

INDEPENDÍZATE



Creadores:

- Cristina Marzo Pardos NIA: 817116
- Daniel Carrizo Pérez NIA: 821674
- Juan Pellicer Barco NIA: 818138
- Ayelen Nuño Gracia NIA: 799301



Universidad
Zaragoza

Implementación: [24/01/2024 - 30/05/2024]

Asignatura: Sistemas y Tecnologías Web

Índice:

1. Resumen del proyecto:	2
2. Propuestas similares:	2
3. Acceso al sistema:	2
4. Relación de módulos/bibliotecas utilizadas:	3
Frontend:	3
Backend:	5
5. Implementación:	6
6. Analíticas:	8
7. Análisis de problemas potenciales:	9
8. Validación:	10
Frontend:	10
Backend:	11
9. Problemas encontrados durante el desarrollo y bugs conocidos:	12
Dificultades Frontend:	12
Dificultades Backend:	12
Problemas de la aplicación:	13
10. Despliegue del sistema:	14
Frontend:	14
Base de datos:	14
API:	15
11. Valoración personal de cada miembro:	15
Cristina Marzo:	15
Daniel Carrizo:	15
Ayelen Nuño:	15
Juan Pellicer:	16
12. Distribución de tiempo:	16
Sumatorio de horas totales:	16
Desglose de tareas por participante:	17
13. Conclusión:	17
14. Anexos de los desarrollos opcionales:	18
Anexo I: Implementación de WebSockets:	18
Anexo II: Login social:	18
Anexo III: Despliegue del sistema sobre docker y una infraestructura en la nube:	19

1. Resumen del proyecto:

Independizate es un servicio web creado con el fin de ser el stack overflow de la vida adulta. Es decir, buscamos que los jóvenes que van a dar este importante paso en sus vidas tengan un lugar seguro al que acudir para consultar dudas básicas, como pueden ser consejos de limpieza, de economía doméstica, recetas y otros. Además consideramos que era una buena oportunidad para centralizar la búsqueda de pisos, y de compañer@s para irse a vivir, dado que hasta ahora no existe ningún servicio que aúne todas estas características.

Respecto a buscar pisos, hemos pensado que sería útil situarlos en el mapa, para así poder ver la zona en la que está situada, los transportes cercanos, los supermercados... es decir, conocer la zona que rodea el piso.

Para implementar *Independizate* se ha usado el stack MEAN, siendo estas las siglas de *Mongodb*, *Express*, *Angular* y *Node.js*. Sobre esta base se han ido añadiendo distintos frameworks, los cuales se explicarán detalladamente en el apartado "relación de módulos/bibliotecas". El tiempo en el que se ha implementado *Independizate* queda recogido en el marco de la asignatura de Sistemas y Tecnologías Web del plan de estudios del Grado en Ingeniería Informática en la Universidad de Zaragoza, cuya primera clase fue el 24/01/2024.

2. Propuestas similares:

Antes de comenzar el proyecto se ha llevado a cabo un pequeño estudio de mercado para saber si nuestro sitio web tenía posibilidades en el mercado actual. Encontramos que esta funcionalidad está cubierta por otros sitios web, pero de forma parcial, encontrando que no existe una sola página que reúna todas las funcionalidades.

A la hora de encontrar pisos y compañeros existen páginas como [amigosMadrid](#), [el tablón de idealista](#), [milanuncios](#), [piso compartido](#) o impresos en los tablones de la universidad. Sin embargo, en la mayoría de ellos observamos que se alquila la habitación, y ya existe gente en esa casa. *Independizate* también ofrece la oportunidad de conocer gente con la que de manera conjunta encontrar un piso.

Por otro lado en lo que a los foros respecta, en el caso de cocina encontramos muchos foros, de expertos de cocina con muchos ingredientes variados, pero suelen ser recetas que requieren de mucho tiempo, o ingredientes muy específicos que de primeras no se tiene claro dónde se pueden comprar. En nuestro caso buscamos un foro de aficionados que quieran sobrevivir al mundo culinario. En lo que respecta a limpieza, hemos encontrado algún foro, pero poco actualizado, o con pocas interacciones e interfaces poco amigables, como [foroactivo](#) y [cotilleando](#). Por último, en economía doméstica hemos encontrado blogs de consejos pero ningún lugar concreto en el que preguntar dudas.

3. Acceso al sistema:

En esta sección vamos a redactar cómo se puede acceder al sistema, tanto a la API desarrollada para este proyecto como a la página de *Independizate*.

La URL de la aplicación es:

<http://independizate.hrd8hgbhf5ayfta2.spaincentral.azurecontainer.io:4200/>

Para poder acceder y ver el funcionamiento del sistema se proporciona un usuario ya creado y su correspondiente contraseña, de igual manera se van a indicar los datos de ingreso de la cuenta del administrador para que sea posible ver los paneles creados y las diversas acciones que como administrador puedes realizar.

- Se puede acceder a modo de demostración con este usuario y contraseña:
demo / demo
- Se puede acceder al panel de administración con este usuario y contraseña:
fabra / fabra

La URL de la documentación Swagger de la API REST es:

<http://backend-independizate.eggtf5e6dvh8hngq.spaincentral.azurecontainer.io:3000/api/docs/>

4. Relación de módulos/bibliotecas utilizadas:

Frontend:

En este caso se van a especificar los módulos externos y las dependencias que se han instalado en el módulo de frontend, haciendo un repaso del fichero *packages.json*.

- ★ @abacritt/angularx-social-login: Proporciona métodos y componentes para la autenticación a través de redes sociales como Google y Facebook en aplicaciones Angular.
- ★ @angular/animations: Módulo de Angular para habilitar y manejar animaciones dentro de la aplicación.
- ★ @angular/cdk: Angular Component Dev Kit, proporciona herramientas y utilidades comunes para desarrollar componentes personalizados.
- ★ @angular/common: Módulo central que incluye directivas y servicios comunes para aplicaciones Angular.
- ★ @angular/compiler: Utilizado internamente por Angular para compilar componentes y plantillas.
- ★ @angular/core: Núcleo del framework Angular que contiene las funcionalidades esenciales y principales.
- ★ @angular/forms: Módulo que proporciona herramientas y directivas para la gestión de formularios en Angular.
- ★ @angular/material: Conjunto de componentes de interfaz de usuario basados en Material Design.
- ★ @angular/platform-browser: Servicios y utilidades necesarios para ejecutar aplicaciones Angular en navegadores.
- ★ @angular/platform-browser-dynamic: Módulo que permite la compilación y ejecución de aplicaciones Angular en tiempo de ejecución.
- ★ @angular/router: Módulo de enrutamiento de Angular para manejar la navegación y rutas dentro de la aplicación.

