Unichat: aplicación de chat para universidades

Pol Gómez Nogués

Resumen– A día de hoy, las aplicaciones web son una herramienta fundamental para ofrecer un servicio online. Paralelamente, ha aumentado la demanda de los usuarios para poder comunicarse entre sí utilizando dichos servicios. Debido a esta necesidad nacen muchos paradigmas para confeccionar dicho tipo de aplicación, haciendo que la elección de éstos varíe en función del fin a alcanzar. Si se indaga sobre servicios de comunicación entre universidades, actualmente no existe una aplicación eficaz que permita comunicarse, a partir de un chat, con los diferentes integrantes de una universidad. Por eso mismo, se idea Unichat, un proyecto que permite a los usuarios de una misma organización, registrarse y empezar a conversar entre ellos. Ahora bien, teniendo en cuenta que cada usuario puede desempeñar un rol dentro de la aplicación ya sea como: alumno, profesor o administrador de la organización.

Palabras clave– Crypto, chat, mySQL, nodeJS, pug, redis, rol, servicio web, software y universidad.

Abstract– Today, web applications are a fundamental tools to offer a service. The demand of users to be able to communicate with each other using these services has increased. Due to this, many paradigms have been designed to resolve these situations, considering the specific purpose to be achieved for each organization. If you inquire about communication services between universities, there is currently no effective application that allows you to communicate, through a chat, with the different members of a university. For this reason, Unichat was created, as a project that allows users of the same organization to register and start conversing with each other. However, the users roles within the application must be considered, such as: student, teacher or administrator of the organization.

Keywords– Crypto, chat, mySQL, nodeJS, pug, redis, role, web service, software and university.

1 Introducción

Durante los últimos años las aplicaciones de comunicación entre usuarios han incrementado su popularidad hasta el punto que, a día de hoy, es imprescindible disponer de algún programa para ello. España es un ejemplo, dado que el 94,6 % de ciudadanos usa este tipo de tecnologías [1]. Esta necesidad de comunicación, se hace más notoria en las universidades, donde los alumnos y profesores se ponen en contacto a través de un foro o correo electrónico con todos los inconvenientes que plantea. Por otro lado, la facilidad de poder usar aplicaciones web sin una instalación previa se ha convertido en una necesidad por la comodidad y ventajas que ofrece al usuario. Dicha facilidad también se aplica a los lenguajes usados para el desarrollo, siendo NodeJS uno de los más relevantes [2]. Ahora bien, al crecer, también aumenta el riesgo de recibir ataques y alterar el funcionamiento normal, por lo que se debe tener en cuenta la seguridad, dado que una pequeña vulnerabilidad puede suponer un gran riesgo [2].

Así pues, para resolver la problemática de comunicación entre alumnos y profesores de una universidad, se plantea el desarrollo de Unichat. Dicho servicio se encuentra realizado en NodeJS, que permite establecer diferentes canales de chat. Éstos se organizan según las materias que cursen, creando una sala sólo de alumnos y otra de alumnos y profesores. Además, tiene otras funcionalidades como acceder a el perfil de usuario, gestionar las diferentes salas existentes y, otras características que se verán en las siguientes secciones. El desarrollo de la aplicación, se realiza con diferentes tecnologías, como Express o Redis, las cuales se explican

• E-mail de contacto: polgn1995@gmail.com
• Mención realitzada: Tecnologias de la Informació
• Trabajo tutorizado por: Jordi Casas Roma (Área de Ciéncias de la Computación e Inteligencia Artificial)
• Curso 2019/20

más adelante. Además, la aplicación estará publicada en un servidor en la nube, por lo que se deberá tener en cuenta que configuración realizar para proveer dicho servicio a los usuarios sin que haya problemas. Cabe remarcar la utilización de buenas prácticas, metodologías, un buen diseño y herramientas que permitan crear una aplicación segura, tanto para el usuario como para ella misma. Todo queda reflejado en este documento a partir de los objetivos, la planificación, el estado del arte y el desarrollo de la propia aplicación.

2 Objetivos

El objetivo principal del proyecto es crear una aplicación de chat que sea útil para los usuarios. No obstante, para alcanzar dicha meta, se deben cumplir los siguientes objetivos que se muestran a continuación.

- Seleccionar las tecnologías a utilizar debe hacerse pensando en cuales de ellas ofrecen una mejor experiencia para el usuario. Para ello, se debe comparar entre diferentes opciones y ver cual de ellas se adapta más al proyecto que se quiere realizar.
- Diseñar la estructura interna del programa, utilizando buenas prácticas para evitar futuros problemas de consistencia, pérdida de tiempo y seguridad [3]. Dichas buenas prácticas, consisten en la utilización de patrones, en el uso las herramientas necesarias en cada ocasión y en la modularización para poder realizar cambios futuros.
- Diseñar la interfaz de usuario. Ésta debe cumplir algunos requisitos, como ser intuitiva, fácil de usar y permitir que el usuario se sienta cómodo con la aplicación. Además, esto implica que debe ser escalable y en todo momento el usuario debe tener la percepción que la aplicación funciona correctamente o, de lo contrario, informar con el mensaje correspondiente.
- Securizar la aplicación y los datos es un aspecto muy importante a tener en cuenta, puesto que una vulnerabilidad podría perjudicar tanto a los usuarios como la propia aplicación. Para ello, se debe escoger un proxy, garantizar que el sitio web utiliza HyperText Transfer Protocol Secure (HTTPS) y, encriptar parte de los datos internos.
- Realizar una buena configuración de red, del sistema y realizar pruebas para rectificar los posibles problemas que se puedan encontrar.

3 Metodología

Para desarrollar este proyecto, se ha planteado seguir una metodología en espiral [16], lo que implica realizar diferentes fases de forma cíclica hasta que se de el proyecto por terminado. La primera iteración comienza determinando los diferentes objetivos y el abasto total de la aplicación. También, se fijan las restricciones, se identifican los puntos más sensibles del proyecto y se realiza una planificación inicial.

A continuación, se buscan los riesgos potenciales y se valoran las posibles soluciones o rutas alternativas que se pueden tomar. Una vez se tienen definidos los puntos anteriores, se empieza el desarrollo, la validación y la prueba. Durante esta fase se implementa la aplicación y se valida la cobertura de todos estos riesgos. Además, también se deben realizar pruebas para comprobar que sea correcto el trabajo realizado.

Por último, se debe planificar que se hará en la siguiente iteración, la cual, empezará por la definición de objetivos en función a la última iteración que se ha realizado.

4 Planificación

Para desarrollar este proyecto, se ha necesitado un total de dieciocho semanas donde se ha seguido la planificación descrita en la Tabla 1.

Semana	Objetivo
1-4	Definición del proyecto.
4-9	Diseño interno, externo y primera versión simple de la aplicación.
9-15	Implementación de la versión final y subida a producción.
15-18	Redacción del artículo final y cierre del proyecto.

TABLA 1: Planificación.

