Huella digital de navegadores

Browser Fingerprinting

Por
Adrián Agudo García-Heras, Jesús Martín González, Rubén Peña García,
Samuel Solo de Zaldívar Barbero

UNIVERSIDAD COMPLUTENSE
MADRID

Trabajo de fin de grado
Grado en Ingeniería Informática
Facultad de Informática

Directores
Enrique Martín Martín, Adrián Riesco Rodríguez

Madrid, 2019–2020

Resumen

El proyecto consiste en el desarrollo de una aplicación web que recopilará la información del navegador donde se ejecuta y obtendrá su huella digital. Esto se conseguirá mediante distintas técnicas que permiten captar en su práctica totalidad los datos del dispositivo, y el conjunto de estos permitirá elaborar y singularizar un perfil. Los distintos datos obtenidos se almacenarán en una base de datos y se hará saber al usuario en cuestión si su perfil es único o no. A su vez, se informará al usuario sobre cuánto de único es en referencia a cada dato de su dispositivo y respecto al global de los datos ya almacenados en el sistema. El proyecto cuenta con la licencia Apache versión 2.0.

Puede acceder respectivamente a la aplicación web y a todos los recursos del proyecto en los siguientes enlaces:

http://browserfingerprinting.educationhost.cloud/

https://github.com/rbn423/browserFingerprinting/

Palabras clave

Huella digital, navegador, canvas, cookies, JavaScript

Abstract

The project consists of developing a web application that collects the information from the browser and gets its fingerprint. This will be achieved through different techniques that permit the capture of practically all the data of the device, and that data set will allow us to create and singularize a profile. The collected data will be stored in a database and it will let the user know whether his profile is unique or not. In turn, the user will be informed about how much unique it is in reference to each data of his device and with respect to the global data already stored in the system. The project is licensed under the Apache License version 2.0.

You can access respectively the web application and all the resources of the project in the following links:

http://browserfingerprinting.educationhost.cloud/

https://github.com/rbn423/browserFingerprinting/

Keywords

Fingerprint, browser, canvas, cookies, JavaScript

Índice general

Capítulo 1

Introducción

La huella digital es la recopilación sistemática sobre un determinado dispositivo con la finalidad de identificarlo, singularizarlo y perfilarlo. Este conjunto de datos permite prácticamente, de forma unívoca, identificar dicho terminal y en cuestión a la persona o grupo de personas que puedan estar usándolo. Por lo general, los terminales como los teléfonos móviles, tabletas, portátiles y ordenadores de sobremesa, son usados por una sola persona y por ello, podemos asumir que los datos recopilados de cierto dispositivo pertenecen a una persona en concreto [1].

Actualmente, las aplicaciones web prestan servicios de forma totalmente gratuita a cambio de los datos que recopilan de los usuarios. En la mayoría de casos esta información se rentabiliza a través de servicios de marketing y publicidad que desean vender un producto o servicio determinado. Por ello, estos servicios necesitan perfilar al usuario en cuestión para así ofrecerle un producto que les pueda interesar. Las entidades usan estos mecanismos de recopilación de datos con todos los terminales que se conecten a sus servidores, para así poder hacer un seguimiento del usuario y elaborar un perfil.

La técnica de seguimiento más conocida es mediante *cookies*, las cuales se almacenan en el propio dispositivo y que luego son usadas para estudiar la estadística de uso de la aplicación web y para mejorar la experiencia del usuario. Es común encontrar cláusulas de privacidad que permiten al usuario dar o no su consentimiento para el uso de estas. Algunos exploradores ofrecen la posibilidad de deshabilitar su uso y ciertos antivirus realizan borrados periódicos de los ficheros de rastreo, pero la mayoría de aplicaciones web no dejan acceder a sus servicios, o la totalidad de los mismos, si el usuario no permite su uso. El uso de las técnicas de huella digital permite que, en el caso de que las *cookies* sean eliminadas, no se pierda la trazabilidad sobre el usuario. Tecnologías como JavaScript, Flash y Microsoft Silverlight facilitan la implementación de métodos para recoger información muy concreta del dispositivo, como es el tamaño de la pantalla o la versión del sistema operativo. La combinación de estas características permite perfilar al usuario e identificarlo.

1.1. Objetivos

El principal objetivo de este proyecto es crear una aplicación web que permita recopilar los datos de un terminal, aplicando las distintas técnicas de *fingerprinting*, y elaborar un perfil

en base a ellos. En concreto tenemos las siguientes metas:

- Crear una aplicación escalable, para que en el futuro se pueda continuar desarrollando según se requiera.
- Investigar la variedad de métodos de huella digital e implementarlos.
- Almacenar la información recogida, de la cual se hará un perfilado y se guardará en una base de datos.

1.2. Plan de trabajo

Para el desarrollo del proyecto se ha fijado una planificación de tal forma que se cubre las fases de investigación, análisis de requisitos, implementación, testing y la memoria.

En la figura 1.1 se puede observar el diagrama de Gantt. Tras la ampliación de los plazos de entrega, se actualizaron las tareas posteriores y sus respectivos tiempos. Las tareas desglosadas consisten en:

- **Investigación**: Recogida de información acerca de las distintas técnicas existentes para obtener información de los dispositivos. Esto incluye ciertas páginas que determinan la huella digital del navegador [2].
- **Análisis de requisitos**: Valoración sobre qué lenguajes de programación y herramientas vamos a usar para el desarrollo del proyecto.
- **Implementación**: Puesta en marcha del proyecto. En esta etapa del desarrollo se implementaría la aplicación web con su respectiva base de datos.
- **Pruebas**: Comprobación del funcionamiento de la aplicación y resolución de las posibles incidencias.
- **Memoria**: Documento donde se refleja el desarrollo y resultados del proyecto.

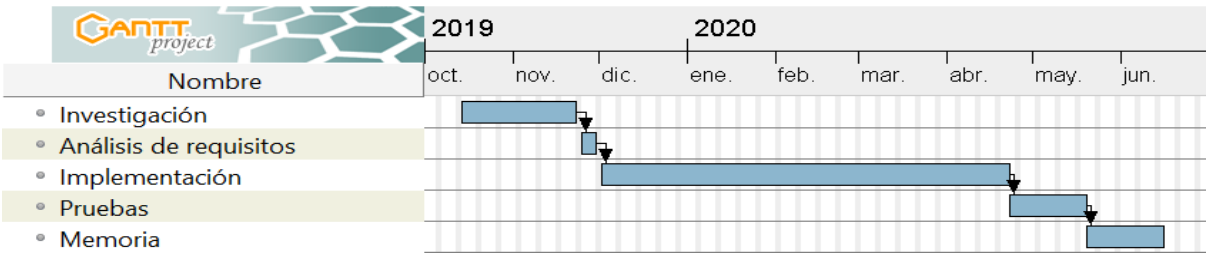


Figura 1.1: Diagrama de Gantt

1.3. Organización de la memoria

El documento incluye los siguientes capítulos:

- **Preliminares**: Situación por la que empieza el proyecto y presentación de las distintas tecnologías y herramientas que se han explorado.
- **Fuentes de información**: Exposición de las principales fuentes de datos utilizadas para el *fingerprinting*.
- **Diseño e implementación**: Descripción sobre cómo funciona la aplicación.
- **Uso del sistema**: Breve demostración donde se observa el uso la aplicación.
- **Contribución individual**: Contribución detallada de cada integrante del proyecto.
- **Conclusiones**: Discusión razonada de los resultados obtenidos y el futuro del proyecto.

Capítulo 2

Introduction

Fingerprint is the systematic compilation on a given device in order to identify it, singularize it and profile it. This data set practically allows, univocally, to identify that device and the person or group of people who may be using it. In general, devices such as mobile phones, tablets, laptops and desktop computers are used by a single person and therefore, we can assume that the data collected from a certain device belongs to a specific person [1].

Currently, web applications provide services completely free of charge in exchange for the data they collect from users. In most cases, this information is made profitable through marketing and advertising services that desire to sell a specific product or service. Therefore, these services need to profile the user in order to offer them a product that may be of their interest. Entities use these data compilation mechanisms with all devices that connect to their servers, in order to monitor the user and create a profile.

The best-known tracking technique is by cookies, which are stored on the device itself and then used to study the statistics of web application usage and to improve the user experience. It is common to find privacy clauses that allow the user to give or not consent to their use. Some browsers offer the possibility of disabling their use and some antivirus perform periodic deletions of the trace files, but most web applications do not allow access to their services, or all of them, if the user does not allow their use. Using fingerprint techniques allows that, in the case that the *cookies* are deleted, the traceability on the user is not lost. Technologies such as JavaScript, Flash and Microsoft Silverlight, facilitate the implementation of methods to collect very specific information from the device, such as the screen size or the operating system version. The combination of these characteristics allows the user to be profiled and identified.

2.1. Objectives

The main objective of this project is to create a web application that allows the collection of data from a device, applying the different fingerprinting techniques, and to elaborate a profile based on them. Specifically, we have the following goals:

- Create a scalable application, so that it can be further developed as required in the future.
- Research the variety of fingerprint methods and implement them.

- Store the collected information, from which will be profiled and saved in a database.

2.2. Workplan

For the development of the project, a plan has been established in such a way that the research, requirements analysis, implementation, testing and report phases are covered.

The Figure 2.1 shows the Gantt chart. After the deadlines were extended, subsequent tasks and their respective times were updated. The broken down tasks consist of:

- **Research**: Information gathering about the various available techniques to obtain data from the devices. This includes certain pages that determine the browser's fingerprint [2].
- **Requirements analysis**: Assessment about which programming language and tools we are going to use for the development of the project.
- **Implementation**: Start up of the project. At this stage of development the web application would be implemented with its respective database.
- **Testing**: Functional verification and testing of the application and resolution of possible incidents.
- **Report**: Reasoned discussion of the results obtained and the future of the project.

2.3. Content

The document includes the following chapters:

- **Preliminary**: Starting point and presentation of the different technologies and tools that have been explored.
- **Information sources**: Briefing of the main data sources used for fingerprinting.
- **Design and implementation**: Description on how the application works.

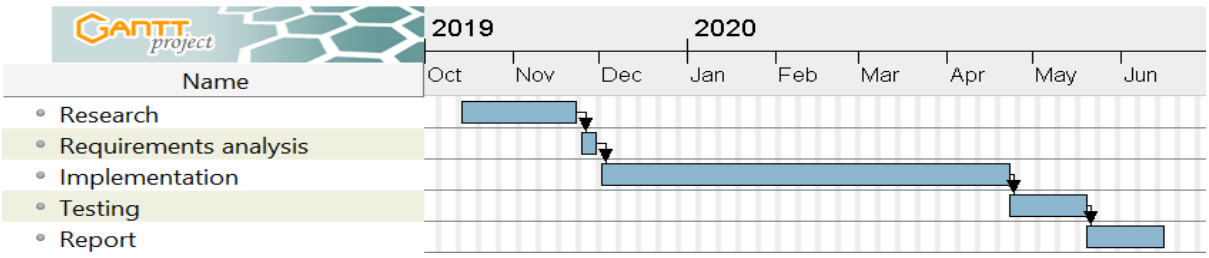


Figura 2.1: Gantt chart

- **System use**: Brief demonstration where the use of the application is observed.
- **Individual contribution**: Detailed contribution of each member of the project.
- **Conclusions**: The obtained results are reasonably discussed.

Capítulo 3

Preliminares

En este capítulo vamos a presentar las diferentes huellas digitales que se pueden extraer de los navegadores web. Discutiremos las implicaciones que esto suscita y explicaremos las distantas técnicas de extracción y evasión que existen. También detallaremos las tecnologías que se han utilizado en nuestro trabajo y, por último, mencionaremos otras herramientas existentes que son similares a nuestro proyecto.

3.1. Huella digital de navegadores

Cuando navegamos por internet, es posible identificar características que nos permitan reconocer si un determinado usuario ha visitado nuestro sitio. El método más común para detectarlo es mediante el uso de *cookies*. Este dato nos permite establecer una sesión con el visitante de la página y poder guardar así un estado de la conexión. Además, este identificador de sesión nos permite almacenar las configuraciones personalizadas para esa sesión específica. De esta forma controlamos en todo momento el rastro de nuestros usuarios. Ese trozo de información, llamado *HTTP cookie* [3], se genera en el servidor y se intercambia al establecer la conexión. Esto es, la *cookie* se almacena también en el navegador del cliente.

A pesar de ello, esta técnica no es suficiente. El usuario puede borrar las *cookies* de su navegador web. En este caso el servidor no le podrá recordar, puesto que la sesión previamente establecida no se puede recuperar y habría que establecer una nueva. Por consiguiente, necesitamos otro enfoque distinto para poder identificar a nuestro usuario con una característica menos volátil.

Otro de los parámetros que históricamente se ha empleado para identificar a los internautas en la red ha sido la dirección IP pública origen. Se trata de un identificador utilizado en el protocolo de red TCP/IP, concretamente en la capa de red. Nos permite determinar el origen de las comunicaciones, ya que esta dirección es única. Por tanto, sería razonable obtener este atributo para poder identificar al usuario que está navegando en nuestra página.

Esto tampoco es tan sencillo. Hoy en día la mayoría de las direcciones IP suelen ser dinámicas y son asignadas por nuestro *ISP* o Proveedor de Servicios de Internet. Esto quiere decir que cada cierto intervalo de tiempo, el *ISP* va a ir asignando una dirección IP pública a nuestra interfaz de red. En consecuencia, tampoco nos va a ser posible utilizarlo como atributo

identificativo, pues para un mismo usuario este valor va a ir variando a lo largo del tiempo.

Además de la asignación dinámica de direcciones IP, otro problema asociado a considerar la dirección IP como atributo identificativo es la utilización de técnicas de CG-NAT por algunos *ISP*. Como resultado, distintos dispositivos fuera de nuestra red doméstica podrían compartir la misma dirección IP pública, ya que el *ISP* estaría conectando muchos dispositivos de su red como si de una red interna se tratase. De esta forma nosotros no seríamos capaces de identificar nuestro dispositivo objetivo.

Por último, también podríamos considerar posibles intrusiones en una red interna. La dirección IP que resolveríamos pertenecería a un tercer usuario víctima que puede no tener nada que ver con el auténtico dispositivo visitante.

Tanto las direcciones IP públicas como el seguimiento a través de *cookies* fueron consideradas las huellas digitales más fiables durante varios años. Sin embargo, existen otros enfoques que nos permiten llegar a mejores conclusiones a partir de otras técnicas. La alternativa más conocida a las expuestas anteriormente es *browser fingerprinting* o huella digital del navegador web.

El método de *browser fingerprinting* [4] consiste en recopilar información del navegador web en función de las características y configuraciones de este. Esta nueva óptica de identificar al usuario a través de las diversas tecnologías incluidas en los navegadores sin *cookies* [5], nos permite afinar más y ser más minuciosos en nuestro objetivo de identificar a los usuarios y monitorizar la actividad que estos realizan en una determinada página web. Es posible encontrar más información sobre la efectividad del *browser fingerprinting* en [6, 7, 8].

En las conclusiones encontramos una paradoja: si bien los primeros estudios aseguraban que la efectividad de este método estaba por encima del 80 % de precisión en cuanto a huellas digitales únicas [6, 7], estudios posteriores no han podido reproducir estos mismos resultados, obteniendo solo un 33.6 % [8].

La explicación a este fenómeno proviene de la relación entre el desarrollo de recursos efectivos de recopilación de huella digital en navegadores web con el desarrollo de las técnicas de mitigación de las mismas.

Cuantos más procedimientos diferentes de *browser fingerprinting* se emplean, más probable es obtener una huella identificativa de las características del navegador web. Hoy en día, el uso de ciertas métodos aislados no resulta del todo identificativo. Entonces no es necesario hallar un atributo único, sino que es suficiente con encontrar una combinación de atributos que resulte única para poder identificar a un usuario.

3.1.1. Impacto: peligros y casos de uso

En esta sección analizaremos las implicaciones que tiene el uso de la identificación de la huella digital en navegadores. Expondremos los casos de usos más frecuentes y los peligros que puede suponer el uso de este recurso. Asimismo, comentaremos algunas posibles implicaciones legales.

Al tratarse de un sistema tan efectivo, cabe preguntarse si puede ocasionar a su vez algunos peligros. Sin duda, el principal inconveniente que nos viene a la cabeza es el riesgo que

supone para la privacidad. Este método se puede realizar de forma transparente al usuario sin que se dé cuenta ni haga falta pedirle ningún tipo de permiso [9].

La parte legal supone también otro desafío. En Europa, existe el Reglamento General de Protección de Datos (GDPR) que restringe la recopilación de datos personales a no ser que se haga de conformidad a las seis formas legítimas para el tratamiento de datos personales. Estas seis formas son las siguientes:

a). Bajo el consentimiento inequívoco del individuo.

b). Por interés vital del individuo.

c). Por interés público.

d). Por necesidad contractual.

e). En cumplimiento de obligaciones legales.

f). Por interés legítimo del responsable del tratamiento de datos.

La GDPR evita mencionar tecnologías específicas para poder estar al día del desarrollo tecnológico más allá de las huellas digitales y las *cookies* [10].

No obstante, en el reglamento de la GDPR aparecen una serie de considerandos no vinculantes que mencionan el hecho de que las *cookies* y otras huellas digitales pueden ser utilizadas para tanto para elaborar perfiles de personas físicas como para identificarlas. Esto se menciona concretamente en el considerando número 30 [11].