- ★ @ng-bootstrap/ng-bootstrap: Integra Bootstrap con Angular proporcionando componentes de UI adaptados a Angular.
- ★ @types/gapi.auth2: Tipos TypeScript para la biblioteca de autenticación Google API.
- ★ @types/leaflet: Tipos TypeScript para la biblioteca de mapas Leaflet.
- ★ angular-cli-ghpages: Permite desplegar aplicaciones Angular en GitHub Pages.
- ★ angular-oauth2-oidc: Proporciona autenticación y autorización OAuth2 y OpenID Connect para aplicaciones Angular.
- ★ apexcharts: Biblioteca de gráficos interactivos.
- ★ bcryptjs: Biblioteca para encriptación y verificación de contraseñas usando bcrypt.
- ★ gapi-script: Script loader para las bibliotecas de Google API.
- ★ glob: Utilidad para emparejar archivos usando patrones similares a los de shell.
- ★ leaflet: Biblioteca para crear mapas interactivos.
- ★ ng-apexcharts: Integración de ApexCharts con Angular.
- ★ ng2-cordova-oauth: Proporciona autenticación OAuth2 para aplicaciones híbridas construidas con Cordova y Angular.
- ★ ngx-cookie-service: Servicio Angular para manejar cookies.
- ★ npm: Administrador de paquetes para Node.js.
- ★ rxjs: Biblioteca para programación reactiva, utilizada ampliamente en Angular.
- ★ tslib: Biblioteca de ayuda para TypeScript que contiene funciones auxiliares comunes.
- ★ twitter-api-v2: Cliente para interactuar con la API de Twitter.
- ★ zone.js: Biblioteca que facilita el manejo de contextos asíncronos en JavaScript, crucial para la detección de cambios en Angular.
- ★ @angular-devkit/build-angular: Herramientas de desarrollo para construir aplicaciones Angular.
- ★ @angular-devkit/core: Conjunto de herramientas centrales para el desarrollo de Angular.
- ★ @angular/cli: Interfaz de línea de comandos para Angular que facilita la creación y gestión de proyectos Angular.
- ★ @angular/compiler-cli: Proporciona herramientas de línea de comandos para compilar aplicaciones Angular.
- ★ @cypress/schematic: Proporciona esquemas de Angular para integrar Cypress en proyectos Angular.
- ★ @types/bcryptjs: Tipos TypeScript para bcryptjs. Para hashear contraseñas.
- ★ @types/jasmine: Tipos TypeScript para Jasmine, un framework de pruebas unitarias.
- ★ cypress: Framework de pruebas end-to-end.
- ★ jasmine: Framework de pruebas unitarias para JavaScript.
- ★ jasmine-core: Núcleo del framework Jasmine.
- ★ karma: Ejecutador de pruebas para JavaScript. Utilizado en el testeo.
- ★ karma-chrome-launcher: Adaptador de Karma para lanzar pruebas en el navegador Chrome.
- ★ karma-coverage: Genera informes de cobertura de código para pruebas ejecutadas con Karma.
- ★ karma-jasmine: Adaptador para usar Jasmine con Karma.
- ★ karma-jasmine-html-reporter: Reporter HTML para mostrar resultados de pruebas de Jasmine en el navegador.

Backend:

En este caso se van a especificar los módulos externos y las dependencias que se han instalado en el módulo de backend, haciendo un repaso del fichero *packages.json*.

- ★ ajv: Esta librería ha sido utilizada para la validación de los datos una vez se reciben del frontend (o los servicios que llamen a la API), creando esquemas y subesquemas.
- ★ axios: Es una librería de JavaScript utilizada para realizar solicitudes HTTP desde el navegador o desde Node.js. Ha sido utilizada para minimizar dependencias internas entre ficheros del backend, haciendo llamadas a nuestro propio servidor de aquellas funcionalidades que ya se encontraban implementadas.
- ★ [chai](#): Librería utilizada para la validación del código de la API desarrollada, los detalles de para qué ha sido utilizada esta librería han quedado detalladas en el apartado de validación, subapartado backend.
- ★ cookie-parser: Esta librería de middleware de Express se utiliza para analizar las cookies adjuntas a la solicitud HTTP del cliente. Viene instalado al crear el proyecto de Express, y por si existían dependencias internas hemos decidido mantenerlo.
- ★ cors: Es un middleware de Express utilizado para habilitar el acceso a recursos desde un origen distinto al del servidor que sirve la página. Se ha utilizado para subsanar el error de CORS procedente de las páginas externas que realizan peticiones a nuestra API.
- ★ dotenv: Esta librería se utiliza para cargar variables de entorno desde un fichero .env a nuestra aplicación en Node.js.
- ★ express: Express es un marco de aplicación web de Node.js que proporciona un conjunto de características para construir aplicaciones web y API de manera más sencilla y rápida.
- ★ express-session: Es un middleware de Express que facilita la gestión de sesiones de usuario en una aplicación web. Viene instalado al crear el proyecto de Express, y por si existían dependencias internas hemos decidido mantenerlo.
- ★ jsonwebtoken: Es una implementación de JSON Web Token (JWT) para Node.js. Se utiliza para generar y verificar tokens de autenticación. De esta forma se consigue que los usuarios tengan tokens que duran entre 12 y 24h para permitirles acceso a determinadas funcionalidades de la web.
- ★ less-middleware: Este middleware de Express se utiliza para compilar archivos LESS en CSS y servirlos automáticamente a través de Express.
- ★ mocha: Es un popular framework de pruebas para JavaScript. Se utiliza para realizar pruebas unitarias y de integración en Node.js y en el navegador. En el proyecto se ha utilizado para ejecutar las pruebas unitarias realizadas.
- ★ mongoose: Esta librería ha sido usada para la creación de los esquemas de datos que se almacenan en la base de datos de *Independizate* subida en el servidor de

Atlas. Estos esquemas se han implementado en el directorio `./independizate/app_server/models/schemas_mongooseDB`.

- ★ `swagger-jsdoc`: Es una herramienta que se utiliza para escribir la documentación de una API en formato OpenAPI para poder introducir la especificación de las funciones directamente como cabeceras de las funciones de Javascript.
- ★ `swagger-ui-express`: Es un middleware de Express que se utiliza para servir la interfaz de usuario de Swagger UI para visualizar y probar una API documentada con Swagger.
- ★ [Winston](#): Librería utilizada para llevar a cabo un registro de la actividad de los usuarios, marcando la hora y la llamada realizada al sistema.
- ★ `ws`: Esta es una librería de Node.js para implementar WebSockets, que son una tecnología que permite establecer una comunicación bidireccional y en tiempo real entre un cliente y un servidor a través de una conexión TCP persistente. Se ha usado para crear notificaciones con una arquitecturas del tipo push, la cual se detalla en el [Anexo I](#).

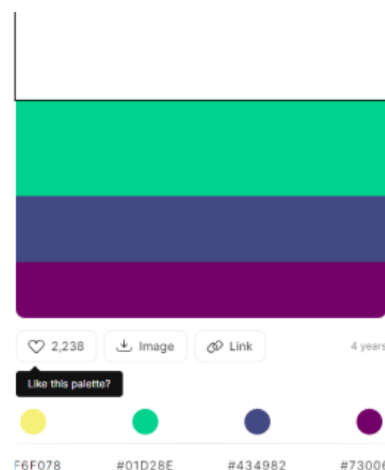
5. Implementación:

Para el desarrollo de este proyecto se ha decidido crear dos repositorios separados en GitHub, uno para el frontend y otro para el backend del sistema, asignando dos miembros a cada uno de los repositorios como responsables. A comienzos de la implementación del proyecto se hizo una excepción en la que Daniel Carrizo se encargó de la definición de los esquemas de mongo y posteriormente se unió al equipo de frontend.

