



TRACER

Javier Gámez Mendoza

[Repositorio Github](#)

1. Introducción.....	3
2. Identificación de las necesidades de Tracer.....	4
3. Análisis alternativas en mercado.....	6
4. Justificación del proyecto.....	7
5. Stack tecnológico.....	8
6. Modelo de datos.....	10
7. Prototipo Figma.....	13
8. Documentación API.....	17
9. Manual de despliegue.....	20
10. Conclusión.....	25

1. Introducción

Cualquier aficionado al motor sabe que en ocasiones, encontrar la solución a un problema concreto puede volverse una tarea tediosa, en algunos casos llega a ser frustrante y en el peor de ellos, hace que una bonita afición acabe por convertirse en una fuente inagotable de desgracias.

De esa idea nace **Tracer**, una herramienta con una premisa muy simple, ayudar a todos aquellos amantes del motor a compartir sus experiencias y conocimientos, para hacer la vida del resto un poco más fácil.

El principal motivo por el que he decidido abordar un proyecto de estas características es que yo soy uno de esos aficionados. No es nada sencillo encontrar valiendote de foros por qué una moto de hace 30 años no carbura de forma correcta. La información está, pero está terriblemente dispersa en decenas de plataformas centradas en una marca o un modelo concreto de vehículo.

Ese es el problema que pretende solucionar **Tracer**.

2. Identificación de las necesidades de Tracer

Tracer busca convertir el anticuado modelo de foros en una modernizada plataforma Restful con capacidad de escalabilidad, además de la capacidad de una futura integración en dispositivos móviles y un ambiente más social .

Tracer permite:

1. La identificación de usuarios por medio de credenciales contrastadas en base de datos, que permiten la securización de la aplicación y la privacidad de los usuarios.
2. La interacción del usuario con los elementos de la aplicación, mayoritariamente la lectura y participación en foros de resolución de dudas referentes a vehículos.
3. El seguimiento personalizado de preferencias del usuario mediante un sistema de favoritos que engloba tanto a los vehículos y a los hilos como a los propios usuarios.
4. La gestión de usuarios, hilos, vehículos, respuestas y demás recursos generados por los administradores de la aplicación.
5. La distinción de usuarios **Fiabiles** que permiten localizar rápidamente respuestas de calidad.
6. La resolución de hilos, facilitando la búsqueda de la información
7. La protección de usuarios mediante un sistema de vetos y reportes manejados por los administradores.
8. La aplicación plantea una interfaz intuitiva, rápida y simple que permite a los usuarios una rápida adaptación al entorno.

Requisitos funcionales concretos:

- RF01 - El sistema debe permitir un login
- RF02 - El sistema debe permitir un logout
- RF03 - El sistema debe permitir el registro de usuarios
- RF04 - El sistema debe permitir editar el perfil de usuario.
- RF05 - El sistema debe permitir añadir vehículos (Administrador)
- RF06 - El sistema debe permitir crear un hilo dentro de un vehículo
- RF07 - El sistema debe permitir el seguimiento de hilos y vehículos (añadir a favoritos)
- RF08 - El sistema debe permitir leer y responder en un hilo (NO a tiempo real, tipo foro)

- RF09 - El sistema debe permitir concluir un hilo (Creador y Administrador)
- RF10 - El sistema debe permitir ver el listado general de vehículos/hilos de un vehículo.
- RF11 - El sistema debe permitir que el administrador elimine comentarios ofensivos.
- RF12 - El sistema debe permitir el veto de usuarios problemáticos de la aplicación.
- RF13 - El sistema debe permitir la búsqueda de vehículos por nombre y el filtrado por marca, año, etc.
- RF14 - El sistema debe permitir el seguimiento de usuarios concretos (añadir a favoritos).
- RF15 - El sistema debe permitir un feed donde se muestre contenido de interés/favoritos ordenado por antigüedad.
- RF16 - El sistema debe permitir buscar usuarios por nombre.
- RF17 - El sistema debe permitir que el usuario creador de un hilo marque como solución la respuesta de otro usuario en su hilo, esta acción concluye el hilo.
- RF18 - El sistema debe permitir que un usuario bloquee a otro usuario para no recibir sus comunicaciones, reportarlo a él o a algún mensaje concreto.
- RF19 - Sistema de usuarios **Fiables**. Usuarios veteranos o con varias respuestas correctas registradas serán distinguidos con insignias.
- RF20 - El administrador puede consultar estadísticas referentes a la resolución de hilos en función de marcas y modelos.