Las cuatro primeras semanas han servido para definir el proyecto. Esto implica indicar cuál es el problema que se quiere resolver y ver que soluciones se brindan actualmente, marcar los objetivos a los que se quiere llegar, seleccionar las diferentes herramientas que se quieren utilizar, así como contrastarlas con otras para elegir las que sean óptimas para este proyecto, y documentarlo todo de forma escrita.

A partir de las siguientes cinco semanas se procede a diseñar la estructura interna del programa, la interfaz de usuario, implementar una primera versión sencilla del chat con un inicio de sesión y la creación de usuarios nuevos. Además, durante este periodo de tiempo se habrán definido todas las rutas necesarias de la página inicial.

Durante las siguientes seis semanas se procederá a mejorar la primera versión del chat, establecer roles, optimizar el código fuente, realizar ajustes de seguridad, hacer pruebas para verificar que su funcionamiento es correcto y adquirir el dominio que se usará para el programa. Además, se procederá a poner la aplicación en producción, probar el funcionamiento con usuarios reales, observar el desarrollo de ésta y reajustar diferentes parámetros para cumplir con los objetivos marcados.

Finalmente, en las tres últimas semanas, se realizará el documento final donde se explica y detalla todas las decisiones que se han tomado, así como el desarrollo del propio proyecto. Cabe destacar que en el Apéndice se encuentra un diagrama de Gantt donde se detalla como se ira desarrollando la aplicación a lo largo del tiempo y las tareas a realizar.

5 Estado del arte

Actualmente existen muchas tecnologías para crear aplicaciones web, como NodeJS que se caracteriza por crear

programas de red rápidos y asíncronos [4]. No obstante, para crear Unichat se necesita algún tipo de *framework* para la gestión del servidor web, entre los que destacan Koa, Express y Strapi. El primero ofrece un buen rendimiento y una estructura modularizada. Aun así, no dispone de múltiples herramientas propias, sino que se deben buscar a partir de fuentes terceras aumentando el riesgo de exponerse a vulnerabilidades [5]. Strapi ofrece unas prestaciones similares a Koa, un entorno minimalista, poder generar aplicaciones *Representational state transfer* (REST) de forma sencilla y controlar los permisos de usuarios de forma simple. Sin embargo, no tiene soporte para *plugins*, no dispone de un gran ecosistema de *middleware* y es un *framework* que lleva poco tiempo en el mercado [6]. Por otro lado, Express ofrece un rendimiento similar a los dos anteriores y una serie de ventajas para los desarrolladores. Además, el propio *framework* permite un mayor grado de seguridad integrando distintas herramientas y gran variedad de *middlewares* para poder adaptar la aplicación a cada situación. Dado que este último ofrece más ventajas, es el *framework* que se utiliza para este proyecto [5][6].

Al ser una aplicación de chat se necesita mantener diferentes datos como el registro de conversaciones o, los usuarios registrados, por lo que se necesita un gestor de base de datos. Entre los más destacados están MySQL, PostgreSQL y Redis. Para este proyecto nos hemos decantado por una mezcla entre Redis y MySQL, dónde el primero es un gestor muy simple y rápido que permite usarlo como una caché. Por otro lado, MySQL, será la base de datos principal. La exclusión de PostgreSQL se justifica por la necesidad de rapidez, un sistema que sea asíncrono y un mayor rendimiento [7][8]. Además, ésta no ofrece tanta variedad de fuentes de documentación como MySQL. No obstante, Redis no ofrece un método de encriptación, por lo que se deberá escoger entre PassportJS o Crypto. El primero, es un middleware que permite controlar si una persona se encuentra o no registrada permitiendo definir como realizar dicho proceso de forma segura. Además, permite la integración con inicios de sesión de varias plataformas como Facebook, Google o Twitter. No obstante, para este proyecto toda esta infraestructura no es necesaria, sino que únicamente se requiere encriptar unos datos para que un posible atacante no pueda obtener su contenido. Por eso, se ha considerado mejor el uso de Crypto [1], una herramienta simple pero potente. Esta permite encriptar los datos con distintos algoritmos y operar con los resultados obtenidos. Al ser más simple, permite un mayor control y se adapta mejor al proyecto.

Adicionalmente, es necesario implementar un sistema para intercambiar diferentes mensajes entre el cliente y el servidor (WebSocket). Para ello, se ha decidido utilizar un conjunto de herramientas denominadas Socket.io que permiten establecer una conexión a tiempo real entre varios clientes web y un servidor. Cabe destacar que se encuentra desarrollada en JavaScript y tiene algunas ayudas de soporte como autoreconexion o detección de errores de red [14]. No obstante, existe SocketCluster como una posible alternativa que también permite realizar aplicaciones con varios clientes web que puedan comunicarse entre sí. Ahora bien, Socket.io ofrece varias analíticas a tiempo real del cliente, obtener un flujo de datos binario (permitiendo el envío de cualquier tipo de archivo) y diseñar una aplicación potente y segura, de forma más rápida. Además, dado que el *framework* principal va a ser Express, esta opción resulta una de las mejores por la integración que tiene con dicho *framework* [15].

Un aspecto muy importante es la interfaz gráfica que verá el usuario. Para ello, se usa la tecnología PUG, una forma de renderizar páginas HTML de forma más dinámica. Además, a diferencia de un archivo HTML puro, permite realizar funciones para decidir cuando mostrar un contenido o no. No obstante, existen soluciones parecidas como Embedded JavaScript templating (EJS) que ofrecen el mismo rendimiento y potencia pero, su sintaxis no es tan clara [9]. También se han escogido tres estilos de CSS: WebSlides, Materialize y Bootstrap. El primero, ofrece una muy buena solución para realizar páginas informativas y visualmente atractivas. Materialize es un CSS creado por Google [10], que permite crear interfaces muy cómodas y agradables para el usuario. Por último, Bootstrap ha sido la opción seleccionada para este proyecto, ya que es el paquete más completo de los analizados. Su elección se ha hecho pensando en cuando deba implementarse la vista para dispositivos móviles, dado que su filosofía consiste en priorizarlos [11].

El dominio se obtiene a partir de 1And1[2], dado que al comprarlo también se proporciona una cuenta de correo que se usa para verificar los registros de usuarios nuevos a la aplicación. Por otro lado, el servidor en la nube se encuentra alojado en DigitalOcean[3], una empresa que ofrece un servicio escalable y múltiples guías de configuración. Finalmente, se usa Nginx como proxy para controlar todas las peticiones entrantes, que a diferencia de Apache es más rápido y consume menos memoria [12].

6 Diseño

A partir de esta sección se verán los diferentes aspectos de diseño, gestión y configuración, para que la aplicación cumpla con todos los objetivos marcados inicialmente.

6.1. Base de datos

El diseño de la base de datos es uno de los más importantes dado que esta será la que contenga toda la información necesaria para el correcto funcionamiento de la aplicación. Cabe destacar que un mal diseño del modelo puede comportar el incumplimiento del objetivo de confeccionar una estructura usando buenas prácticas o el objetivo relacionado con la seguridad. Tal y como se ha mencionado en secciones anteriores, se utilizan dos sistemas de bases de datos diferentes: MySQL y Redis. Siendo la primera la principal que va a necesitar un diseño mucho más complejo y, la segunda un soporte que simplemente contiene una relación de llaves con información.