La división de los equipos queda de la siguiente manera:

- Backend:
 - Cristina Marzo Pardos
 - Ayelen Nuño Gracia
- Frontend:
 - Daniel Carrizo Pérez
 - Juan Pellicer Barco

Para la implementación del frontend se ha decidido llevar a cabo el proyecto usando angular, teniendo de guía el prototipo de [Figma](#), en el que se detallaron las pantallas de móvil y de ordenador que se implementarían a futuro, creando una guía de estilos detallada, y definiendo una paleta de colores concreta. La paleta se muestra en la figura del lateral. Para optimizar la creación de la web se ha optado por implementar módulos los cuales se pueden reutilizar en las diferentes páginas, lo cual ha permitido ahorrar tiempo y mantener una uniformidad en las páginas de Independizate. De igual forma, se quiere destacar que las pantallas creadas son **responsive**, llegando a tener un comportamiento diferenciado para todos aquellos componentes que podían llegar a generar problemas, a la hora de visualizarlas desde la web. Esto ha supuesto un



esfuerzo adicional dado que los elementos de Angular Material, que se suponía que en primera instancia eran responsive, no se comportan como tal con pantallas demasiado pequeñas.

En lo que corresponde al backend se ha seguido la guía de organización proporcionada en clase creando la carpeta de `app_server`, que se divide en tres subcarpetas: *controllers*, en la que se implementan todas las funciones de la API divididas en ficheros por temática. *Routes*, en la que encontramos la redirección y el enrutamiento de las funciones. En los mismos ficheros de enrutamiento se ha implementado un pequeño middleware en el que se verifica que el token se ha recibido en las funciones en las que es pertinente. Por último tenemos la carpeta *models*, en la que se ha hecho una diferenciación, entre *schemas_mongooseDB* y *schemas_json*. A la hora de realizar verificaciones en el backend del correcto funcionamiento de los esquemas, se detectó que los esquemas de mongooseno siempre establecen un límite estricto en lo que se refiere a incluir datos adicionales en el cuerpo que se quiere almacenar en la base de datos. Para solventar este percance se buscaron diversas formas de indicar a través de los esquemas que no admitiese más parámetros, siendo infructuosos todos los intentos realizados. Debido a esto se optó por realizar sus JSON-schemas con ajv, tanto obligatorios como con los campos opcionales para poder asegurar los datos que se recibían, acabando finalmente teniendo una doble validación de los datos.

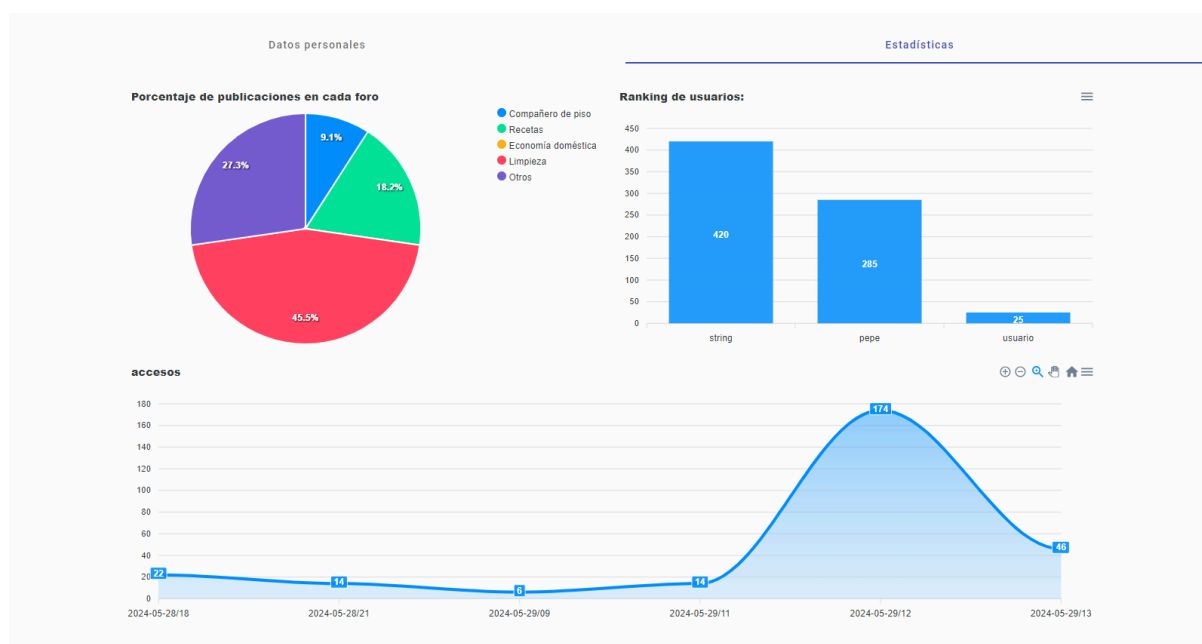
Durante la implementación de las funciones que aparecen en el swagger del proyecto, se ha buscado mantener desacoplados los accesos a base de datos. Para ello se han seguido dos formas distintas de implementación, a modo de estudio. La primera de ellas consiste en aislar el acceso a bd, y llamar a esa función aislada tantas veces como sea necesario. La segunda de ellas es hacer llamadas HTTP a nuestro propio servicio. A futuro se buscaría una unificación de métodos para mantener una consistencia en el código, pero de momento se ha dejado esta división.

Otra decisión que se ha llevado a cabo consiste en implementar como métodos *put* funciones que bloquean y ocultan información (usuarios, posts, mensajes, etc.) y que sustituyen a los métodos *delete*. Esto se debe a que no es aconsejable eliminar totalmente de la capa de persistencia esta información, sino que, aun notificando a los usuarios del sistema la eliminación exitosa, conservar la información marcada como deshabilitada, y no mostrarla. De esta forma, por temas de legalidad se mantiene un registro de toda la información que pasa a través del sistema. Con el mismo fin de mantener la web segura se han creado funciones específicas a las que solo los usuarios registrados pueden acceder, siendo estas escribir en el foro, contactar con el administrador, guardar mensajes, darles me gusta... A los usuarios registrados se les asigna un token que dura 12h en caso de no haber marcado el campo llamado *RememberMe* y 24h en caso de haberlo marcado.

En lo que respecta a los desarrollos opcionales, nos hemos decantado por el despliegue de la API en cloud a través de una imagen de *docker*, y en la creación de un servidor de WebSockets para avisar a los usuarios de las notificaciones a través de modales (arquitectura push). Adicionalmente, se ha optado por realizar login social con google. La implementación de estos opcionales se detalla en los anexos.