3. Análisis alternativas en mercado

La idea de Tracer nace precisamente de las buenas ideas mal implementadas de la competencia:

1. Forocoches

Forocoches es quizá la plataforma forera dedicada al motor más reconocida al menos en España, y no es para menos ya que fue una de las primeras en abordar la idea.

Aún y con todo ello, forocoches peca de una larga lista de errores que abordamos:

La antigüedad de su stack tecnológico la encasilla casi automáticamente como una vieja gloria. Existen muchos casos de aplicaciones que quedaron atrás por no saber adaptarse a los tiempos actuales. Forocoches es uno de ellos.

Diseño poco íntegro derivado de su antigüedad, lo que provoca discrepancias visuales y una difícil adaptación a dispositivos móviles.

Dispersión en sus temas, aunque se ha comentado que forocoches es una plataforma nacida para el motor como su nombre indica, realmente se ha convertido en un foro multipropósito en el que los temas divagan tanto como quieran los usuarios.

2. Otras plataformas

Forocoches no es la única plataforma que existe ni mucho menos, pero básicamente pecan de lo mismo. El mundo del motor fue introducido a la web en una época muy temprana, por lo que las tecnologías usadas entonces ahora se consideran prácticamente obsoletas en comparación.

ForoClub, AudiSport-Iberia, Club Seat, Club Toyota, BmwFaqClub y un largo etcétera.

4. Justificación del proyecto

El proyecto busca amenizar la búsqueda de información, proporcionando un ambiente más familiar para el usuario moderno, con portabilidad a dispositivos móviles, con interfaces agradables y acordes a los tiempos que corren y a velocidades actuales.

Además de todo lo anteriormente mencionado, también plantea la solución a un problema común, **la dispersión de la información**.

Tradicionalmente, la información era condensada en foros concretos dedicados a marcas o modelos específicos de vehículos, donde las motos por supuesto no tenían demasiada cabida. **Tracer** busca unificar todos estos aspectos en una **experiencia global** en la que se tengan en cuenta todas las necesidades de los usuarios.

5. Stack tecnológico

1. ReactJS para el Frontend:

a. Versatilidad y Flexibilidad: ReactJS es una biblioteca de JavaScript extremadamente versátil que permite el desarrollo de interfaces de usuario interactivas y dinámicas. Su enfoque en componentes reutilizables y su arquitectura basada en el concepto de unidireccionalidad de datos (unidirectional data flow) hacen que sea una opción ideal para construir aplicaciones web modernas y escalables.

b. Eficiencia en el Desarrollo: ReactJS ofrece un flujo de trabajo eficiente gracias a su capacidad para dividir la interfaz de usuario en componentes independientes, lo que facilita la colaboración entre equipos de desarrollo y la reutilización de código.

c. Compatibilidad con Dispositivos Móviles: Con el creciente uso de dispositivos móviles, es crucial que las aplicaciones web sean responsivas y se adapten a diferentes tamaños de pantalla. ReactJS ofrece la posibilidad de crear interfaces de usuario adaptables que proporcionan una experiencia de usuario óptima en dispositivos móviles y de escritorio.

d. Comunidad Activa y Ecosistema Amplio: ReactJS cuenta con una gran comunidad de desarrolladores y una amplia variedad de bibliotecas y herramientas complementarias que facilitan el desarrollo y la optimización de aplicaciones web.