El diseño interno del primer tipo queda reflejado por el modelo de entidad relación de la figura 1. Dicho modelo contempla las organizaciones como entidades, las cuales, son identificadas a partir de un número y nombre. También, se encuentran los usuarios que son definidos por un identificador, nombre, correo electrónico, una contraseña y

[1] Más información en: https://nodejs.org/api/crypto.html

[2] Más información en: https://www.ionos.es/

[3] Más información en: https://www.digitalocean.com/products

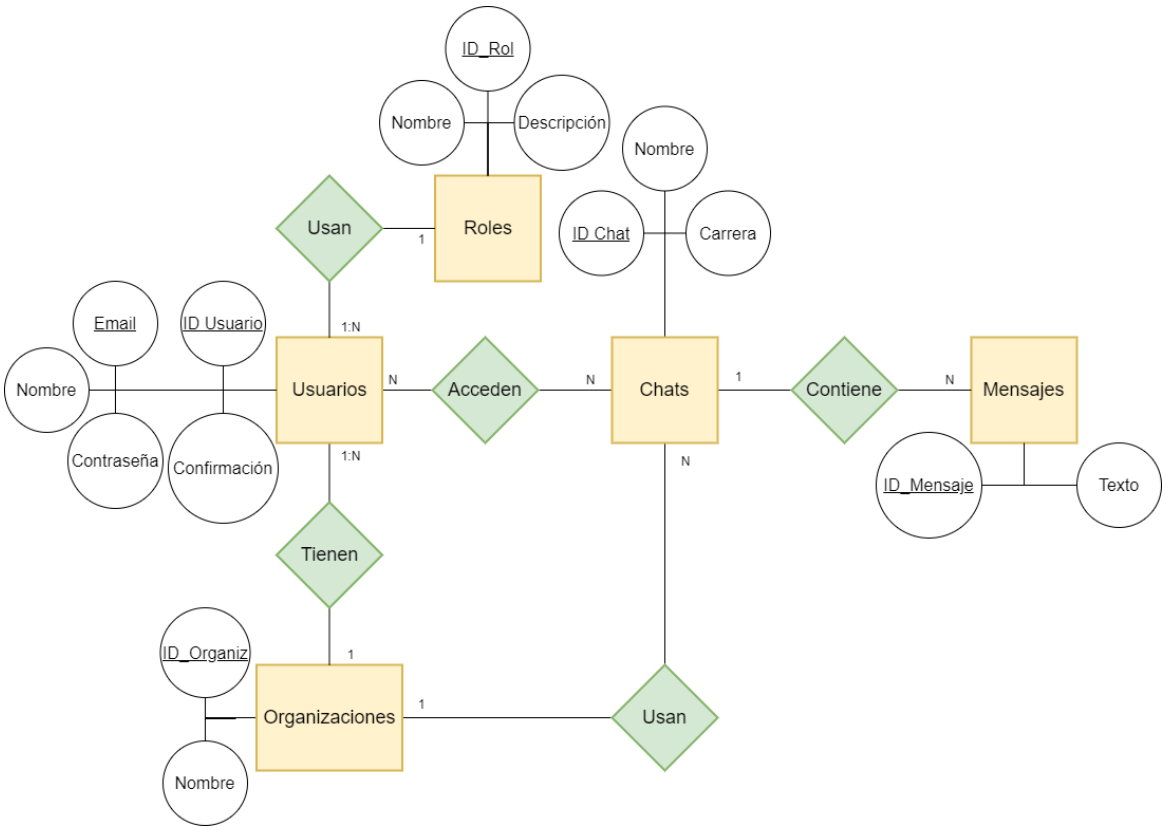

Fig. 1: Diseño de la base de datos MySQL.

la confirmación por correo electrónico (para evitar posibles suplantaciones de identidad). Las otras entidades representadas en el diagrama son: los chats a los que diferentes participantes pueden acceder, los roles que permiten identificar que tipo de usuario puede acceder a un chat determinado y, los mensajes que se envían, los cuales serán almacenados para garantizar que si hubiera algún fallo, no se pierdan datos.

Por otro lado, la segunda base de datos sigue un diseño mucho más simple, ya que tal y como se ha mencionado, se utiliza a modo de caché para toda esa información que se necesita de modo inmediato y asiduamente. Dichos datos corresponden a las conversaciones de los chats. El diseño, pasa por establecer diferentes llaves con nombres clave y asignarles el valor correspondiente.

A continuación, para implementar los dos diseños, simplemente se han instalado las versiones de las bases de datos correspondientes y, se ha confeccionado un *script* a partir del cual se ha creado toda la organización descrita anteriormente.

Es importante notar que, que cuando se implemente el código interno de la aplicación se debe tener en cuenta que hay dos tipos de base de datos. Esto provoca que se deba diferenciar el rol que va a seguir cada una de ellas y como se gestiona la información que se obtiene de cada una de ellas.

6.2. Estructura interna

La lógica interna de la aplicación está descrita en la figura 2, donde se contempla un diagrama de componentes con las diferentes relaciones entre éstos y como se organizan. Dichos elementos están distribuidos según la función que van a desarrollar, siendo `Index` el principal.

El la agrupación de componentes, denominada `Index`, hay un gestor de peticiones que se encarga de redirigirlas hacia el módulo correspondiente, de la inicialización de todos los *middleware* necesarios y, de la gestión de posibles peticiones erróneas que se realicen. No obstante, se debe tener en cuenta que no sean peticiones maliciosas y, establecer algún método de seguridad para evitar conexiones no deseadas o accesos no autorizados. Para ello, se incorpora el componente `Seguridad` que permite desempeñar dicho rol.

A continuación, como se puede observar en la figura 2, hay un grupo de componentes que representa la interfaz de usuario. En ella, se encuentran dos elementos clave, siendo uno el que contiene todas las vistas posibles (hechas de forma que puedan variarse dinámicamente con facilidad) y, el segundo, que permite gestionar toda la información que, más tarde, será añadida a la vista correspondiente dependiendo de la petición recibida. Además, cabe destacar que dicha agrupación siempre va a necesitar de la existencia de `Index` para funcionar, por eso, se le asigna una relación de dependencia.