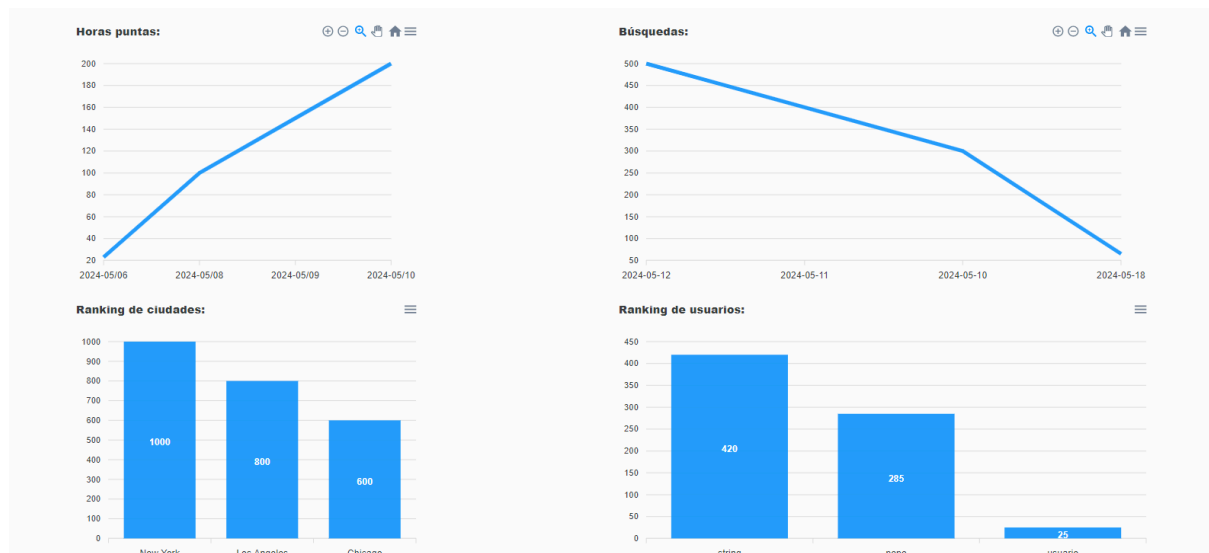
6. Analíticas:

En lo que respecta a las analíticas de la web, se ha optado por realizar dos secciones diferenciadas. La primera de ellas son los gráficos de los usuarios, que se pueden ver en la imagen inferior, en la que se indica el tanto por ciento de publicaciones que tienen en cada foro. De esta manera pueden saber en qué sección están más interesados, y son usuarios más activos. El segundo gráfico mostrado es un ranking de los tres usuarios con mayor reputación en el sistema actualmente. Se ha considerado interesante mostrar este gráfico para fomentar la participación de los usuarios en la web. Por último se muestran los accesos de los usuarios a la web. En este caso, dado que se está usando un sujeto de prueba, el número de accesos es muy elevado, puesto que es el que se usa para todas las pruebas.



La segunda sección de analíticas es la zona del administrador, en la que hemos optado por mostrar 6 gráficos, una de ellas repetida, que es la del ranking de los usuarios. Adicionalmente, mostramos el ranking de las tres ciudades más buscadas a través de la web, para saber así qué ciudades son las que más les interesan a nuestros usuarios. El siguiente gráfico que tenemos es el de las horas punta. Se ha considerado importante para el administrador dado que si a futuro se quisiese monetizar la web con anuncios, se sabría a qué horas es más caro. También es útil para saber en qué momentos del día hay más usuarios, y por tanto pueden surgir problemas por demasiadas peticiones en la web.

Por otro lado tenemos el gráfico de búsquedas de pisos realizadas. Esto sirve para realizar un estudio del mercado y saber en qué fechas y que meses es más común buscar pisos y compañeros.



Los últimos dos gráficos que encontramos son para mantener un registro de la tendencia de los usuarios en la web, para saber si los registros son constantes y si los inicios de sesión varían mucho. Se ha considerado interesante que sean gráficos diferenciados de los anteriormente mencionados dado que la funcionalidad principal de la web son los foros. Por ello es probable que los usuarios no busquen pisos, pero mantengan una actividad frecuente.



Para la implementación de esta sección se ha creado una colección aparte en la que se van almacenando los datos. De manera adicional, para el gráfico de las horas puntas se almacena en memoria los usuarios activos cada hora del día. Tras el paso de este periodo de tiempo (una hora) se ejecuta un trigger que se ha desarrollado en MongoDB Atlas para que contabilice el número de estos usuarios, reiniciar la lista y almacene los datos en las estadísticas pertinentes (el código se encuentra en el repositorio de backend, en *doc/triggers*). Para la implementación en la parte del frontend se ha usado [apexcharts.js](https://apexcharts.com/), una biblioteca de gráficos interactivos en JavaScript que permite crear una amplia variedad de gráficos y visualizaciones de datos, siendo la mayoría de ellos responsive.

7. Análisis de problemas potenciales:

Como principal limitación en nuestro proyecto, en la parte del backend hemos encontrado que la API de *Idealista* de la que se leen los datos tan solo proporciona 100 peticiones al mes. Para poder subsanar este problema se ha tomado la decisión de cargar en memoria los datos de idealista una vez al día, implementando un trigger en Atlas que se ejecuta dos veces al día para cargar los datos en memoria. Como desventaja encontramos que la búsqueda de pisos es muy competitiva y que las últimas actualizaciones no se recibirán de

forma inmediata desde nuestra web, dado que cada día añadimos 20 pisos a la base de datos.

Adicionalmente en lo que respecta al frontend se han encontrado limitaciones a la hora de usar *openStreetMaps*. Esto se debe a que en el primer diseño de la web se pensó en aplicar filtros que permitieran marcar los puntos de interés en el mapa, cercanos a los pisos, y poder destacar aquellos que le interesaban a los usuarios. Sin embargo, estos puntos de interés se encontraban marcados en *openStreetMaps* de forma permanente, sin ninguna alternativa real para emplear un sistema de filtrado que permitiese mostrar y ocultar al gusto del usuario, razón por la cual se acabó dejando de lado la idea de filtrado.

8. Validación

Frontend

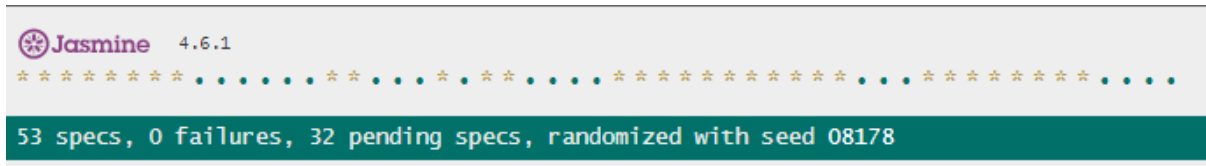
Se ha llevado a cabo una validación E2E, para ello se ha usado [Cypress](#), en el que se han definido 5 pruebas sobre los componentes que se han considerado oportunos, siendo estos:

Componente	Tests	Descripción
BuscarPisoComponent	"should create" "debería filtrar los datos correctamente" "debería formatear los datos de respuesta correctamente"	Conjunto de test para comprobar el correcto filtrado y formateo de la búsqueda.
CompaneroPisoComponent	"should create" "Debería manejar los posts favoritos" "Debería manejar darle like a un post" "Debería actualizar los posts en la búsqueda" "Debería manejar la recogida de post de invitados" "Debería abrir el dialogo para reportar un post"	Conjunto de test que comprueban la obtención de posts, interacciones con post al marcarlos como favoritos, darle like, reportar o actualizar todos los post en la búsqueda.
ListaComponent	"should create" "Debería navegar a info piso" "Debería actualizar los datos del paginador"	Conjunto de test que aseguran la paginación de la lista de pisos y la correcta navegación al piso seleccionado.
DenunciasDelForoComponent	"should create" "Debería obtener las denuncias" "Debería actualizar el paginador" "Debería reformatear los datos de las denuncias"	Conjunto de test que comprueban la obtención de las denuncias, la aplicación del formato correcto y la paginación.
EstadísticasComponent	"should create" "Debería situar las gráficas"	Conjunto de test para asegurar la definición y carga de datos

	correctamente" "Debería obtener las gráficas del usuario" "Debería obtener las gráficas del administrador"	en las estadísticas tanto de usuario como de administrador
--	--	---

El resto de bases de test creadas por defecto al emplear comandos para generar componentes han sido marcados como test en desarrollo y no se tienen en cuenta.