2. Spring Boot Java para el Backend:

a. Robustez y Escalabilidad: Spring Boot es un framework de Java que ofrece un entorno de desarrollo robusto y altamente escalable para la construcción de aplicaciones empresariales. Su arquitectura modular y su amplia gama de funcionalidades integradas permiten desarrollar y desplegar aplicaciones de manera eficiente y confiable.

b. Gestión de Dependencias Simplificada: Spring Boot simplifica la gestión de dependencias y la configuración del proyecto mediante el uso de anotaciones y convenciones predefinidas. Esto permite a los desarrolladores

centrarse en la lógica del negocio sin tener que preocuparse por la configuración manual del entorno.

c. Integración con Tecnologías Emergentes: Spring Boot ofrece soporte para una variedad de tecnologías y estándares de la industria, incluyendo RESTful APIs, seguridad, bases de datos relacionales y no relacionales, entre otros. Esto permite integrar fácilmente nuevas tecnologías y adaptarse a los requisitos cambiantes del negocio.

d. Seguridad y Gestión de Transacciones: Spring Boot proporciona características integradas de seguridad y gestión de transacciones que ayudan a proteger los datos sensibles y a garantizar la integridad de las operaciones comerciales.

3. MySQL para la Base de Datos:

a. Fiabilidad y Estabilidad: MySQL es un sistema de gestión de bases de datos relacional ampliamente utilizado que ofrece fiabilidad y estabilidad probadas en entornos de producción. Su robusta arquitectura y su amplio conjunto de características garantizan un rendimiento óptimo y una gestión eficiente de los datos.

b. Escalabilidad Horizontal y Vertical: MySQL es altamente escalable y puede adaptarse a las necesidades cambiantes de las aplicaciones web mediante la escalabilidad horizontal (mediante la adición de más servidores) y la escalabilidad vertical (mediante la mejora de los recursos del servidor).

c. Compatibilidad con Estándares de la Industria: MySQL es compatible con los estándares SQL de la industria, lo que facilita la migración de datos entre diferentes sistemas de gestión de bases de datos y garantiza la interoperabilidad con otras tecnologías.

d. Comunidad Activa y Recursos Abundantes: MySQL cuenta con una gran comunidad de usuarios y una amplia variedad de recursos educativos y de soporte disponibles en línea. Esto facilita el aprendizaje y la resolución de problemas relacionados con el desarrollo y la administración de bases de datos MySQL.

creator_id: Identificador del usuario que creó el hilo.
vehicle_id: Identificador del vehículo asociado al hilo.
title: Título del hilo.
message: Mensaje del hilo.

3. ThreadPost

id: Identificador único del post en un hilo.
creation_date: Fecha de creación del post.
thread_id: Identificador del hilo al que pertenece el post.
user_id: Identificador del usuario que creó el post.
message: Contenido del post.

4. Vehicle

id: Identificador único del vehículo.
creation_date: Fecha de creación del vehículo en el sistema.
brand: Marca del vehículo.
model: Modelo del vehículo.

5. Ticket

id: Identificador único del ticket.
close_date: Fecha de cierre del ticket.
creation_date: Fecha de creación del ticket.
creator_id: Identificador del usuario que creó el ticket.
infractor_id: Identificador del usuario infractor.
message: Mensaje del ticket.

6. User_Thread (Entidad de relación para la relación muchos a muchos entre usuarios y hilos)

thread_id: Identificador del hilo.
user_id: Identificador del usuario.
User_Vehicle (Entidad de relación para la relación muchos a muchos entre usuarios y vehículos)

user_id: Identificador del usuario.
vehicle_id: Identificador del vehículo.

7. **User__User** (Entidad de relación para la relación de seguimiento entre usuarios)

followers__id: Identificador del usuario que sigue.

follows__id: Identificador del usuario seguido.

Relaciones

User-Thread: Un usuario puede crear muchos hilos (relación uno a muchos), y esta relación se almacena en la tabla thread mediante el campo creator__id.

User-ThreadPost: Un usuario puede crear muchas publicaciones en hilos (relación uno a muchos), y esta relación se almacena en la tabla threadpost mediante el campo user__id.

Thread-ThreadPost: Un hilo puede tener muchas publicaciones (relación uno a muchos), y esta relación se almacena en la tabla threadpost mediante el campo thread__id.

