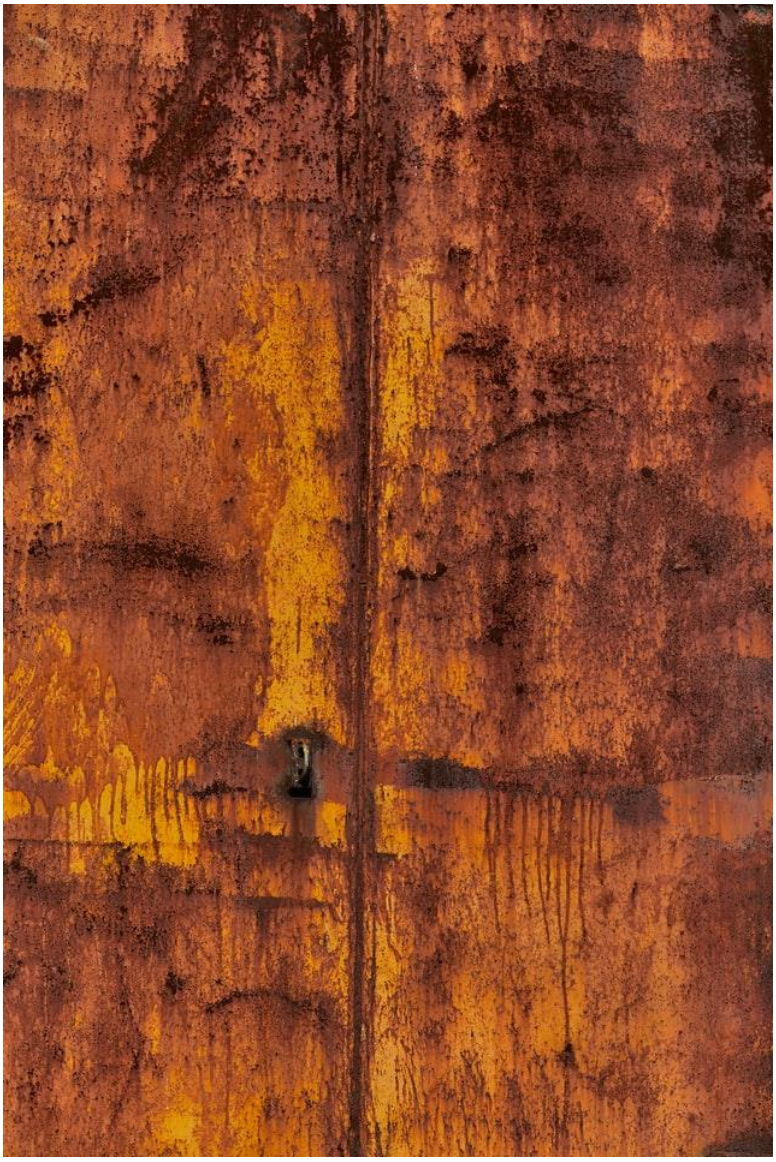


Proyecto Final de Ciclo

Raúl Callado Montero | Desarrollo de aplicaciones web | 16/05/2022

Extracto

Este documento describe la implementación de una nueva funcionalidad en un sitio web existente. Pasando por la motivación inicial, la estimación de tiempos, el análisis comercial, la descripción de los componentes y las conclusiones. También cubre opiniones personales sobre las tecnologías aplicadas y cómo están cobrando importancia en la actualidad.

Abstract

This document describe a way to implement a new feature in an existing website. It goes through the initial motivation, time estimation, business analysis, components descriptions and conclusions. It also covers personal opinions about the technologies applied and how then are getting importance in nowdays.

Tabla de contenidos

RESUMEN DEL PROYECTO

El proyecto se inicia con la hipotética situación en la que nuestro cliente acude a nosotros para actualizar la página web de su negocio. Este nos explica que regenta un restaurante llamado *sensacion* y que ya dispone de un sitio web informativo sobre su restaurante. Debido a la gran cantidad de usuarios que visitan su sitio, ha decidido implementar un formulario de reserva de mesa.