Al ejecutarlos se ha obtenido el siguiente resultado:



Backend

A lo largo del desarrollo se han realizado diversas pruebas de cada una de las funciones implementadas, haciendo uso del API de Swagger de forma manual al igual que algunas pruebas en lo que respecta a los webSockets que se ha implementado con postman. Adicionalmente se ha optado por desarrollar test específicos en aquellas secciones que se han considerado críticas en el sistema, siendo estas la creación de nuevos usuarios y el funcionamiento básico de los foros. Para ello se han usado [mocha y chai](#). Los resultados obtenidos se muestran en la tabla a continuación:

```

● cmarzo@LAPTOP-IGSCBL6J:/mnt/c/Users/crisim/Desktop/PracticasUni/Independizate/independizate$ npm test --env-file=.env

> independizate@0.10.0 test
> ./node_modules/.bin/mocha --reporter spec

Registrar un nuevo usuario
  ✓ Registrar un nuevo usuario con datos correctos (303ms)
  ✓ Iniciar sesión con el nuevo usuario (119ms)

Iniciar sesión con el usuario de pruebas
  ✓ Iniciar sesión con el usuario de pruebas (111ms)

Crear un nuevo post
  ✓ Crear un nuevo post con la sesión de usuario (294ms)

Comprobar el post creado
  ✓ Comprobar los campos del post creado (747ms)

Crear un mensaje en el nuevo post
  ✓ Crear un mensaje en el nuevo post (624ms)

Comprobar el nuevo mensaje
  ✓ Comprobar el nuevo mensaje (113ms)

Responder al mensaje
  ✓ Responder al mensaje (964ms)

Comprobar la respuesta al mensaje
  ✓ Comprobar la respuesta al mensaje (102ms)

9 passing (3s)

```

Se quiere destacar que al hacer los tests se ha tenido que migrar el proyecto de backend a ESM, mediante la introducción de **"type": "module"** en el fichero *packages.json*.

9. Problemas encontrados durante el desarrollo y bugs conocidos:

Dificultades Frontend

En lo que respecta a la implementación del frontend la mayor dificultad encontrada ha sido con Angular Materials, una biblioteca de componentes proporcionada para angular que asegurar ser responsive, sin embargo, tras la implementación de componentes de la misma en la página web, se detectó que no son elementos responsive, o que al menos en nuestro caso no actuaban como tal. Esto ha supuesto un incremento de las horas de trabajo dado que se ha tenido que hacer de forma manual el estudio del comportamiento de cada uno de los elementos en pantallas de tamaño reducido. Adaptando el comportamiento de los componentes al tamaño de pantalla de forma manual en toda la Web. Adicionalmente, ciertos de estos materiales tenían reacciones con los test del frontend, lo que retrasó su desarrollo notablemente.

Otro limitante importante ha sido a la hora de implementar el segundo logging social, tras la implementación del primero, se barajó diferentes opciones, decantándose finalmente por Facebook, pero esto resultó ser una mala idea debido a la necesidad de confirmar tu identidad como empresa, lo que no era posible. Tras esto se trató de emplear X (antiguamente Twitter) pero la falta de documentación oficial enfocada a Angular dificultó el progreso con esta opción. Finalmente se trató de emplear github como logging social, pero a esas alturas se consideró que existían otras prioridades y se acabó abandonando la implementación de un segundo login social.

La última dificultad encontrada ha sido que las imágenes en base64 cuando nos conectamos a la base de datos desplegada en Atlas da problemas, mientras que esas mismas imágenes en local no han supuesto ningún problema a la hora de realizar las peticiones de las funciones. Para solventar este problema, hemos revisado la forma de enviar la información desde el frontend hacia el backend, y por último hemos tenido que quitar el middleware *perfect-express-sanitizer* del servidor backend.

Dificultades Backend

A la hora de llevar a cabo este proyecto se han encontrado diferentes dificultades, la primera de ellas fue conseguir conectarse a la API de idealista, dado que no quedaba claro en la documentación proporcionada por los desarrolladores como funcionaba el Token, implementado con Oauth2, que es una variante de Bearer token. Al carecer de experiencia a la hora de utilizar este método de autenticación supuso una dificultad añadida hasta que se consiguió adquirir el token que permite realizar la solicitud a la función que proporcionan para adquirir los datos de los pisos. Tras esto el equipo se percató de que el Token caduca cada 12 horas, por tanto si se quiere usar la función para adquirir los pisos previamente hay que solicitar el Token. A lo que se le añade que tan solo se aceptan 100 peticiones al mes. Esto ha supuesto un reto a la hora de obtener los datos y gestionarlos. Como solución final se optó por crear un trigger que descarga la información de idealista una vez al día, limitando así el uso del token, y evitando problemas de falta de peticiones.

Otra dificultad que se ha encontrado ha ocurrido a la hora de desplegar el backend. Como primera opción se seleccionó Render, pero nos dimos cuenta de que no era posible dado que usábamos dos puertos diferentes, uno para el servidor de WebSockets, y otro para el servidor de las peticiones HTTP. Se intentó redirigir las peticiones a través de un mismo puerto y gestionarlas de manera interna, pero Render continuaba detectando el uso de dos servidores y forzaba la promoción del plan a una versión de pago. Al final se optó por usar Microsoft Azure y desplegar una imagen docker de la API en la que se podían usar ambos puertos como en la implementación original.

Por último, al implementar algunos tests unitarios ha habido que migrar el proyecto a ESM.

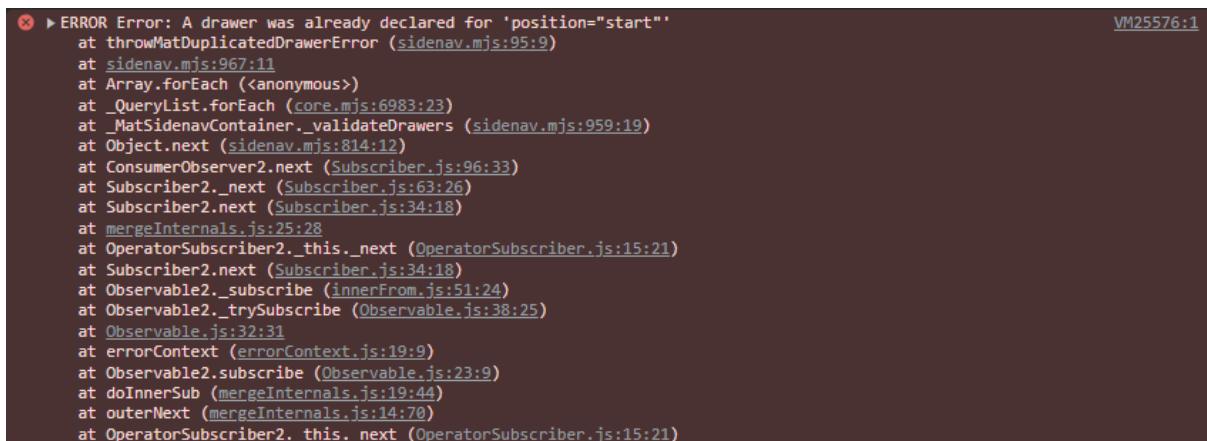
Problemas de la aplicación:

Se han detectado dos problemas principales que arrastra la aplicación en la parte del backend. El primero de ellos es que los pisos de idealista los almacenamos de 20 en 20, esto limita mucho el alcance de búsqueda de los usuarios, dado que la búsqueda de pisos en zonas cercanas a la universidad son muy competitivas y la velocidad es un factor que puede marcar la diferencia.

