**UNIVERSIDAD AUTÓNOMA DE COAHUILA**

**UNIDAD CAMPUS ARTEAGA**

**FACULTAD DE SISTEMAS**



DESARROLLO DE APLICACIONES WEB

**PROYECTO MULTICASA**

BRANDON EMMANUEL DELABRA SALINAS

JESÚS ÁNGLE RODRÍGUEZ MARTÍNEZ

ALÍCIA MONTSERRT MONJARAS GRANADOS

BRYAN BALDERAS PEÑA

02 DE JUNIO DEL 2022

# **Tabla de Contenido**

[**Tabla de Contenido** 2](#_Toc104483358)

[**Introducción** 3](#_Toc104483359)

[**Antecedentes** 4](#_Toc104483360)

[**Análisis de requerimientos** 5](#_Toc104483361)

[**Requisitos específicos** 5](#_Toc104483362)

[**Justificación** 7](#_Toc104483363)

[**Diseño de arquitectura de microservicios** 7](#_Toc104483364)

[**¿Qué son los microservicios?** 7](#_Toc104483365)

[**Ventajas de implementar microservicios** 9](#_Toc104483366)

[**Metodología** 10](#_Toc104483367)

[**Diseño y arquitectura** 10](#_Toc104483368)

[**Programación** 10](#_Toc104483369)

[**Pruebas** 10](#_Toc104483370)

# **Introducción**

# **Antecedentes**

La compañía Saltillense, líder a nivel nacional de los bienes raíces, que lleva por nombre “Bienes Raíces Multicasa”; reconocida primeramente por la venta de sus innovadoras viviendas acorde a su diseño y construcción arquitectónica, ha querido expandir su prestigio y servicios, para ser implementados en una página web que pueda cumplir con la función de darlos a conocer, exponiendo los ideales y conceptos que la compañía maneja. Su plan es poder brindar la información necesaria para los nuevos clientes que busquen una vivienda que brinde la comodidad y bienestar que la compañía promete cumplir.

Se espera que el servicio web pueda funcionar como una herramienta para los clientes, al buscar las mejores opciones acorde al catálogo sugerido, siendo este obtenido mediante el registro de datos conforme sean las necesidades de los usuarios o incurrir en la opciones presentadas en las imágenes y los rangos de precios; así mismo, los datos de la localización del inmueble dentro de cada uno de los estados de la república que pertenezcan a la compañía; número de habitaciones y baños contemplados, todo acorde a las necesidades básicas para los compradores.

# **Análisis de requerimientos**

## **Requisitos específicos del proyecto**

1. Sitio web acorde al diseño (Figura 1).
2. Implementar un slider en la sección marcada en el Photoshop.
3. Implementar estrategia de búsqueda de casas mostrada en el diseño.
4. Implementar el mantenimiento de datos (altas, bajas y actualizaciones de los registros) solo para administradores.
5. Los administradores tendrán su forma de autenticación, por lo cual el sitio tendrá actividad pública y privada.
6. Deberá contar con un formulario de contacto por correo electrónico.
7. Deberá ser responsivo (puede utilizar cualquier estrategia).
8. Deberá proporcionar el mapa de ubicación geográfica de las casas al momento de las búsquedas. (puede utilizar API Google maps o alguna otra alternativa).
9. Deberá contener imágenes por viviendas para mejor percepción para el comprador.
10. Generar un video donde explique el objetivo de la página y cuál es la finalidad, alguna promoción etcétera o consejos, video de 2 a 4 minutos, y claro, ubicarlo en la sección o link que usted crea conveniente en el sitio web.
11. Poder generar un pdf con la ficha técnica de los datos de una casa de interés en específico.
12. Generar un reporte de todo el listado las casas que estén en la base de datos como con el estatus correspondiente (venta o vendida).
13. Obtener un dashboard (con gráfico) de la cantidad de casas por rango de costo y otro de las casas vendidas y las que aún están en venta.
14. Cuidar la compatibilidad con los navegadores más importantes.
15. Implementar la base de datos correspondiente.

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

**Figura 1:** Diseño de la página.

# **Justificación**

## **Diseño de arquitectura de microservicios**

Los microservicios constituyen un estilo arquitectónico popular para la compilación de aplicaciones resistentes, altamente escalables, que se pueden implementar de forma independiente y que evolucionan rápidamente. Pero una arquitectura de microservicios correcta requiere un enfoque diferente para diseñar y compilar aplicaciones.

Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Cada uno de servicio es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado. Un contexto delimitado es una división natural de una empresa y proporciona un límite explícito dentro del cual existe un modelo de dominio.

