**UNIVERSIDAD PRIVADA DEL VALLE**

**FACULTAD DE INGENIERÍA**

**INGENIERÍA DE SISTEMAS INFORMÁTICOS**

**UNIVALLE – COCHABAMBA**

**PROYECTO DE SISTEMAS III**

**TOTEM**

**DOCENTE:** Gaston Silva Sanchez

**Cochabamba noviembre de 2023**

**Gestión 2-2023**

**Manual Técnico**

**Equipo: Tótem**

1. **Introducción.**

El proyecto Tótem es una innovadora solución diseñada para facilitar la ubicación de lugares deseados de manera rápida y sencilla. Consta de un tótem interactivo equipado con una pantalla, un botón y un micrófono. Al presionar el botón, los usuarios podrán utilizar comandos de voz para indicar la ubicación que desean encontrar. El tótem realizará búsquedas en línea y presentará los resultados en la pantalla, mostrando el nombre del lugar, un carrusel de imágenes relacionadas, una descripción detallada y un mapa con la ruta para llegar al destino.

Este proyecto tiene como objetivo principal ayudar a las personas a encontrar fácilmente los lugares a los que desean llegar, brindándoles información completa y visualmente atractiva. Además, busca simplificar el proceso de navegación y eliminar la necesidad de realizar múltiples consultas en dispositivos móviles o computadoras. Con el tótem, la ubicación deseada estará al alcance de la mano, proporcionando una experiencia intuitiva y eficiente para los usuarios en diversos ámbitos, como turismo, navegación en interiores o simplemente descubrir nuevos lugares en su ciudad.

1. **Descripción de proyecto.**

El proyecto Tótem se desarrollará utilizando una combinación de hardware y software para lograr su funcionalidad integral. En cuanto al hardware, se utilizará una pantalla de alta resolución para mostrar los resultados de búsqueda y la interfaz de usuario. Además, se empleará un micrófono para capturar los comandos de voz de los usuarios y un botón para activar el proceso de búsqueda. Todo este hardware estará conectado a un dispositivo de procesamiento, una Raspberry y una computadora, que se encargará de controlar las interacciones y ejecutar el software.

En cuanto al software, se utilizarán diversas técnicas y herramientas para lograr las funcionalidades clave del proyecto como: React, SQL Server, visual Studio, visual studio Code, Docker.

La presentación de los resultados se llevará a cabo mediante la combinación de elementos visuales y textuales en la pantalla. El nombre del lugar se mostrará de manera destacada, seguido de una descripción detallada que proporcionará información adicional sobre el lugar. Asimismo, se incluirá un mapa con la ruta para llegar al destino, el cual se generará utilizando una API de mapas y se mostrará en la pantalla del tótem.

1. **Roles / integrantes.**

|  |  |
| --- | --- |
| Integrantes | Rol |
| Fernando Daniel Aparicio Estrada | GIT Master/Developer A cargo de la continuación |
| Alan Abdel Montaño Cruz | Team Leader anterior |
| Luciana Mayra Blanco Aranibar | Integration anterior |
| Nicolas Tadeo Meneses Barbery | DB Architect / infrastructure anterior |
| Alicia Natalia Virreira Idina | Scrum Master anterior |

1. **Arquitectura del software.**

La arquitectura de nuestro proyecto se compone de varios componentes clave que interactúan entre sí para lograr nuestros objetivos. A continuación, presento una explicación de la estructura y organización del software, considerando los elementos mencionados:

Componentes principales:

En nuestro proyecto utilizamos una base de datos SQL Server para almacenar y gestionar los datos. La base de datos actúa como una fuente centralizada de información. Utilizamos el modelo relacional de SQL Server para definir la estructura de la base de datos y establecer relaciones entre las tablas.

Para la comunicación entre el frontend y la base de datos, desarrollamos una API utilizando ASP.NET Core. Esta API proporciona puntos de acceso para interactuar con los datos almacenados en la base de datos. Definimos controladores y rutas en la API para recibir solicitudes desde el frontend y realizar operaciones correspondientes en la base de datos.

