title: 'Tiempo',
```

```
            format: 'decimal',
            gridlines: {color: '#EEE', count: 10}
          },
          vAxis: {
            title: 'Reflectividad',
            format: 'decimal',
            gridlines: {color: '#EEE', count: 10}
          },
          series: {
            1: {curveType: 'function'}
          },
          legend: { position: 'top' },
          explorer: { actions: ['dragToZoom', 'rightClickToReset'] },
          chartArea: {
              width: '85%'
          }
      }
   });

   var control = new google.visualization.ControlWrapper({
       controlType: 'ChartRangeFilter',
       containerId: 'control_div_ccq',
       options: {
           filterColumnIndex: 0,
           ui: {
               chartOptions: {
                   height: 50,
                   chartArea: {
                       width: '85%'
                   }
               }
           }
       }
   });

   var dashboard = new
google.visualization.Dashboard(document.querySelector('#dashboard_div'));
   dashboard.bind([control], [chart]);
   dashboard.draw(data);
 }

 google.charts.load('visualization','1' ,{packages: ['corechart','controls']});
 google.charts.setOnLoadCallback(drawCurveA);
 google.charts.setOnLoadCallback(drawCurveCQ);

 $(document).ready(function () {
     $(window).resize(function(){
       drawCurveA();
       drawCurveCQ();
     });
 });

</script>
```