Por tanto, en la práctica, con informar al usuario del empleo de este método, sería suficiente para cumplir con la GDPR [12]. Sin embargo, aquellas corporaciones sin acuerdos con Europa podrán seguir recopilando los datos de huellas digitales sin tener en cuenta esta ley.

Por supuesto, el agente detrás del empleo este procedimiento va a contar con sus propias motivaciones. De todas maneras, los casos de uso principales suelen ser los siguientes:

- Realizar un perfil específico de clientes para adaptarse a sus necesidades. Una de las formas es rastrear de qué página viene el usuario y establecer si ese usuario ya ha visitado nuestro sitio anteriormente. En este caso no estamos necesariamente interesados en averiguar la identidad del usuario en sí, tan solo en realizar un perfil. Es el sistema más utilizado por los *data brokers* [13]. Este es el caso de uso fundamental de la mayoría servicios de publicidad de terceros.
- Entender la actividad del usuario a través de patrones de comportamiento. Analizando cómo es el flujo de interacción del usuario con nuestra página, podemos llegar a conclusiones muy valiosas. La tecnología de JavaScript es la más utilizada en este aspecto, ya que nos permite analizar la atención del usuario sobre la página (saber si la pestaña se encuentra activa, cuál es la posición del ratón, comprobar si nuestras sugerencias son de verdad atractivos e interesantes para el usuario). Se utiliza mucho en tiendas en línea y redes sociales como estrategia de mercadotecnia.
- Como consecuencia de los dos casos de uso, la huella digital del navegador puede determinar patrones de comportamiento. De esta forma es posible también precisar si la actividad web está siendo llevada a cabo por un *bot* [14].

3.1.2. Técnicas de *fingerprinting*

Podemos clasificar las distintas técnicas de detección de huellas digitales en navegadores web en dos grupos distintos:

- *Fingerprinting* pasivo: Se limita a explorar el contenido de la petición web sin ejecutar código de forma activa en el navegador del cliente. En este grupo incluiremos la recolección de campos de la cabecera HTTP. En el capítulo 4 explicaremos detalladamente cada uno de los elementos que conforman dicha cabecera.
- *Fingerprinting* activo: Se realiza ejecutando código JavaScript en el navegador web del cliente que está realizando la petición web. Entre todas las opciones posibles, el sistema operativo, la configuración del idioma, la resolución de pantalla, la detección de *plugins* y el lienzo suelen ser los atributos elegidos para realizar el método activo de identificación de huella digital. En el capítulo 4 se explicará detalladamente cada uno de los elementos JavaScript que hemos obtenido para poder perfilar los navegadores web en nuestro proyecto.

También podríamos clasificar los distintos sistemas de *browser fingerprinting* teniendo en cuenta el tipo de información a obtener. Por una parte, podríamos obtener información del *hardware* de la máquina directamente a través de la API de JavaScript. Algunos de estos atributos son: la memoria RAM, el tipo de procesador de la máquina, el nombre de la tarjeta gráfica, el número de núcleos de la CPU o el estado de la batería. Por otra parte, podemos obtener información del sistema operativo, configuraciones del navegados o fuentes y *códecs* de audio y vídeo instalados, entre otros. Todos estos elementos se obtienen del *software* del sistema del cliente. Entre todo este conjunto de técnicas, las más usadas son:

- Recopilar el *User-Agent* del dispositivo: Esto se puede realizar tanto de forma activa como de forma pasiva. En general, en este atributo se anuncia el nombre del navegador web que se está utilizando. Una vez lo conozcamos, sabemos las tecnologías soportadas y podemos profundizar más en nuestra detección de la huella digital.
- Obtener el lienzo o *Canvas*: La propiedad que tiene el lienzo es que una misma imagen o un mismo texto puede representarse de forma diferente en distintos clientes. Esto depende del número de píxeles del dispositivo, del motor del navegador web, de los formatos de compresión, del sistema operativo y de la unidad de procesamiento gráfico (GPU) que se utilice, principalmente. Se considera esta técnica como una de las más efectivas [15]. Si bien no garantiza la unicidad de un usuario, combinado con otros campos casi puede garantizar la identificación inequívoca del mismo.

Por último, queremos mencionar cuales son los últimos métodos de detección de huella digital en los navegadores que han llamado la atención tanto en estudios académicos como para la industria. Las ideas más disruptivas son:

- WebGL API: Es una API de JavaScript diseñada para renderizar gráficos 3D. Además, los componentes de WebGL pueden combinarse con otros elementos HTML, como el lienzo [16]. Al igual que ocurre con el lienzo, la granularidad de la huella digital dependerá estrechamente del *hardware* del dispositivo, concretamente de la GPU.

- Web Audio API [17]: Esta API está pensada para dar soporte a aplicaciones más avanzadas capaces de sintetizar audio, como controladores de sonido MIDI, y para mejorar la experiencia de usuario al jugar a videojuegos web. La idea que hay detrás de obtener una huella digital a través de este sistema es que las señales de audio son procesadas de formas distintas en cada navegador del cliente según el *hardware* y *software* del que disponga. Por eso, en una determinada máquina, los parámetros serán siempre los mismos.
- Utilización de los niveles de batería: HTML5 implementa la API *Battery Status* que permite extraer valores de carga y descarga de la batería. Si esto es utilizado por un *script* de terceros, pueden vincular esos niveles de batería con las visitas a determinadas web en un intervalo de tiempo. De esta forma se podría obtener la traza de navegación de un usuario [18].

3.1.3. Técnicas de evasión

Como indican [Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan y Claudia Diaz] en [15], "las opciones actuales de los usuarios para mitigar estas amenazas son limitadas, en parte debido a la dificultad de distinguir el rastreo no deseado del comportamiento benigno."

La mitigación más sencilla para aumentar la seguridad y privacidad de nuestro navegador es por supuesto eliminar funcionalidades. No obstante, elevar nuestro nivel de protección sin degradar la experiencia del usuario siempre es un desafío.

Como resultado, no existe una solución simple, sino que tan solo se pueden implementar medidas de mitigación [19].

- Modificar los ajustes del navegador. Cada navegador cuenta con unas opciones configurables. En la mayoría de ellos podemos habilitar funciones de anti-seguimiento.

  Es posible acceder a las configuraciones avanzadas de nuestros navegadores y deshabilitar funcionalidades y configuraciones que puedan dar demasiados detalles de nuestra huella digital. En Firefox bastaría con ingresar en el recurso about:config. Google Chrome cuenta con otra opción equivalente, chrome://flags/

  Otra opción es utilizar navegadores que ya vengan con configuraciones por defecto para evadir varias técnicas de detección de huella digital, como Tor Browser.

- Utilizar complementos y extensiones para el navegador que modifiquen el comportamiento del navegador [20]). Existe una gran diversidad de complementos que realizan esta labor. Algunos de los más conocidos son *NoScript* y *uMatrix*. Estos complementos permiten una configuración flexible por lo que es posible bloquear solo el código JavaScript y los elementos de Flash que deseemos, sin tener que prescindir de la funcionalidad completa. Es una solución de compromiso que busca el equilibro entre privacidad y funcionalidad.

  Además del bloqueo de elementos Flash y JavaScript, existen más complementos que modifican las cabeceras HTTP y que alteran algunas APIs de JavaScript para evitar el *fingerprinting*. De esta forma podemos falsear los datos que se van a recopilar sobre nosotros. Como resultado, tanto los métodos activos como pasivos serían mucho menos eficaces.

- Utilizar máquinas virtuales: Es posible virtualizar distintos sistemas operativos con sus respectivos navegadores. De esta forma mantener por separado nuestros distintos intereses, como puedan ser trabajo, ocio, educación, etc. y utilizar así cada perfil para un fin específico. Por supuesto, siempre podemos borrar la máquina virtual y crear otra nueva [21].

Todas estas medidas de mitigación son tan solo esfuerzos técnicos. Cambios en las tecnologías provocarían que las medidas que hoy son eficaces para evitar la huella digital mañana estén desfasadas y resulten inútiles. Por esta razón, sin una regulación efectiva en el campo del derecho, estos trucos y ajustes están destinados a fracasar a largo plazo.

3.2. Tecnologías empleadas

Existe un abanico muy amplio de tecnologías [22] relacionadas con la navegación web. Para dar una primera visión general, vamos a realizar una clasificación de las distintas tecnologías que hemos utilizado en las siguientes categorías:

- Tecnologías básicas:
  - HTML.
  - CSS.
  - HTTP.
- Lenguajes de programación y de descripción de datos:
  - JavaScript (AJAX y Web API).
  - PHP.
  - JSON.
- Backend:
  - XAMPP.
    - Apache.
    - MariaDB.
    - PhpMyAdmin.
  - Education Host.
- Otras tecnologías:
  - Google Charts.
  - Git.
  - PhpStorm.
  - LaTeX.

A continuación explicamos brevemente en qué consiste cada tecnología utilizada y porqué nos hemos decantado por usarla.

3.2.1. Tecnologías básicas

HTML

HTML significa Lenguaje de Marcado de Hipertextos (*HyperText Markup Language*). Es el lenguaje estandarizado por el *World Wide Web Consortium (W3C)* que define el código de una página web. Este lenguaje de marcado está formado de elementos o etiquetas que separan los distintos componentes de la página.

Usamos HTML5 en nuestro proyecto ya que necesitamos construir una página web y éste es el lenguaje estándar para realizarlo. HTML es el lenguaje más extendido. No es tan estricto como otros lenguajes de marcado similares, como XHTML, que no son retrocompatibles.

CSS

CSS significa Hojas de Estilo en Cascada (*Cascading Style Sheets*). Se utiliza para definir el diseño de la presentación de un lenguaje de marcado. Es otro de los lenguajes estandarizados por el *W3C*. Existen varias versiones. Nosotros utilizamos la versión más reciente, que es CSS3.

Utilizamos CSS en nuestro proyecto para colocar los elementos HTML de forma estática y darles el estilo de presentación que deseamos. Usar CSS resulta mucho más flexible que fijar el valor del estilo dentro la etiqueta HTML, haciendo nuestro código más sencillo de mantener.

HTTP

HTTP es Protocolo de Transferencia de Hipertexto (*Hypertext Transfer Protocol*). Es el protocolo que se emplea para el envío de documentos HTML. En la pila TCP/IP se encuentra en la capa de aplicación. HTTP es un protocolo sin estado y sigue un modelo de cliente-servidor.

En nuestro proyecto usamos la versión HTTP/1.1 , que es la que viene predeterminada en nuestro servidor. La ventaja que tenemos con esta versión es que el formato es texto plano, por tanto legible para el ser humano.

3.2.2. Lenguajes de programación y de descripción de datos

JavaScript (AJAX y Web API)

JavaScript es un lenguaje de programación multipropósito que está soportado por todos los navegadores web modernos. Aparece por primera vez en el navegador Netscape en 1995. Se puede utilizar tanto para ejecutar código en el navegador del cliente como en funcionalidades del servidor. A través de JavaScript también es posible acceder a multitud de APIs que existen en nuestro navegador web. Estas permiten la configuración de ajustes y el acceso a varias funcionalidades.

En nuestro proyecto utilizamos WebAPI para recolectar los datos de la huella digital del

navegador del cliente. Algunas de las WebAPIs que más utilizamos son: window y document (pertenecientes a Document Object Model (DOM)), navigator, screen y canvas.

También usamos JavaScript para colocar elementos HTML de forma dinámica y asíncrona modificando el árbol DOM HTML. Esto nos permite cargar los elementos sin actualizar la página completamente ejecutando el código en el navegador web del cliente.

PHP

PHP (proviene del acrónimo recursivo *PHP: Hypertext Preprocessor*) es otro lenguaje de programación multipropósito. Es el lenguaje de desarrollo web por excelencia. La primera referencia que se tiene de PHP data también de 1995. En nuestro caso lo empleamos en el lado del servidor. Una característica muy útil es que podemos combinar código PHP con JSON y HTML.

Elegimos PHP porque se trata de un lenguaje que ya conocíamos todos los miembros del equipo. Además nos iba a facilitar el uso del software con el que ya teníamos en mente trabajar.

JSON

Las siglas JSON significan Notación de Objeto de JavaScript (*JavaScript Object Notation*). Es un formato de representación de texto. Se caracteriza por contener colecciones de datos no ordenadas. Estas colecciones tienen una estructura de pares <clave>:<valor>.

Utilizamos JSON en nuestro proyecto para intercambiar objetos desde la lógica a las vistas. En ellos incluimos toda la información que queremos mostrar al usuario.

3.2.3. *Backend*

En nuestro servidor hemos optado por instalar XAMPP [23]. Consiste en un paquete de herramientas para combinar las principales tecnologías que se utilizan en un servidor web. Hemos elegido este paquete de *software* debido a la facilidades que brinda tanto para instalar como para usarse con el sistema operativo Microsoft Windows. Los componentes que hemos utilizado en XAMPP son los siguientes:

Apache

Es el servidor web más popular. Cuenta con una perfecta integración con los el lenguaje PHP. Se trata de un servidor con el que ya habíamos trabajado anteriormente así que tomamos la decisión de utilizarlo para poder comenzar el desarrollo de nuestra aplicación rápidamente. También consideramos otras opciones que finalmente descartamos. Esto lo explicaremos en la sección 3.2.5.

MaríaDB

Se trata de un sencillo sistema gestor de base de datos relacionales. Es una bifurcación del proyecto MySQL, que fue adquirido por Sun Microsystems en 2009. El proyecto de MariaDB trata de dar continuación al desarrollo abierto de dicha tecnología a través de su comunidad.

En nuestro proyecto se usa para albergar la huella digital que recopilamos de los navegadores web que conectan con nuestra aplicación. Como en el punto anterior, también consideramos otro tipo de bases de datos que finalmente fueron descartadas.

PhpMyAdmin

Es una interfaz web para administrar la base de datos. Está escrita en PHP y soporta bases de datos tanto MariaDB como MySQL.

Está contenida dentro de la instalación de XAMPP. Nos resulta más cómodo observar el contenido de nuestra base de datos de un vistazo a través de una interfaz web y por eso utilizamos esta herramienta y no un cliente CLI.

Education Host

Education Host [1] es un proveedor que ofrece servicios de alojamiento web. Elegimos este servicio ya trae de serie los servicios Apache, MySQL y PhpMyAdmin, por lo que no tenemos que preocuparnos por la instalación ni el mantenimiento de los mismos. Como MariaDB es totalmente compatible con MySQL, nuestra aplicación funciona perfectamente. Por si no fuera suficiente, este servicio está incluido en el pack de estudiante de Github y podemos utilizarlo de forma gratuita.

3.2.4. Otras tecnologías

Además de las tecnologías que hemos usado relacionadas con los navegadores, mencionamos el resto de tecnologías que hemos incorporado a nuestro proyecto:

Google Charts

Utilizamos la herramienta Google Charts [24] para representar estadísticas sobre la información que hemos almacenado en nuestra base de datos. A través de gráficas de tartas mostramos porcentajes de los atributos más comunes. Este herramienta la importamos a través de código JavaScript.

[1] https://educationhost.co.uk/

Git

Es un *software* de control de versiones. Nos permite organizar nuestro flujo de trabajo y guardar un historial de cambios para saber el motivo y la persona que realizó las nuevas mejoras. Además nos hace más sencilla la coordinación entre todo el equipo en el desarrollo de software.

El código lo alojamos en la plataforma Github. Elegimos Github porque la licencia de estudiante ofrece una serie de extras de forma gratuita. Uno de estos extras es la opción de utilizar repositorios privados, en los que estábamos interesados. Igualmente, cuenta con una aplicación de escritorio que nos facilita la revisión de código antes de adoptar los cambios definitivos.

PhpStorm

Se trata de un entorno de desarrollo integrado (IDE) desarrollado por la compañía Jetbrains. Este IDE nos sorprendió por la cantidad de ayudas que nos ofrece al trabajar con PHP, JavaScript, HTML y CSS. Soporta todos esos lenguajes y formatos, además de contar con opciones de autocompletado de código y detección de errores sintácticos.

Este entorno comercial es de pago. Sin embargo, cuenta con una licencia de estudiante que permite su uso gratuito.

LaTeX

LaTeX es un sistema para componer textos. Se suele utilizar en textos técnicos y de divulgación científica. Su característica principal es que permite escribir documentos centrándose en el contenido del mismo, sin tener que preocuparse por los detalles del formato. Otra ventaja que ofrece es que la salida que ofrece es invariable, por lo que no depende del dispositivo con el que estamos trabajando.

Usamos LaTeX para componer el documento de la memoria del proyecto desde la página Overleaf. Nos decantamos por esta opción porque nos permite prescindir de descargar *software* adicional en nuestra máquina. Asimismo, cuenta con diversos compiladores online, por lo que no tenemos que preocuparnos de los conflictos con las dependencias de los paquetes.

3.2.5. Tecnologías consideradas y finalmente descartadas

Antes de elegir estuvimos probando con diversas tecnologías que finalmente fuimos descartando hasta quedarnos con las que hemos mencionado en el apartado anterior. Realizamos aquí un pequeño resumen.

Django

Django es un framework de desarrollo web. Una de las ventajas es que respeta el patrón de diseño Modelo-Vista-Controlador (MVC). Por otra parte, el lenguaje de programación Pyt-

hon es usado en todas las partes del *framework* y no todos los integrantes del equipo tenemos el dominio suficiente sobre él. Finalmente consideramos que acostumbrarnos al lenguaje iba a resultar complicado y lo descartamos al estar más familiarizados con PHP.