Como segundo error encontramos que dada la dependencia de determinadas funciones (desde unas se llama a otras), falta la documentación de algunos casos de error en el swagger.

En lo que respecta al frontend de la aplicación se han encontrado diversos problemas con el tratamiento de imágenes. En el caso concreto de las estampas de la aplicación no hemos sido capaces de transmitir las desde el frontend hasta el backend en un formato adecuado lo que ha generado que se tome la decisión de no implementar el cambio de las estampas favoritas de los usuarios. A futuro este punto se tendría que mejorar, pero dado el tiempo de desarrollo y la proximidad de la entrega, se ha optado por marcarlo como tarea pendiente.

Además, en la consola de la aplicación aparecen dos tipos de errores que no hemos podido eliminar de la consola. El primero trata sobre la superposición de elementos por la configuración de Angular Material al hacerlo responsive a mano, y el segundo sobre la forma de inicializar las gráficas de las analíticas.



```
✖ ERROR Error: A drawer was already declared for 'position='start''
    at throwMatDuplicatedDrawerError (sidenav.mjs:95:9)
    at sidenav.mjs:967:11
    at Array.forEach (<anonymous>)
    at _QueryList.forEach (core.mjs:6983:23)
    at _MatSidenavContainer._validateDrawers (sidenav.mjs:959:19)
    at Object.next (sidenav.mjs:814:12)
    at ConsumerObserver2.next (Subscriber.js:96:33)
    at Subscriber2._next (Subscriber.js:63:26)
    at Subscriber2.next (Subscriber.js:34:18)
    at mergeInternals.js:25:28
    at OperatorSubscriber2._this._next (OperatorSubscriber.js:15:21)
    at Subscriber2.next (Subscriber.js:34:18)
    at Observable2._subscribe (innerFrom.js:51:24)
    at Observable2._trySubscribe (Observable.js:38:25)
    at Observable.js:32:31
    at errorContext (errorContext.js:19:9)
    at Observable2.subscribe (Observable.js:23:9)
    at doInnerSub (mergeInternals.js:19:44)
    at outerNext (mergeInternals.js:14:70)
    at OperatorSubscriber2._this._next (OperatorSubscriber.js:15:21)
```

Imagen del error de superposición de elementos de angular material

```
3 ▶ ERROR TypeError: Cannot read properties of undefined (reading 'series') VM25576:1
    at EstadisticasComponent_Template (estadisticas.component.html:5:28)
    at executeTemplate (core.mjs:11268:9)
    at refreshView (core.mjs:12791:13)
    at detectChangesInView$1 (core.mjs:13015:9)
    at detectChangesInViewIfAttached (core.mjs:12978:5)
    at detectChangesInComponent (core.mjs:12967:5)
    at detectChangesInChildComponents (core.mjs:13028:9)
    at refreshView (core.mjs:12841:13)
    at detectChangesInView$1 (core.mjs:13015:9)
    at detectChangesInViewIfAttached (core.mjs:12978:5)
    at detectChangesInEmbeddedViews (core.mjs:12935:13)
    at refreshView (core.mjs:12815:9)
    at detectChangesInView$1 (core.mjs:13015:9)
    at detectChangesInViewIfAttached (core.mjs:12978:5)
    at detectChangesInComponent (core.mjs:12967:5)
    at detectChangesInChildComponents (core.mjs:13028:9)
    at refreshView (core.mjs:12841:13)
    at detectChangesInView$1 (core.mjs:13015:9)
    at detectChangesInViewIfAttached (core.mjs:12978:5)
    at detectChangesInEmbeddedViews (core.mjs:12935:13)
```

Imagen del error de inicialización de gráficos

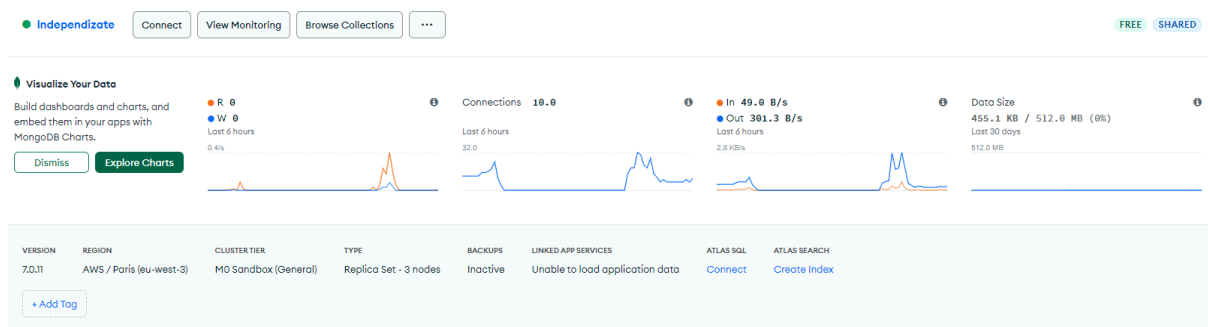
10. Despliegue del sistema:

Frontend:

Se ha decidido subir la aplicación a Microsoft Azure como imagen de contenedor. En primer lugar se había intentado utilizar GitHub Pages, pero la página principal no cargaba. Como ya se había adquirido experiencia desplegando el backend, se ha seguido el mismo procedimiento, descrito en el [Anexo III](#).

Base de datos:

Para poder acceder de forma remota a los datos del sistema y que estos tengan persistencia se ha subido la base de datos creada en mongodb a Atlas, que ofrece un tamaño de almacenamiento que va desde los 512MB a los 5GB. Como punto positivo dado el carácter del proyecto, encontramos que no pide tarjeta de crédito para el uso de la versión gratuita. En el caso de Amazon Web Services encontramos el cobro de 1 euro por empezar a usar sus servicios, y una limitación de espacio en la que se cobra sin avisar si te excedes. Por otro lado en Azure solo se han encontrado opciones de subida de base de datos al cloud para MySQL. En conclusión, se ha considerado Atlas como la mejor opción para la base de datos, bajo el nombre de Independizate.



API:

Se ha creado una imagen de los servicios de la API para utilizar Azure. Se ha tomado esta decisión dado que se utilizan dos puertos en la aplicación, puesto que se despliegan dos servidores (HTTP y WebSockets). Esto hace que la versión gratuita de Render no nos sirviese dado que sólo proporciona un puerto en la versión gratuita. Los pasos seguidos se encuentran en el [Anexo III](#).