User-Vehicle: Un usuario puede seguir a muchos vehículos y un vehículo puede ser seguido por muchos usuarios (relación muchos a muchos), y esta relación se almacena en la tabla user__vehicle.

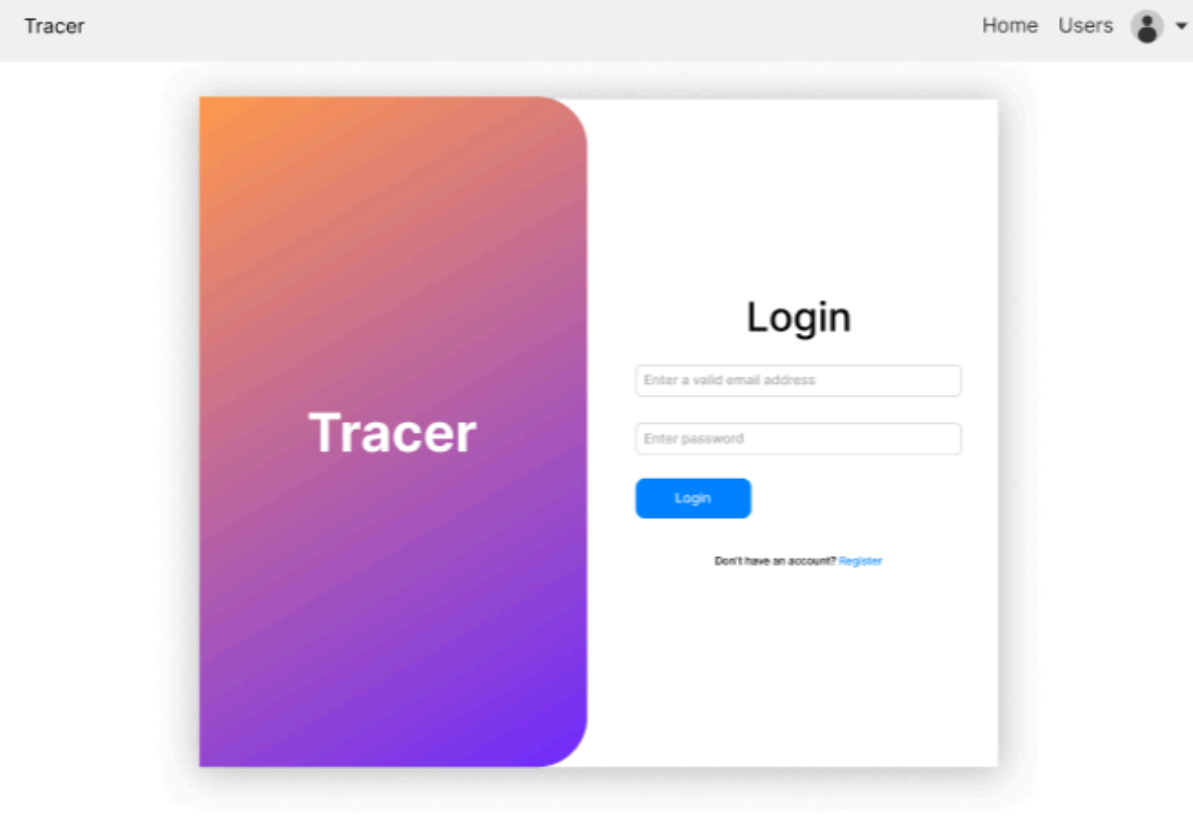
User-Ticket: Un usuario puede crear muchos tickets de vetos (relación uno a muchos) y esta relación se almacena en la tabla ticket mediante los campos creator__id e infractor__id.

User-User: Los usuarios pueden seguir a otros usuarios (relación muchos a muchos) y esta relación se almacena en la tabla user__user.

