PROYECTO

DRCARS
LOGOTIPO

CICLO FORMATIVO DE GRADO SUPERIOR

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

AUTORES
RICARDO ROMERO LOBATO Y MIGUEL BLANCO JIMENEZ

TUTOR DEL PROYECTO
VICTOR GARCÍA DELGADO

Licencia

Índice de contenido

1 INTRODUCCIÓN 5
2 ALCANCE DEL PROYECTO 6
3 ESTUDIO DE VIABILIDAD 7
3.1 Estado actual del sistema 7
3.2 Requisitos del cliente 7
3.3 Posibles soluciones 7
3.4 Solución elegida 7
3.5 Planificación temporal de las tareas del proyecto [nuevo proyecto] 7
3.6 Planificación de los recursos a utilizar 7
4 ANÁLISIS 8
4.1 Requisitos funcionales 8
4.2 Requisitos no funcionales 8
5 DISEÑO 9
5.1 Estructura de la aplicación 9
5.2 Componentes del sistema 9
5.3 Arquitectura de la red 9
5.4 Herramientas 9
6 IMPLEMENTACIÓN 10
6.1 Entorno de implementación 10
6.2 Tablas creadas 10
6.3 Carga de datos 10
6.4 Ficheros de configuración actualizados 10
6.5 Configuraciones realizadas en el sistema 10
6.6 Implentaciones de código realizadas 10
7 PRUEBAS 11
7.1 Casos de pruebas 11
7.2 Plan de ejecución…… 11
8 EXPLOTACIÓN 13
8.1 Planificación 13
8.2 Preparación para el cambio 13
8.3 Plan de formación (Manual) **13**
9 DEFINICIÓN DE PROCEDIMIENTOS DE CONTROL Y EVALUACIÓN 14
11 FUENTES (Páginas de referencia) 16
12 ANEXOS 17
12 CONCLUSIONES

1 INTRODUCCIÓN

Este documento recoge el trabajo realizado para el **módulo de Proyecto** del CFGS en **Desarrollo de Aplicaciones Multiplataforma**. Este módulo profesional complementa la formación establecida para el resto de los módulos profesionales que integran el título en las funciones de **análisis** del contexto, **diseño** del proyecto y organización de la **ejecución**.

En este documento se mostrará todo el proceso realizado para el desarrollo de un software de gestión y venta de vehículos de importación para una empresa llamada DRCars.

2 ALCANCE DEL PROYECTO

El presente proyecto tiene como **propósito** fundamental desarrollar una solución integral para la gestión de citas y el inventario de vehículos en un servicio de importación, proporcionando dos interfaces diferenciadas y optimizadas para las necesidades de clientes y personal interno.

Se **pretende conseguir** una mejora sustancial en la eficiencia operativa del negocio, agilizando el proceso de solicitud de citas por parte de los clientes y centralizando la gestión de citas, vehículos y usuarios para el personal de la empresa. Esto se traducirá en una mejor experiencia para el cliente, una reducción de la carga administrativa y una mayor capacidad de organización y control para la empresa.

El **objetivo general** del proyecto es el desarrollo e implementación de un sistema multiplataforma, compuesto por:

- Una aplicación web accesible para clientes, que permita la solicitud y gestión autónoma de citas para la compra de vehículos.
- Una aplicación de escritorio para el personal de la empresa, que facilite la gestión integral de citas, la administración del catálogo de vehículos y la gestión de usuarios y empleados.
- Una API RESTful que actúe como intermediario entre las aplicaciones y la base de datos, garantizando una comunicación eficiente y segura.

- Una base de datos MySQL para el almacenamiento persistente de la información (citas, vehículos, usuarios, etc.).
- Una arquitectura basada en contenedores Docker para la API y la base de datos, que asegure la portabilidad y escalabilidad del sistema.
- El uso de ngrok para permitir el acceso externo a la API, superando las limitaciones de una IP dinámica.