Node.js

Es un entorno construido sobre JavaScript, normalmente utilizado en el lado del servidor web. La principal ventaja que veíamos era la futura integración que podíamos tener más adelante con el código que recopila la información de la huella digital de los navegadores. Otra virtud a tener en cuenta es que con muy pocas líneas de código podíamos tener un servidor web escuchando y listo para producción. En comparación con Django que te obligaba a crear todos los componentes necesarios para respetar el modelo MVC, esto era mucho más rápido y sencillo. La complicación que le vimos era entender a fondo el lenguaje JavaScript, ya que al comienzo del proyecto nos sentíamos mucho más cómodos trabajando con PHP y Apache en el lado del servidor que con JavaScript y Node.js.

Nginx

Es un servidor web. En cuanto a las características pensamos que es muy similar a Apache, por lo que no veíamos en Nginx ventajas considerables. Además, Apache ya viene incluido en el pack de herramientas de XAMPP.

MongoDB

Nos planteamos la posibilidad de utilizar una base de datos NoSQL como MongoDB. La ventaja que le veíamos era el poder añadir atributos a medida que nos los íbamos encontrando, ya que en un primer momento al estudiar la diversidad de navegadores y configuraciones de los mismos, pensamos que podíamos encontrarnos con datos no estructurados.

Sin embargo, vimos que no nos hacía falta ya que podíamos diseñar un esquema fijo relacional con tablas SQL. Además, la cantidad a datos a almacenar no tiene nada que ver con el entorno Big Data y MariaDB se adaptaba mejor a nuestras necesidades.

3.3. Herramientas similares

Existen diversas herramientas similares a la que hemos desarrollado en nuestro proyecto, ya que todas realizan métodos de *browser fingerprinting*. Sin embargo, las técnicas realizadas y los objetivos de cada plataforma son muy dispares. Hemos reunido la siguiente lista de utilidades web que realizan *browser fingerprinting*.

AmIUnique

AmIUnique [2] es una plataforma que recopila información de los diversos navegadores que visitan la web. De esta forma, son capaces de proveer estadísticas de uso. Entre sus objetivos encontramos:

- Indicar si nuestra huella digital es única y si podemos llegar a ser rastreables.
- Mostrar el contenido de los parámetros más utilizados en *browser fingerprinting*. Para cada parámetro, nos indica el porcentaje de registros similares encontrados en su base de datos.
- Estadísticas de uso globales: la herramienta nos muestra gráficos y porcentajes de los navegadores y sistemas operativos.
- Estadísticas de uso para ayudar a los desarrolladores para ayudarles ofrecer una experiencia optimizada para cada plataforma.

Los desarrolladores de esta plataforma también han diseñado una extensión de navegador para alertarnos de cuando nuestra huella digital ha cambiado.

BrowserSpy.dk

Encontramos aquí otra página diferente que nos permite averiguar la información que el navegador revela sobre nuestro dispositivo.

Cuenta con un panel a la izquierda donde se hallan los distintos elementos que *BrowserSpy* [3] es capaz de obtener de nosotros. Pinchando sobre cada uno de ellos obtenemos el valor que se utiliza para rastrear nuestra huella digital. Destaca en especial por recopilar información sobre el soporte de AJAX en nuestro navegador. También tiene un método sofisticado de obtener información de las fuentes, incorporando tecnologías Java y Flash.

BrowserLeaks

BrowserLeaks [4] es un portal web muy similar al anterior. En su página principal nos explica detalladamente cuales son los atributos que utiliza para obtener la huella digital del navegador web.

Cuenta con una serie de opciones que la diferencian del resto, como el uso de la WebAPI *Geolocation*, las distintas opciones SSL/TLS que soporta el navegador y la obtención de datos sobre plugins Java, Silverlight y Flash.

[2] https://amiunique.org/
[3] http://browserspy.dk/
[4] https://browserleaks.com/

Panopticlick

Se trata de un proyecto de investigación de la *Electronic Frontier Foundation* (EFF). Está desarrollado a partir la biblioteca *Fingerprint2* [25] e incorpora partes de *BrowserSpy.dk* para la detección de fuentes.

El objetivo es concienciar a los usuarios de que son susceptibles a ser perfilados a través de su navegación. Para proteger nuestra privacidad, *Panopticlick* [5] recomienda la instalación de una extensión para el navegador, llamada *Privacy Badger*, la cual ha sido también desarrollada por la EFF. Está disponible para Google Chrome, Mozilla Firefox y Opera Browser.

Audio Context Fingerprint TestPage

Este proyecto forma parte del *Princeton Web Transparency and Accountability Project*, de la Universidad de Princeton [6].

Al contrario que las plataformas anteriores, las cuales realizaban una comprobación más o menos completa de casi todas las habilidades conocidas de *browser fingerprinting*, esta se focaliza tan solo es dos aspectos concretos.

El objetivo es obtener la huella digital del navegador web analizando los valores de audio y detección de fuentes. En cuanto al audio, analiza las propiedades *AudioContext*, *DynamicsCompressor* y *OscillatorNode*. También es posible obtener información de las distintas fuentes instaladas en el cliente, información del lienzo y detección de las fuentes de *Flash*.

DeviceInfo

Device Info [7] es una utilidad que nos permite comprobar la privacidad de nuestros ajustes en el navegador.

El diseño de la página es muy austero; pero, a pesar de ello, cumple perfectamente con su cometido. Posiblemente sea la página más completa de todas en cuanto a la diversidad de técnicas que realiza para obtener la información del usuario.

Por contrapartida, no proporciona consejos para aumentar la privacidad de nuestro navegador web. Asimismo, tampoco nos devuelve los datos en un contexto en relación con otros usuarios.

[5] https://panopticlick.eff.org/
[6] https://audiofingerprint.openwpm.com/
[7] https://www.deviceinfo.me/

Capítulo 4

Fuentes de información

Las fuentes de información son los elementos que definen la huella digital del usuario. A continuación vamos a indicar cuales son estos elementos y vamos a hacer una breve explicación de cada uno de ellos. También vamos a mencionar cuales hemos decidido descartar y el porqué.

Dividimos las fuentes de información en dos grupos en base al método que hemos empleado para su obtención. El primer grupo será el de los datos obtenidos a partir de la cabecera HTTP y el segundo grupo será el de los datos extraídos de elementos o funciones JavaScript.

4.1. Datos de la cabecera HTTP

Los elementos de la cabecera HTTP que hemos decidido utilizar han sido:

- **Accept**: Informa al servidor sobre los diferentes tipos de datos que el cliente puede procesar mediante una estructura MIME (*Multipurpose Internet Mail Extensions*). Las estructuras MIME son un estándar que indican el formato de un documento o fichero y están compuestas de uno o varios elementos con la siguiente estructura *tipo/subtipo*. Algunos de los valores que puede tomar este campo son *text/html* o *application/xhtml+xml*.
- **Accept-Language**: Anuncia qué idiomas puede entender el cliente y qué variante de región prefiere de este. Algunos ejemplos de valores que puede tener este campo son *es-ES* o *es*.
- **Upgrade-Insecure-Requests**: Envía una señal al servidor en la que se indica que el cliente tiene preferencia por una respuesta cifrada y autenticada, y que puede manejar la directiva CSP *upgrade-insecure-request*. Esta directiva instruye al navegador del usuario para que trate las URLs de sitios no seguros que son aquellas que utilizan el protocolo HTTP como URLs seguras que son las que utilizan el protocolo HTTPS. El valor *1* indica que el usuario maneja la directiva *upgrade-insecure-request* mientras que el *0* representa que no puede manejarla.
- **User-Agent**: Es una cadena característica que identifica el protocolo de red mediante el cual se pueden obtener los datos del tipo de aplicación utilizada, el sistema operativo, el proveedor del software o la versión utilizada. Un ejemplo sobre el formato que tiene este

elemento es *Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.97 Safari/537.36*.

- **Accept-Encoding**: Anuncia la codificación del contenido que el cliente puede entender. Puede estar formado por un valor o por varios y estos valores son *gzip*, *compress*, *deflate*, *br*, *identity* y *.
- **Connection**: Comprueba si la conexión a la red permanece abierta o no al finalizar la transacción actual. Los valores que puede tomar son *keep-alive* si se mantiene la conexión o *close* si la conexión está cerrada.
- **Sec-Fetch-Mode**: Es una cabecera de metadatos que indica el modo de la respuesta. Algunos de los valores que puede contener son *navigate* o *nested-navigate*.
- **Sec-Fetch-User**: Es una cabecera de metadatos que indica si la solicitud de navegación fue realizada por una activación de usuario. Los valores posibles para este campo son *?0* si fue activada por un usuario o *?1* si no lo fue.
- **Sec-Fetch-Site**: Es una cabecera de metadatos que indica la relación entre el origen de la consulta y el origen de la respuesta. Los valores que puede tomar son *none*, *cross-site*, *same-origin* y *same-site*.
- **DNT**: Indica la preferencia de seguimiento del usuario, le indica si prefiere la privacidad al contenido personalizado. Sus posibles valores son *1* si el usuario permite ser reconocido, *0* si el usuario no lo permite o *null* si el usuario no lo especifica.

Estos son los elementos que hemos decidido utilizar para nuestro estudio pero vamos a mencionar también los elementos que extraemos de la cabecera y que hemos descartado porque no tenían relevancia en nuestro trabajo, estos elementos son:

- **Host**: Indica el dominio al que se está realizando la petición. Decidimos descartar este elemento porque el dominio al que se va a realizar la petición es el nuestro y por tanto todas las peticiones van a contener el mismo valor por lo que no decreta ningún criterio de unicidad del usuario.
- **Cookie**: Es un elemento que se utiliza generalmente para identificar a un servidor que varias peticiones vienen del mismo origen. En nuestro caso al usar php este nos genera una cookie propia que tiene el formato *PHPSESSID=mmfvtd8ccr1ud7ksp6r91vmlni* y permite a la web guardar datos serializados del estado. Concluimos no utilizar este dato porque el fin que tiene nuestro TFG es reconocer al usuario mediante su huella digital y este puede expresamente eliminar sus cookies por lo que acabaríamos recogiendo un dato vacío que no tiene ninguna utilidad en nuestro estudio.

4.2. Datos de elementos o funciones JavaScript

Hemos realizado una subdivisión dentro de los elementos extraídos de JavaScript según el objeto o la función que hemos utilizado para obtenerlo.

Los elementos que se extraen de los atributos del objeto *navigator*[26] son:

- **Plataforma**: Indica para qué plataforma está compilado el navegador. Se obtiene mediante el atributo *platform*. Algunos ejemplos de valores que puede devolver este campo son *Win32*, *Linux armv7l* o *MacIntel*.
- **User-Agent**: Representa el mismo objeto que en la cabecera HTTP pero en este caso está obtenido mediante el atributo *userAgent*. Se utiliza para comprobar si el atributo que se envió en la cabecera HTTP no fue falseado y además separando los caracteres de la cadena que lo forma se pueden obtener los siguientes valores:
  - **Navegador**: Indica la plataforma web mediante la cual se está realizando la navegación. Unos posibles valores que puede contener este campo son *Chrome* o *Opera*.
  - **Versión**: Indica la versión del software web con el que se está realizando la navegación y se representa mediante un número.
- **Cookies habilitadas**: Este elemento indica si el navegador tiene las cookies habilitadas por defecto. Se consigue mediante el atributo *cookieEnabled*.Se representa con un booleano.
- **Lenguaje**: Muestra la versión del lenguaje del navegador. Se obtiene a través del atributo *language*. Algunos ejemplos de este campos son *es-ES* o *en-US*.
- **Lenguajes soportados**: indica los lenguajes que es capaz de soportar el navegador. Contiene el mismo valor que el elemento Accept-Language a excepción de que utiliza elementos con formato *q=* con el que indican la preferencia. Se obtienen extrayendo los elementos del atributo *language*. Se representa como una cadena de valores que tiene un formato como este *es-ES // es*.
- **Navegador en línea**: Indica si la máquina del usuario se encuentra conectada a la red o no. Puede devolver los valores *true* si se encuentra conectada o *false* en caso de que no lo esté, y se obtiene a través del atributo *onLine*.
- **AppName**: Muestra el nombre del navegador, por lo general el nombre de todos los navegadores modernos es *Netscape*, pero los navegadores de versiones anteriores devuelven el valor *Microsoft Internet Explorer*. Se obtiene con el atributo *appName*.
- **Información de la batería**: indica si es posible obtener información de la batería. Este elemento se obtiene comprobando la existencia del atributo *getBattery*. Su representación se realiza mediante un booleano.
- **Do-not-track JavaScript**: comprueba cuáles son los ajustes del do-not-track que indica si el usuario quiere ser seguido o mantener su privacidad y se indica mediante un booleano. Se almacena en el atributo *doNotTrack*. Se representa con un booleano.
- **Número máximo de puntos táctiles soportados**: representa el número máximo de zonas sobre las se puede contactar de forma simultánea. Está contenido en el atributo *maxTouchPoints*. Se representa con un valor numérico.
- **Motor del navegador**: indica el motor de navegación y siempre tiene el valor *Gecko*. Se obtiene a través del atributo *product*.
- **ProductSub**: contiene el número de compilación del navegador actual. Se consigue a través del atributo *productSub*. Algunos posibles valores son *20100101* para *Firefox* o *20030107* para *Chrome*.

- **Sistema Operativo**: muestra el Sistema Operativo sobre el que se está ejecutando el navegador. Se obtiene con el atributo *oscpu*. Algunos ejemplos de este campos son *Windows NT x.y; Win64; x64* o *Intel Mac OS X or macOS version x.y*.
- **Vendedor**: indica el proveedor del navegador y puede ser *Google Inc.*, *Apple Computer, Inc.* o ninguno en el caso de Firefox. Su obtención se realiza con el atributo *vendor*.
- **Concurrencia hardware**: es el número de procesadores disponibles que se pueden utilizar para ejecutar subprocesos en la máquina del usuario. Está contenido en el atributo *hardwareConcurrency*. Se representa con un valor numérico.
- **Build Id**: retorna el valor de identificación de compilación del navegador. La mayoría de navegadores devuelve un dato indefinido pero otros devuelven una marca de tiempo fija como medida de privacidad. Se obtiene con el atributo *buildID*. Un posible ejemplo para este campo es *20181001000000* que es el valor que obtiene en *Firefox*.
- **Memoria del dispositivo**: devuelve el tamaño aproximado de la memoria en Gigabytes. Se obtiene del atributo *deviceMemory*. Se representa con un valor numérico.
- **Plugins disponibles**: listado de plugins disponibles por el navegador que utiliza el usuario. Se obtiene con el tratamiento del atributo *plugins*. Se representa como un listado de Plugins entre los que podemos encontrar *Chrome PDF Plugino* o *edge pdf viewer*.

El elemento que se obtiene a partir de la función *getTimezoneOffset()* del objeto *Date* propio de JavaScript es:

- **Diferencia entre la UTC y la hora local en minutos**: diferencia en minutos de la configuración horaria de la zona en la que se encuentra el usuario y la hora media de Greenwich(GMT). Se presenta con un valor númerico que toma los valores *0* si está en GMT o *-120* si está en en GMT+2.

Los elementos que se obtienen a partir del objeto *screen*[27] son:

- **Ancho de la pantalla**: ancho total de la pantalla del usuario en píxeles. Se obtiene con el atributo *width*. Se representa con un valor numérico.
- **Altura de la pantalla**: alto total de la pantalla del usuario en píxeles. Es devuelto por el atributo *height*. Se representa con un valor numérico.
- **Ancho de pantalla disponible**: ancho de la pantalla del usuario en píxeles sin contar con los elementos de la interfaz como la barra de tareas. Se obtiene a través del atributo *availWidth*. Se representa con un valor numérico.
- **Altura de pantalla disponible**: alto de la pantalla del usuario en píxeles sin tener en cuenta los elementos de la interfaz como pueden ser la barra de tareas. Se obtiene con el atributo *availHeight*. Se representa con un valor numérico.
- **Profundidad del color de la pantalla**: devuelve la profundidad de bits de la paleta de colores para mostrar imágenes en bits por píxel. Se obtiene con el atributo *colorDepth*. Se representa con un valor numérico.

- **Profundidad del color en pixel de la pantalla**: devuelve la resolución del color en bits por pixel de la pantalla del usuario. Se consigue a través del atributo *pixelDepth*. Se representa con un valor numérico.

Los elementos que se puede acceder a partir del objeto *window*[28] son:

- **Barra de localización visible**: indica si la barra de localización se encuentra visible o no representado con un booleano. Se obtiene mediante el atributo *locationbar.visible*.
- **Ratio de píxel**: relación que existe entre la altura de un pixel físico de la pantalla que está utilizando el usuario y la altura de un pixel de un dispositivo independiente (dips). Dips, que significa *Density-Independent Pixel*, es una medida que varía según la densidad de píxeles que contiene una pantalla para ajustar el tamaño de los objetos a esta. Se encuentra con el atributo *devicePixelRatio*. Se representa con un valor numérico.
- **Barra de menú visible**: muestra si la barra de menú de la ventana es visible mediante un booleano. Está contenido en el atributo *menubar.visible*.
- **Barra personal visible**: indica si la barra personal de la ventana del usuario es visible o no con un booleano. Se obtiene con el atributo *personalbar.visible*.
- **Barra de estado visible**: muestra si la barra de estado del usuario es visible o no con un booleano. Se consigue con el atributo *statusbar.visible*.
- **Barra de herramientas visible**: muestra si la barra de herramientas está visible en la ventan del usuario o no mediante un booleano. Este valor se obtiene mediante el atributo *toolbar.visible*.
- **Almacenamiento local del dispositivo**: indica si los datos de las diferentes sesiones de navegación se almacenan de forma persistente en el equipo del usuario. Se obtiene comprobando la existencia del atributo *localStorage*. Su representación se realiza mediante un booleano.
- **Almacenamiento de la sesión disponible**: representa si se almacena la información de la sesión. La diferencia con el almacenamiento local es que en este los datos se elimina cuando se cierra la sesión. Se consigue comprobando la existencia del atributo *sessionStorage*. Su representación se realiza mediante un booleano.
- **Se puede usar base de datos indexadas**: muestra si es posible almacenar datos en el navegador del usuario lo que permite a los usuarios seguir navegando sin tener conexión. Se obtiene comprobando la existencia del atributo *indexedDB*. Se representa con un booleano.
- **Objeto results de windows disponible**: este elemento indica si el objeto result de windows está disponible. Se obtiene comprobando la existencia del atributo *results*. Se representa con un booleano.