7. Prototipo Figma

[Enlace al proyecto figma.](#)


Login:



The image shows a login form prototype for an application named 'Tracer'. The form is displayed within a browser window mockup. The browser's address bar shows 'Tracer'. The top navigation bar includes 'Home' and 'Users' links, along with a user profile icon and a dropdown arrow. The login form itself is centered and features a large, rounded rectangle on the left with a vertical orange-to-purple gradient and the word 'Tracer' in white. To the right of this graphic, the word 'Login' is displayed in a bold, black font. Below the title, there are two input fields: the first is labeled 'Enter a valid email address' and the second is labeled 'Enter password'. A blue 'Login' button is positioned below the password field. At the bottom of the form, there is a link that reads 'Don't have an account? Register'.

Registro:

Tracer

Home Users 

Tracer

Register

Enter a username

Enter a valid email address

Enter a biography


Enter password

Login


Already have an account? [Login](#)

Inicio(Vehículos) :

Tracer

Home Users 

Search a vehicle by name or date



Favoritos

Brand: Audi
Model: A4
Resales Date: 1/1/1999

Threads

Add to fav

Brand: Mitsubishi
Model: Montero
Resales Date: 1/1/1999

Threads

Add to fav

Brand: Citroën
Model: Saxo
Resales Date: 1/1/1999

Threads

Add to fav

Brand: Toyota
Model: Corolla
Resales Date: 1/1/1999

Threads

Add to fav

Brand: Jeep
Model: Wrangler
Resales Date: 1/1/1999

Threads

Add to fav


Brand: Hyundai
Model: Tucson
Resales Date: 1/1/1999

Threads

Add to fav

Ventana de hilos de un vehículo:

Tracer

HomeUsers

BMW - E30

Add Thread +

Where is BMW the central office located?

ToyotaFan_0304

Please i need to know where is bmw from.

Get in

Where can i find BMW E30 mechanic?


ToyotaFan_0304

Im searching for a E30 expert to start a project on my own bmw.

Get in

Ventana de respuestas de un hilo:

Tracer

HomeUsers

Where is toyota the central office located?

ToyotaFan_0304

Please i need to know where is toyota from.

Get in

2024-4-1 15:36


Toyota is a Japanese company


ToyotaFan_0304

Get in

Ventana de perfil propio.

Tracer

Home Users 



Profile

Username

BobJohnson

Biography

E30 enthusiast


Email


bob-johnson@gmail.com

Edit

Ventana de usuarios:

Tracer

Home Users 

Search a user by name or date 

Username	email	Biography	Followers	Follows
Alice_Johnson	alice-johnson@gmail.com	A car lover	124	34
Frank_miller	frank@gmail.com	A bike lover	10	23
BobJohnson	bob-johnson@gmail.com	E30 enthusiast	52	774

8. Documentación API

Endpoint	Tipo	Parámetros	Descripción
/users	GET	Ninguno	Obtener todos los usuarios
/users/:id	GET	id: Id del usuario	Obtener un usuario por ID
/users/:id	PUT	id: ID del usuario (URL Path),body: { "username": "NEW USERNAME", "bio": "NEW BIOGRAPHY" }	Actualizar un usuario por ID
/users/:id	DELETE	id: Id del usuario	Eliminar un usuario por ID
/users/follow/thread/:id	PATCH	id: Id del hilo	Seguir un hilo por ID
/users/unfollow/thread/:id	PATCH	id: Id del hilo	Dejar de seguir un hilo por ID
/users/follow/vehicle/:id	PATCH	id: Id del vehículo	Seguir un vehículo por ID
/users/unfollow/vehicle/:id	PATCH	id: Id del vehículo	Dejar de seguir un vehículo por ID
/tickets	GET	Ninguno	Obtener todos los tickets
/tickets/:id	GET	id: Id del ticket	Obtener un ticket por ID
/tickets	POST	body: { "creatorId": 1, "infractorId": 1, "threadpostId": 1, "message": "Thats offensive" }	Crear un ticket
/tickets/:id	PATCH	id: Id del ticket	Cerrar un ticket por ID
/tickets/:id	DELETE	id: Id del ticket	Eliminar un ticket por ID
/threads	GET	Ninguno	Obtener todos los threads