El desarrollo de este proyecto se llevará a cabo en varias fases: estudio de viabilidad, análisis, diseño, implementación y pruebas, y explotación o ejecución. Las actividades/tareas/procedimientos de cada una de estas fases se detallarán en las siguientes secciones de este documento.

3 ESTUDIO DE VIABILIDAD

Tras realizarse un exhaustivo estudio de viabilidad se ha concluido que el proyecto puede ser realizado sin costes monetarios adicionales. Además, se cuenta con dos desarrolladores en el proyecto por lo que se ha determinado que es posible la entrega del software en el plazo establecido por el cliente.

Para poder llegar a la conclusión del actual estudio de viabilidad se ha tenido que realizar una valoración de los recursos actuales que dispone el cliente en la actualidad y presentar un estudio de posibles soluciones siendo una de esta elegida por el cliente.

3.1 Estado actual del sistema

El sistema actual del cliente consta de múltiples hojas de cálculo (Excel) y cuadernos tradicionales para la gestión interna de su empresa y una página web de mala calidad para el cliente y sin posibilidad de realizar ventas online sobre un catálogo mal estructurado.

Tampoco se cuenta con una seguridad sobre los datos, dado que si alguna hoja de cálculo y/o cuaderno se pierde no existiría redundancia de datos y se podría estar violando la ley de protección de datos (si dicho caso se produjese fuera de las instalaciones).

3.2 Requisitos del cliente

Tras ponernos en contacto con el cliente, se han establecido sus necesidades. La primera necesidad es un software que permita gestionar de forma eficiente, centralizada y segura la lógica del negocio, gestionando así las cuentas del personal, el stock de los vehículos (pudiendo cambiar entre distintos estados, precios, características y más), las citas que soliciten los clientes y las ventas realizadas.
Por otra parte, el cliente tiene otra necesidad, una remodelación de su página web, realizando una remaquetación completa y desarrollando un UX intuitivo y sencillo. El usuario final debe poder visualizar el catálogo de forma eficiente y rápida, proporcionándole los detalles esenciales como el precio, los kilómetros, el año y el nombre y modelo desde el propio catálogo, si el usuario lo considera debe poder acceder a los detalles específicos de un vehículo y si decide comprarlo, podrá solicitar una cita desde la página de detalles del vehículo seleccionado, rellenando un pequeño cuestionario con algunos datos, esta solicitud se mostrara directamente en el software empresarial nombrado anteriormente. El usuario también debe poder solicitar citas relacionadas con otros motivos como solicitar información.
El software debe asegurar una persistencia de datos seguros con validaciones y reglas de seguridad conectando todas las interfaces disponibles la lógica de negocio junto a el servicio de almacenamiento de datos.

3.3 Posibles soluciones

Las soluciones posibles comentadas con el cliente fueron las siguientes:

- Utilizar un software de gestión empresarial ya existente.
- Desarrollar un software completo utilizando una base de datos NoSQL
- Desarrollar un software completo utilizando una base de datos SQL

3.4 Solución elegida

Tras pensar en la mejor opción para el cliente se ha concluido que la opción correcta es desarrollar un software completo utilizando una base de datos SQL.
El motivo principal de esta decisión es que dado el alcance del negocio y el bajo presupuesto era inviable contratar los servicios de cualquier software empresarial.

El proyecto que se desarrollara constara de 4 partes que formaran un software empresarial completo, persistente y seguro, las distintas partes son las siguientes:

- Lo primero será desarrollar el modelo de datos, ya que es la base del software y el que se encargara de almacenar cualquier dato.
- Lo segundo será desarrollar una API REST que se encargará de procesar los datos entrantes y almacenarlos de forma segura.
- Después continuaremos con la página web con un UX intuitivo y sencillo para que el usuario no se pierda en funcionalidades que no sean necesarias.
- Por último, se desarrollará la aplicación de escritorio desde la que los trabajadores podrán gestionar vehículos, usuarios, citas, ventas y estadísticas.