Tras un estudio previo se ha determinado que para la inclusión de esta nueva mejora se necesitarán 40 horas para el desarrollo del proyecto, así como otras 40 horas para un mantenimiento inicial del sitio, hasta que se haya depurado suficientemente bien en producción. Este estudio arroja la siguiente distribución del tiempo en secciones del proyecto:

Nombre	Tiempo
Crear entorno desarrollo (git, github, directorios)	5 Horas
Creación servidor (postgresql, servidor web, api)	20 Horas
Integración y desarrollo cliente (Angular)	10 Horas
Contenerización (docker, kubernetes)	5 Horas
Lanzamiento y mantenimiento inicial	40 Horas

* Tabla 1

Con estos datos el departamento financiero generará un presupuesto previo al inicio del proyecto, que daremos por aceptado y que todas las cuestiones legales serán resueltas por el departamento o persona apropiado.

Idea inicial

El cliente describe que actualmente los usuarios de la web, si quieren reservar deben ponerse en contacto directamente con el restaurante, y que

muchos de los usuarios desisten de reservar por tener que usar medios externos a la web. En principio, quiere integrar las reservas desde la web solo para el servicio de comidas (no cenas) y sin selección de hora. El usuario solo tendría que poner una fecha ver si hay disponibilidad y si es así, escribir su nombre y enviar. No quiere que los usuarios tengan que registrarse para poder reservar y en caso de querer hacer alguna modificación en la reserva, el usuario llamaría al restaurante.

Este proyecto solo se encarga de mostrar la disponibilidad dada una fecha y permitir crear una reserva a partir de dicha disponibilidad. Otro equipo se encargaría de desarrollar un administrador que permita al cliente gestionar las reservas y las mesas.

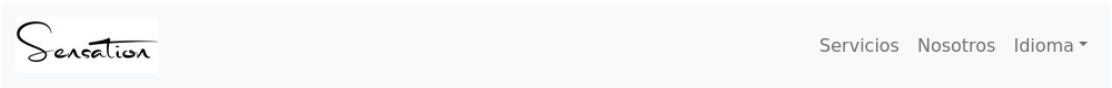
Estado previo

El cliente nos entrega el código fuente de la página actual para que sirva de inicio para el proyecto. Es un proyecto web sencillo de páginas html que usa bootstrap 5 para los estilos. También integra un fichero *Dockerfile* ya que actualmente la web está desplegada como un contenedor sobre un servidor de kubernetes.

Objetivos requeridos

Tras el estudio y algunas reuniones con el cliente hemos determinado que en el menú de navegación de la web, se añadirá un nuevo botón *reserva*, que al ser pulsado iniciará Angular para mostrar la disponibilidad y ayudará al usuario a reservar. Para que esto funcione también se ha determinado que necesitaremos una base de datos con las mesas y las reservas. Para comunicar la base de datos con el cliente se creará una api en el lado del servidor.

La imagen muestra estado actual del menú de navegación de la web:



Una vez todo esto funcione se generarán las imágenes de docker necesarias para el posterior despliegue de la aplicación en un servidor de kubernetes que el cliente ya tiene contratado.

Se entiende que la siguiente lista son los objetivos requeridos por el proyecto.

TODO:

- [ ] Integrar botón reserva en web actual.
- [ ] Crear CRUD para mesa y reserva.
- [ ] Crear endpoint consulta disponibilidad.
- [ ] Integrar Angular en la web.
- [ ] Usar Angular para interactuar con el servidor.
- [ ] Contenerizar la aplicación.

JUSTIFICACIÓN Y OBJETIVOS DEL PROYECTO

Tras el estudio de los requerimientos del cliente se ha llegado a la decisión de dividir el proyecto en las siguientes secciones:

1. Front-end
2. Back-end
3. Despliegue
4. Actualización

Cada uno de estas secciones se describe más adelante.

