ADMINISTRACIÓN DE SISTEMAS DE INFORMACIÓN 2025

Alumna: Isabella Bresciani

Profesores:

- Prof. Ing. Gustavo Cerveri - Profesor Adjunto
- Prof. Ing. Martín Jorge - JTP
- Ing. Franco Kral - Ayudante

PRACTICA N° 5
Desarrollo Full Stack

Temas de la unidad

a. Docker.

b. Azure.

c. Desarrollo web.

d. Máquinas virtuales.

e. Nginx.

Contexto

Imagina que has sido contratado/a por una startup tecnológica que necesita una aplicación web funcional y lista para producción. Tu rol como desarrollador/a Full Stack será fundamental para construir toda la solución desde cero. La empresa requiere un equipo ágil capaz de desarrollar tanto el backend como el frontend, asegurando la comunicación eficiente entre ambas partes y desplegando la solución final en la nube.

Objetivos principales del trabajo

- Aplicar conocimientos reales de desarrollo web full stack.
- Aprender sobre integración y despliegue en entornos de producción.
- Entender la importancia del versionamiento de código, la contenerización y el despliegue en la nube.

Cumplenda

Cumplenda es una aplicación web diseñada para resolver un problema simple pero universal: no olvidar nunca más el cumpleaños de un ser querido. En un mundo lleno de notificaciones y distracciones, Cumplenda ofrece un espacio centralizado, privado y enfocado. Permite a los usuarios crear una cuenta segura para registrar y gestionar las fechas de nacimiento de amigos, familiares y colegas. Su principal ventaja es la simplicidad: una interfaz limpia que muestra los próximos cumpleaños de forma ordenada, ayudando al usuario a planificar con antelación y a mantener vivas sus relaciones importantes.

1. Repositorio

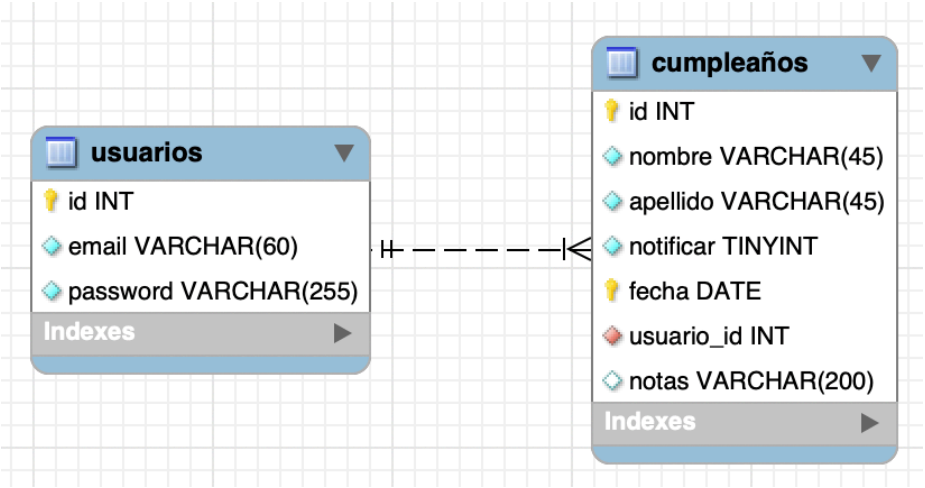
Link Repositorio Github: https://github.com/IsabellaBresciani/Cumplenda