Antes de contar el resto de elementos hay que mencionar que hemos definido apartados en el DOM de la página a los que hemos denominado regiones sobre los que realizaremos pruebas para los datos que buscamos. Los elementos que hemos obtenido a partir de estas pruebas son:

- **Canvas**: definimos un dibujo en una región canvas que se representará de forma diferente dependiendo del equipo y del navegador que lo reproduzca y de esta representación obtenemos un resumen. Este resumen consiste en obtener un hash del tipo *MD5* que utilizamos para definir el elemento canvas y así establecer relaciones entre la forma de representar de cada usuario. Ofrecemos diferentes elementos a pintar, tres cuadrados de color rojo, azul y amarillo, un emoticono de una carita sonriente y la siguiente frase ”PrUeBa De CaNvAs En Tu NaVeGaDor”. Este elementos lo obtenemos con una función creada por nosotros a la que hemos llamado *pintar()*.

- **Formatos de video soportados**: en una región de video realizamos pruebas para ver si es posible reproducir diversos formatos de video. Para obtener esta información urilizamos una función a la que llamamos *formatosSoportadosVideo()*. La prueba para cada formato puede retornarnos uno de estos valores *probably*, *maybe*, o vacío. Los formatos de video que hemos utilizado para hacer pruebas han sido los siguientes:

Formato	Codecs
video/ogg	theora
video/ogg	vorbis
video/ogg	opus
video/mp	avc1.4D401E
video/mp4	mp4a.40.2
video/mp4	flac
video/webm	vp8.0
video/webm	vp9
video/webm	cvorbis

- **Formatos de audio soportados**: sobre una zona de audio probamos si es posible reproducir diversos formatos en la máquina del usuario. Para ello utilizamos una función a la que hemos puesto el nombre de *formatosSoportadosAudio()*. Para cada formato puede devolvernos los valores *probably*, *maybe*, o vacío. Los fomatos de audio que utilizamos para realizar las pruebas son:

Formato	Codecs
audio/ogg	vorbis
audio/ogg	opus
audio/3gpp	
audio/mp4	mp4a.40.5
audio/mp4	mp3
audio/mp4	ac-3
audio/mp4	ec-3
audio/aac	
audio/pcm	
audio/flac	
audio/wave	
audio/aac	
audio/webm	vorbis
audio/mp3	mp3

- **Lista de fuentes detectadas**: este elemento se consigue a partir de la función *fingerprint-fonts()* que realiza pruebas para ver si el navegador que esta usando el usuario soporta diversas fuentes. Se hace una comprobocación con un listado de más de 500 fuentes.

En cuanto a los elementos JavaScript que hemos descartado no es el mismo caso que con los de la cabecera, ya que en ella recibimos ciertos elementos y descartamos algunos por que no nos eran útiles. En este caso los elementos descartados son aquellos que descubrimos y que en un principio podrían tener utilidad para nuestro estudio pero que tras realizar pruebas con ellos no nos parecieron del todo provechosos. Estos elementos descartados de código JavaScript son:

- **Bateria**: en cuanto a elementos más concretos sobre la batería como pueden ser la carga total del equipo o si el equipo se encuentra cargando en estos momentos podrían haber sido datos útiles. Pero, tras hacer pruebas con ellos, decidimos descartarlos porque los valores que recibimos de estos parámetros eran siempre los mismos para todos los usuarios y lo único que logramos con ellos era que las huellas de usuarios diferentes fueran un poco más parecidas.
- **Geolocalización**: sobre este elemento pensamos que podría sernos de bastante utilidad conocer qué localización tenía el usuario a la hora de conectarse a nuestra web. Tras investigar con ella descubrimos que para poder acceder a este dato el usuario debía darnos un permiso concreto. Fue este hecho el que nos hizo descartalo porque nos parecía que perdía bastante importancia puesto que la huella que pretendemos sacar es solo con la conexión al servidor sin más acciones del usuario.
- **Historial de la sesión**: en cuanto a este elemento lo utilizamos porque pensamos que podría concretar más la huella porque si sabíamos que paginas suele visitar normalmente el usuario es mucho más probable que sepamos qué usuario está al otro lado de la pantalla.Tras investigar con él descubrimos que solo nos proporcionaba el número de webs visitadas antes que la nuestra. Esto nos hizo ver que ese número podría ser cualquiera para el mismo usuario y lo único que conseguimos con él fue que el ratio de similaridad de dos huellas iguales se hiciese un poco más diferente cuando en realidad pertenecen al mismo.

Capítulo 5

Diseño e implementación

La aplicación recopila información del usuario en distintas etapas. Se trata de una aplicación web que ejecuta código tanto en cliente como en servidor para obtener dicha información.

- **Cliente**: Es la parte de la aplicación ejecutada en el navegador del usuario. Obtiene información del navegador mediante código JavaScript y lo envía al servidor de forma asíncrona.
- **Servidor**: Aloja la base de datos y el código que obtiene la información de las cabeceras HTTP.
    - **Base de datos**: Una base de datos relacional MariaDB/MySQL en la que se almacenarán los datos obtenidos de cada una de las conexiones de los usuarios.

5.1. Funcionamiento

Nuestra aplicación se ejecuta tanto en cliente como en servidor, en distintas etapas y comunicándose de manera asíncrona en algunas de ellas.

El cliente hace una petición HTTP al servidor. El servidor analiza las cabeceras de la petición y las inserta en la base de datos con un identificador asociado al usuario para definir su huella. Devuelve al usuario una página estática en HTML con JavaScript embebido en la que el usuario verá los datos de su cabecera HTTP y el código JavaScript se ejecutará automáticamente para obtener el resto de datos del navegador.
Una vez terminada la ejecución de todo el código JavaScript, este actualizará la apariencia de la página para que el usuario pueda ver las características obtenidas mediante JavaScript y enviará de forma asíncrona, sin recargar la página del usuario, toda la información obtenida al servidor.
El servidor recibe la información obtenida en JavaScript y actualiza todas las filas de las tablas de la base de datos para el id del usuario.
Por último, el servidor ejecuta el código encargado de calcular la unicidad de cada uno de los atributos del usuario por separado y la unicidad total del conjunto de atributos. Tras calcularlo devolverá como respuesta de la petición asíncrona del código JavaScript todos los valores de la unicidad, que volverán a actualizar la página para mostrarle al usuario cuánto de único es en cada uno de los campos de su navegador y cuánto de único es su navegador en conjunto.

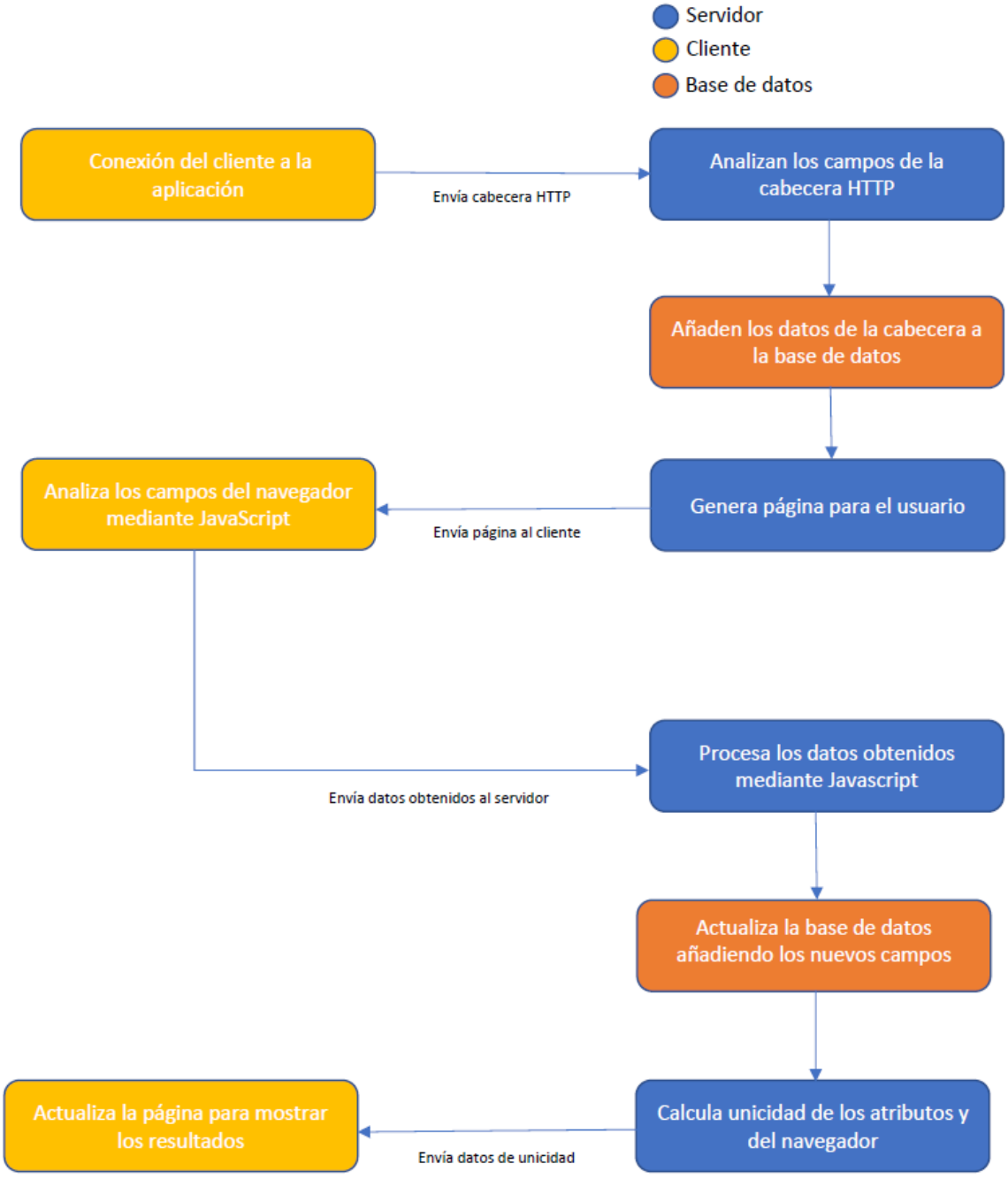


Figura 5.1: Diagrama de flujo de la aplicación

5.2. Componentes del sistema

El proyecto cuenta con distintos componentes. Principalmente se dividen en cliente y servidor.

5.2.1. Servidor

El servidor se encarga de gestionar las peticiones HTTP enviadas por el usuario al utilizar nuestra aplicación web.

La página consta de dos rutas, que serán generadas por el servidor mediante PHP:

- /: Es la base de la aplicación. En ella se analizan y muestran todos los elementos del navegador del usuario, así como su unicidad general e individual de cada uno de los atributos del navegador.
- **/graficas**: En esta página se muestra al usuario gráficas con los datos globales de los atributos obtenidos por la aplicación.

Para la funcionalidad principal de la aplicación el servidor actúa en dos ocasiones.

La primera se encarga de recopilar la información de la cabecera HTTP, generar la página que verá el usuario e insertar en la base de datos esta información. Esta parte corresponde a la ruta principal.

La segunda ocasión recibe una petición asíncrona con los datos obtenidos en la ejecución de código en cliente. El servidor se encarga de actualizar la base de datos añadiendo esos datos a las tablas para el id del usuario. Una vez añadida esta información, genera un objeto JSON que devolverá como respuesta de la petición. En este objeto se encuentra la unicidad de cada uno de los elementos analizados y la unicidad general del navegador del usuario. Esta acción se lleva a cabo haciendo una llamada a la ruta /cargaJS.

Lenguaje de programación

Para el código del servidor hemos utilizado el lenguaje PHP. Con él se gestiona todo el funcionamiento de la aplicación en el servidor.

En el servidor se generan las páginas HTML que se envían al cliente. Al ser necesario que cada cliente reciba una página única y personalizada estas se generan de manera dinámica en el servidor con este lenguaje. Por otra parte, PHP nos permite obtener la información de la cabecera HTTP que recibe el servidor y tratarla para añadir esta información en la página que verá el cliente y a su vez insertarla en nuestra base de datos.

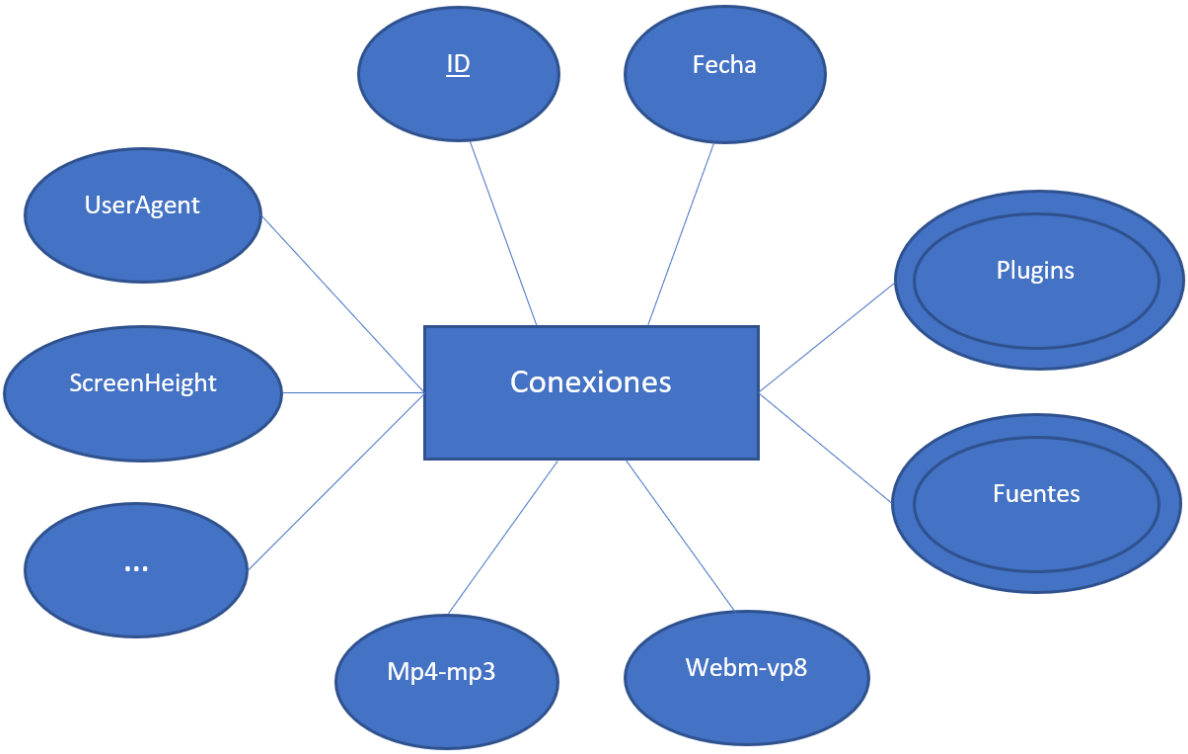


Figura 5.2: Diagrama entidad-relación de la base de datos

Base de datos

La base de datos de nuestra aplicación es de tipo relacional. En la figura 5.2 se puede observar el diagrama entidad-relación de nuestra base de datos, en el que omitimos muchos los atributos de las tablas al ser su número tan elevado que dificultaría la legibilidad del diagrama. Este diagrama consta de una única entidad con una cantidad elevada de atributos, algo normal puesto que la aplicación está centrada en almacenar y comparar conexiones. La traducción del diagrama entidad-relación ha dado lugar a 5 tablas, las cuales detallaremos a continuación.

- **Conexiones**: La tabla principal de la base de datos. En ella se almacenan el ID del usuario, la fecha de la inserción, los datos obtenidos de la cabecera HTTP (detallados en la tabla 5.1) y la mayoría de atributos obtenidos mediante JavaScript (tabla 5.2). El ID es la clave primaria de la tabla e identifica al usuario y a todos sus atributos.

  Los atributos de la cabecera HTTP varían en los distintos navegadores o incluso en un mismo navegador. Por esa razón examinamos los atributos de las cabeceras HTTP de distintos navegadores y en distintos dispositivos, y seleccionamos los mas comunes para insertarlos en la base de datos. Estos elementos se detallan en la tabla 5.1. El resto de elementos se pueden observar en la aplicación, pero no los almacenamos.

  La fecha se almacena por seguridad y estaba pensado utilizarla en una opción para comprobar las huellas almacenadas en distintos intervalos de tiempo, funcionalidad que ha quedado como trabajo futuro.

  El resto de atributos de la tabla *Conexiones* son los obtenidos mediante JavaScript. Estos elementos están detallados en la tabla 5.2.