Front-end

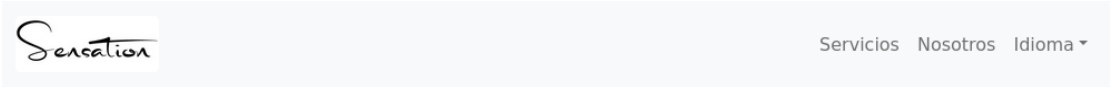
He decidido usar Angular como framework de javascript. Lo implementaré solo en una ruta concreta de la aplicación (/reserva), ya que el resto de la web seguirán siendo html y css. Con esto evitamos que el proyecto crezca en complejidad y solo cargue la aplicación en un punto determinado en el que usuario de la web está interesado en consultar disponibilidad para realizar una reserva.

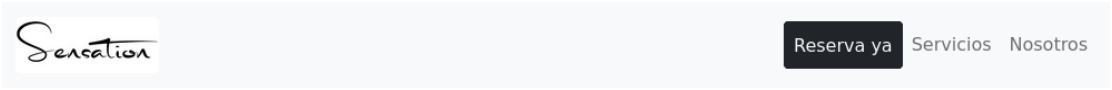
A nivel de desarrollo Angular construye una SPA que se alojará en el directorio *reserva*. Mientras que el directorio *Angular* será ignorado en el despliegue y por lo tanto solo se usa para el desarrollo y construcción de la SPA.

Actualización

En general la web se verá igual que hasta ahora. Tan solo se observará un nuevo botón en la barra de navegación *reserva*. Este será el punto de entrada para el nuevo gestor de reservas que pretendemos implementar.

En las imágenes se muestra la diferencia entre el menú de navegación inicial y el final:

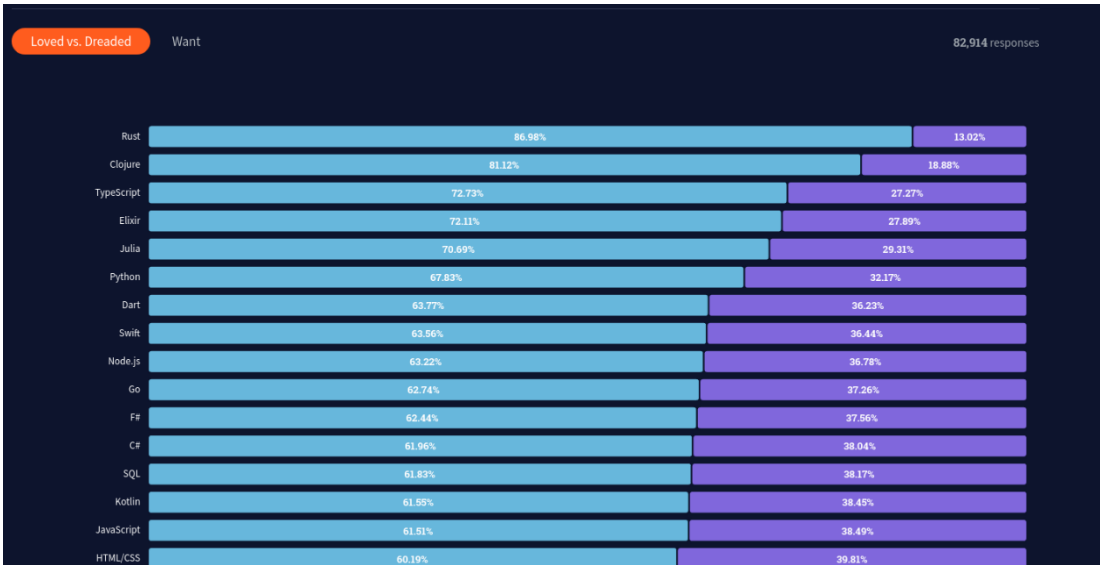




Back-end

Para poder almacenar la información necesitamos una base de datos y para comunicarnos con ella necesitamos crear una servidor que entregue la información a la aplicación cliente. Esta aplicación, que se conoce como servidor api, se escribirá en Rust y he decidido usar rocket como framework.