Partiendo del grupo de componentes `Index`, se encuentran muy ligados a él las agrupaciones que representan el chat y el acceso. La primera contiene elementos que se encargan de gestionar los chats a los cuales los usuarios pueden acceder, obtener información sobre las diferentes salas a las que tienen acceso y, organizar los usuarios para que no haya problemas de interferencia de datos. Además, contiene un elemento para gestionar las conversaciones, el cual se encarga de recopilar todos los mensajes pertenecientes a la sala del chat y procurar que no se produzcan errores al prepararlos para mostrarlos. El último componente referen-

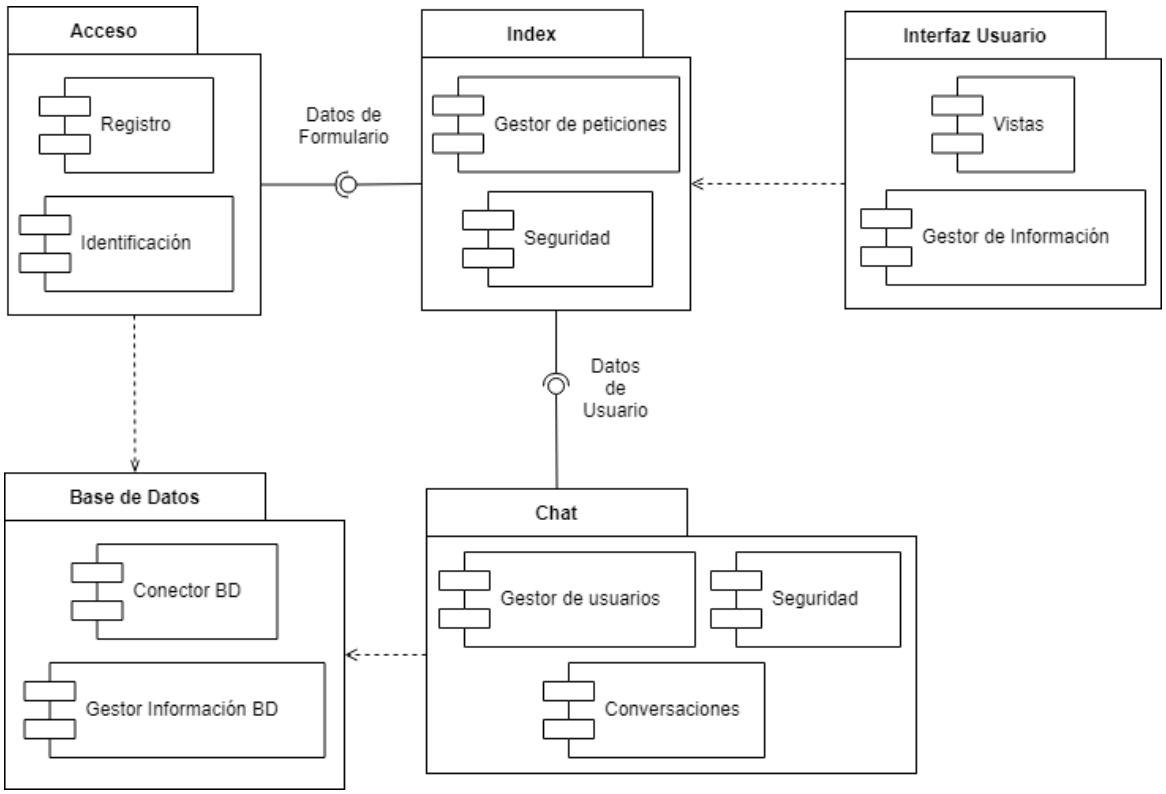

Fig. 2: Diagrama de componentes de la estructura interna.

te al chat, se encuentra relacionado con la seguridad para evitar que un usuario pueda acceder a una sala que no tenga permiso o realizar cualquier acción no autorizada. Por otro lado, se encuentra la agrupación de acceso, la cual se divide en registrar un nuevo usuario o acceder al sistema con uno ya creado anteriormente. Estos dos componentes son muy parecidos, únicamente varían en la información que gestiona cada uno de ellos y como se aplica la seguridad. Cabe destacar, que estos dos grupos tienen una relación de dependencia hacia la última agrupación que representa la base de datos.

El conjunto de módulos que ayudan a obtener información sobre la base de datos, se encuentran divididos en dos: el conector y el gestor de información. El primero proporciona una conexión a la base de datos (ya sea Redis o MySQL) y varios mecanismos de seguridad para evitar que se realicen conexiones no autorizadas. El segundo, el gestor de información, permite manipular los datos extraídos de la base de datos y proporcionarles el formato necesario según la finalidad, lo que implica que éste componente gestiona varias estructuras de datos propias de la aplicación.

6.3. Interfaz

Uno de los objetivos es el diseño de la interficie y la usabilidad, que implica que que el usuario debe sentirse cómodo con la aplicación. Para ello, se ha diseñado una interfaz amigable, sencilla, que permita no perderse y utilizar la aplicación sin tener conocimientos sobre esta.

Los *frameworks* utilizados para crear la interfaz inicial (figura 3) han sido las descritos ya anteriormente, adaptándolos a las necesidades de este proyecto. Hay que decir, que éstos ofrecen herramientas muy potentes para ayudar a confeccionar la visualización en dispositivos

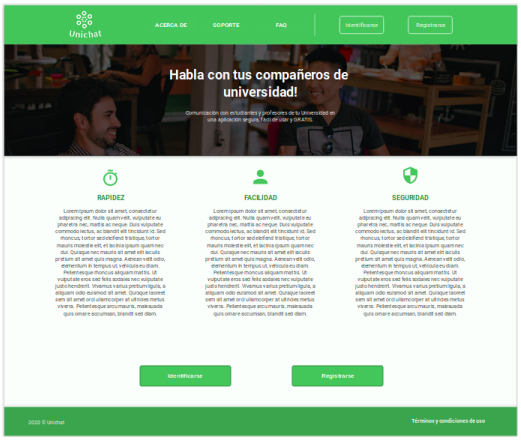

Fig. 3: Vista inicial.

móviles, haciendo que independientemente del dispositivo que se use, el usuario tenga una buena experiencia. En este caso, cuando se accede desde un dispositivo móvil, los elementos de navegación aparecen en un menú desplegable. Además, tal y como se ha mencionado, se ha optado por un diseño simple, el cual con pocos elementos ayuda a que el usuario se sienta cómodo y no se vea abrumado por mucha información.

Así mismo, para poder tener la interfaz dividida en pequeños fragmentos y así poder realizar cambios de forma más simple, se han creado varios archivos con elementos de las vistas que, importándolos al fichero principal, construyen la interfaz. Por ejemplo, en la página de inicio se tienen cuatro archivos que corresponden a: cabeceras, el pie de página y varios ficheros en JavaScript necesarios para el funcionamiento de estos *frameworks*. Todos ellos se importan a la vista principal haciendo que formen la vista desea-

da.

Por último, cabe destacar que la tecnología PUG permite crear plantillas de las vistas que, más tarde, podrán ser rellenadas con los datos que se reciban y así ofrecer una experiencia personalizada para cada usuario. Dichos datos se encuentran almacenados en la sesión del usuario para evitar que la información se colapse entre diferentes conexiones.

7 Desarrollo

En esta sección se muestra como se ha implementado la construcción de la aplicación y el motivo de las decisiones tomadas para construir un servicio de calidad. En las siguientes subsecciones se detallan los diferentes módulos que se han comentado anteriormente.