Atributo	Tipo
ID	Int
Fecha	Timestamp
Accept	Varchar
AcceptLanguage	Varchar
UpgradeInsecureRequest	Varchar
UserAgent	Varchar
AcceptEncoding	Varchar
Connection	Varchar
SecFetchMode	Varchar
SecFetchUser	Varchar
SecFetchSite	Varchar
DNT	TinyInt

Tabla 5.1: Atributos de la cabecera HTTP en la tabla *Conexiones*

Entre los atributos de JavaScript destacan tres en concreto:

- **Canvas**: Es el resumen del resultado del elemento *canvas* dibujado en el navegador. Este elemento se puede leer como una cadena de caracteres de una longitud considerable. Por ello decidimos resumirla utilizando el algoritmo «MD5» para almacenarla en la base de datos sin ocupar tanto espacio.
- **ResumenFuentes**: Se trata de un resumen utilizando el algoritmo «MD5» de todas las fuentes detectadas en el navegador. Para ello, a la hora de almacenar las fuentes, también las ordenamos y concatenamos, para después resumirlas e insertarlas en esta tabla. Esto se hace para ahorrar tiempo a la hora de calcular la unicidad de la huella del usuario.
- **ResumenPlugins**: Hacemos un resumen de todos los plugins encontrados en el navegador de la misma forma que hacemos con las fuentes y por la misma razón. Para el resumen de este elemento también utilizamos el algoritmo «MD5».

- **FormatosAudio**: En esta tabla se almacenan los resultados de la comprobación de los distintos formatos de audio soportados por el navegador. Para hacer esta comprobación analizamos los formatos mas comunes en la actualidad en navegadores [29]. Estos elementos forman parte de la tabla *Conexiones*, pero decidimos almacenarlos en una tabla alternativa para trabajar con conjuntos de elementos mas reducidos. En la tabla 5.3 se detallan los atributos de esta.

- **FormatosVideo**: En esta tabla se almacenan los resultados de la comprobación de los distintos formatos de vídeo soportados por el navegador. De la misma forma que con los formatos de audio, analizamos los formatos de vídeo mas comunes en navegadores [29]. Por el mismo motivo que en la tabla *FormatosAudio*, decidimos almacenar los resultados de los formatos de vídeo en una tabla aparte.Se pueden ver en detalle sus atributos en la tabla 5.4

- **Plugins**: En esta tabla se almacenan los nombres de los plugins encontrados en un navegador. Se trata de un atributo multivaluado de la tabla *Conexiones*. Los atributos de esta tabla están detallados en la tabla 5.5.

Atributo	Tipo
Plataforma	Varchar
UserAgentJS	Varchar
Navegador	Varchar
Version	Int
CookieEnabled	TinyInt
Language	Varchar
Lenguajes	Varchar
OnLine	TinyInt
AppName	Varchar
Bateria	TinyInt
DNTJS	TinyInt
TouchPoints	Int
Product	Varchar
ProductSub	Varchar
OS	Varchar
Vendor	Varchar
HardwareConcurrency	Int
BuildID	Varchar
DevMemory	Float
ResumenPlugins	Varchar

Atributo	Tipo
ZonaHoraria	Int
ScreenWidth	Int
ScreenHeight	Int
ScreenAvailWidth	Int
ScreenAvailHeigth	Int
ScreenColorDepth	Int
ScreenPixelDepth	Int
LocationBar	TinyInt
PixelRatio	Double
MenuBar	TinyInt
PersonalBar	TinyInt
StatusBar	TinyInt
ToolBar	TinyInt
LocalStorage	TinyInt
SessionStorage	TinyInt
IndexDB	TinyInt
WindowResults	TinyInt
Canvas	Varchar
ResumenFuentes	Varchar

Tabla 5.2: Atributos obtenidos mediante JavaScript en la tabla *Conexiones*

Columna	Tipo
ogg-vorbis	Varchar
3gpp	Varchar
mp4-mp4a	Varchar
mp4-mp3	Varchar
mp4-ac3	Varchar
mp4-ec3	Varchar
acc	Varchar
pcm	Varchar
mpeg	Varchar
flac	Varchar
wave	Varchar
webm-vorbis	Varchar
mp3-mp3	Varchar

Tabla 5.3: Tabla *FormatosAudio*

Columna	Tipo
ogg-theora	Varchar
ogg-vorbis	Varchar
ogg-opus	Varchar
mp4-avc1	Varchar
mp4-mp4a	Varchar
mp4-flac	Varchar
webm-vp8	Varchar
webm-vp9	Varchar
webm-vorbis	Varchar

Tabla 5.4: Tabla *FormatosVideo*

- **Fuentes**: Tabla en la que se almacenan todos los nombres de las fuentes encontradas en el navegador del usuario.En ella se guardan todas las fuentes encontradas en cada análisis del navegador. Es un atributo multivaluado de la tabla *Conexiones*. Sus atributos se detallan en la tabla 5.6.

Columna	Nombre
ID	Int
NombrePlugin	Varchar

Tabla 5.5: Columnas de la tabla *Plugins*

Columna	Nombre
ID	Int
NombreFuente	Varchar

Tabla 5.6: Columnas de la tabla *Fuentes*

5.2.2. Cliente

En cliente se carga la página del usuario, en la que se muestran los datos del navegador en el que se está ejecutando la aplicación.

La página que recibe el usuario es dinámica. Al principio en ella se muestra la información obtenida de la cabecera HTTP mientras se ejecuta código JavaScript encargado de obtener el resto de datos del navegador. Cuando se termina de ejecutar esta parte, se realiza una llamada asíncrona al servidor, en la que se envían todos los datos obtenidos y la página queda a la espera de la respuesta de este. Al recibir la respuesta en la que se encuentran todos los valores de la unicidad del navegador, se actualiza la apariencia de la página para que el usuario pueda terminar de ver toda la información relacionada a su navegador y cuánto de única es su huella.

Lenguajes de programación

El lenguaje utilizado en cliente para generar la página es HTML, con JavaScript para toda la parte dinámica.

El código JavaScript se encarga de obtener datos del navegador mediante clases y funciones predefinidas en el JavaScript y otros métodos desarrollados por nosotros. Además JavaScript nos permite realizar las peticiones asíncronas al servidor de forma que se puede actualizar la página sin necesidad de recargarla una vez recibe la respuesta del servidor [30].

Para la generación de los gráficos de las estadísticas de la aplicación utilizamos la herramienta «Google Charts» [24], la cual importamos mediante JavaScript y que permite crear distintos tipos de gráficas.

5.3. Problemas

Durante el desarrollo de la aplicación nos hemos encontrado con distintos problemas, los cuales detallaremos a continuación junto con las soluciones que hemos encontrado para estos.

Asincronía

Este problema surgió durante la implementación de la parte asíncrona de nuestra aplicación. Para la realización de esta utilizamos el objeto «XMLHTTPRequest» de JavaScript. Este permite realizar peticiones al servidor mientras se sigue ejecutando código en cliente y, una vez recibe la respuesta del servidor, puede ejecutar funciones que estaban a la espera de la respuesta. El código JavaScript que utilizamos está dividido en distintas secciones, dependiendo de los objetos que se utilizan o el tipo de información que se obtiene. Una vez se termina de obtener todos los datos de cada sección se realiza el envío de información al servidor de esa parte. Solo en el último envío, cuando ya están todos los datos alojados en el servidor se espera la información devuelta por este para actualizar la página.

Al ejecutarse el código de manera asíncrona en el servidor, en algunas ocasiones se ejecutaba la última petición antes que el resto, por lo que se retornaba una unicidad errónea del navegador, ya que se recibía la respuesta antes de que el resto de datos hubiesen sido añadidos en el servidor.

Para solucionarlo hicimos que una vez lanzada una petición asíncrona no lanzase la siguiente hasta haber recibido la respuesta del servidor, de forma que se lanzasen síncronamente.

Adaptación a los cambios en los navegadores

Para obtener los datos que identifican a los navegadores, dependemos completamente de su implementación. Esto quiere decir que en el momento en que se introduzcan cambios en las WebAPIs o en las cabeceras de cada navegador, deberemos actualizar nuestro proyecto en consonancia. Del mismo modo, los elementos de la cabecera HTTP pueden variar de un navegador a otro, e incluso dentro de un mismo navegador. Nuestra aplicación analiza todos los campos de la cabecera HTTP de manera automática, por lo que en ocasiones pueden aparecer elementos con los que no contamos.

La solución fue analizar los elementos mas comunes a todos los navegadores y controlar que solo fuesen esos los que tratamos dentro de la aplicación. Esta comprobación la hacemos mediante PHP en el servidor. El resto de datos solo se muestran en la página al usuario para comprobar los elementos que conforman la cabecera HTTP de su navegador.

Campos multivaluados

En nuestra base de datos existen dos tablas que corresponden a atributos multivaluados de la tabla *Conexiones*. Estas tablas son *fuentes* y *plugins*. En estas tablas se almacenan en cada fila un nombre de fuente o plugin y el ID del resultado al que pertenecen. El problema surgió

a la hora de comprobar la unicidad de la huella del usuario, ya que se complicaba la consulta a realizar por la cantidad de filas de la tabla.

La solución que encontramos fue añadir un resumen de estos elementos en la tabla *Conexiones*. Para ello, además de insertar todos los elementos de fuentes y plugins en sus respectivas tablas, concatenamos estos elementos ordenados y los resumimos. De esta forma se simplificaba mucho el cálculo de la unicidad de la huella del usuario.

Capítulo 6

Uso del sistema

Es este capítulo realizaremos un recorrido guiado sobre el uso de nuestra herramienta. Explicaremos paso a paso cada una de las opciones que hay a nuestra disposición y adjuntaremos capturas de pantalla de cada acción realizada. Recordamos que el sistema está disponible en http://browserfingerprinting.educationhost.cloud/

6.1. Bienvenida

Lo primero que nos encontramos al acceder a la *URL* es una pantalla de bienvenida como la que aparece en la figura 6.1. Aquí damos una breve explicación de qué es la huella digital de un navegador web, introducimos en qué consiste nuestro proyecto e informamos al usuario del tratamiento que vamos a hacer con sus datos. Para continuar, pincharemos en el botón Ver huella digital de tu navegador.

6.2. Índice

Tras pinchar en el botón de la sección anterior, la página nos carga un índice con distintas secciones. También aparece arriba a la izquierda el botón de Gráficos, que detallaremos en el siguiente apartado.

Resultado

La primera de las secciones es la de Resultado, figura 6.2, que nos indica cuántas huellas digitales iguales a la del usuario se han encontrado en nuestra base de datos. Esto es, nos da el porcentaje de unicidad total del usuario. A partir de aquí se muestran varias secciones con tablas. En cada tabla aparece:

- Cada elemento de esa sección.
- El porcentaje de similaridad que existe entre el valor del elemento y los que tenemos guardados en la base de datos.

Figura 6.1: Pantalla de bienvenida.

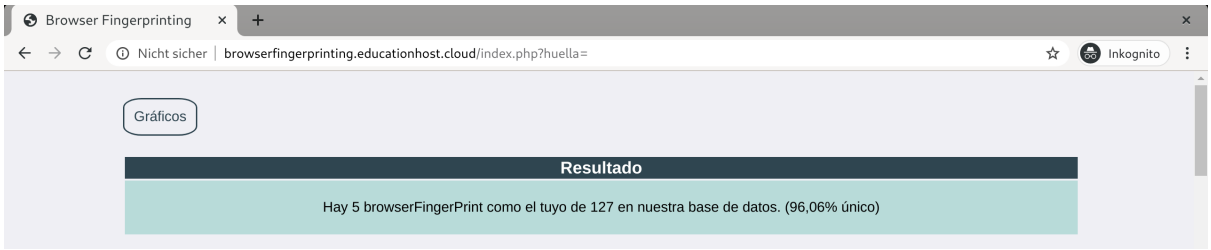


Figura 6.2: Porcentaje de unicidad total.

Atributos de la cabecera HTTP

Elemento		Similaridad	Valor
Connection	info	96,85%	keep-alive
Upgrade-Insecure-Requests	info	90,55%	1
User-Agent	info	10,24%	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.106 Safari/537.36
Accept	info	52,76%	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding	info	96,85%	gzip, deflate
Accept-Language	info	8,66%	de-DE,de;q=0.9,en;q=0.8
Content-Length	info		0

Figura 6.3: Cabecera HTTP.

- El valor que toma el elemento en nuestro dispositivo.

Cabecera HTTP

En la figura 6.3 se nos muestra una tabla con los distintos campos que contiene la cabecera de petición HTTP del cliente. Si situamos el cursor encima del botón info que aparece en cada elemento, podemos leer una pequeña explicación sobre su significado.

JavaScript

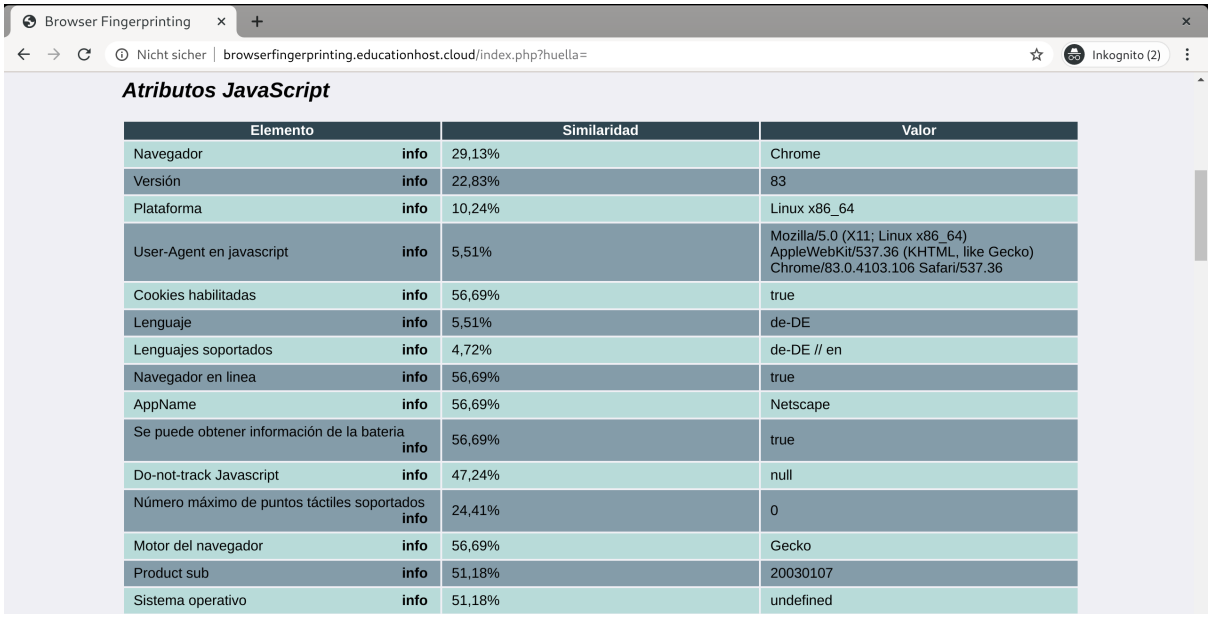
Aquí mostramos información sobre los elementos a los que accedemos desde WebAPI a través de JavaScript, que se puede observar en la figura 6.4. En el atributo de canvas pintamos sobre el lienzo del navegador y le mostramos el dibujo al usuario, como se puede ver en la figura 6.5.

Formatos de vídeo

Cada navegador es capaz de reproducir unos formatos de vídeo específicos. En la figura 6.6 se pueden ver los valores que se obtienen. El valor probably indica que es muy probable que dicho formato esté soportado, en cambio el valor maybe nos da una respuesta más incierta. Se realiza esta distinción porque aunque a veces el formato está soportado, algunos códecs en concreto puede que no lo estén. Si el formato no está soportado por el navegador, se muestra el resultado No soportado.

Formatos de audio

De forma análoga a los formatos de vídeo, en la figura 6.7 mostramos una tabla con los formatos de audio a los que el navegador es capaz de dar soporte.



Atributos JavaScript

Elemento		Similaridad	Valor
Navegador	info	29,13%	Chrome
Versión	info	22,83%	83
Plataforma	info	10,24%	Linux x86_64
User-Agent en javascript	info	5,51%	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.106 Safari/537.36
Cookies habilitadas	info	56,69%	true
Lenguaje	info	5,51%	de-DE
Lenguajes soportados	info	4,72%	de-DE // en
Navegador en linea	info	56,69%	true
AppName	info	56,69%	Netscape
Se puede obtener información de la bateria	info	56,69%	true
Do-not-track Javascript	info	47,24%	null
Número máximo de puntos táctiles soportados	info	24,41%	0
Motor del navegador	info	56,69%	Gecko
Product sub	info	51,18%	20030107
Sistema operativo	info	51,18%	undefined

Figura 6.4: Elementos accedidos desde WebAPI a través de Javascript.