Rust es un lenguaje de programación moderno, confiable y eficiente. Permite crear aplicaciones muy eficientes gracias a que no usa interprete ni garbage collector, en su lugar implementa ownership para garantizar un acceso seguro a la memoria siempre. Con todo esto y muchas otras ventajas creo que es uno de los lenguajes más potentes en la actualidad. Además de ser el lenguaje más amado desde 2015, en la conocida encuesta de "stackoverflow survey" [Enlace].



Rocket es un web framework desarrollado en Rust que facilita el desarrollo de aplicaciones web a nivel de servidor. Está en su versión 0.5 por lo que puede introducir cambios drásticos en futuras versiones, sin embargo es muy consistente y sencillo. No es necesario tener un conocimiento profundo del lenguaje para crear servidores web sencillos y funcionales.

Despliegue

La tecnología más actual y potente para el despliegue de aplicaciones son los contenedores. Con docker podemos crear imágenes de nuestra aplicación que se ejecutaran en servidores en forma de contenedor. Para orquestar los contenedores su utiliza Kubernetes, el cual permite proveer a la aplicación de los requisitos de hardware que necesite (cpu, ram, memoria).

Esto hace que esta tecnología se esté implantando rápidamente en la actualidad como reza en el siguiente artículo [Enlace], el 75% de las grandes empresas usan kubernetes y por consiguiente la contenerización de aplicaciones.

DESARROLLO DEL PROYECTO

Análisis del mercado

Para el estudio previo se analizó el mercado, consultando algunas páginas que ofrecen la misma funcionalidad en la actualidad.

Ruta del veleta [Enlace].

Tras estudiar un poco esta página se determinó que la experiencia de usuario no es buena, a pesar de que se pueden realizar reservas en el restaurantes, resulta difícil realizar la transacción por lo complejo de su formulario y en ningún caso muestra disponibilidad.

Bodega el Capricho [Enlace]

Esta otra opción es mucho más atractiva visualmente y permite en pocos pasos realizar la reserva de una mesa en el restaurante. Pero finalmente el cliente se encuentra con un complejo formulario por rellenar antes de terminar el proceso.

Conclusión

Tras observar estas y otras páginas se decidió optar por un modelo mucho más sencillo en el que el usuario no tuviera que rellenar un gran formulario con muchos datos. Al igual que al reservar por teléfono, normalmente se pide hora, número de comensales y un nombre para la reserva, en la web no se mostrarán complejos formularios, solo lo mínimo.

Este enfoque expone la posibilidad de que se reciban reservas falsas o que malintencionadamente alguien bloqueé el sistema. Por esta razón se ha decidido con el cliente, mostrar solo una parte de la disponibilidad en la web para evitar que esta vulnerabilidad suponga un desastre financiero.

Metodologías utilizadas

Al constar de una sola persona el grupo de trabajo, se a optado por implementar la metodología en cascada. Desde un principio se han separado

las diferentes partes del proyecto y se han definido sus funcionalidades. Con esta metodología cada etapa finaliza con cada una de las partes o funcionalidades. Se puede decir que el proyecto ha seguido el siguiente orden:

1. Entorno de desarrollo
2. Creación apis
3. Contenedor api
4. Proyecto angular
5. Contenedor web
6. Despliegue.

Si al desarrollar una etapa se ha detectado un error en la anterior se a reparado indistintamente ya que no existe un sistema de generación de tickets (issue tracking system) para su posterior reparación.

Creo que para proyectos unipersonales es una buena opción y cualquier alternativa ágil supondría un incremento en complejidad que debería estudiarse.

Descripción de los componentes

Para este proyecto he creado tres repositorios diferentes, front, server, root. Cada uno de ellos tiene una función específica dentro del conjunto pero se puede trabajar en ellos de forma independiente. Para esto he usado *git submodule*, esta funcionalidad permite especificar dentro de un repositorio si algún directorio hace referencia ha otro repositorio diferente. De este modo el módulo *root* además de su propio contenido también hace referencia a un commit concreto de cada uno los otros repositorios. Este enfoque permite trabajar de forma independiente en los módulos y según avanza el proyecto actualizar el commit al que hace referencia la raíz para que así el repositorio principal quede actualizado.