3.5 Planificación temporal de las tareas del proyecto [nuevo proyecto]

Planificación de los requisitos mínimos (MVP) → 3 días

Arquitectura del software y estructura → 7 días

Desarrollo de la BBDD → 4 días + futuras modificaciones

Desarrollo de la API REST, Aplicación de escritorio, web → 67 días en paralelo

Implementación en los servidores → 7 días

Pruebas → 1 día

Despliegue → 2 días

3.6 Planificación de los recursos a utilizar

En este proyecto no será necesaria la contratación de ningún empleado, por ende, tampoco hará falta formar personal. No obstante, se necesitará invertir en herramientas de desarrollo de despliegue web y en un servidor dedicado.

Se necesitará invertir también en un domino para que la pagina pueda ser accesible desde la web.

4 ANÁLISIS

En esta fase se establecen los requisitos que el sistema debe cumplir.

4.1 Requisitos funcionales

1. Autenticación y autorización:
    - RF 1.1: Los usuarios internos (administradores y empleados) deben iniciar sesión en la aplicación de escritorio.
    - RF 1.2: Los clientes deben poder registrarse y autenticarse en la página web.
    - RF 1.3: La API REST debe validar credenciales
2. Gestión de usuarios internos:
    - RF 2.1: Crear, modificar, eliminar y listar usuarios internos desde la aplicación de escritorio.
    - RF 2.2: Asignar roles (ADMIN, USER) a cada usuario interno.
3. Gestión de clientes:
    - RF 3.1: Registrar nuevos clientes desde la aplicación de escritorio.
    - RF 3.2: Consultar datos de clientes existentes.
4. Gestión de vehículos:
    - RF 4.1: Registrar nuevos vehículos importados, con datos: marca, modelo, año, kilometraje, matrícula, chasis, precio de compra, estado, proveedor, tipo de combustible y transmisión.
    - RF 4.2: Actualizar datos de vehículos existentes.
    - RF 4.3: Cambiar estado de vehículos (STOCK → GARAJE → VENTA → VENDIDO).

- RF 4.4: Consultar catálogo de vehículos en la página web con filtros por marca, modelo, año y precio.

5. Gestión de reservas:
   - RF 5.1: Cliente puede realizar una cita indicando el id de vehículo, fecha y hora, motivo y descripción.
   - RF 5.2: Aplicación de escritorio muestra lista de reservas pendientes.
   - RF 5.3: Empleado puede confirmar o rechazar reservas.
6. Gestión de ventas:
   - RF 6.1: Actualizar estado del vehículo a VENDIDO una vez realizada la venta.
7. Gestión de favoritos (web):
   - RF 7.1: Cliente autenticado puede marcar vehículos como favoritos.
   - RF 7.2: Consultar lista de favoritos del cliente.
8. Reportes y dashboards:
   - RF 8.1: Aplicación de escritorio muestra dashboard con estadísticas de ventas y catálogo.

4.2 Requisitos no funcionales

1. Rendimiento:
   - RNF 1.1: La API REST debe atender hasta 50 peticiones concurrentes sin degradar la respuesta por debajo de 200 ms.
   - RNF 1.2: La página web debe cargar el catálogo de vehículos en menos de 3 segundos con 10.000 registros en la base de datos.
2. Escalabilidad:
   - RNF 2.1: El sistema debe permitir añadir nuevos módulos (piezas) sin reestructurar la base de datos principal.
3. Seguridad:
   - RNF 3.1: Comunicaciones entre componentes deben realizarse sobre HTTPS.
   - RNF 3.3: Los datos sensibles (contraseñas) deben almacenarse cifrados.
4. Usabilidad:

- RNF 4.1: Interfaces web y de escritorio deben ser intuitivas, con diseño responsive en la web y layout adaptado a escritorio.