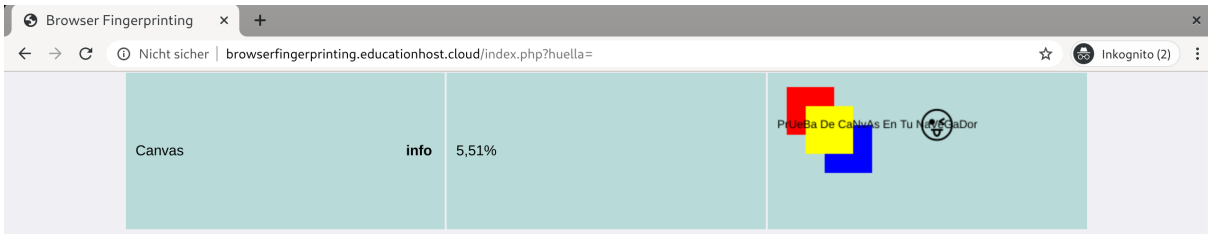
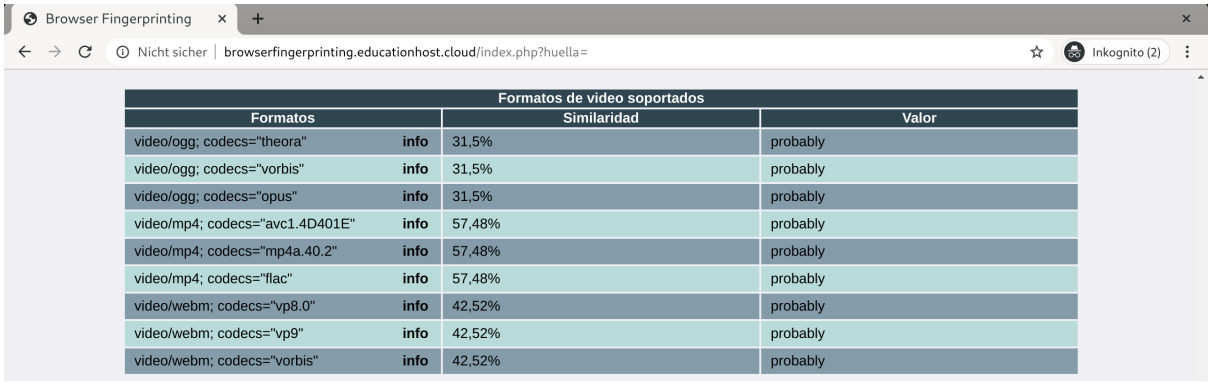
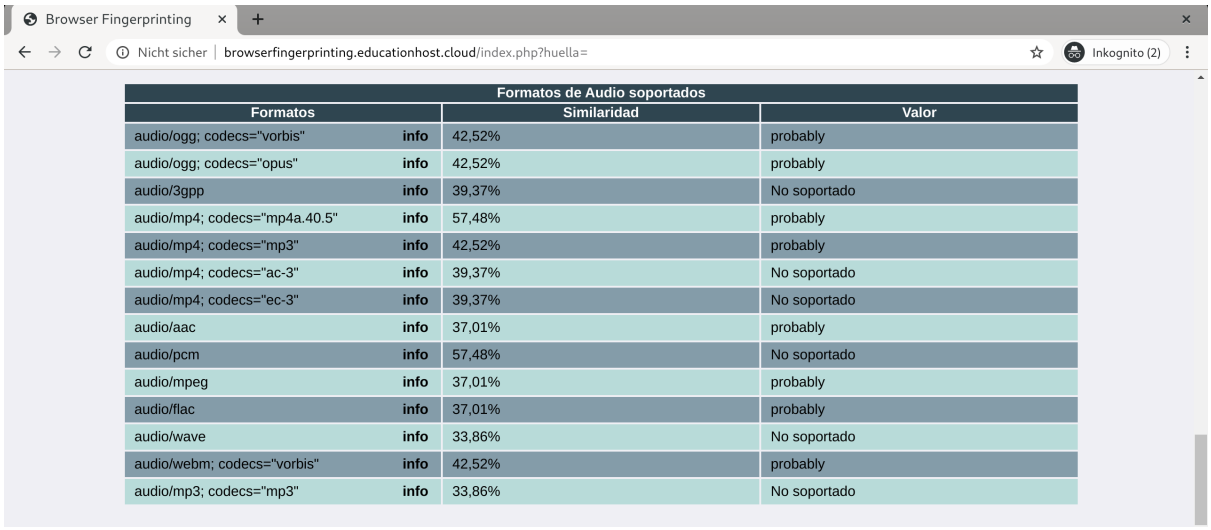


Figura 6.5: Obtención de la huella digital a través del lienzo.



Formatos de video soportados			
Formatos		**Similaridad**	**Valor**
video/ogg; codecs="theora"	info	31,5%	probably
video/ogg; codecs="vorbis"	info	31,5%	probably
video/ogg; codecs="opus"	info	31,5%	probably
video/mp4; codecs="avc1.4D401E"	info	57,48%	probably
video/mp4; codecs="mp4a.40.2"	info	57,48%	probably
video/mp4; codecs="flac"	info	57,48%	probably
video/webm; codecs="vp8.0"	info	42,52%	probably
video/webm; codecs="vp9"	info	42,52%	probably
video/webm; codecs="vorbis"	info	42,52%	probably

Figura 6.6: Formatos de vídeo soportados.



Formatos de Audio soportados			
Formatos		Similaridad	Valor
audio/ogg; codecs="vorbis"	info	42,52%	probably
audio/ogg; codecs="opus"	info	42,52%	probably
audio/3gpp	info	39,37%	No soportado
audio/mp4; codecs="mp4a.40.5"	info	57,48%	probably
audio/mp4; codecs="mp3"	info	42,52%	probably
audio/mp4; codecs="ac-3"	info	39,37%	No soportado
audio/mp4; codecs="ec-3"	info	39,37%	No soportado
audio/aac	info	37,01%	probably
audio/pcm	info	57,48%	No soportado
audio/mpeg	info	37,01%	probably
audio/flac	info	37,01%	probably
audio/wave	info	33,86%	No soportado
audio/webm; codecs="vorbis"	info	42,52%	probably
audio/mp3; codecs="mp3"	info	33,86%	No soportado

Figura 6.7: Formatos de audio soportados.

Plugins		
Formatos	Similaridad	Valor
Plugins disponibles	4,19%	Shockwave Flash
Plugins totales instalados		1
Versión de flash instalada		32.0.0.387
AdBlock activado		false

Figura 6.8: Información sobre *plugins* del navegador.

Plugins

En este apartado se recopilan los nombres de las distintas extensiones que hay instaladas en el navegador del cliente. En la tabla de *Plugins*, correspodiente a la figura 6.8, mostramos también información acerca de los complementos Flash y AdBlock en caso de que estén instalados.

Fuentes

En la figura 6.9 se indican cuáles son las fuentes que están instaladas en el sistema. También nos indica el número total de fuentes que han sido detectadas.

Ya hemos observado las distintas secciones de nuestra tabla con los valores correspondientes. Volveremos a la parte superior de la página y selecionaremos el botón Gráficos para cambiar de ventana (figura 6.10).

6.3. Gráficos

Una vez hemos seleccionado el botón Gráficos, aparece en pantalla un gráfico circular en tres dimensiones. Este gráfico representa estadísticas para algunos de los parámetros que he-

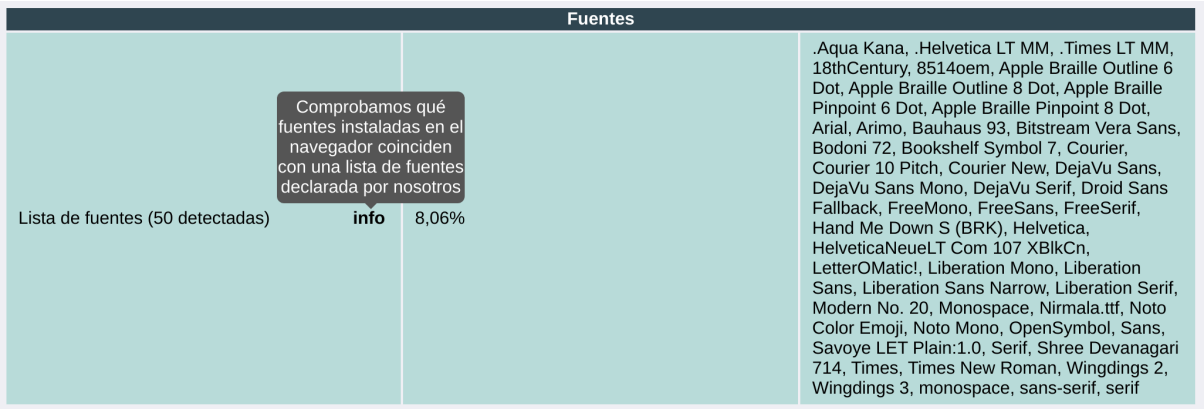


Figura 6.9: Fuentes del navegador.

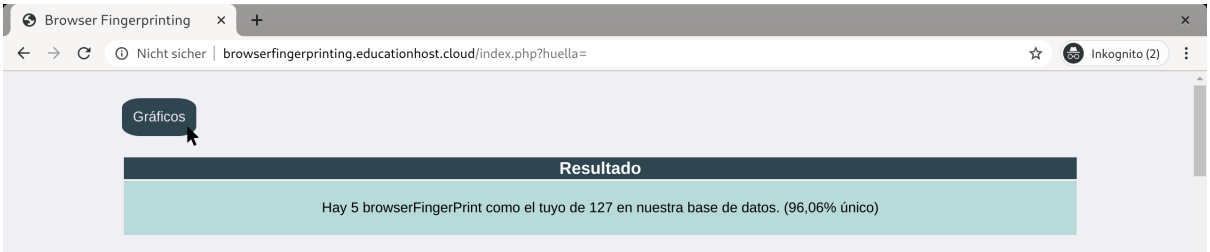


Figura 6.10: Ir a la ventana Gráficos.

mos almacenado en nuestra base de datos. Por defecto, se muestran los porcentajes del atributo navegador. En la figura 6.11 se puede observar la distribución de valores de las conexiones que ha recibido la aplicación. Observamos que a la derecha del gráfico aparece una leyenda de colores que indica cuál es el valor de cada color en el gráfico.

Podemos acceder al menú desplegable que se encuentra arriba a la izquierda. Pinchando sobre el atributo seleccionado, se despliegan las distintas opciones, como se puede ver en la figura 6.12. Podemos elegir ver las estadísticas en forma de diagrama de tartas para cada parámetro que se muestra en el menú. En nuestro caso, decidimos ver las estadísticas de pixelRatio, así que seleccionaremos la opción que se muestra en la figura 6.13.

Las opciones que nos quedan ahora son escoger otro atributo del menú para seguir viendo más diagramas o volver a la pantalla de bienvenida pinchando en el botón BrowserFingerprint de la figura 6.14 en la parte superior izquierda de la página.

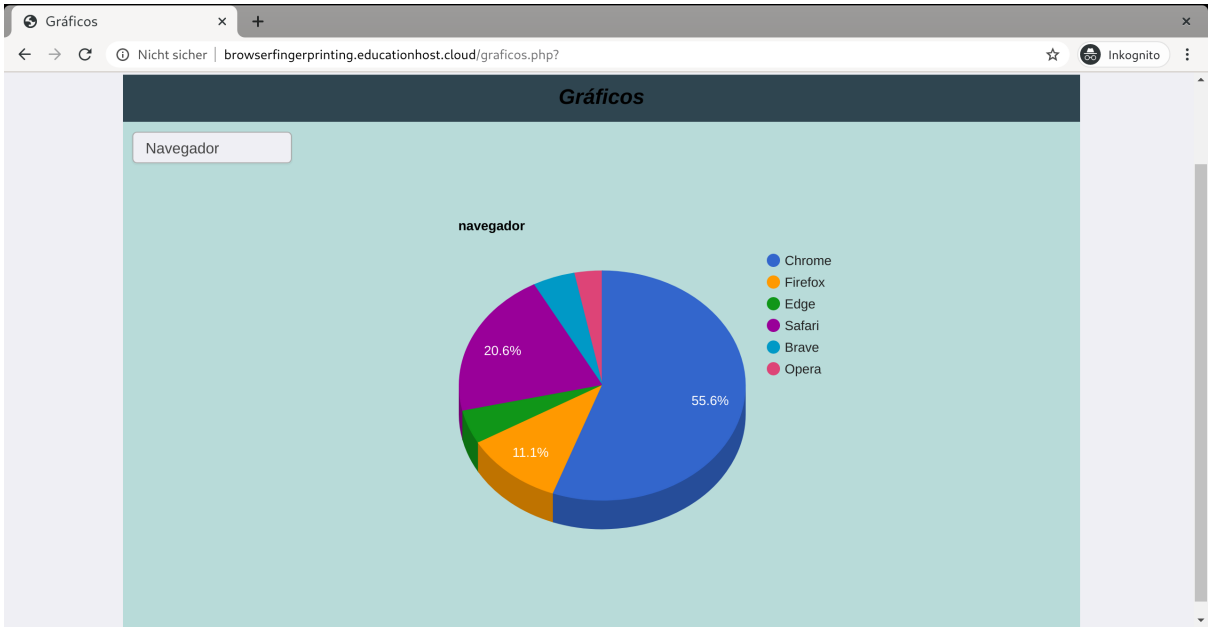


Figura 6.11: Gráfico del atributo navegador.

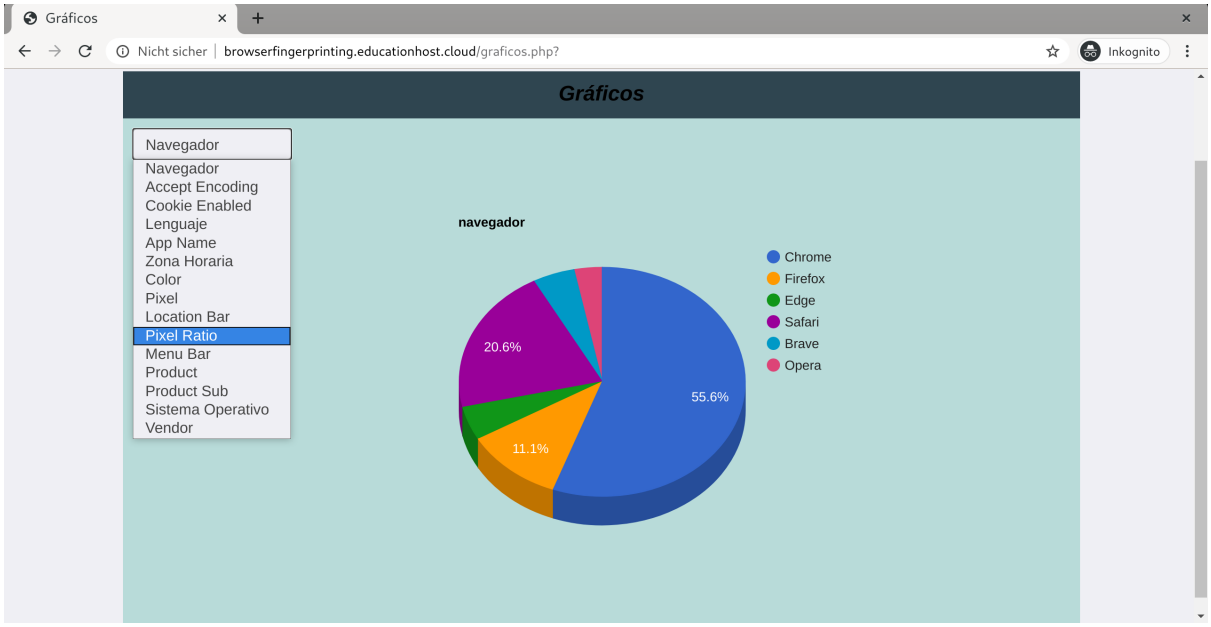


Figura 6.12: Menú desplegable del apartado de Gráficos.

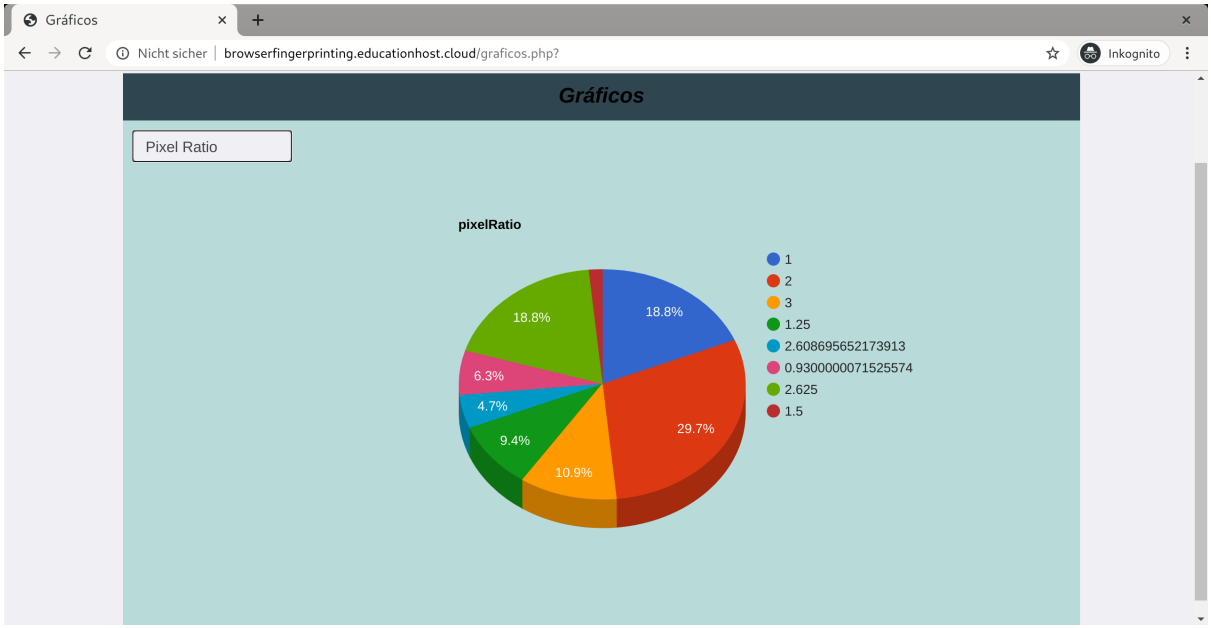


Figura 6.13: Gráfico del atributo pixelRatio.