/threads/:id	GET	id: Id del hilo	Obtener un thread por ID
/threads	POST	body: { "title": "New Thread", "message": "Thread content", "creatorId": 1, "vehicleId": 1 }	Crear un thread
/threads/:id	PUT	id: ID del thread (URL Path), body: { "title": "Updated Thread", "content": "Updated content", "userId": 1 }	Actualizar un thread por ID
/threads/:id	PATCH	id: Id del hilo	Cerrar un thread por ID
/threads/:id	DELETE	id: Id del hilo	Eliminar un thread por ID
/threadposts	GET	Ninguno	Obtener todos los threadposts
/threadposts/:id	GET	id: Id del post	Obtener un threadpost por ID
/threadposts	POST	body: { "content": "New post content", "creatorId": 1, "threadId": 1 }	Crear un threadpost
/threadposts/:id	PUT	id: ID del threadpost (URL Path), body: { "content": "Updated post content" }	Actualizar un threadpost por ID
/threadposts/:id	DELETE	id: Id del post	Eliminar un threadpost por ID
/vehicles	GET	Ninguno	Obtener todos los vehículos
/vehicles/:id	GET	id: Id del vehículo	Obtener un vehículo por ID
/vehicles	POST	id: ID del vehículo (URL Path), body: { "brand": "New Brand", "model": "New Model", "creationDate": "2023-10-10" }	Crear un nuevo vehículo

/vehicles/:id	PUT	id: ID del vehículo (URL Path), body: { "brand": "New Brand", "model": "New Model", "creationDate": 2023-10-10 }	Actualizar un vehículo por ID
/vehicles/:id	DELETE	id: ID del vehículo (URL Path)	Eliminar un vehículo por ID
/auth/login	POST	body: { "username": "user", "password": "pass" }	Iniciar sesión
/auth/logout	POST	Ninguno	Cerrar sesión
/auth/register	POST	body: { "username": "user", "password": "pass", "email": "email@example.com" }	Registrar un nuevo usuario

9. Manual de despliegue

El despliegue de la aplicación se ha realizado con el gestor de contenedores Docker con los siguientes archivos y dependencias:

1. Docker Engine

Indispensable contar con una instalación de docker Engine en su sistema ya sea Windows, Linux o Mac.

2. Archivos utilizados

a. Dockerfile tracer-frontend

Este archivo se ubica en la carpeta raíz de el frontend de tracer, y contiene toda la información necesaria para la creación de un contenedor docker con un despliegue correcto de la app.

```
# Usa una imagen de Node.js para construir la aplicación React
FROM node:20.12.2 as build

# Establece el directorio de trabajo
WORKDIR /app

# Copia el resto de los archivos del proyecto
COPY . .

# Instala las dependencias
RUN npm install

# Construye la aplicación React
RUN npm run build --prod

# Usa una imagen de Nginx para servir la aplicación React
FROM nginx:stable-alpine
LABEL authors="Javier Gámez Mendoza"

# Copia los archivos contruidos desde la fase de construcción
COPY --from=build /app/build /usr/share/nginx/html

# Copia la configuración de Nginx
```

```

COPY ./nginx.conf /etc/nginx/nginx.conf

#RUN nginx -t

# Exponer el puerto en el que Nginx servirá la aplicación
EXPOSE 80

# Comando para iniciar Nginx
CMD ["nginx", "-g", "daemon off;"]

```

b. Dockerfile tracer-backend

Este archivo se ubica en la raíz del proyecto backend y proporciona un entorno de ejecución java para el despliegue de la API. Crea un contenedor en el que construye un ejecutable en base al proyecto Spring.

```

FROM maven:3-openjdk-17 as builder

COPY src /usr/src/app/src
COPY pom.xml /usr/src/app

RUN mvn -f /usr/src/app/pom.xml clean package -DskipTests

FROM openjdk:17-alpine
LABEL authors="Javier Gámez Mendoza"

ENV BBDD_HOST="db"
ENV BBDD_NAME="tracer"
ENV APP_PORT=8080
ENV LOG_LEVEL="INFO"
ENV DLL_AUTO="update"

VOLUME /tmp
VOLUME /mediafiles
EXPOSE 8080
RUN mkdir -p /mediafiles
COPY --from=builder /usr/src/app/target/tracer-0.0.1-SNAPSHOT.jar
/app/app.jar
#ADD ./target/api_store-0.0.1-SNAPSHOT.jar app.jar
WORKDIR /app

```

```
ENTRYPOINT ["java", "-jar", "/app/app.jar"]
```

c. Configuración Nginx.conf

Es el archivo de configuración del servidor web que ejecutará el frontend.