Es especialmente útil para trabajar con diferentes equipos sobre el mismo proyecto, pero he pensado que podría ser muy buena idea implementarlo para mantener siempre la independencia entre los diferentes módulos sin que el trabajo en uno afecte a otro.

De este modo se puede determinar que el cliente recibirá una aplicación completa y funcional divida en tres componentes principalmente. Cada uno de ellos está acompañado de un *README* con anotaciones sobre su compilación y puesta en funcionamiento. Los componentes son:

1. front: Contiene aplicación web con la que el usuario interactúa directamente.
2. server: Contiene servidor api para el intercambio de información entre el front y la base de datos.
3. root: Contiene los dos módulos anteriores, además de los manifiestos para su despliegue.

Front

Aplicación web con partes estáticas y otras dinámicas. Algunas partes de esta aplicación son ficheros html y css, los cuales sirven para dar información al usuario sobre la empresa. Esta información está compuesta por diferentes apartados describiendo los servicios que se prestan, métodos de contacto y otra información importante para el usuario.

También incluye otra parte dinámica, la cual se ha construido usando el conocido framework de javascript Angular. Esta parte sirve para mostrar la disponibilidad y permite realizar reservas en el establecimiento. Una vez que el usuario entra en la ruta */reserva/* la aplicación se conecta al servidor para obtener la disponibilidad de mesas y con los datos obtenidos, procede a mostrarlos en pantalla.

El usuario tiene la posibilidad de consultar otra fecha diferente y una vez elegida la mesa deseada puede pulsar un botón para reservar. Esta acción le mostrará un formulario con algunos datos requeridos, nombre y número de comensales. Una vez completado y enviado, si todo ha ido bien se le confirmará que ha sido aceptada la reserva y se actualizará la disponibilidad para que el usuario pueda comprobar que la mesa ya no está disponible.

Server

Este componente sirve de intermediario entre la base de datos y la aplicación web. Su función es gestionar las peticiones que llegan desde el front lo más eficientemente posible, consultando la base de datos y devolviendo la información correcta.

Para este cometido he decidido usar el no tan conocido framework de Rust, Rocket. Se caracteriza por su seguridad, estabilidad y velocidad. A pesar de lo complejo que pueda parecer, una vez que conocer un poco el lenguaje y lees la documentación resulta muy sencillo para la creación servidores web.

Ya que este módulo se encarga de comunicarse con la base de datos, también integra las migraciones, que permiten al servidor crear las tablas y las relaciones necesarias en la base de datos previamente creada. Se ha decidido usar Postgresql como gestor de base de datos por sus muchas funcionalidades pero sobre todo por la gran velocidad de respuesta que tiene.

Para intermediar entre el servidor web, el cual recibe las peticiones y devuelve los datos, y la base de datos he usado _diesel_ que funciona como orm en rust, facilitando escribir las peticiones a la base de datos en este lenguaje. Sinceramente esta ha sido quizá la parte más difícil, por la complejidad de esta librería, la cual además depende de una librería de C *libpq*, por lo que compilar para arquitecturas arm64 y sistemas operativos tipo *musl* resulta complejo.

Tras la compilación de este módulo se obtiene un binario que se puede ejecutar directamente en el servidor, siempre y cuando exista la base de datos y tenga los permisos adecuados.

root

El componente raíz del proyecto tiene dos funcionalidades. Una es contener la documentación relacionada con el proyecto, así como describir el proceso de despliegue de la aplicación. Y por otro lado sirve para mantener las referencias a los commit concretos de cada submodulo. Esto quiere decir, que al haber implementado git submodule, cada vez que uno de los submodulos se actualiza no afecta directamente al proyecto, sino que desde la raíz deberá actualizarse la referencia de cada submodulo directamente. Esto permite actualizar, testear la aplicación y se algo no funciona volver a apuntar a un commit anterior hasta que se repare el error y se pueda volver a actualizar testear y finalmente desplegar.