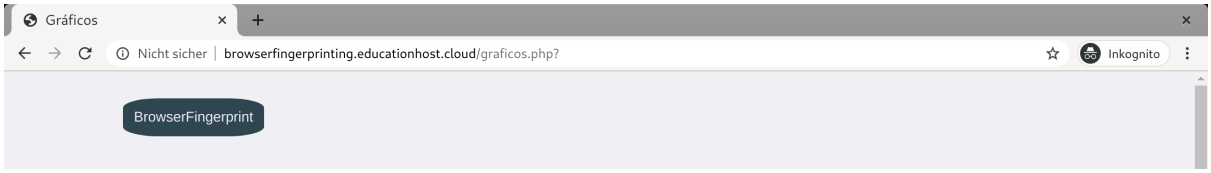


Figura 6.14: Volver a la pantalla de bienvenida.

Capítulo 7

Contribución individual

A lo largo del desarrollo del trabajo la carga de tareas ha sido equitativa, trabajando de forma paralela cada uno de los integrantes en distintas tareas. A continuación se detalla la aportación individual de cada uno de los autores de este trabajo.

7.1. Adrián Agudo García-Heras

Investigación

En este primer apartado lo primero que hice fue buscar información sobre qué es el *fingerprint* y qué utilidad tiene, ya que al ser la base del trabajo era un concepto que tenía que tener bastante claro. Durante esta investigación aprendí que para conseguir el *fingerprint* es necesario averiguar varios valores referentes al software y al hardware que utiliza el usuario. Fue en esta parte en la que observamos aplicaciones en las que podríamos basarnos como por ejemplo *AmIUnique* [2], que ha sido el gran espejo en el que nos hemos mirado.

Los primeros valores sobre los que se centró la investigación fue sobre los referentes a la cabecera HTTP. Una vez encontrados qué campos de la cabecera podía obtener había que ver cuáles eran compartidos por varios navegadores y cuáles eran más específicos con el fin de utilizar los que fueran a aportarnos una mayor utilidad. Tras esto, comencé la investigación sobre qué elementos podríamos obtener sobre JavaScript. En este apartado centré la investigación en buscar qué objetos nos ofrece este lenguaje de los cuales podamos obtener datos que nos fueran útiles para la obtención de la huella digital. Tras observar qué objetos podemos utilizar y haberlos comparado con los datos de las aplicaciones en las que nos hemos fijado, comencé la investigación para averiguar esos datos que nos parecían relevantes pero que no habíamos encontrado como, por ejemplo, elementos como el Navegador o la versión. Fue ahí donde descubrí funciones que estaban ya realizadas y que podían aportarnos esos datos extra para afinar más aún la huella que queríamos obtener del usuario.

Análisis de requisitos

Una vez terminado el proceso de investigación comenzamos con la selección de las herramientas que íbamos a utilizar, que en nuestro caso fue algo que hicimos todos de forma común. Sabíamos que íbamos a realizar una aplicación web y por tanto necesitábamos definir qué lenguajes iban a trabajar sobre el servidor y cuáles sobre el cliente.

En esta parte la selección de la herramienta gráfica corrió íntegramente a cargo mía. Decidí seleccionar Google Charts porque me pareció que era bastante intuitiva y se acoplaba perfectamente al uso que nosotros queríamos hacer de ella. Aparte, la documentación que aportan está bastante completa lo que me permitió aprender a utilizarla sin mucha dificultad.

Implementación

Esta etapa, al ser la más larga, estuvo dividida en varias fases. La primera fase consistió en adquirir todos los elementos que habíamos visto útiles para el trabajo. En este apartado mi tarea consistió en conseguir diversos datos aportados por JavaScript. Algunos de los elementos sobre los que yo trabajé fueron el navegador, la versión o el listado de las fuentes disponibles.

Más tarde comenzamos una fase de diseño en la que yo me encargue de crear la vista de los gráficos. Primero hice un pequeño montaje para mostrar solo los valores posibles de un campo. En esta prueba generé la ventana para las gráficas, añadí la biblioteca y el código necesarios para generar el diagrama y realicé las llamadas pertinentes para extraer los datos del campo escogido. Después, llegó el momento de generalizar el código para que fuera capaz de extraer los datos de todos los campos que quisiéramos mostrar. Primero realicé cambios en el fichero que genera las consultas para obtener todos los datos sin tener que generar llamadas a la base de datos para cada elemento. También tuve que realizar cambios en el código para generar el diagrama y que este pudiera pintarlo sin saber de antemano qué elemento iba a mostrar.

Pruebas

Las pruebas han sido una fase que se ha ido realizando un poco de la mano de la implementación. Por ello, las pruebas más exhaustivas que he realizado han sido sobre todo las que están relacionadas con la parte que he realizado yo. Pero también he probado las partes que han hecho mis compañeros, ya que cada vez que iba a realizar una tarea mía antes observaba el estado en el que mis compañeros habían dejado el proyecto y podía comprobar si sus partes se comportaban de forma correcta.

Memoria

En esta etapa, una vez obtenida la plantilla por mi compañero Rubén, listamos los capítulos según los cuales íbamos a dividir la memoria. Nos dividimos estos para que cada uno hiciese ciertas partes. Yo en concreto me encargué del capítulo 4. Aparte de escribir mi capítulo también he leído la parte que han hecho mis compañeros con el fin de contribuir o corregir algún detalle que estuviese pendiente.

7.2. Jesús Martín González

Investigación

En la fase de investigación todos los integrantes del grupo comenzamos a introducirnos en los aspectos básicos que rodean ya no solo al *browser fingerpriting* sino en general a las formas de detección de huellas digitales. En mi caso, ya contaba con una serie de nociones básicas sobre estos procedimientos, ya que soy un entusiasta de la privacidad en dispositivos digitales. Aun así, mis conocimientos no iban mucho más allá de haber explorado distintas extensiones de navegador que podían evadir ciertas técnicas de detección de huella digital.

La fase de investigación fue muy general, ya que iniciamos una búsqueda de todo tipo de material entre todos que nos pudiera ser de utilidad, como herramientas, bibliotecas y demás. En mi caso me centré en recopilar publicaciones y documentación de interés, que luego han sido aprovechados para ayudarnos a escribir nuestra memoria. Esto me ayudó a comprender la diferencia entre las técnicas activas y pasivas de rastreo de huella digital y a distinguir las ventajas e inconvenientes que plantea frente al uso de *cookies*. Por último, también colaboré en la elaboración de una lista de posibles tecnologías a usar en nuestro proyecto.

Análisis de requisitos

En la fase de análisis de requisitos valoramos las distintas opciones que barajábamos, y decidir así con cuales nos quedaríamos. Mi tarea consistió en realizar pruebas de concepto con diversas tecnologías de servidores web, concretamente con Django y Node.js. Finalmente descartamos ambas y nos quedamos con Apache, como hemos explicado en la sección 3.2.5. Además, también planteé la adopción del sistema de base de datos MongoDB. La conclusión fue que una base de datos relacional se adaptaba mejor a nuestras necesidades, por lo que también descartamos esta opción. Elegimos PHP y JavaScript, como los lenguajes de programación que utilizaríamos. Aunque ya había trabajado con ellos anteriormente, hacia bastante tiempo que no los utilizaba así que en mi caso también tuve que ponerme al día y repasar conceptos que tenía olvidados.

También se analizó qué componentes del navegador recolectar con JavaScript a través de WebAPI. Me ocupé de identificar cómo podríamos obtener información relacionada con el protocolo de red, como es la IP origen, los DNS utilizados o información sobre la WebAPI WebRTC. Descartamos recopilar estos atributos al no considerarlos tan relevantes como otras características del propio navegador. Por último, me encargué de analizar cuáles son los elementos obtenidos desde distintas fuentes que se repetían. La intención era comprobar si arrojaban valores diferentes para detectar posibles técnicas de evasión. Esta funcionalidad se ha dejado como trabajo futuro.

Implementación

Mi primera labor relacionada con la implementación del proyecto fue entender la estructura de la aplicación, que había sido diseñada por Rubén. Este primer esquema se basaba en la primera prueba de concepto que había sido realizada por Rubén, Adrián y Samuel utilizando

Apache y PHP. Como ya he comentado anteriormente, me llevó un tiempo el volver a familiarizarme con el lenguaje PHP. Mi primera contribución de código fue colaborar con Rubén para hacer la extracción de datos de la cabecera HTTP lo más flexible posible. Una vez me iba poniendo más al día con el lenguaje, pude contribuir a más tareas. Tras esto, colaboré en la extracción de atributos con JavaScript a través de WebAPIs: primero mejorando la detección de fuentes del navegador, tarea que había sido iniciada por Adrián; y posteriormente mediante la detección de elementos Flash. Esta tarea fue desarrollada junto con Rubén, ya que él mejoró bastante mi primera implementación inicial.

Mi contribución más importante en este apartado es el cálculo del porcentaje de similaridad que hay entre el usuario que accede a la página web y el resto de valores de los usuarios que han accedido anteriormente. Lo interesante de la implementación es que se pueden reutilizar secciones del código y consultas SQL que en un principio estaban destinadas solo para identificar si un usuario había visitado nuestra plataforma y para pintar los valores de los atributos. Más tarde, mejoré esta funcionalidad protegiendo las consultas ante posibles ataques de *SQLInjection*. También he contribuido a mejorar otras funcionalidades de la aplicación en momentos puntuales. Otras tareas también fueron llevadas a cabo pero finalmente no se adoptaron, como la generación de certificados SSL en local.

Pruebas

Hacer pruebas con sistemas no tan comunes para el público general fue una de mis tareas a realizar. En concreto, se efectuaron pruebas desde los sistemas operativos Ubuntu y Debian, y desde los navegadores Brave, Waterfox, Tor Browser y Lynx, además de pruebas puntuales desde la herramienta cURL. El objetivo era verificar si nuestra aplicación era capaz de funcionar fuera de la plataforma XAMPP en Windows y obtener información de los navegadores menos comunes. También colaboré en alojar nuestra aplicación en una web de *hosting*, aunque sin éxito. Rubén retomó esta tarea y la llevó a cabo de forma satisfactoria. En la fase de pruebas tan solo encontré pequeños fallos no muy importantes que fueron subsanados.

Memoria

Respecto a la memoria, me he encargado de escribir los capítulos 3 y 6 de Preliminares y Uso del sistema, respectivamente. A la hora de repartir el trabajo de la memoria, pensamos que iba a ser más interesante que me encargara yo de los preliminares porque algunos de los documentos académicos que habíamos recopilando anteriormente ya me resultaban familiares y me iba a ser más sencillo redactar los antecedentes del *browser fingerprinting*. También he participado en la elaboración del capítulo 8 de Conclusiones junto con Samuel. Tal como nos habíamos asignado los capítulos, tenía sentido que nosotros dos comenzáramos a redactarlo, enunciando los resultados finales y que lo concluyeran Adrián y Rubén con más detalles sobre el trabajo futuro de la implementación y sobre los problemas con los que nos habíamos encontrado.

Por último, además de participar en la revisión del resto de partes redactadas por mis compañeros, destacar también mi colaboración en la traducción al inglés de los capítulos 1 y 8 de Introducción y Conclusiones, respectivamente. Aunque esta tarea ha sido realizada por Samuel, yo he prestado apoyo introduciendo algunas mejoras en la traducción.

7.3. Rubén Peña García

Investigación

En esta fase del proyecto comencé informándome acerca de los distintos métodos de *browser fingerprinting* que existen, ya que mi conocimiento de la materia era casi nulo. Principalmente examiné el funcionamiento de otras aplicaciones capaces de obtener la huella digital del navegador, como *AmIUnique* [2] o *Panopticlick* [6]. También me documenté acerca de los usos del reconocimiento de la huella digital de navegadores y su impacto gracias al estudio publicado por la Agencia Española de Protección de Datos [1].

Después, mi investigación consistió en comprobar y listar todos los elementos que íbamos a obtener en nuestra aplicación y su método de obtención, para poder determinar el alcance de nuestro proyecto.

Para los datos que obtenemos de las cabeceras HTTP analicé los elementos que conformaban las cabeceras HTTP en los navegadores más utilizados, como *Google Chrome*, *Microsoft Edge*, *Microsoft Internet Explorer* o *Mozilla Firefox*. De todos los resultados obtenidos listé los más comunes para su almacenamiento en nuestra base de datos.

También investigué elementos que podíamos obtener mediante el uso de JavaScript, comprobando en otras aplicaciones los atributos que obtenían mediante esta tecnología. Algunas, como *AmIUnique*, listan todos los elementos para que el usuario pueda verlos, pero en muchos casos no explican su método de obtención. Mi investigación en este ámbito consistió en examinar los distintos atributos que obtienen estas aplicaciones por un lado y encontrar su forma de obtención por otra parte. Además investigué distintos objetos de JavaScript, examinando los métodos de sus clases por si alguno podía servirnos a la hora de identificar a un usuario añadiendo información a su huella.

Análisis de requisitos

Una vez tuvimos claro el funcionamiento que tendría nuestra aplicación y la manera de obtener los datos para generar la huella investigamos qué tecnologías utilizaríamos. Repartimos la investigación de tecnologías, sobre todo en lado de servidor, ya que estaba claro que en cliente tendríamos que utilizar HTML y JavaScript.

Mi tarea fue investigar *frameworks* JavaScrpt para el servidor. El que más estudié fue *Node.js* [31], ya que es uno de los *frameworks* más extendidos y famosos. Al concluir todas nuestras investigaciones y poner nuestras conclusiones en común decidimos utilizar *Apache* para nuestro servidor, ya que ofrecía muchas comodidades y todos los integrantes del trabajo habíamos trabajado con él anteriormente, lo que nos permitía trabajar de manera inmediata sin necesidad de aprender a utilizar un *framework* desconocido.

Implementación

Esta etapa del proyecto fue la más larga y compleja. Al comienzo de esta me encargué de diseñar la estructura de una aplicación de prueba para obtener datos del navegador. Se trataba de

una versión básica de lo que finalmente terminaría siendo nuestra aplicación. En ella programé lo necesario para identificar cada uno de los atributos de las cabeceras HTTP y mostrarlos por pantalla. Todo este código se almacenó en un repositorio de *GitHub* que yo creé.

Posteriormente diseñé e implementé la base de datos con PhpMyAdmin, que en principio solo almacenaba en una única tabla el ID y los elementos de la cabecera HTTP. Esta base de datos iría aumentando su número de tablas y atributos a lo largo del desarrollo, ya que añadíamos las columnas según íbamos implementando los métodos de obtención de atributos de la aplicación.

Tras terminar la versión de prueba comenzamos la programación de los métodos encargados de obtener información mediante JavaScript. Me encargué de seguir desarrollando la estructura del código de la aplicación, de forma que este fuese lo más modular posible para que cada uno de los integrantes pudiésemos trabajar en paralelo de la forma más cómoda posible.

Después de completar la estructura del código comenzamos a programar las funciones para la obtención de elementos con JavaScript. Los integrantes del grupo nos fuimos repartiendo estas tareas, las cuales teníamos listadas desde la etapa de investigación. Cada uno programábamos por separado estas funciones y la íbamos añadiendo al código de la aplicación según las íbamos terminando. Entre otros me encargué de muchos de los elementos obtenidos del objeto *Navigator*, del objeto *Screen*, de *Canvas*, de algunos *plugins* como *AdBlock* o *Flash*, la batería, el objeto *Window*, los códecs de audio y vídeo o los dispositivos del sistema.

Una vez consideramos que no era necesario la obtención de más atributos para nuestra aplicación pasamos a implementar el resto de funcionalidades. Entre estas funcionalidades yo me dediqué a programar el código necesario para obtener la unicidad de la huella del usuario. También añadí la herramienta de información que hay en nuestra aplicación, la cual detalla la utilidad de cada uno de los elementos analizados en la página. Además me encargué de que el código en cliente pudiese enviar la información al servidor de manera asíncrona para no tener que recargar la página.

Aunque el encargado de la apariencia estética de la aplicación fue Samuel, yo también programé parte de esta, cambiando algunos colores y aplicando la misma apariencia al apartado de gráficas de nuestra aplicación.

Finalmente buscamos un alojamiento en línea que fuese gratuito para nuestra aplicación. Jesús encontró uno en *Education Host* y yo me encargué de crear una cuenta en este servicio e instalar nuestra aplicación en él. Hubo que hacer algunos cambios en el código, ya que el servidor no era del todo compatible con algunas partes de nuestro código, pero en general fue bastante sencillo y cómodo, ya que este servicio ofrece herramientas para ejecutar PHP y bases de datos *MariaDB*.

Pruebas

Aunque durante el desarrollo del código de la aplicación fuimos comprobando que el código funcionase de manera correcta, fue cuando terminamos de programar cuando nos encontramos con varios problemas. Durante estas pruebas encontramos el que posiblemente era el mayor problema, la asincronía. Debido a la forma en la que realizábamos la asincronía, en algunas ocasiones se ejecutaban las llamadas asíncronas fuera del orden deseado, lo cual afectaba al resultado de la unicidad del usuario. Yo me ocupé de solucionar este problema, haciendo que las llamadas asíncronas se ejecutasen en orden.