7.1. Index

Para realizar este módulo, se han creado varias secciones que ayudan a tener un mayor control, siendo el gestor de peticiones la más relevante. Éste, se ha implementado, tal y como se muestra en el código 1, a partir del direccionamiento de rutas que ofrece Express para procesar la petición que se realizará y el resultado que se obtendrá. Para garantizar que no se puede acceder de forma simple a las peticiones que se encargan de procesar los datos, se han tratado como POST. Ahora bien, todas aquellas que no necesitan hacer uso de datos sensibles o no se requiere de ninguna condición especial para ser accedidas, se han tratado como GET. Además, en esta sección se inicializan todos los parámetros necesarios para el correcto funcionamiento de las peticiones, entre ellos: las sesiones de usuarios, la conexión a la base de datos, diferentes indicaciones para que Express sepa dónde encontrar las vistas a renderizar, que directorios son públicos, y el servicio de chat. Este último se ha considerado como una petición, ya que Socket.io establece un servidor que recibirá peticiones y en función del tipo que reciba, enviará una respuesta.

```
1  app.get('/', function(req, res){
2      if(req.session.datosView.logged && !
           req.session.datosView.primerInicio){
3          res.redirect('/chat')
4      }
```

Código 1: Direccionamiento a partir de Express.

La siguiente sección consiste en implementar la seguridad, que se basa en observar cuando un usuario puede procesar una petición o no. Para ello, se ha hecho uso de la sesión, dónde se almacena información de control, como un booleano para saber si se encuentra identificado o si ha elegido las asignaturas que cursa. Cabe destacar que este modo de funcionamiento provoca que la sesión sea muy vulnerable, haciendo que un robo de ésta sea fatal para el usuario. Para evitar dicha problemática, se ha inicializado utilizando un pequeño módulo que permite encriptar la sesión de forma que solamente el propio usuario y servidor saben el contenido de ésta. Además, la inicialización de la base de datos, se realiza desde un fichero externo. Es decir, cuando la conexión a la base de datos se establece, se hace a partir de un fichero especial para ello, haciendo que no se pueda observar el servidor donde se encuentra alojada, el usuario o la contraseña.

7.2. Interfaz Usuario

Tal y como se ha especificado en secciones anteriores, para crear las vistas para los usuarios se ha utilizado la tecnología PUG. Esto implica utilizar plantillas, como la del código 2, que se rellenan con la información que se obtiene a partir del renderizado.

```
1  for val in asign
2      div(class="row")
3          div(class="input-field col s6")
4              label
5                  input(class="with-gap green-
                       text" name=val type="
                       radio")
6                  span= val
```

Código 2: Fragmento de código de la plantilla SelectAsign.pug.

Dicha información se obtiene a partir de la sesión de usuario, la cual contiene en todo momento el identificador de usuario y los datos para elegir que elementos deben mostrarse (dado que esta tecnología permite establecer condicionales a partir de variables). Ha de decirse que para hacer que sea modularizado, cada plantilla se divide en los elementos del `head`, `header`, `banner`, las características propias de cada vista, `footer` y los *scripts* necesarios para el funcionamiento del CSS o del chat.

Por otro lado, se necesita una forma de organizar todos los datos que irán a la sesión y saber como pueden ser encontrados. Por eso, se ha creado una estructura de datos denominada `datosView` la cual permite saber en todo momento como se debe interpretar la información. Además, también se han implementado una serie de funciones que permiten recuperar varios datos necesarios para el correcto funcionamiento, como toda la lista de usuarios que se encuentran en un centro para gestionarlos. No obstante. aunque se almacene cierta información en la sesión, nunca se guardan todos aquellos datos que pueden suponer un riesgo para un usuario.

7.3. Acceso

Una de los módulos más importantes es el acceso, dado que tanto la seguridad como la forma en que se gestionen los datos, determinarán gran parte de la calidad final de la aplicación. Para ello, se ha dividido en dos grandes secciones: el registro y la identificación.

7.3.1. Registro

El registro se realiza a partir de un formulario que envía, mediante una petición POST, los datos de éste para ser procesado. Una vez el servidor obtiene dichos datos, entra en juego el módulo de Seguridad descrito en el código 3. Este módulo comprueba que cada campo que se ha introducido tenga una dimensión mínima y, no contenga caracteres especiales (así pudiendo evitar un posible ataque de SQL injection). Si en alguno de estos pasos se encuentra alguna

discrepancia, se procede a rellenar una variable de la sesión indicando dónde se encuentra el error. A continuación, se contacta con la base de datos y comprueba si el correo electrónico introducido ya se encuentra en uso, además de comprobar si el correo electrónico forma parte de alguna organización. Así como en el caso anterior, si se encuentra una discrepancia, se notifica mediante una variable de sesión que activará un mensaje de error en la vista del usuario.

```
1  if(req.session.datosView.logged){
2      res.redirect('/')}
3  else{
4      if(compruebaFormulario(req)){
5          procesarNuevoUsuario(req)
6          enviarMailConfirmacion(req)
7          anadirDatosBasicosSesion(req)
8          res.redirect('/confirmacion')
9      }
10     else{
11         res.redirect('/registro')}
```

Código 3: Proceso de registro de un usuario.

Ahora bien, si todos los datos son correctos, se procede a insertarlos en la base de datos encriptados con la ayuda de crypto y una palabra secreta que corresponde a el correo del propio usuario más, una pequeña serie de números y letras generadas de forma aleatoria. Cabe destacar que el campo de `Confirmacion` estará desactivado, haciendo que se deba confirmar la cuenta creada a partir de un correo electrónico que contiene una URL dinámica. Dicha dirección, se almacena en una variable global del servidor para, más tarde, cuando un usuario acceda a ella pueda procesarse y confirmar la cuenta de usuario. Una vez se haya confirmado la cuenta de correo, se actualiza la base de datos marcando el campo `Confirmacion` y quitando de la lista de URL dinámicas la que ya se ha usado. Ahora bien, si un usuario no se encuentra confirmado, no podrá acceder a ninguna de las secciones que requieran identificación. Ésta medida se ha adoptado para evitar que se creen una cuenta falsa y así poder acceder a la aplicación.

7.3.2. Identificación

La identificación de un usuario se realiza a partir de un formulario que recoge, mediante una petición POST, el correo electrónico y la contraseña. En el código 4 se observa como se procesan los datos. Primeramente se ejecuta, de nuevo, el módulo de `Seguridad` para comprobar que no hay caracteres especiales y las dimensiones de éstos son correctas. A continuación, se comprueba si existe un usuario con su mismo correo y contraseña (teniendo en cuenta que se están comparando encriptaciones, en ningún momento se almacenan en texto plano). En el caso que haya una discrepancia en cualquiera de los dos procesos descritos anteriormente, se marca que hay un error en una variable de la sesión y se vuelve a cargar la vista de identificación indicándolo.