La interfaz de usuario de nuestro proyecto se implementó utilizando React, un framework de JavaScript. React nos permite construir una interfaz de usuario interactiva y receptiva, proporcionando componentes reutilizables y una gestión eficiente del estado de la aplicación. Nuestro frontend se comunica con la API para obtener y enviar datos al backend.

Interacciones entre componentes:

Nuestro frontend desarrollado con React se comunica con la API utilizando solicitudes HTTP. Utilizamos bibliotecas como Axios para realizar llamadas a la API y enviar datos a través de métodos como GET, POST, PUT o DELETE. Recibimos las respuestas de la API en formato JSON y las utilizamos para actualizar la interfaz de usuario según sea necesario.

La API establece una conexión con la base de datos SQL Server para realizar operaciones de lectura y escritura. Utilizamos consultas SQL para recuperar datos de la base de datos o modificarlos según las solicitudes recibidas desde el frontend. Para simplificar las operaciones de acceso a la base de datos y mapear objetos a tablas y viceversa, utilizamos un ORM (Object-Relational Mapping) como Entity Framework Core.

Patrones de diseño:

Para organizar la estructura de nuestra aplicación, seguimos el enfoque de arquitectura de tres capas. Tenemos una capa de presentación (React frontend), una capa de lógica de negocio y una capa de acceso a datos (API y base de datos SQL Server). Este enfoque facilita la escalabilidad y el mantenimiento del sistema.

En el desarrollo de la API, aplicamos el patrón de diseño MVC (Modelo-Vista-Controlador) para separar la lógica de negocio de la presentación y el control de las solicitudes. También utilizamos el patrón Repository para abstraer el acceso a la base de datos y facilitar las operaciones CRUD.

1. **Requisitos del sistema.**

**Requerimientos de Hardware (mínimo):** 500 MB de almacenamiento, Intel Pentium 4, 512 md de RAM.

**Requerimientos de Software: (cliente):**

Windows 7, Docker, Visual studio Code 2019.

**Requerimientos de Hardware (server/ hosting/BD)**

Procesador 1.4 GHz (64 bits), RAM 512 MB, Disco duro 32 GB.

**Requerimientos de Software (server/ hosting/BD)**

Docker,Sql Server

1. **Instalación y configuración.**

En este proyecto, los componentes están contenidos en contenedores docker, para lo cual se tendrá que importar las imágenes de la base de datos, API y la app de react o si no se dispone de estas crearlas con los dockerfile de cada componente. Si es necesario, en el proyecto de React, deberemos modificar las URL o direcciones que se conectan con la API, y también cambiar la cadena de conexión en la API para establecer la conexión con la base de datos.

**Instalación de la dependencia Node.js**

768 install

**Instalación de la librería de mandado de emails**  
npm install @emailjs/browser --save

**Instalación de la librería Sweet Alert para el uso de alertas**

npm install --save sweetalert2 sweetalert2-react-content

**Instalación de la librería de cifrado de contraseñas**  
*npm install crypto-js*

**Instalacion de librería de axios**

npm install axios

Estas son las configuraciones que deberemos realizar para poder utilizar u instalar nuestra aplicación.

1. **Procedimiento de dockerizado.**

**Pasos para crear el contenedor de la Base de datos en docker:**

1. Descargar la imagen de la base de datos de la aplicación tótem
2. Una vez obtenido el archivo “totem-sqlserver.tar” abrimos una consola en la ubicación del archivo con el servicio de docker corriendo
3. En esta consola ejecutaremos el siguiente comando: “docker load tótem-sqlserver.tar” para asi tener importada la nueva imagen.
4. Una vez realizado esto ejecutaremos el siguiente comando: “**docker run -d -p 8433:1433 --name sqlserver totem-sqlserver**”
5. Hecho esto ya se nos habrá creado la base de datos en docker.