11. Valoración personal de cada miembro:

Cristina Marzo

La asignatura es muy completa y me parece básica para la formación de un graduado en ingeniería informática hoy en día. Además, habría sido muy útil haberla cursado antes de *Proyecto Software*, ya que nos habría ayudado a controlar los aspectos técnicos básicos y poder cumplir los plazos de forma más sencilla.

Se acompaña el proceso de aprendizaje de los alumnos desde el primer momento, explicando en clase la tecnología a utilizar en el proyecto de la asignatura. Las clases de prácticas y los ejemplos de sistemas funcionales son muy útiles a la hora de empezar a implementar el proyecto, para ir entendiendo cómo hacer las cosas.

Daniel Carrizo

Me parece una asignatura con objetivos claros y una metodología relacionada al objetivo final de la asignatura. El realizar un proyecto en vez de ir a examen nos permite enfrentarnos a las tecnologías de mejor forma que con prácticas individuales.

En relación al proyecto, creo que ha sido un proyecto interesante, que nos ha hecho tener que enfrentarnos a trabajar en equipo, buscarnos la vida para aprender a implementar lo que queríamos conseguir en nuestra aplicación, tomando decisiones como a qué llegamos y a qué no.

Considero que hemos logrado los objetivos de la asignatura, del trabajo realizado por los diferentes integrantes, puesto que ha habido buena comunicación y los resultados han sido satisfactorios.

Ayelen Nuño

En conjunto, me ha parecido una asignatura muy completa, que proporciona las bases necesarias para aprender a crear un servicio web y estructurarlo correctamente, además de aprender a documentar el código y a buscar módulos de interés que resultan muy útiles a la hora de programar. Además, está acompañada en todo momento por el profesorado, quienes hacen una gran labor, explicando las lecciones, mostrando una forma práctica de aplicación y resolviendo las dudas que surgen en todo momento, buscando siempre que las clases sean amenas y que los conceptos queden claros.

Me ha parecido muy interesante desarrollar el proyecto, y se ha agradecido mucho haber usado previamente los módulos necesarios en prácticas, lo que nos ha dado una base con la que trabajar y una línea definida de pasos a seguir. Considero que se han logrado los

objetivos de la asignatura gracias al trabajo realizado por los miembros del equipo, quienes han dado su mejor esfuerzo en todo momento, ayudando siempre a tener una buena comunicación y buscando los mejores resultados posibles.

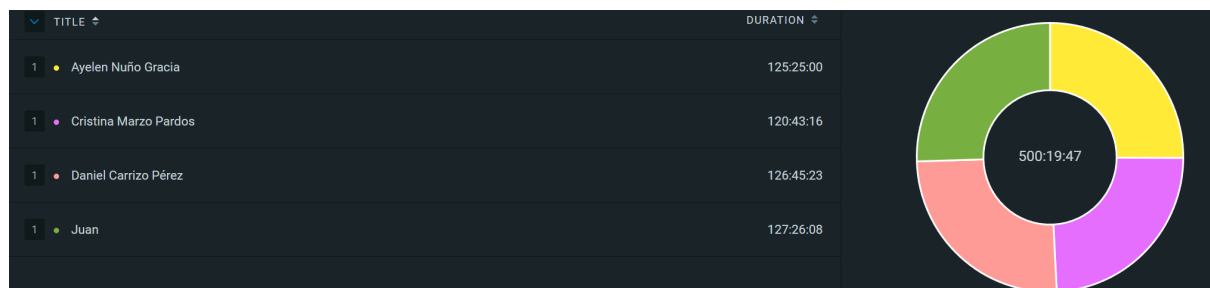
Juan Pellicer

Esta asignatura ha mostrado un gran número de tópicos durante estos meses, tratando el tema de los servicios web desde diversos puntos de vista, permitiéndonos aprender tanto en el ámbito del front como del back, manteniendo siempre la cohesión y la relación entre todas las lecciones aprendidas para converger en un entendimiento de un proceso de desarrollo completo.

Este proyecto ha sido mi primera vez empleando Angular, y si bien las advertencias de que la curva de aprendizaje de este framework era bastante extraña han sido ciertas, he podido afianzar conocimientos y experiencia en su uso y obtener confianza a la hora de probar entornos nuevos. El proyecto ha sido una manera interesante de plantear la asignatura, al conocer desde el principio el objetivo y importancia de este, he podido apreciar con mayor facilidad la relevancia de los diferentes temas dados a medida que los veía, en vez de ver la importancia de cara a las últimas semanas del curso como suele ocurrir con otras asignaturas, por lo que espero que se tome este tipo de planificaciones más en cuenta para el futuro de esta carrera.

12. Distribución de tiempo:

Sumatorio de horas totales



Desglose de tareas por participante

Usuario	Descripción	Tiempo (h)	Tiempo (decimal)	Usuario	Descripción	Tiempo (h)	Tiempo (decimal)
Ayelen Nuño Gracia		125:25:00	125,42	Cristina Marzo Pardos		120:43:16	120,72
	Rediseño de la interfaz	02:00:00	2,00		Reunión definición caracteri	00:59:10	0,99
	Diseño pantallas perfil del u	03:15:00	3,25		Memoria	06:50:03	6,83
	Entrega final	01:23:12	1,39		Preparar presentación no fir	00:22:06	0,37
	Revisión reunión Fabra	00:30:00	0,50		Prototipo tablet, estructura	01:30:00	1,50
	Conceptos API	04:22:24	4,37		Reunión tomar decisiones	01:35:43	1,60
	API	03:00:00	3,00		Programación API	77:35:57	77,60
	Especificación contenido de	01:30:00	1,50		Esquema de datos	01:54:00	1,90
	Arreglos última revisión	01:35:00	1,58		Reunión	00:22:53	0,38
	Triggers y Memoria	03:10:34	3,18		Testing API	04:50:48	4,85
	Reunión revisión del diseño	02:30:00	2,50		Reunión establecer tema pr	01:00:00	1,00
	Reunión tomar decisiones	01:34:40	1,58		Figma	08:09:23	8,16
	Diseño pantalla login y logo	02:00:00	2,00		Conceptos API	07:22:10	7,37
	Programación API	80:39:58	80,67		Empezar Figma	02:00:00	2,00
	Limpeza de código	00:36:00	0,60		Revisión del proyecto	00:22:48	0,38
	Memoria	05:02:05	5,03		Despliegue Backend	02:30:47	2,51
	Diseño pantallas móvil	04:40:00	4,67		Reunión figma	03:17:28	3,29
	Despliegue backend	01:05:06	1,09				
	Esquema de datos	01:54:27	1,91				
	Reunión Figma	00:48:00	0,80				
Daniel Carrizo Perez	Reunión definición caracteri	01:00:00	1,00	Juan		127:26:08	127,44
	Testeo	02:48:34	2,81		Figma - Pasar notificaciones	00:07:59	0,13
		128:07:22	128,12		Figma - Estadísticas y come	00:55:55	0,93
	Reunión con Fabra para apr	00:30:00	0,50		Figma - Contacta	00:39:24	0,66
	Buscar piso v1 y arreglos de	01:58:16	1,97		Reunión definición caracteri	01:00:00	1,00
	Avances	13:20:40	13,34		Figma - FAQ	00:35:47	0,60
	Continuamos con la realizac	09:13:40	9,23		Reunión prototipo movil	00:53:31	0,89
	Familiarizarse con Angular	03:42:18	3,71		Modelo estático - saltos	00:18:31	0,31
	Avance de pantallas	26:00:00	26,00		Frontend - Login	30:10:03	30,17
	Aprendiendo angular mater	03:30:00	3,50		Modelo - estático Admin	02:16:15	2,27
	Creación de los esquemas a	01:15:00	1,25		Figma - arreglos	01:02:21	1,04
	Estadísticas	04:12:19	4,21		Modelo estático - Saltos/For	02:09:42	2,16
	Integración pantallas	33:35:47	33,60		Modelo estático - Foro / Bus	03:53:12	3,89
	Avances (CSS, integrar back	17:02:36	17,04		Modelo estático - Side bar /	04:19:44	4,33
	Documentación de los esqui	00:45:00	0,75		Modelo estático - Foro	02:06:54	2,12
	Actualización de los códigos	01:17:25	1,29		Modelo estático - Foro / Cor	03:57:46	3,96
	Reunión para decidir los est	01:54:00	1,90		Reunión análisis figma	02:31:00	2,52
	Comienzo de la realización c	02:00:00	2,00		Frontend	47:24:08	47,40
	Avances (CSS)	00:28:22	0,47		Frontend - Inicio	03:22:22	3,37
	Foro v1	06:00:00	6,00		Modelo estático	14:18:46	14,31
	Partes finales	01:21:59	1,37		Figma - Pasar a movil	00:53:29	0,89
					Reuniones + figma	04:29:19	4,49