Una vez realizado el inicio de sesión, se comprueba si el usuario se encuentra inscrito en alguna sala de chat. De no ser así, se obtienen los diferentes grados que ofrece la organización y, al escoger uno, se visualizan todas las asignaturas de dichos estudios. Cuando los estudios ya se han

```
1  if(req.session.datosView.logged){
       res.redirect('/')}
2  else{
3      if(compruebaFormularioLogin(req) &&
           compruebaUsuario(req)){
4          procesaInicioSesion(req)
5          if(compruebaPrimerInicio(req)){
6              cargaPrimerInicio(req)
7              cargaCursos(req)
8              res.redirect('/
                   seleccionAsignatura')}
9          else{
10             cargaLoginCorrecto(req)
11             res.redirect('/')}
12     }
13     else{
14         res.redirect('/identificarse')}
15 }
```

Código 4: Proceso de identificación de un usuario.

elegido, se procede a mostrar un chat con las diferentes salas.

7.3.3. Recuperar contraseña

En el caso que un usuario no se acuerde de su contraseña, puede solicitar recuperarla. Para ello, será necesario que introduzca el correo electrónico. De nuevo, se realizarán las comprobaciones de seguridad pertinentes, y si todo es correcto se genera un correo electrónico de recuperación de contraseña. De éste modo un usuario cualquiera no podrá restablecer dicho campo de un tercero. Cuando el usuario entra en la dirección mandada en el correo, automáticamente se le avisa conforme ha solicitado restablecerla y se le envía por correo electrónico una nueva, la cual se genera con un pequeño módulo de Crypto que dispone de una función que genera una cadena de valores aleatoria. A continuación, se realiza el cambio pertinente en la base de datos y se envía la nueva contraseña por correo electrónico al usuario.

7.3.4. Perfil

El último módulo de esta agrupación, permite gestionar el perfil de usuario, el cual puede cambiar su contraseña en todo momento introduciendo la antigua y la nueva. Cabe destacar que, en este proceso también se comprueba que todos los datos introducidos no tengan ningún tipo de carácter especial, no excedan el limite permitido, no tengan una longitud demasiado corta y la nueva contraseña introducida no sea la misma que la anterior. Además, en el perfil del usuario se puede observar el nombre de usuario y el nombre que muestra a los demás miembros del chat. En el caso que se quisiera cambiar se debería contactar con un administrador, puesto que la idea es que cada usuario utilice el correo institucional y el nombre y apellidos para que pueda ser identificado con más facilidad.

7.4. Base de Datos

Para poder gestionar toda la información de la aplicación, se usan dos tipos de bases de datos, las cuales van enfocadas a propósitos distintos. La primera se basa en el gestor

MySQL y se usa como base de datos principal. Su implementación ha sido a partir de un *script* que se ha introducido en un servidor remoto para aligerar la carga del servicio de chat. Para poder usarla en la aplicación, se ha creado un fichero con toda la información de acceso y se ha importado al módulo para usar dichas variables e iniciar la conexión. Seguidamente, se ha creado una función que permite realizar peticiones de todo tipo.

Por otro lado, la segunda base de datos que se utiliza, es Redis, la cual se enfoca a guardar los historiales de los chats. De este modo, siempre que un usuario inicie sesión, accederá a Redis, recogerá la información de la sala de chat activa y la mostrará por pantalla. Cabe destacar que se ha configurado un modo automático de *backup*, cada hora, para sincronizar los historiales de las salas de chat de una base de datos con otra y así asegurarse que si hubiera algún fallo, se pierda la menor información posible. Además, toda la información que se obtiene de los historiales de cada sala de chat, se encuentra cifrada una vez se almacena a la base de datos. En este caso, no se ha implementado una función que sirva para realizar peticiones, sino que al ser una base de datos de llave-valor, se guarda y retira la información directamente.

7.5. Chat

La parte principal de la aplicación reside en éste conjunto de módulos. Por eso, se ha dividido en varias secciones para poder controlar y gestionar de forma más ordenada cada una de ellas. Tal y como se especifica en secciones anteriores, en la aplicación hay varios roles, los cuales pueden realizar tareas diferentes. Para implementarlos, se ha creado una tabla en la base de datos con los roles, junto con la relación de que rol tiene asignado cada usuario. De esta forma, cuando un usuario se identifica, se guarda el valor en una variable de la sesión. Además, una vez se ha iniciado sesión, si se trata de un `Profesor` o un `Administrador`, verá un menú de administración, a partir del cual cada uno de ellos podrá realizar varias acciones.

7.5.1. Roles

Por definición, solamente habrá tres tipos de roles: el de `Alumno`, el cual únicamente puede chatear y crear mensajes privados con otros alumnos; y los roles de `Profesor` y `Administrador` que podrán acceder a tareas administrativas.

Dichas tareas dependen del rol que tenga un usuario. En el caso de un `Profesor`, únicamente podrá eliminar a todo aquél usuario que sea alumno de las salas a las que él se encuentra suscrito. De ésta forma, podrá gestionar si hay alguna persona que no debería pertenecer a dicho grupo o, si hay algún usuario que tenga una conducta inadecuada. Por otro lado, el `Administrador` puede gestionar cualquier tipo de usuario eliminándolo de cualquier sala de chat de la organización, añadiéndolo o, simplemente, eliminando la cuenta de un usuario. No obstante, éste rol también puede gestionar las salas de chat existentes, de forma que puede eliminar y crear de nuevas.

Para lograr todo esto, se ha implementado el módulo `Gestor de usuarios`, el cual a partir de diferentes peticiones se decide que se debe renderizar (dependiendo tanto del rol que tenga el usuario, como de la opción que se seleccione del menú). No obstante, en el caso del `Administrador`, se ha tenido en cuenta que al crear salas nuevas, el nombre introducido no exista ya. Además de controlar en todo momento las salas que se muestran para añadir a un usuario a una nueva asignatura, para evitar problemas de duplicado. Cabe destacar, que se ha optado por el uso de botones seleccionables, ya que de éste modo el usuario tiene más claro que opciones puede seleccionar y para procesar las peticiones se puede identificar de forma más concisa la acción a realizar. Aun así, se debe tener muy en cuenta el acceso a estas tareas, por eso, a partir del módulo de `Seguridad` se ha restringido el acceso, comprobando en todo momento que las peticiones que se realizan son de un usuario legítimo y no se trata de un atacante.

7.5.2. Gestión del chat

El módulo del Chat está creado principalmente con el *framework* Socket.io. Los ficheros que lo representan son funciones que están divididas, tal y como muestra en el código 5, entre el servidor y el cliente. En el lado del servidor se incluyen algunas, como por ejemplo: cuando un usuario se conecta por primera vez (se debe dar la bienvenida y cargar el historial del canal al que se acceda), cuando se envía un mensaje (se capta el texto, se añade a la base de datos de Redis del historial y se manda a los clientes para que sea procesado y mostrado), cambiar de sala de chat (se debe desconectar al cliente de la sala actual y conectar a la sala que desea) y desconectarse.

```
1  socket.on('joinRoom', function(){
2          entraSalaUsuario(socket, req)
3     })
4  socket.on('sendChat', function(data){
5      enviaMensaje(io, socket)
6  })
7  socket.on('switchRoom', function(newroom){
8      variaSala(socket)
9  });
10 socket.on('disconnect', function(){
11   desconectaChat(socket)
12 });
```

Código 5: Proceso de identificación de un usuario.