**Diagrama

Descripción generada automáticamente**

**Figura 2:** Diagrama del empleo de los microservicios.

## **¿Qué son los microservicios?**

* Los microservicios son pequeños e independientes, y están acoplados de forma imprecisa. Un único equipo reducido de programadores puede escribir y mantener un servicio.
* Cada servicio es un código base independiente, que puede administrarse por un equipo de desarrollo pequeño.
* Los servicios pueden implementarse de manera independiente. Un equipo puede actualizar un servicio existente sin tener que volver a generar e implementar toda la aplicación.
* Los servicios son los responsables de conservar sus propios datos o estado externo. Esto difiere del modelo tradicional, donde una capa de datos independiente controla la persistencia de los datos.
* Los servicios se comunican entre sí mediante API bien definidas. Los detalles de la implementación interna de cada servicio se ocultan frente a otros servicios.
* Admite la programación políglota. Por ejemplo, no es necesario que los servicios compartan la misma pila de tecnología, las bibliotecas o los marcos.

## **Ventajas de implementar microservicios**

* Agilidad. Dado que microservicios se implementan de forma independiente, resulta más fácil de administrar las correcciones de errores y las versiones de características. Puede actualizar un servicio sin volver a implementar toda la aplicación y revertir una actualización si algo va mal. En muchas aplicaciones tradicionales, un error en una parte de la aplicación puede bloquear todo el proceso de lanzamiento. Es posible que se requieran nuevas características a la espera de que se integre, pruebe y publique una corrección de errores.
* Equipos pequeños y centrados. Un microservicio debe ser lo suficientemente pequeño como para que un solo equipo de características lo pueda compilar, probar e implementar. Los equipos pequeños favorecen la agilidad. Los equipos grandes suelen ser menos productivos, porque la comunicación es más lenta, aumenta la sobrecarga de administración y la agilidad disminuye.
* Base de código pequeña. En las aplicaciones monolíticas, con el paso del tiempo se da la tendencia de que las dependencias del código se acaben enredando. Para agregar una característica nueva, es preciso modificar el código en muchos lugares. Al no compartir el código ni los almacenes de datos, la arquitectura de microservicios minimiza las dependencias y resulta más fácil agregar nuevas características.
* Mezcla de tecnologías. Los equipos pueden elegir la tecnología que mejor se adapte al servicio de una combinación de pilas de tecnología, según corresponda.
* Aislamiento de errores. Si un microservicio individual no está disponible, no interrumpe toda la aplicación, siempre que los microservicios de nivel superior estén diseñados para controlar los errores correctamente (por ejemplo, mediante la implementación de disyuntores).
* Escalabilidad. Los servicios se pueden escalar de forma independiente, lo que permite escalar horizontalmente los subsistemas que requieren más recursos, sin tener que escalar horizontalmente toda la aplicación. Mediante un orquestador como Kubernetes o Service Fabric se puede empaquetar una mayor densidad de servicios en un solo host, lo que aumenta la eficacia en el uso de los recursos.
* Aislamiento de los datos. Al verse afectado solo un microservicio, es mucho más fácil realizar actualizaciones del esquema. En una aplicación monolítica, las actualizaciones del esquema pueden ser muy complicadas, ya que las distintas partes de la aplicación pueden tocar los mismos datos, por lo que realizar modificaciones en el esquema resulta peligroso.

# **Metodología**

## **Método de trabajo**

Para el constante cambio de versiones implementadas dentro del desarrollo del proyecto, así mismo el poder colaborar desde diferentes entornos de desarrollo, tomando en cuenta las distintas tareas asignadas a cada uno de los miembros del equipo de desarrollo; se optó crear un repositorio en Github: un servicio basado en la nube que aloja un sistema de control de versiones (VCS) llamado Git. Éste permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantienen un seguimiento detallado de su progreso.

A simple vista se puede observar la creación de 3 carpetas, cada una de ellas destinada para los diferentes entornos de desarrollo como el Frontend, Backend, database, así mismo las instrucciones para operar con las carpetas.

Captura de pantalla de computadora

Descripción generada automáticamente

**Figura 3:** Repositorio del proyecto en GitHub.

## **Desarrollo de la página**

### **Frontend**

**R1:** Sitio web acorde al diseño (Figura 5).

Con el fin de poder interactuar con los componentes propios que establece la página, así mismo establecer interfaces interactivas de manera sencilla que los usuarios puedan manipular, se utilizo la librería React de JavaScript, que fusiona el código de HTML, CSS y JavaScript en un solo archivo. Esta librería es implementada en varias de las aplicaciones que conocemos actualmente. Por mencionar algunas como: WhatsApp, Instagram, Netflix.