13. Conclusión:

Creemos que el proyecto ha alcanzado un nivel satisfactorio de cumplimiento, tomando en cuenta los resultados obtenidos y los requisitos previamente establecidos. Consideramos que es una idea original, enfocada de una forma gamificada, lo que incentiva a los usuarios a interactuar con el sistema. Durante el desarrollo, hemos adquirido conocimientos sobre la estructura del stack MEAN, así como habilidades para utilizar cada una de sus tecnologías y organizar un proyecto de estas características.

A lo largo del proceso, hemos realizado ajustes y tomado decisiones en función de los datos y servicios proporcionados por las fuentes seleccionadas, sin embargo, logramos mantener en su mayoría la fidelidad al diseño original. Este proyecto ha sido una experiencia de aprendizaje continuo y colaborativo, donde contamos con las herramientas necesarias en todo momento para llevarlo a cabo. Aunque se trató de un proyecto ambicioso, creemos haber estado a la altura de los desafíos que presentó.

14. Anexos de los desarrollos opcionales:

Anexo I: Implementación de WebSockets

Para que los usuarios reciban notificaciones de tipo modal, cuando se les responde a un post del foro, a un mensaje o bien para que sepan que el administrador les ha enviado una respuesta a su duda y/o denuncia, se ha implementado un servidor de Websockets, en el que se utiliza el propio id de los usuarios, siendo este su identificador único proporcionado por la base de datos para establecer el nombre del canal. De esta forma conseguimos que el canal para cada usuario sea único, y sea sencillo automatizar el envío de los mensajes. Los pasos que se tiene que crear para establecer este canal de comunicación son los siguientes:

1. **const socket = new WebSocket('ws://localhost:4000?clientId=123');**

Se quiere destacar que es necesario sustituir el campo 123 por el id del usuario proporcionado por la API al realizar el proceso de logging.

2. **socket.addEventListener('open', () => { console.log('Conexión establecida con el servidor WebSocket: ', event.data);});**
3. **socket.addEventListener('message', (event) =>{console.log('Mensaje recibido del servidor:', event.data);});**

Los distintos mensajes que se pueden recibir son:

- sugerencia/queja respondida
- mensaje respondido
- mensaje posteo
- denuncia respondida
- Has ganado una nueva estampa

En el frontend se han manejado estos errores con un modal que se muestra cuando llega una notificación vía WebSocket.

Anexo II: Login social

Para poder lograr el login social por google, se ha tenido que crear una aplicación en el servicio de apis y credenciales de google, proporcionando a esta diferentes urls como orígenes y redireccionamientos autorizados, adquiriendo así la id de cliente necesaria para proporcionar el login social en nuestra aplicación

La autenticación se realizaba mediante el SDK de google authentication que permitía la inicialización de un objeto auth2 con la información de la id de cliente, permitiendo enlazar dicha autenticación con nuestra aplicación.

Tras interactuar con el proceso de login por google, el front era presentado con diferentes datos del usuario, los más interesantes siendo el "id" y el "token", siendo el id un

identificador permanente para el usuario, y el token siendo un elemento temporal. Al permitir google que envíes una petición con un token para validarlo, se puede emplear dicho token como una contraseña temporal, nuestra metodología para ello fue realizar una validación en el front end para posteriormente enviar el token junto a la id y otros datos al backend, los cuales procedían a realizar una segunda validación con ese mismo token, probando así la veracidad del usuario y pudiendo ahora interactuar con su cuenta.

Hay que destacar que por razones que no conocemos a veces no se realiza el login social.

Anexo III: Despliegue del sistema sobre docker y una infraestructura en la nube

Se ha desplegado el sistema como contenedor de docker sobre Microsoft Azure. En el caso del backend, tal y como se ha comentado anteriormente, debido a los problemas surgidos con Render. En el caso del frontend, debido a problemas surgidos con GitHub Pages.

Para poder llevar a cabo el despliegue se han seguido los siguientes pasos (se indican los seguidos para la imagen del backend, pero son análogos para el frontend):

1. Nos hemos registrado como usuarios en Azure usando la versión de estudiante que proporciona 100\$
2. Hemos creado un grupo de recursos (a través de la interfaz de Azure)
3. Posteriormente se ha procedido a crear un registro de contenedores dentro de este grupo de recursos, al que se le ha asignado el nombre de independizate.
4. El siguiente paso consiste en loguearse desde nuestros equipos dentro del registro de contenedores que se ha creado en Azure. Para ello se han ejecutado los siguientes comandos:

```
az login
az acr login --name independizate --expose-token
docker login independizate.azurecr.io --username 00000000-0000-0000-0000-000000000000 -p <accessToken>
```

5. Se construye la imagen de docker desde el código del proyecto. Los ficheros utilizados se encuentran en el directorio *docker* del proyecto: *Dockerfile* y *build.sh*. Se ha optado por crear este ejecutable bash para copiar todos los ficheros de código fuente del proyecto dentro del directorio docker y eliminarlos tras crear la imagen. La creación de la imagen se hace con el siguiente comando:

```
docker build -t independizate/backend:latest .
```

6. Una vez construida la imagen, se asigna una nueva etiqueta para preparar la subida al registro de contenedores de Azure, con el siguiente comando:

```
docker tag independizate/backend:latest independizate.azurecr.io/independizate/backend:latest
```

7. Después, se sube esta imagen al registro de contenedores con este comando:

```
docker push independizate.azurecr.io/independizate/backend:latest
```

8. Por último, se crea desde el portal web de Azure una instancia de contenedor que utiliza la imagen creada, con los puertos 3000 (HTTP) y 4000 (Websockets) redireccionados.

Hay que destacar que la primera vez que se intentó subir la imagen a Azure se obtuvo un error, dado que el registro de contenedores no tenía configurado un rol de administrador. Esto se ha solucionado con el siguiente comando:

```
az acr update -n independizate --admin-enabled true
```