Tanto el módulo front como server están pensados para ser incrustados en una imagen de docker y ser consumidos como contenedores en un servidor de kubernetes. Por esta razón en el módulo principal se acompañan los manifiestos necesarios para que todo se despliegue correctamente en la nube.

La base de datos se contrataría a parte y en estos manifiestos se pegarían las credenciales de conexión. También se deberían de ejecutar las migraciones previamente al despliegue, ya que son estas (las migraciones), las que generan las tablas y relaciones que la aplicación necesita, pero este proceso se describe con más detalle en el *README* del módulo server.

También se ha agregado el borrador del proyecto en este mismo repositorio para la entrega final de este, pero el cliente recibiría un manual de usuario y un manual técnico en su lugar.

Resultados obtenidos

Los objetivos establecidos para el proyecto se ha cumplido en su totalidad y en los plazos impuestos, 40 horas de desarrollo.

TODO:

- [x] Integrar botón reserva en web actual.
- [x] Crear CRUD para mesa y reserva.
- [x] Crear endpoint consulta disponibilidad.
- [x] Integrar Angular en la web.
- [x] Usar Angular para interactuar con el servidor.
- [x] Contenerizar la aplicación.

Como se aprecia en la lista, todos los _todo_ se han completado y la aplicación actualmente se está ejecutando correctamente en un laboratorio domestico online, para que el cliente pueda testearla antes de la entrega final. Una vez que el cliente diese su visto bueno, se procesaría el pago final de la aplicación y se le entregaría un fichero con todo el código fuente, documentación así como los derechos legales sobre lo entregado. Como parte del contrato también se desplegaría la aplicación y se mantendría durante los primeros meses haciendo revisiones y comprobando la telemetría que ofrecen los servidores kubernetes. Al concluir este periodo se le presentaría al cliente un dosier de posibles mejoras y una propuesta de mantenimiento a largo plazo.

CONCLUSIONES

Con este proyecto no solo he alcanzado las metas propuestas por el cliente, sino que he mejorados mis conocimientos tanto a nivel de programación, como a nivel de investigación y documentación de nuevas tecnologías. Algunas de estas las describo a continuación.

Angular

He aprendido a implementar Angular en un proyecto ya existente y configurarlo para que se ejecute solo en una ruta especifica, lo cual me parece muy interesante para empresas que ya tienen una página web con información de su empresa pero quieren ofrecer algo más para sus usuarios.

Rust

En el proceso de aprendizaje de Rust también he adquirido mejores habilidades para crear código de calidad y confiable.

También he aprendido a usar un nuevo framework para crear servidore en este lenguaje. Me ha parecido muy sencillo de usar una vez que entiendes funciona y aunque está es una estapa de desarrollo muy temprana, he encontrado respuesta para todas las dudas que me han ido surgiendo durante el desarrollo. Seguiré trabajando con este framework ya que me parece uno de los mejores que he probado a pesar de no tener tantos añadidos como otros framework más conocidos (symphony, laravel, adonis, ect...).

Docker

Desde que empecé a usar docker me ha parecido una herramienta genial para el desarrollo de software y ahora que he aprendido también a desplegar la imágenes en producción, me parece aún más potente y versatil. Estoy seguro que su uso seguirá creciendo y recibiendo más funcionalidades nuevas.

A pesar de las compleja estructura que he creado para este proyecto con gracias a la documentación y foros en internet he conseguido completar los objetivos de generar una aplicación que economice los recursos requeridos al máximo.

Kubernetes

He aprendido a crear los manifiestos necesarios para desplegar aplicaciones sobre un servidor de kubernetes. No ha sido sencillo pero con la abundante documentación que se encuentra en la página oficial he reforzado mis conocimientos en esta tecnología y logrado el objetivo.