**Pasos para crear el contenedor de la API en docker:**

1. Descargar la imagen de la API de la aplicación tótem
   1. **En caso de no tener la imagen docker ejecutar el siguiente comando dentro del directorio de la API: “docker build –t totem-api .” esto creará una imagen con las configuraciones actuales del proyecto, entonces saltea el paso 2 y 3**
2. Una vez obtenido el archivo “totem-api.tar” abrimos una consola en la ubicación del archivo con el servicio de docker corriendo
3. En esta consola ejecutaremos el siguiente comando: “**docker load totem-api.tar**” para asi tener importada la nueva imagen.
4. Una vez realizado esto ejecutaremos el siguiente comando: “**docker run -d -p 5001:5001 -p 5000:5000 --name totemapi totem-api**”
5. Hecho esto ya se nos habrá creado la API en docker.

**Pasos para crear el contenedor de la aplicación de react en docker:**

1. Descargar la imagen de la API de la aplicación tótem
   1. **En caso de no tener la imagen docker ejecutar el siguiente comando dentro del directorio de la aplicación: “docker build –t totem-react .” esto creará una imagen con las configuraciones actuales del proyecto entonces saltea el paso 2 y 3**
2. Una vez obtenido el archivo “totem-react.tar” abrimos una consola en la ubicación del archivo con el servicio de docker corriendo
3. En esta consola ejecutaremos el siguiente comando: “**docker load totem-api.tar**” para asi tener importada la nueva imagen.
4. Una vez realizado esto ejecutaremos el siguiente comando: “**docker run -p 5173:5173 --name totem totem-react**”
5. Hecho esto ya se nos habrá creado la aplicación en docker.

Una vez creados todos los contenedores deberemos de conectarlos usando docker network con los siguientes comandos:

1. **docker network create dockernetwork**
2. **docker network connect dockernetwork sqlserver**
3. **docker network connect dockernetwork totemapi**
4. **docker network connect dockernetwork tótem**

Con estos comandos haremos que los contenedores puedan comunicarse entre sí y así no tengamos problemas.

1. **GIT.**

Este es el link de repositorio de la API:

<https://github.com/Totem-Univalle/Totem_Api_Univalle.git>

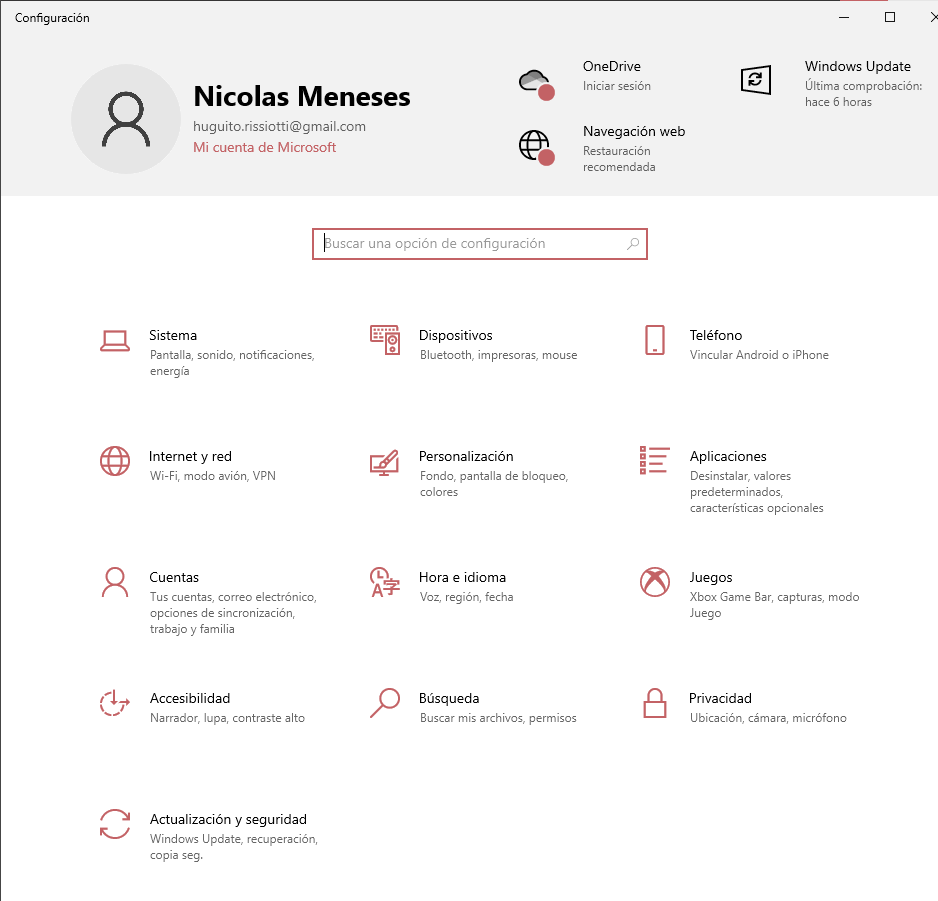
Este link es del proyecto:

<https://github.com/Totem-Univalle/Totem.git>

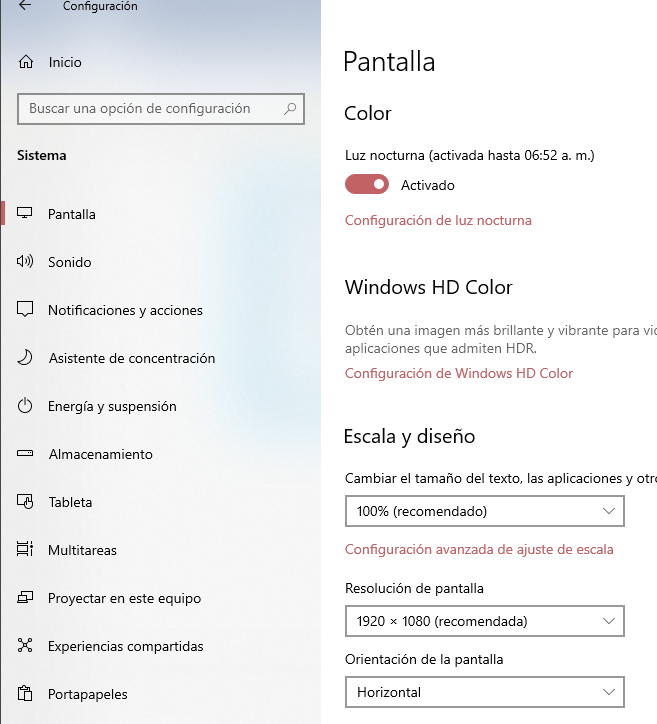
1. **Personalización y configuración.**

Lo único que la persona tiene que configurar es que vaya a Ajustes en la computadora

Y seleccione la opción de sistema.



Y en pantalla seleccionar Orientación de la pantalla seleccionamos la “opción de Vertical”



1. **Seguridad.**

El presente proyecto ha implementado un sistema de autenticación y obtención de información de una API mediante el uso de tokens y cookies. Estos mecanismos son indispensables para asegurar que solo los usuarios autorizados puedan acceder a la API.

El proceso de inicio de sesión se lleva a cabo mediante la generación y validación de tokens. Los usuarios deben contar con los tokens correspondientes para poder acceder a la API. Estos tokens actúan como credenciales de autenticación, garantizando la legitimidad del usuario y autorizando su acceso a los recursos protegidos.

Además de los tokens, se utilizan cookies para facilitar el inicio de sesión y mantener la sesión activa. Las cookies son pequeños archivos de texto almacenados en el navegador del usuario. Al iniciar sesión, se genera una cookie que contiene información relevante, como un identificador único de sesión.

Durante las solicitudes subsiguientes a la API, el navegador adjunta automáticamente la cookie pertinente, permitiendo así que la API identifique la sesión del usuario y valide su acceso en base a la información almacenada en la cookie.