Como se puede mostrar en la figura 4, el código implementado para la página de inicio usa HTML para el desarrollo estructural, a su vez utiliza elementos de React para establecer los componentes, los elementos más importantes de la librería que en esencia es una pieza de la interfaz de usuario.

Como norma general, al diseñar una aplicación con React, lo que se hace es crear componentes independientes y reusables para, poco a poco, crear interfaces de usuario más complejas. En la figura 5 se puede observar el inicio de la página, del cual se despliega los componentes (inicio, compañía, servicios, requisitos, contactos) que serán utilizados en cada despliegue de ventanas.

Texto

Descripción generada automáticamente

**Figura 4:** Código en React para inicio de página.

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

**Figura 5:** Frontend de la página de inicio.

**R14:** Cuidar la compatibilidad con los navegadores más importantes.

Ya que el explorador no puede interpretar el tipo de archivo para la programación en React (tsx), se implementó del uso del NPM (Node Package Manager), un gestor de paquetes de Nodejs un entorno de ejecución multiplataforma para ejecutar JavaScript no sólo en un navegador web (como se concibió originalmente) sino fuera de él, y poder utilizarlo en sistemas de escritorio o servidores web.

Este gestor de paquetes permite instalar de forma muy sencilla y automática paquetes JavaScript (tanto de Node como JavaScript para el navegador) y utilizarlo dentro del proyecto.

**R7:** Deberá ser responsivo (puede utilizar cualquier estrategia).

Cada una de las ventanas de contenido cumplen con un formato establecido para que, al momento de manipular cualquiera de las ventanas de contenido, el formato será ajustable a cualquier tamaño de la pantalla, en pocas palabras siendo responsivo. Esto es posible a un archivo creado en React llamado “Skeleton”; que es exportado en cada archivo relacionado con el Frontend, permitiendo mantener el formato establecido.

**Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente**Texto

Descripción generada automáticamente

**Figura 6:** Programación y vista de ventana responsiva.

En la figura 6 se puede contemplar la programación empleada para que los archivos de vistas puedan ser responsivos, como se observa en la imagen de la derecha, en donde ha sido alterado el tamaño de la ventana y la página conjunta su formato conforme esta se altera.

**R2:** Implementar un slider en la sección marcada en el Photoshop.

Imagen que contiene edificio, firmar, calle, tablero

Descripción generada automáticamente

**Figura 7:** Slider de inicio.

El inicio de la página cuenta con un slider que muestra las diferentes casas disponibles dentro del catálogo sugerido para el usuario. Este puede reproducirse en automático, al igual que puede agrandarse la pantalla de visualización, igualmente, si es que el usuario lo desea, puede cambiar o atrasar la aparición de las imágenes.

**Texto

Descripción generada automáticamente**

**Figura 8:** Archivo Content.

El archivo “Content” (Figura 8) contiene el formato establecido para la creación del slider, al cual se le cargan las imágenes de las casas que se van a presentar, siendo estas importadas desde la carpeta de imágenes.

### **Backend**

### **Base de datos**

**R15:** Implementar la base de datos correspondiente.

Diagrama

Descripción generada automáticamente

**Figura 9:** Modelo entidad relación

## **Arquitectura**

La arquitectura implementada dentro del desarrollo del proyecto es la de microservicios, utilizando la herramienta de Docker.

La razón del trabajar con Docker es por la simplicidad que maneja al trabajar con sus contenedores los cuales incluyen todo lo necesario para que el software se ejecute, incluyendo bibliotecas, herramientas del sistema, código y le tiempo de ejecución independiente mente de sistema operativo utilizado.

**Figura :** Dockerfile de la aplicación.

Antes de comenzar a operar con los contenedores, es necesario crear un DockerFile como se muestra la figura 4.

Un Dockerfile es un documento de texto sobre el que se pueden agrupar una serie de comandos con el fin que se ejecuten todos a la vez evitando así tener que ejecutarlos uno a uno manualmente, con el fin de que el proceso de crear una imagen de Docker sea mucho más rápido y eficiente. En este caso el Dockerfile para la aplicación transpilará el código de React incluyendo sus librerías y el comando de ejecución del NPM, generando un empaquetado final ligero, el cual se montará sobre un servidor Nginx preparado para producción, así mismo, designado el puerto al cual va a ejecutar la aplicación.

Una vez que es desarrollado el Dockerfile, es posible el poder crear una imagen de Docker la cual contiene las librerías, junto al código de la aplicación que contiene todo lo necesario para ejecutar nuestra aplicación y así poder ejecutar de un contenedor de Docker.

**Figura :** Contenedores para el proyecto Multicasa.

La figura 5 muestra los diferentes contenedores implementados para la ejecución de los servicios de la página (se dará una descripción de uno de los contenedores de la imagen)