5. Mantenibilidad:
   - RNF 5.1: El código debe seguir estándares de estilo (C# para escritorio, Java para API, TypeScript/React para web).
   - RNF 5.2: Documentación (JavaDoc, XML Docs) para facilitar futuras modificaciones.
6. Fiabilidad:
   - RNF 6.1: La base de datos MySQL debe contar con backups diarios.
   - RNF 6.2: Logs de errores y auditoría disponibles en API y escritorio.

5 DISEÑO

En esta fase se describe cómo se van a materializar los requisitos establecidos.

5.1 Estructura de la aplicación

La estructura de la aplicación se basa en un modelo vista-controlador.
La estructura del controlador está compuesta de una base de datos MySQL y de una API REST que se encarga de manejar los datos y formatearlo de forma correcta para su posterior visualización o almacenamiento. Por otra parte, tenemos las vistas, que están compuestas de la página web y de la aplicación de escritorio, estas vistas se encargan de visualizar los datos y poder interactuar con ellos.

5.2 Componentes del sistema

1. **Base de datos (MySQL 5.7):**
   Una base de datos con un conjunto de 11 tablas para poder almacenar toda la información y datos del negocio, en estas tablas se pueden almacenar información de vehículos, usuarios, clientes, etc.

2. **API RESTful (Spring Boot):**
   Servidor local encargado de modelar los datos para enviarlos a las vistas o recibirlos y almacenarlos en las tablas correspondientes de la base de datos.

3. **Ngrok:**
   Encargado de permitir accesos a peticiones externas desde las vistas del servidor dedicado que contiene la API superando las limitaciones que contienen las IP dinámicas.

4. **Vistas:**
   Las vistas se componen de una página web y una aplicación de escritorio, estas encargan de recibir los datos y mostrarlos de forma intuitiva para que tanto clientes como empleados puedan consultarlos, modificarlos o eliminarlos.

5.3 Arquitectura de la red

La red contiene la siguiente configuración para cada componente:

- La base de datos está instalada en un servidor dedicado cuyo puerto ha sido habilitado para recibir y/o proporcionar los datos que requiera el software
- La API, al igual que la base de datos, está instalada en el servidor dedicado, conectado a ngrok, de esta forma se podrán realizar conexiones cifradas a la api desde IP públicas.
- La página web esta hosteada en un servicio llamado Vercel, este proporciona un servidor público y seguro con un dominio con el que se puede acceder, cuando realiza cualquier conexión cifra los datos sensibles y los envía en formato JSON al servidor, este procesa los datos que recibe y almacena en las tablas correspondientes los datos recibidos.

5.4 Herramientas

Lenguajes de Programación:

- **Backend (API):** Java.
- **Backend y Frontend (Aplicación de Escritorio):** C#.
- **Frontend (Aplicación Web):** TypeScript y React

Frameworks/Librerías:

- **API (Backend):** Spring Boot y Maven (Desarrollo de la API RESTful).
- **Escritorio (Frontend):** .NET Core con WPF/Windows Forms
- **Web (Frontend):** React.js, CSS.
- **Base de Datos:** MySQL (Gestión y almacenamiento de datos relacionales).
- **Contenerización:** Docker (Creación y gestión de contenedores de la API y la base de datos MySQL), y Docker Compose (Orquestación y despliegue de los servicios contenerizados).
- **Exposición de API Externa:** Ngrok (Establecer un túnel seguro que permita el acceso remoto a la API desde entornos con IP dinámica).
- **Plataforma de Despliegue Web:** Vercel (Despliegue y hosting de la aplicación web).
- **Entornos de Desarrollo Integrado (IDE):**
- **Aplicación de escritorio:** Visual Studio.
- **API:** Eclipse.
- **Aplicación web:** Vercel y Visual Studio Code
- **Control de Versiones:** Git (Gestión de cambios en el código fuente) y GitHub (Plataforma de alojamiento de repositorios).

Otras Herramientas:

- Postman (Pruebas y depuración de la API).
- MySQL Workbench (para la gestión y administración de la base de datos MySQL).
- Herramienta de ingeniería inversa de MySQL Workbench (Diagramas de arquitectura).