Se ubica dentro del directorio raíz del frontend, junto al Dockerfile.

```
# Global settings
worker_processes 1;

events {
    worker_connections 1024;
}

http {
    include mime.types;
    default_type application/octet-stream;

    # Angular app server block
    server {
        listen 80;
        server_name localhost;

        location / {
            root /usr/share/nginx/html;
            index index.html index.htm;
            try_files $uri $uri/ /index.html;
        }

        # Additional configurations, if any
    }
}
```

d. Docker-compose

Por último, el archivo Docker-compose.yml será el encargado de poner en funcionamiento el resto de contenedores y crear el stack de docker. Debe estar colocado en el directorio directamente superior a tracer-frontend y tracer-backend.

```

version: '3.1'

services:
  db:
    image: mysql:latest
    container_name: bbdd-tracer
    restart: unless-stopped
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: {your_password}
      MYSQL_DATABASE: tracer

  tracer_api:
    image: tracer_api:latest
    build:
      context: ./tracer-backend
      dockerfile: ./Dockerfile
    container_name: api-tracer
    restart: unless-stopped
    ports:
      - 8080:8080
    environment:
      BBDD_HOST: db
      BBDD_NAME: tracer
      DATABASE_USERNAME: root
      DATABASE_PASSWORD: {your_password}
      APP_PORT: 8080
      LOG_LEVEL: DEBUG
      DLL_AUTO: create
    volumes:
      - ./src/main/resources:/app/resources
    depends_on:
      - db
    links:
      - db

  tracer_react:
    build:
      context: ./tracer-frontend
      dockerfile: ./Dockerfile
    container_name: react-tracer
    restart: unless-stopped

```

```
ports:
  - 80:80
depends_on:
  - tracer_api
links:
  - tracer_api
```

e. Ejecución del despliegue

Una vez creados todos los archivos y colocados en la siguiente estructura

```
tracer:
  docker-compose.yml
  tracer-backend:
    Dockerfile (backend)
  tracer-frontend:
    Dockerfile (frontend)
    nginx.conf
```

Y teniendo el servicio de docker en ejecución, debemos ubicarnos en el directorio **tracer** y ejecutar el siguiente comando:

docker-compose up -d

Es necesaria una conexión a internet para ello, ya que las dependencias serán descargadas e instaladas en el momento de la creación del contenedor.

10. Conclusión

Repositorio Github (rama develop):

<https://github.com/JavierGamezMendoza/Tracer-TFG-JGM/tree/develop>

El mayor esfuerzo realizado en la aplicación es sin duda la organización.

Al componer “un ejército de un solo hombre” te ves en la tesitura de cargar con el trabajo que normalmente abarcaría un equipo de desarrollo completo, y por tanto, con todos los problemas que esto acarrea en las fases de desarrollo, diseño y despliegue.

La aplicación sirve como ejemplo de cómo el paso del tiempo y las nuevas tecnologías pueden aprovecharse para revivir viejas glorias y aportar nuevas funciones.

Tracer se plantea como una posible aplicación móvil, explorando la posibilidad de su portabilidad a **React Native**

Además de todo ello, se busca en ella un ambiente de **Red social** por lo que las posibles mejoras como chat incorporado, sistema de feed algorítmico, sistema de ventas embebido o incluso monetización por medio de publicidad o suscripciones queda a la vista.

Un especial agradecimiento al equipo docente del IES Alixar, y a Eviden por la formación impartida como parte de las practicas FCT.