En el lado cliente se ha implementado un *script* que permite enviar una petición (ya sea de cambio de sala, nueva conexión o envío de mensaje) al servidor y que se procese. No obstante, hay que tener en cuenta que se deben implementar medidas de seguridad para evitar que dichos mensajes puedan ser leídos por terceras personas y, limitar el acceso a las salas a los usuarios suscritos. Para ello, en el módulo de Seguridad se describen una serie de funciones que permiten obtener dicha privacidad en los mensajes y gestionar que cada usuario solamente se pueda ver las salas a las que se ha suscrito (mediante la sesión, se indican las asignaturas que tiene el usuario).

8 SUBIDA A PRODUCCIÓN

Una vez la aplicación ha sido desarrollada, el próximo paso es hacer que cualquier usuario pueda acceder a través de Internet. Para ello, se ha adquirido un dominio y se ha

alquilado un droplet de DigitalOcean con Ubuntu. No obstante, se debe configurar el servidor para garantizar el correcto funcionamiento de nuestra aplicación e implementar un certificado electrónico. A continuación, se muestra como se han realizado todos estos procesos.

8.1. Configuración del servidor

Antes de desplegar la aplicación se debe proporcionar una vía de acceso mediante Secure Shell (SSH). Dicha configuración consiste en acceder al *droplet* a través de la página web y crear un nuevo usuario que será al que se conectará de forma remota. A continuación, se le añade a un grupo de usuarios el cual tendrá permisos para utilizar el droplet a niveles básicos (de éste modo se asegura que dicho usuario no pueda acceder a archivos confidenciales propios del droplet). Seguidamente, se añade la clave SSH del dispositivo que desea acceder remotamente, haciendo que cada vez que se desee iniciar la conexión no sea necesario especificar la contraseña. Por último, se editan los permisos de conexión del usuario *root* para evitar que se puedan hacer conexiones remotas con dicho usuario.

Una vez se puede acceder de forma remota y operar en el droplet sin ser el usuario root, se procede a configurar el firewall para garantizar que únicamente se acepten las peticiones de los puertos 80 (para HTTP) y 443 (para HTTPS). Dicha configuración consiste en utilizar tres comandos de Uncomplicated Firewall (ufw) para habilitar el tráfico de OpenSSH, HTTP y HTTPS. Dichos comandos pueden observarse en el código 6.

```
1 sudo ufw allow OpenSSH
2 sudo ufw allow http
3 sudo ufw allow https
4 sudo ufw enable
```

Código 6: Comandos para la configuración del firewall.

Llegados a éste punto, el siguiente paso es subir la aplicación al droplet para empezar a utilizarla. Para ello, primero se debe instalar Git y NodeJS para que puedan ser utilizados. El siguiente paso consiste en copiar el contenido del repositorio de la aplicación en nuestro servidor. A continuación, se necesita una forma de iniciar la aplicación pero que deje la máquina operativa para realizar los comandos que se puedan necesitar. Para ello, se usa la herramienta PM2, la cual permite iniciar una aplicación en segundo plano, haciendo que reciba todos los recursos necesarios pero también deje operar en el droplet. No obstante, cada vez que se reinicia la máquina, se debe volver a iniciar el proceso de la aplicación mediante la herramienta. Para automatizar este proceso se debe utilizar un comando que permite establecer que una aplicación se ejecutará cuando se inicie el droplet (en el caso que sea necesario reiniciarlo).

8.2. Certificado y dominio

Para establecer que el servicio web utilice el protocolo HTTPS se debe instalar un certificado Transport Layer Security (SSL). Para ello, se ha utilizado la herramienta Let's Encrypt [4], la cual genera un certificado válido. Además, una vez se ha generado, se procede a marcar que se autorenueve, dado que por defecto expira cada tres meses.

El siguiente paso consiste en instalar NGINX para hacer que las diferentes peticiones que se reciban sean redirigidas a la aplicación que hay instalada en el droplet. Para ello, lo primero que se hace es acceder al fichero de sitios habilitados y especificar que todas las conexiones que provengan del puerto 80 (HTTP) sean redirigidas hacia el 443 (haciendo que todas la conexiones deban utilizar la versión con SSL). A continuación, se debe crear un fichero, como el del código 7, con toda la configuración de SSL, la cual incluye varios parámetros ajustables para determinar como actuará. La configuración de dichos parámetros se ha obtenido de Cipherli [13].

```
1  ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
2  ssl_prefer_server_ciphers on;
3  ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+
       EECDH:AES256+EDH";
4  ssl_ecdh_curve secp384r1
5  ssl_session_cache shared:SSL:10m;
6  ssl_session_tickets off;
7  ssl_stapling on;
8  ssl_stapling_verify on;
9  resolver 8.8.8.8 8.8.4.4 valid=300s;
10 resolver_timeout 5s;
11 add_header Strict-Transport-Security "max-
       age=63072000; includeSubDomains; preload
       ";
12 add_header X-Frame-Options DENY;
13 add_header X-Content-Type-Options nosniff;
```

Código 7: Fichero de configuración SSL para NGINX.

Una vez se tienen los parámetros de configuración del certificado SSL, se debe configurar NGINX para que redirija todo el tráfico que provenga del dominio `unichat.es` hacia la aplicación. Para éste caso, se utiliza el puerto 5000 para la aplicación. La configuración consiste en indicar que contendrán varias cabeceras usadas por el proxy, la localización del programa, saber si se puede reutilizar la sesión y si hay redirecciones. A continuación, para confirmar que funciona correctamente, se realiza una conexión hacia la IP del droplet. No obstante, para que funcione con el dominio `unichat.es` se debe acceder al panel de control de 1&1 y, especificar que los servidores que utiliza el dominio son los de DigitalOcean. Además, también se debe entrar en el panel de configuración del droplet e indicar la lista de subdominios que van a tener acceso al droplet. Una vez realizada dicha configuración, ya se podrá utilizar desde un navegador web externo.

9 Conclusiones

A partir de los diferentes apartados se ha visto como se ha diseñado y creado una aplicación web de chat para universidades. En este trabajo se ha mostrado el diseño de las bases de datos, la lógica interna del sistema, y la interfaz que verá el usuario una vez acceda a la aplicación web. Además, se ha podido apreciar como la elección entre varias tecnologías similares depende de varios factores, como los objetivos iniciales que se hayan planteado. Otro aspecto importante ha sido ver como funciona internamente el aplicativo, pasando por el inicio de sesión, el registro de un usuario, la genera-

[4] Más información en: https://letsencrypt.org

ción de las vistas, la gestión de las bases de datos y el chat en tiempo real.