6 IMPLEMENTACIÓN

A partir del diseño establecido, en esta fase se lleva a cabo la construcción del sistema, incluyendo el despliegue y configuración de sus componentes principales.

6.1 Entorno de implementación

El sistema está alojado en un servidor Ubuntu ubicado en entorno doméstico, el cual contiene tanto la API como la base de datos. Para garantizar una gestión eficiente y portátil de los servicios, se emplea Docker, permitiendo la contenerización de la API y la base de datos, lo que facilita su despliegue y mantenimiento.

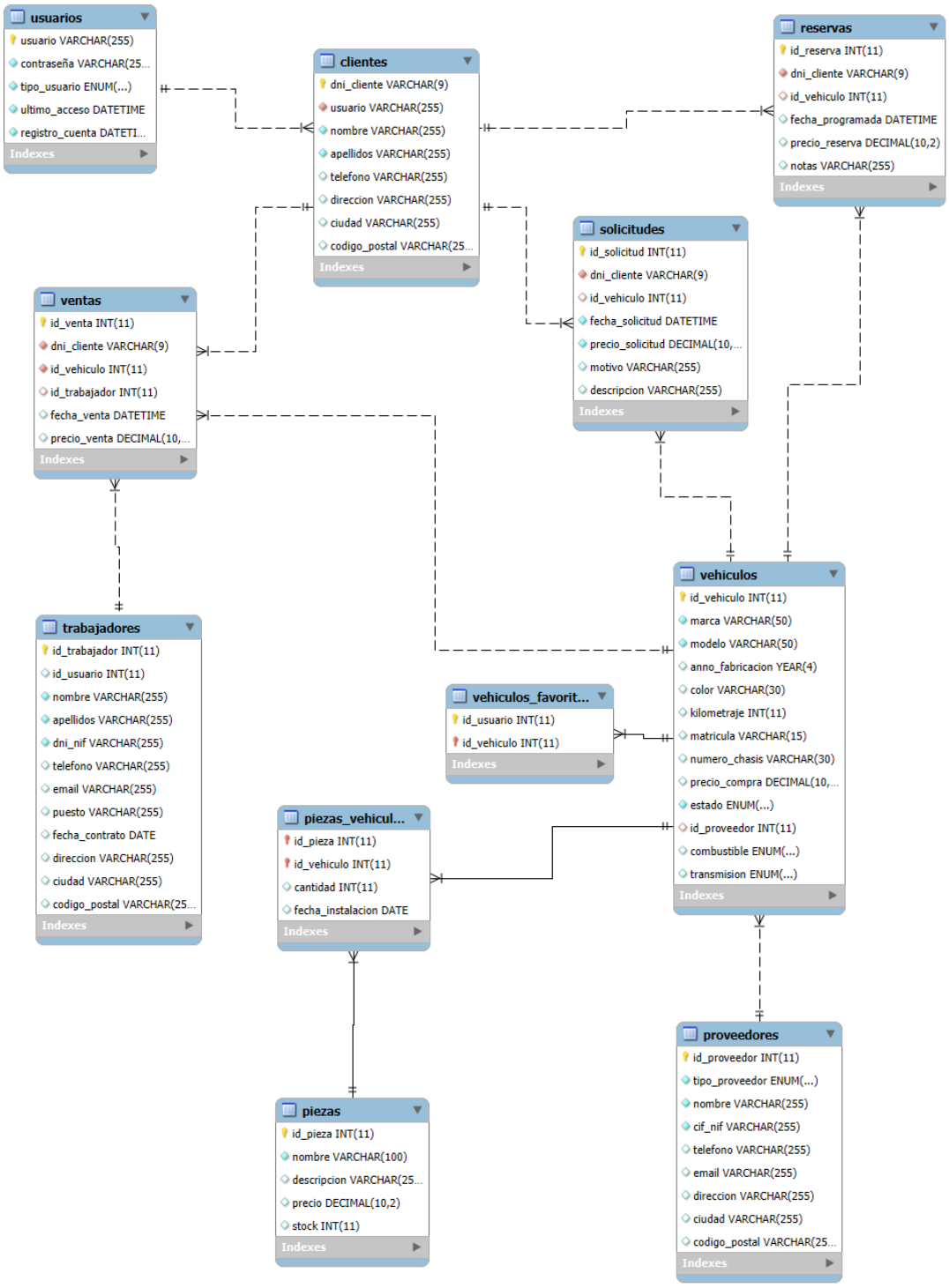
Para que la API pueda ser accedida externamente por los clientes (aplicación de escritorio y web), se utiliza Ngrok, una herramienta que proporciona túneles seguros y expone el servicio en internet. Esto permite que la aplicación de escritorio desarrollada en Visual Studio y la web implementada en Next.js interactúen con la base de datos a través de la API.