Esta tecnología se está implementando rápidamente entre las grandes compañias ya que permite la autoescalada de recursos según los requisitos de la aplicación en cada momento y pagar por uso. Por lo que haber aprendido al menos los conceptos básicos me será muy útil en mi carrera.

Demo

Tras finalizar el proyecto lo he desplegado en un laboratorio doméstico que tengo conectado a un domínio personal. La aplicación está funcionando correctamente y puede ser probada en el siguiente enlace:

https://sensacion.kennycallado.dev

Está sujeta a posibles fallos de conexión y otros ya que es un laboratorio doméstico y muchas veces pruebo cosas nuevas que pueden hacer que otras no funcionen, pero en general está permanentemente accesible y funcionando correctamente.

LÍNEAS FUTURAS

Se proponen una serie de mejoras a futuro, tanto para el apartado de visual y el diseño de la aplicación, como para el servidor web. Con estas mejoras se brindarían más opciones a los usuarios y la aplicación sería más estable.

- Mejorar estado previo de la aplicación (corrección errores).
- Mejorar estilos en front-end.
- Incluir posibilidad de reserva en diferentes horarios.
- Crear tests para prevenir código con errores.

La última de estas mejoras, sería la más importante de implementar cuanto antes, ya que garantizaría que la aplicación funciona perfectamente incluso tras implementar futuras mejoras.

BIBLIOGRAFÍA

Git. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Git

Github. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/GitHub

Postgres. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/PostgreSQL

Servidor web. s.f. *Wikipedia.* Recuperado de: https://es.wikipedia.org/wiki/Servidor_web

Api. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

Html. s.f. *Wikipedia.* Recuperado de: https://es.wikipedia.org/wiki/HTML

Css. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/CSS

Bootstrap. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Bootstrap_(framework)

Docker. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Docker_(software)

Dockerfile. s.f. *Nick Janetakis blog*. Recuperado de: https://nickjanetakis.com/blog/differences-between-a-dockerfile-docker-image-and-docker-container#it-all-starts-with-a-dockerfile

Dockerhub. s.f. *Página oficial*. Recuperado de: https://hub.docker.com

Kubernetes. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Kubernetes

Despliegue de software. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Despliegue_de_software

CRUD. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/CRUD

Endpoint. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Punto_final_de_comunicaci%C3%B3n

Contenerización. s.f. *Rackspace Technology blog*. Recuperado de: https://www.rackspace.com/es/blog/containers-101

Front-end y Back-end. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Front_end_y_back_end

Framework. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Framework_para_aplicaciones_web

Javascript. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/JavaScript

Rutas. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme

SPA. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Single-page_application

Rust. s.f. *Wikipedia*. Recuperado de https://es.wikipedia.org/wiki/Rust_(lenguaje_de_programaci%C3%B3n)

Rocket. s.f. *Web oficial*. Recuperado de: https://rocket.rs/

Interprete. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Int%C3%A9rprete_(inform%C3%A1tica)

Garbage collector. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Recolector_de_basura

Ownership. s.f. *Rust book*. Recuperado de: https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html

Metodología cascada. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software

Issue tracking system. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Sistema_de_seguimiento_de_incidentes

Metodología agile. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software

Commit. s.f. *Wikipedia*. Recuperado de: https://en.wikipedia.org/wiki/Commit_(version_control)

README. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/README

Página web estática. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/P%C3%A1gina_web_est%C3%A1tica

Página web dinámica. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/P%C3%A1gina_web_din%C3%A1mica

Angular. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Angular_(framework)

Diesel. s.f. *Web oficial*. Recuperado de: https://diesel.rs/

Orm. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Asignaci%C3%B3n_objeto-relacional

Librería C. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Glibc

Librería musl. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Musl

Arquitectura arm. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Arquitectura_ARM

Online. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/En_l%C3%ADnea

(Telemetría) monitorización. s.f. *Wikipedia*. Recuperado de: https://es.wikipedia.org/wiki/Monitorizaci%C3%B3n_de_Servidores_de_Internet