Memoria

Para la realización de la memoria me dediqué a buscar una plantilla de LaTeX para que nos sirviese de base para la escritura de esta memoria. Me encargué de modificar algunas cosas y adaptarla a nuestro gusto. Una vez lista nos repartimos la escritura de los capítulos entre los integrantes del grupo. Yo me encargué de la escritura del capítulo 5 y parte del capítulo 8.

Además de la escritura de estos capítulos, todos los integrantes hemos ido revisando los capítulos realizados por el resto de compañeros. De esta forma hemos podido añadir información que se nos hubiese podido olvidar individualmente y corregir errores en común.

7.4. Samuel Solo de Zaldívar Barbero

Investigación

En referente a la primera etapa, la labor primordial era documentarse sobre las distintas formas de captar datos de un dispositivo. Como antecedente, el único método que conocía eran las *cookies*, por lo que ha sido un mundo completamente nuevo para mí y para la mayoría de nosotros. El objetivo era conocer qué datos se podían recoger, de qué tipo son y qué se puede hacer con ellos. Para ello se ha de navegar por las distintas páginas que se encargan de obtener la huella digital del navegador y ciertos estudios relacionados con la misma. Una vez analizadas estas fuentes, se crea una lista con los distintos atributos y se decide en base a ellos si son útiles o no a la hora de determinar la huella digital del navegador.

Análisis de requisitos

En esta segunda etapa del proyecto y teniendo claro cómo iba a funcionar nuestro proyecto, el objetivo era investigar qué tecnologías íbamos a emplear. Mi labor consistió en estudiar tecnologías como Django y Node.js para la parte del servidor web, pero finalmente las descartamos y nos decantamos por usar Apache. Esto es debido a que anteriormente todos los integrantes del proyecto ya habíamos usado esta herramienta, por lo que era mucho más sencillo y cómodo para nosotros.

Implementación

En la tercera etapa y la más laboriosa del proyecto, Rubén elaboró lo que sería la base del mismo. Tras haberla revisado, nos centramos en la recolección de datos de las cabeceras HTTP y posteriormente de los atributos con JavaScript. Mi labor fue identificar los distintos permisos que existen en los navegadores, como la geolocalización o el uso del micrófono y la cámara web. Esta implementación requería de conocimientos de *callbacks*, por lo que resultó bastante difícil en un principio hasta que Rubén logró implementar la idea principal. Luego se descartó el uso de estos atributos porque su implementación resultaba muy complicada y los datos que se podían recoger no resultaban fundamentales para la obtención de la huella digital.

Ya una vez teníamos las distintas funcionalidades de la aplicación, nos centramos en

la apariencia. En esta parte me encargué del CSS del proyecto. El objetivo, a parte de darle un mejor aspecto, es que el usuario pueda identificar con más claridad los elementos de la aplicación. En esta parte Rubén contribuyó de forma notable para introducir ciertos cambios y añadir el estilo a la sección de gráficas.

Pruebas

Respecto a la cuarta etapa, al haberme ocupado del aspecto de la aplicación, mi labor fue verificar que los elementos de la página se mantenían estables probando, por ejemplo, distintos navegadores o distintas resoluciones de pantalla. También era necesario comprobar que, según lo que devolvieran los distintos campos de las tablas, estos no se salían del rango que en teoría estaba determinado y así no descuadraba todo el contenido. Esto último resultó ser un problema pero fue arreglado en actualizaciones posteriores.

Memoria

En cuanto a la última etapa, me he ocupado en concreto del resumen, el capítulo 1 y parte del capítulo 8. A su vez, he realizado las traducciones de las partes que así lo requerían, como en el resumen, el capítulo 2 y el capítulo 9. Esta tarea de traducción se realizó en gran medida con ayuda de Jesús.

Para finalizar, a parte de la escritura de los capítulos citados anteriormente, he repasado las distintas secciones que han redactado los demás integrantes para asegurar que tanto el contenido como el formato eran los correctos.

Capítulo 8

Conclusiones

El principal objetivo de este proyecto ha consistido en crear una aplicación web que permita recopilar los datos de un dispositivo y elaborar un perfil de su huella digital. Tras la implementación de la aplicación, podemos señalar que se han cumplido con los objetivos y las expectativas que habíamos planteado en un principio. Sin embargo, ciertas tareas secundarias se han visto alteradas debido a distintos factores como la tecnología y el tiempo que disponíamos para cumplirlas, y se proponen como tareas futuras.

La técnica de *browser fingerprinting* recopila información del usuario que por lo general permanece invariable, por lo que podemos afirmar que es un método muy efectivo. No es necesario hallar un atributo único sino que es suficiente encontrar una combinación de atributos que resulte única para poder identificar a un usuario. Las *cookies* son un método más eficaz pero si se eliminan, se pierde la trazabilidad sobre el usuario.

Es prácticamente inevitable perder ciertas funcionalidades al evadir técnicas de perfilado, aunque es posible mitigar su impacto ya sea con la configuración de determinados parámetros o el enmascaramiento de la información con extensiones del navegador.

La recopilación de datos que puedan identificar al usuario se consideran datos de carácter personal y está protegidos por la legislación Reglamento General de Protección de Datos (RGPD). Aun así, nuestro proyecto ayuda a que los usuarios tomen conciencia y que comprendan que sus datos pueden ser recopilados de forma silenciosa, lo cual supone un riesgo para la privacidad.

Por otra parte, en el ámbito personal hemos adquirido y asimilado conocimientos en JavaScript, sobre todo el manejo de conexiones asíncronas con AJAX. Además, hemos aprendido a acceder a las distintas WebAPIs que nos brindan los navegadores. También, hemos mejorado nuestra organización del trabajo a través de Git, que nos ha permitido un control de versiones óptimo.

8.1. Trabajo futuro

Durante el trabajo hemos dejado funcionalidades o mejoras que se podrían realizar en el futuro para mejorar el desempeño de la aplicación.

Asincronía

En nuestra aplicación hemos utilizado el objeto «XMLHTTPRequest» de JavaScript para la conexión asíncrona con el servidor. Se trata de un objeto que empieza a estar obsoleto y que ha sido reemplazado por «Fetch», que permite acceder y manipular partes del canal HTTP de manera mas sencilla y cómoda.

Además se puede buscar una mejor solución al problema de la asincronía mencionado anteriormente.

Interfaz gráfica

La interfaz gráfica de nuestro proyecto es un punto en el que se podrían realizar varias mejoras. La página inicial, se puede adornar mediante el uso de Frameworks de JavaScript como React, Vue.js y derivados, dándole así un toque más moderno.

Respecto a la vista de las estadísticas, se puede realizar una modificación de la gráficas, de forma que al dibujarse estas se resalte la porción a la que pertenece el navegador del usuario que la está viendo.

También se puede añadir una mejora que permita al usuario ver una nueva gráfica al seleccionar un campo de otra, en la que se detalle información de la anterior. Por ejemplo, en la gráfica de sistema operativo, al pulsar sobre uno de los sistemas se mostraría una nueva gráfica en la que se pudiese ver el porcentaje de cada una de las versiones de ese sistema.

Analizar otros elementos del navegador

Consideramos que nuestra aplicación obtiene información de suficientes elementos mediante JavaScript, pero durante el desarrollo dejamos varios por implementar por falta de tiempo. Entre ellos se podrían añadir los siguientes.

- **WebGL**: Se trata de una API para JavaScript que permite renderizar gráficos en 3D. Se puede utilizar de forma similar a nuestra implementación de «Canvas».
- **Sensores**: Mediante JavaScript se puede obtener información de sensores del dispositivo, como el acelerómetro o el giroscopio.

Análisis de resultados en intervalos de tiempo

Esta mejora consiste en poder observar distintos ratios de los atributos del navegador para distintos intervalos de tiempo. De esta forma se podrían ver los resultados de las huellas en los últimos 15 días o en el último mes. Es una mejora que teníamos en mente desde el inicio del proyecto, incluso almacenamos la fecha en la tabla *Conexiones* de la base de datos, pero por falta de tiempo no hemos podido llevarla a cabo.

Soporte para HTTP2

Actualmente, nuestro servidor web sólo permite conexiones HTTP1/1. Dar soporte a la versión HTTP/2 implicaría mejoras en el rendimiento, ya que HTTP2 se trata de un formato binario, a diferencia de HTTP1/1 que es texto plano. Otras funcionalidades que incorpora y que creemos que pueden mejorar nuestro proyecto es la multiplexación de conexiones entrantes y la capacidad de enviar datos al cliente antes de que él los pida.

Ha sido una opción que estuvimos investigando y llegamos a la conclusión de que además necesitábamos incluir certificados SSL/TLS. Esto es así ya que, aunque la especificación del protocolo HTTP/2 permite prescindir del cifrado, varias implementaciones declararon que sólo soportarán HTTP/2 cuando se utilice a través de una conexión cifrada, y actualmente ningún navegador soporta HTTP/2 sin cifrar [32].

Detectar técnicas de evasión de rastreo

Existen varios atributos que guardamos duplicados, ya que los recopilamos de fuentes diferentes. Habíamos pensado que se pueden realizar comprobaciones para detectar si el resultado de ambas fuentes es consistente entre sí. En caso contrario, estaríamos ante una evidencia de evasión del rastreo de huella digital y podríamos identificar qué campos se están tratando de enmascarar.

Capítulo 9

Conclusions

The main objective of this project has been to create a web application that will allows the collection of data from a device and the creation of a profile of its fingerprint. After the implementation of the application, we can say that the objectives and expectations we had originally set have been fulfilled. However, certain secondary tasks have been altered due to different factors such as technology and the time, and are proposed as future tasks.

The technique of browser fingerprinting collects information from the user that usually remains unchanged, so we can say that it is a very effective method. It is not necessary to find a unique attribute but it is enough to find a combination of attributes that is unique to identify a user. Cookies are a more effective method but if they are removed, the traceability on the user is lost.

It is practically inevitable to lose certain functionalities when evading fingerprinting techniques, although it is possible to mitigate their impact either by configuring certain parameters or by masking information with browser extensions.

The collection of data that can identify the user is considered personal data and is protected by the General Data Protection Regulation (GDPR). Even so, our project helps users to become aware and understand that their data may be collected silently, which is a risk to privacy.

On the other hand, on a personal level we have acquired and assimilated knowledge in JavaScript, especially the handling of asynchronous connections with AJAX. In addition, we have learned how to access the different WebAPIs provided by browsers. We have also improved our work organization through Git, which has allowed us optimal version control.

9.1. Future Work

While we were working we left functionalities or improvements that could be made in the future to improve the performance of the application.

Asynchrony

We have used the JavaScript object «XMLHTTPRequest» for the asynchronous connection to the server. This is an object that is becoming obsolete and has been replaced by «Fetch», which allows you to access and manipulate parts of the HTTP header more easily and conveniently.

In addition, a better solution can be found to the problem of asynchrony mentioned above.

Graphical interface

The graphical interface of our project is a point where several improvements could be made. The home page can be decorated by using JavaScript Frameworks such as React, Vue.js and derivatives, giving it a more modern look.

With respect to the view of the statistics, a modification of the graphs can be made, so that when they are drawn, the portion to which the browser of the user who is viewing them belongs is highlighted.

You can also add an enhancement that allows the user to view a new graph by selecting a field from another, which details information from the previous one. For example, in the operating system graph, if you click on one of them, a new graph will be shown where you could see the percentage of each one of the versions of that system.

Analyze other browser elements

We believe that our application gets enough information from JavaScript, but during development we left several to be implemented due to lack of time. Among them we could add the following.

- **WebGL**: This is a JavaScript API that allows you to render 3D graphics. It can be used in a similar way to our implementation of «Canvas».
- **Sensores**: Using JavaScript, information can be obtained from device sensors, such as the accelerometer or gyroscope.

Analysis of results in time intervals

This improvement consists in being able to observe different ratios of the browser attributes for different time intervals. In this way you could see the results of the fingerprints in the last 15 days or in the last month. It is an improvement that we had in mind from the beginning of the project, we even stored the date in the table *Conexiones* in the database, but due to lack of time we have not been able to carry it out.

Support for HTTP2

Currently, our web server only allows HTTP1/1 connections. Supporting the HTTP/2 version would mean performance improvements, since HTTP2 is a binary format, unlike HTTP1/1 which is plain text. Other features that we have built in and that we believe can improve our project are the multiplexing of incoming connections and the ability to send data to the client before he requests it.

This was an option that we investigated and concluded that we also needed to include SSL/TLS certificates. This is because, although the HTTP/2 protocol specification allows you to dispense with encryption, several implementations have stated that they will only support HTTP/2 when used over an encrypted connection, and currently no browser supports unencrypted HTTP/2 [32].

Detect tracking avoidance techniques

There are several attributes that we keep duplicates of, as we collect them from different sources. We had thought that checks could be made to detect whether the result from both sources is consistent with each other. If not, we would be looking at evidence of evasion of fingerprint tracking and could identify which fields are being masked.

Índice de figuras

Índice de tablas

Bibliografía

[1] Agencia Española de Protección de Datos. Estudio de fingerprint o huella digital de dispositivo. Technical report, AEPD, 2019.

[2] AmIUnique, Fecha de acceso, octubre de 2019. https://amiunique.org.

[3] Adam Barth. HTTP State Management Mechanism. *RFC*, 6265:1–37, 2011.

[4] Nasser Mohammed Al-Fannah, Wanpeng Li, and Chris J. Mitchell. Beyond Cookie Monster Amnesia: Real World Persistent Online Tracking. *arXiv: Cryptography and Security*, 2019.

[5] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Krügel, Frank Piessens, and Giovanni Vigna. Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. *2013 IEEE Symposium on Security and Privacy*, pages 541–555, 2013.

[6] Peter Eckersley. How Unique Is Your Web Browser? In *Privacy Enhancing Technologies*, 2010.

[7] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting Information in JavaScript Implementations. 2011.

[8] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. *Proceedings of the 2018 World Wide Web Conference*, 2018.

[9] Browser Fingerprinting: An Introduction and the Challenges Ahead, Fecha de acceso, octubre de 2019. https://blog.torproject.org/browser-fingerprinting-introduction-and-challenges-ahead.

[10] The GDPR and Browser Fingerprinting: How It Changes the Game for the Sneakiest Web Trackers, Fecha de acceso, mayo de 2020. https://www.eff.org/es/deeplinks/2018/06/gdpr-and-browser-fingerprinting-how-it-changes-game-sneakiest-web-trackers.

[11] REGLAMENTO (UE) 2016/679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/ce (Reglamento General de Protección de Datos). L 119/1, 2016-04-27.

[12] Browser Fingerprinting [2020 Update] – What Is It How to avoid It, Fecha de acceso, mayo de 2020. https://blokt.com/guides/browser-fingerprinting.

[13] This is Your Digital Fingerprint, Fecha de acceso, mayo de 2020. https://blog.mozilla.org/internetcitizen/2018/07/26/this-is-your-digital-fingerprint/.

[14] Gunes Acar, Marc Juárez, Nick Nikiforakis, Claudia Díaz, Seda F. Gürses, Frank Piessens, and Bart Preneel. FPDetective: dusting the web for fingerprinters. In *CCS '13*, 2013.

[15] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juárez, Arvind Narayanan, and Claudia Díaz. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *CCS '14*, 2014.

[16] Keaton Mowery and Hovav Shacham. Pixel Perfect : Fingerprinting Canvas in HTML 5. 2012.

[17] W3C Audio Working Group. Web Audio Processing: Use Cases and Requirements. Technical report, W3C, January 2013.

[18] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Díaz. The Leaking Battery - A Privacy Analysis of the HTML5 Battery Status API. *IACR Cryptol. ePrint Arch.*, 2015:616, 2015.

[19] W3C Interest Group. Mitigating Browser Fingerprinting in Web Specifications. Technical report, W3C, March 2019.

[20] Browser Fingerprinting: What Is It and What Should You Do About It?, Fecha de acceso, mayo de 2020. https://pixelprivacy.com/resources/browser-fingerprinting/.

[21] Browser Fingerprinting – Explanation, Tests, Solutions, Fecha de acceso, mayo de 2020. https://restoreprivacy.com/browser-fingerprinting/.

[22] Tecnología web para desarrolladores, Fecha de acceso, octubre de 2019. https://developer.mozilla.org/es/docs/Web.

[23] ¿Qué es XAMPP?, Fecha de acceso, junio de 2020. https://www.apachefriends.org/es/index.html.

[24] Guia Google Charts, Fecha de acceso, abril de 2020. https://developers.google.com/chart/interactive/docs.

[25] FingerprintJS2, Fecha de acceso, octubre de 2019. https://github.com/Valve/fingerprintjs2.

[26] Navigator, Fecha de acceso, diciembre de 2019. https://developer.mozilla.org/es/docs/Web/API/Navigator.

[27] Screen, Fecha de acceso, diciembre de 2019. https://developer.mozilla.org/es/docs/Web/API/Screen.

[28] Window, Fecha de acceso, diciembre de 2019. https://developer.mozilla.org/es/docs/Web/API/Window.

[29] Formatos de audio y video, Fecha de acceso, noviembre de 2019. https://developer.mozilla.org/es/docs/Web/HTML/Formatos_admitidos_de_audio_y_video_en_html5.

[30] Rebecca M. Riordan. *Head First Ajax*. O'Reilly Media, Inc., 2008.

[31] George Ornbo. *Node.js*. Anaya Multimedia, 2013.

[32] HTTP/2 Frequently Asked Questions, Fecha de acceso, junio de 2020. https://http2.github.io/faq/.

PASCAL
ENERO 2018
Ult. actualización 26 de junio de 2020
LaTeX lic. LPPL & powered by TEFLON CC-ZERO