Se han desarrollado distintos módulos dentro de la API que permiten gestionar solicitudes de los clientes, realizar validaciones y optimizar el acceso a la base de datos. Se ha priorizado el uso de buenas prácticas de desarrollo, asegurando la correcta modularización y mantenimiento del código.

La aplicación de escritorio y la web consumen la API para modificar y leer datos de la base de datos, garantizando que los cambios sean reflejados de manera inmediata en el sistema.

Este diseño permite una implementación eficiente del sistema, asegurando escalabilidad y facilidad de administración.

6.2 Tablas creadas



Usuarios: Gestiona la autenticación y preferencias.

Clientes: Permite realizar reservas, compras y solicitudes de piezas.

Trabajadores: Encargados de gestionar las operaciones.

Vehículos: Almacena información sobre los modelos disponibles.

Vehículos favoritos: Relaciona a cada cliente con sus vehículos preferidos.

Proveedores: Distribuyen piezas y vehículos al negocio.

Solicitudes: Vinculadas a clientes que solicitan una reserva.

Reservas: Vinculadas a clientes que desean probar o adquirir vehículos.

Ventas: Registra las transacciones realizadas.

Piezas: Registra las diferentes partes de los vehículos

Piezas_Vehiculos: Relaciona las piezas asociadas a un vehículo

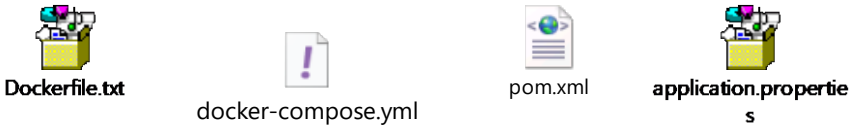
6.3 Carga de datos

Se pueden cargar datos de diferentes maneras:

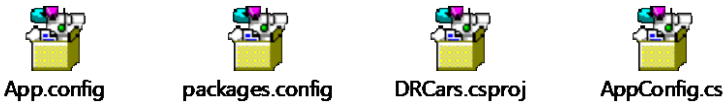
- Creando citas a través de la web como un cliente
- Añadiendo coches y usuarios desde la app de escritorio como un usuario
- Utilizando la herramienta MySQL Workbench para cargar manualmente los registros
- Scripts SQL cargados desde la herramienta previamente dicha

6.4 Ficheros de configuración actualizados

API, Docker y MySQL



App Escritorio



Ngrok

Web

6.5 Configuraciones realizadas en el sistema

Se han realizado configuraciones en Docker para gestionar la persistencia de los datos y optimizar el rendimiento del sistema. Además, la API tiene medidas básicas de seguridad para gestionar el acceso y evitar vulnerabilidades (paquete com.DRCars.config), considerando la posibilidad de uso de certificados SSL para proteger la comunicación con los clientes.

6.6 Implementaciones de código realizadas

Como se ha explicado anteriormente, el código consiste en una app de escritorio junto con una página web que interactúan con una base de datos por medio de una API alojada en un servidor, haciendo peticiones y manejando las respuestas y/o errores.