Así pues, los diferentes objetivos que se habían marcado inicialmente, se han cumplido. Éstos consistían en realizar una buena selección de herramientas para desarrollar la aplicación, utilizar buenas prácticas para crear una estructura interna lo suficientemente sólida para soportar cambios futuros, confeccionar una interfaz de usuario que fuera agradable, realizar diferentes ajustes de seguridad para garantizar que el aplicativo se puede usar con total tranquilidad y, configurar la red en la que se encuentra instalada (en éste caso ha sido en un droplet proporcionado por DigitalOcean).

10 Trabajo futuro

Desde la perspectiva futuras lineas de mejora se pueden considerar las siguientes:

- Poder hacer que el servicio sea autoescalable y no depender de recursos limitados. Dado que actualmente se utiliza un servidor de base de datos remoto gratuito y los medios económicos son muy limitados, se establece que una buena forma de mejorar la aplicación es ésta.
- Realizar un mantenimiento constate, dado que las tecnologías van evolucionando y la seguridad también, es necesario que se vaya analizando el funcionamiento durante bastante tiempo con un gran tráfico de usuarios para observar que posibles mejoras podrían implementarse.
- Actualmente no hay una compartición de archivos, podría ser una buena linea implementar dicha capacidad con un filtro gestionado por una inteligencia artificial, para gestionar si es adecuado o no.

Agradecimientos

En primer lugar, agradezco toda la ayuda prestada por parte de mi tutor. No solamente me ha estado indicando mejoras para el trabajo, sino que en todo momento ha estado apoyándome y preocupándose para que el proyecto pudiera salir adelante, tanto la parte escrita como la parte de desarrollo.

También, me gustaría hacer una mención especial a mi familia y, a cuatro grandes amigos (Carmen, María, Eleazar e Ivette), los cuales han estado animándome, preocupándose por mí, por el desarrollo del proyecto y me han ayudado aportando opiniones y varios puntos de vista sobre el diseño inicial de la interfaz de la aplicación.

Referencias

[1] Fernando Rivero (2017). Informe Ditrendia 2017: Mobile en España y el Mundo [En línea]. Disponible: https://www.amic.media/media/files/file_352_1289.pdf

[2] Ferenc Hámori, Gergely Nemeth (2019, Marzo 9). History of Node.js on a Timeline [En línea]. Disponible: https://blog.risingstack.com/history-of-node-js/

[3] Greg Wilson, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumbley, Ben Waugh, Ethan P. White, Paul Wilson (2014, Enero 7). Best Practices for Scientific Computing [En línia]. Disponible: https://journals.plos.org/plosbiology/ article?id=10.1371/journal.pbio.1001745

[4] Victor Ofoegbu (2018, Enero 23). The only NodeJs introduction you'll ever need [En línea]. Disponible: https://codeburst.io/the-only-nodejs-introduction-youll-ever-need-d969a47ef219

[5] Dave Swersky (2018, Mayo 25). Koa vs Express in NodeJS: 2018 Edition [En línea]. Disponible: https://raygun.com/blog/koa-vs-express/

[6] Slant. Express JS VS Strapi [En línea]. Disponible: https://www.slant.co/versus/1277/12406/ẽxpress-js_vs_strapi

[7] Jason Harris (2018, Abril 25). PostgreSQL Vs. MySQL: Differences In Performance, Syntax, And Features [En línea]. Disponible: https://blog.panoply.io/postgresql-vs.-mysql

[8] Anastasia Raspopina, Sveta Smirnova (2017, Enero 6). PostgreSQL and MySQL: Millions of Queries per Second [En línea]. Disponible: https://www.percona.com/blog/2017/01/06/millions-queries-per-second-postgresql-and-mysql-peaceful-battle-at-modern-demanding-workloads/

[9] Slant. EJS vs Pug Jade [En línea]. Disponible: https://www.slant.co/versus/184/185/ ejs_vs_pug-jade

[10] Materialize. About [En línea] https://materializecss.com/about.html

[11] Ankush Thakur (2018, Octubre 10). 10 Best CSS Frameworks for Front-End Developers [En línia]. Disponible: https://geekflare.com/best-css-frameworks/

[12] Alexandra Leslie (2018, Enero 11). NGINX vs. Apache (Pro/Con Review, Uses, & Hosting for Each) [En línea]. Disponible: https://www.hostingadvice.com/how-to/nginx-vs-apache/

[13] Cipherli. Cipherli.st Strong Ciphers for Apache, nginx and Lighttpd [En línea]. Disponible: https://cipherli.st/

[14] Socket.io. What Socket.IO is [En línea]. Disponible: https://socket.io/docs/

[15] Stackshare.io. Socket.IO vs SocketCluster [En línea]. Disponible: https://stackshare.io/stackups/socket-io-vs-socketcluster

[16] Sayan Kumar Pal. Software Engineering Spiral Model [En línea]. Disponible: https://www.geeksforgeeks.org/software-engineering-spiral-model/

APÉNDICE

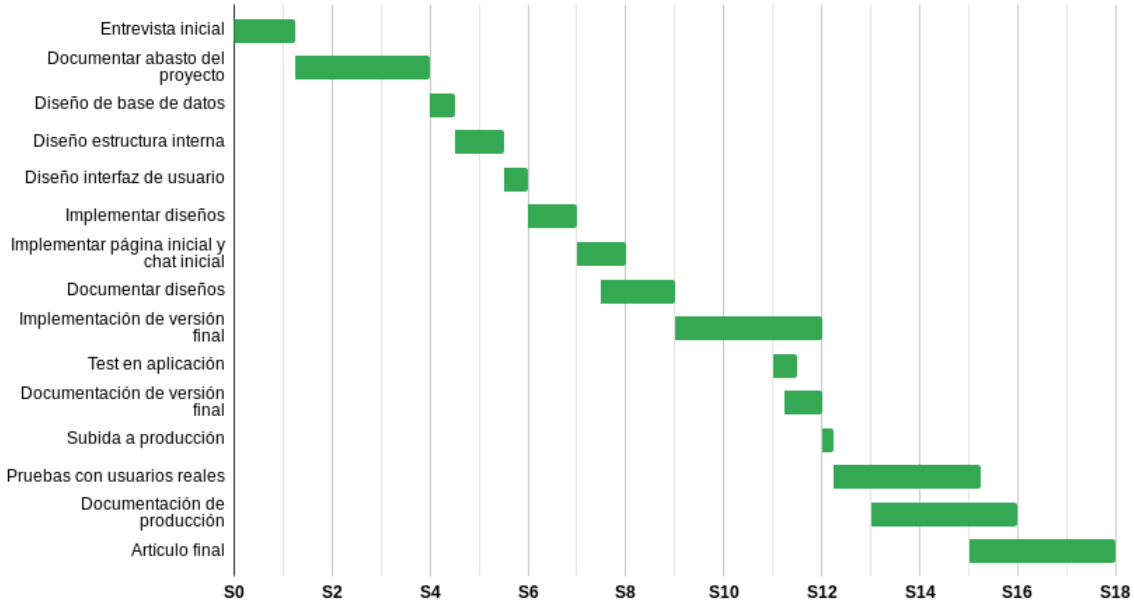

Fig. 1: Diagrama de Gantt de la planificación.