En relación a la duración de la sesión, se ha implementado la funcionalidad de expiración de cookies. Después de transcurrir un período de tiempo determinado, las cookies caducan y, como resultado, el usuario es deslogueado automáticamente del sistema. Esta medida de seguridad se implementa para prevenir accesos no autorizados a la cuenta del usuario en caso de que olvide cerrar sesión o abandone el dispositivo sin protección.

En conclusión, el proyecto ha integrado con éxito el uso de tokens y cookies para el inicio de sesión y la obtención de información de la API. La seguridad proporcionada por docker, garantiza la autenticación y autorización adecuadas de los usuarios. Además, la expiración de cookies brinda una capa adicional de seguridad al desloguear al usuario después de un cierto período de tiempo. Estas implementaciones aseguran la protección de los recursos y la privacidad de los usuarios en el marco de este proyecto.

1. **Depuración y solución de problemas.**

Para identificar y comprender adecuadamente los problemas específicos relacionados con la arquitectura de mi proyecto, se recomienda seguir estos pasos:

a) Reproducción del problema: Intentar recrear el problema de manera sistemática para comprender su origen y los pasos necesarios para replicarlo de manera consistente.

b) Análisis de registros y mensajes de error: Examinar los registros de la aplicación y los mensajes de error para obtener información relevante sobre el problema en cuestión. Esto permitirá comprender dónde y por qué se está produciendo el error en el contexto de la API ASP.NET Core, la base de datos SQL Server y la interacción con el frontend en React.

Solución de problemas:

Una vez que se haya identificado un problema específico, es fundamental seguir estos pasos para su resolución:

a) Verificación de la configuración: Asegurarse de que todas las configuraciones necesarias para la API ASP.NET Core y la conexión a la base de datos SQL Server estén correctamente establecidas. Esto incluye comprobar las cadenas de conexión, la configuración de red y cualquier otro aspecto relevante.

b) Examen del código: Analizar minuciosamente el código relacionado con el problema específico en la API ASP.NET Core y en el frontend de React. Revisar si existen errores de sintaxis, llamadas a funciones incorrectas o lógica inadecuada que puedan estar causando el problema.

c) Depuración: Utilizar herramientas de depuración disponibles en el entorno de desarrollo de la API ASP.NET Core y React para examinar el estado de las variables, rastrear la ejecución del código y detectar posibles problemas específicos en la interacción entre la API, la base de datos y el frontend.

d) Pruebas unitarias: Implementar pruebas unitarias específicas para las diferentes partes de la API ASP.NET Core y el frontend de React. Estas pruebas unitarias deben abarcar escenarios relevantes para identificar rápidamente problemas y errores en segmentos específicos de código.

1. **Glosario de términos.**

**Base de datos:** Programa capaz de almacenar gran cantidad de datos, relacionados y estructurados, que pueden ser consultados rápidamente de acuerdo con las características selectivas que se deseen.

**API:** significa “interfaz de programación de aplicaciones”. En el contexto de las API, la palabra aplicación se refiere a cualquier software con una función distinta. La interfaz puede considerarse como un contrato de servicio entre dos aplicaciones.

**Cookies:** es una pequeña información enviada por un sitio web y almacenada en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del navegador.

1. **Referencias y recursos adicionales.**

<https://youtu.be/MLFU3CkOd1o>

<https://youtu.be/HEFXebTGiMA>

<https://youtu.be/XVcyVnNWUwo>

<https://youtu.be/rLoWMU4L_qE>

<https://codigofacilito.com/cursos/react-api-store>

<https://codigofacilito.com/cursos/javascript-profesional>

1. **Herramientas de implementación.**

Visual Studio.

Visual Studio Code.

SQL Server

GIT HUB

Docker

1. **Bibliografía.**

<https://tailwindui.com/documentation>

<https://tailwindcss.com/docs/installation>

<https://legacy.reactjs.org/docs/getting-started.html>