Enlace al repositorio GitHub: DRCars_Code

7 PRUEBAS

Son muchas las pruebas que pueden realizarse en un proyecto, para eliminar los posibles errores y garantizar su correcto funcionamiento. Los casos de prueba establecen las condiciones/variables que permitirán determinar si los requisitos establecidos se cumplen o no.

7.1 Casos de pruebas

A continuación, se detallan algunos de los casos de prueba que se ejecutarán para comprobar la correcta construcción de este proyecto.


Plan de pruebas DRCars.xlsx

8 EXPLOTACIÓN

La implantación es la fase más crítica del proyecto ya que el sistema entra en producción, es decir opera en un entorno real, con usuarios reales.

8.1 Planificación

Para garantizar una transición eficiente, se han definido las siguientes tareas de implantación:

1. Despliegue del sistema en el servidor Ubuntu con contenedores Docker.
2. Pruebas de conectividad y acceso desde la aplicación de escritorio y la web.
3. Verificación de la seguridad de la API, incluyendo el correcto funcionamiento de autenticación y permisos.
4. Asignación de recursos materiales y humanos necesarios para supervisar la implementación.

Se han identificado posibles riesgos, como fallos en la conexión de Ngrok, errores en la sincronización de datos o dificultades en la integración con los clientes. Para mitigarlos, se han establecido medidas preventivas, incluyendo pruebas previas y un plan de contingencia.

8.2 Preparación para el cambio

Dado que la implantación implica modificaciones en los procesos de gestión de vehículos y citas, se han tomado en cuenta los siguientes puntos:

- Autorizaciones y accesos para los usuarios que interactuarán con el sistema.
- Procedimientos claros para minimizar reticencias al cambio, garantizando una transición fluida.
- Soporte técnico inicial para resolver incidencias en la adopción del sistema.

8.3 Plan de formación

A continuación, se detalla el plan de formación a seguir:

8.4 Implantación propiamente dicha

La implantación ha sido validada mediante:

- Pruebas funcionales que confirman la operatividad del sistema en producción.
- Accesos exitosos desde la API a la base de datos en Docker.
- Interacciones entre la aplicación web y de escritorio con la API en tiempo real.

8.5 Pruebas de implantación

Se han realizado pruebas sobre el sistema implantado en el entorno de producción, incluyendo:

- Validación de seguridad con autenticación y permisos adecuados.
- Prueba de carga del catálogo en la web para verificar el rendimiento del servidor en condiciones de uso real.
- Verificación de acceso externo mediante Ngrok desde la web y pruebas de conectividad.
- Simulación de procesos de negocio, como gestión de citas y visibilidad de las reservas del cliente.

Estos procedimientos garantizan que el sistema está listo para su explotación sin comprometer la estabilidad ni la seguridad de los datos.

9 DEFINICIÓN DE PROCEDIMIENTOS DE CONTROL Y EVALUACIÓN

A lo largo del ciclo de vida del proyecto se producirán cambios e incidencias que deberán controlarse y registrarse.

A continuación, se muestran algunos de ellos:

Incidencia 1: Error de configuración CORS en la API

- **Autor:** Miguel
- **Identificación de la incidencia:** La API bloquea las solicitudes desde la web debido a restricciones de CORS.
- **Descripción de la incidencia:** La aplicación web intenta consumir la API, pero las peticiones son rechazadas por falta de un encabezado en la respuesta que permite el acceso desde distintos dominios.
    - **Valor esperado:** La web debería poder recibir datos de la API sin restricciones.
    - **Valor obtenido:** Error de CORS en el navegador.
- **Posible causa de error:** Falta del encabezado Access-Control-Allow-Origin en las respuestas de la API.
- **Posible corrección:**
    - Agregar en la configuración de la API el encabezado correcto en las respuestas.
    - Configurar CORS permitiendo acceso desde la web específica.
- **Áreas afectadas:**
    - API REST (configuración de respuestas).
    - Web (integración con la API).

Incidencia 2: Error en la deserialización de respuestas en la app de escritorio

- **Autor:** Miguel
- **Identificación de la incidencia:** La app de escritorio falla al intentar deserializar respuestas de la API.
- **Descripción de la incidencia:** Cuando la API responde, la app de escritorio espera recibir un objeto (por ejemplo, de tipo Vehículo), pero en su lugar obtiene un String, provocando un fallo en la conversión.

    - **Valor esperado:** La app debería recibir datos correctamente estructurados y convertirlos en objetos.
    - **Valor obtenido:** Excepción por error de conversión de datos.
- **Posible causa de error:**
    - La API envía respuestas en un formato inesperado.
    - Error en la configuración de la deserialización en la app.
- **Posible corrección:**
    - Ajustar la estructura de datos en la API para que corresponda al tipo esperado.
    - Revisar el código de deserialización en la app para manejar diferentes formatos.
- **Áreas afectadas:**
    - API REST (formato de respuesta).
    - Aplicación de escritorio (parsing de respuestas).

Incidencia 3: Cambio en la estructura de datos de ventas y reservas

- **Autor:** Miguel
- **Identificación de la incidencia:** Se modifica el flujo de gestión de ventas y reservas, añadiendo solicitudes como una etapa previa.
- **Descripción de la incidencia:** Inicialmente, el sistema manejaba las reservas como ventas, lo que generaba confusión en la lógica del negocio. Se decide estructurarlo en tres fases: **solicitudes, reservas y ventas**, para reflejar mejor la realidad del negocio.
    - **Valor esperado:** El sistema debería gestionar correctamente ventas y reservas.
    - **Valor obtenido:** Confusión en los procesos de negocio.
- **Posible causa de error:**
    - Diseño inicial inadecuado de la gestión de ventas.
    - Falta de diferenciación entre la intención del cliente y la reserva real.
- **Posible corrección:**
    - Implementar el nuevo modelo con tres fases (solicitud, reserva y venta).
    - Adaptar la API y la base de datos para reflejar estos cambios.
    - Actualizar la lógica en la web y la app de escritorio.

- **Áreas afectadas:**
  - Base de datos (estructura y relaciones).
  - API REST (endpoints y lógica de negocio).
  - Aplicación de escritorio y web (adaptación del flujo).

10 CONCLUSIONES

El desarrollo de este proyecto ha permitido la creación de un sistema completo, seguro y escalable para la gestión de vehículos, clientes, citas y ventas dentro de la empresa. Desde la definición del modelo de datos**,** hasta la implementación de la API REST, la web intuitiva y la aplicación de escritorio, cada componente ha sido diseñado para maximizar la eficiencia y cubrir las necesidades específicas del negocio.

Se han aplicado buenas prácticas de desarrollo, asegurando que el software sea persistente, modular y de fácil mantenimiento. Además, la contenerización mediante Docker y la exposición segura de la API a través de Ngrok han facilitado su despliegue en un entorno real, garantizando su acceso desde múltiples plataformas.

Durante la fase de explotación, se han realizado pruebas exhaustivas que validan la correcta funcionalidad del sistema, abarcando desde la autenticación de usuarios hasta la gestión integral de vehículos y ventas. También se ha establecido un plan de formación para facilitar la transición y optimizar la adopción del sistema por parte de los empleados.

A pesar de los retos enfrentados, como incidencias en la configuración de CORS, problemas de deserialización y ajustes en la lógica de ventas y reservas, se han identificado y solucionado de manera eficiente, logrando una implementación sólida y operativa.

En conclusión, este proyecto ha cumplido con los objetivos planteados, proporcionando una solución efectiva y accesible para la empresa. Su estructura escalable permitirá futuras mejoras y adaptaciones conforme a la evolución del negocio, asegurando su continuidad en el tiempo.

11 FUENTES

https://vercel.com/docs
https://docs.github.com
https://learn.microsoft.com/es-es/dotnet/csharp/
https://dashboard.ngrok.com/get-started/setup/windows