Link Video: TP5 DEMO ADR - Isabella Bresciani.mov

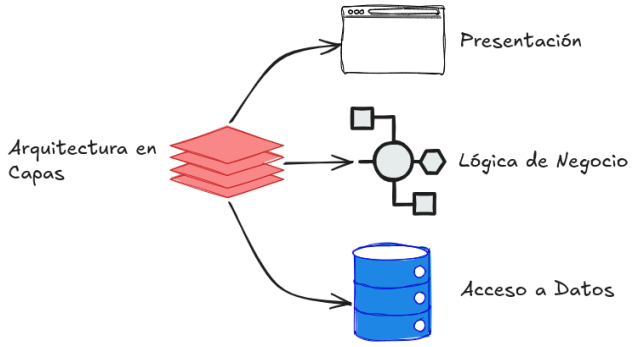
2. Desarrollo

3.1 Tech Stack

3.1 Diagrama de Entidad Relación



3.2 Desarrollo Backend



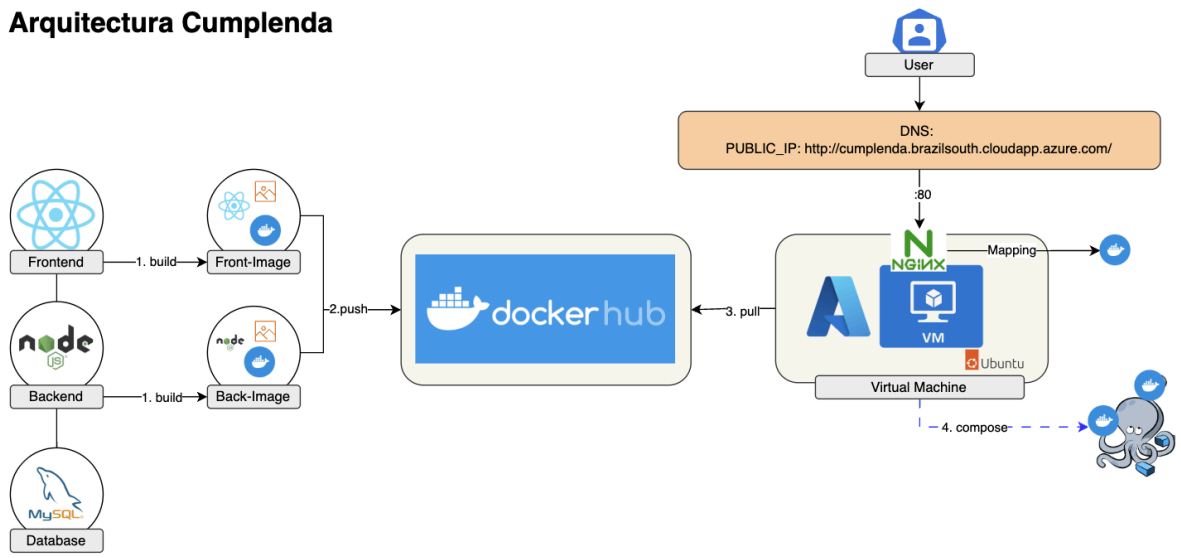
Para el desarrollo del backend se utilizó una arquitectura por capas, un patrón de diseño que promueve la separación de responsabilidades y la modularidad del código. Esta estructura facilita el mantenimiento, la escalabilidad y las pruebas de la aplicación. Como se observa en la estructura de directorios, el flujo de una solicitud atraviesa varias capas bien definidas.

3.3 Desarrollo Frontend

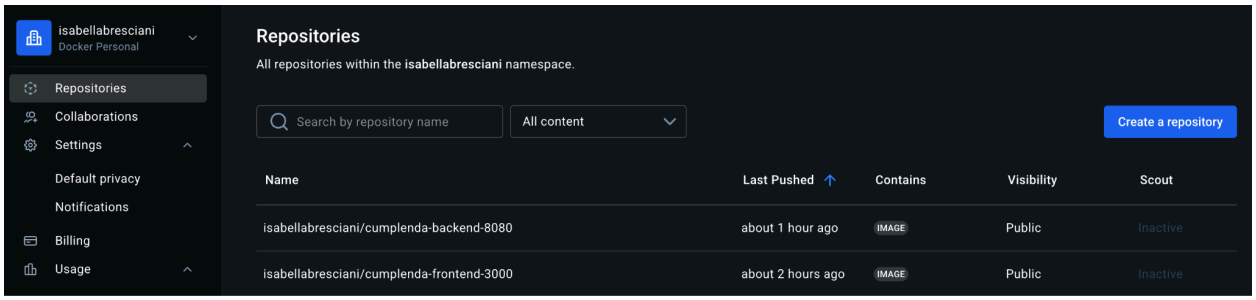
La arquitectura del frontend está construida con **React** y sigue un **diseño modular** que separa claramente las responsabilidades, facilitando su mantenimiento y escalabilidad.

- **components**: Contiene piezas de UI reutilizables como formularios y la barra de navegación.
- **pages**: Define las vistas completas de la aplicación, como la página de inicio (Home) o la de autenticación (Auth).
- **context**: Gestiona el estado global. AuthContext es clave, ya que controla la información de sesión del usuario para toda la aplicación.
- **routes**: Protege las rutas de la aplicación. PrivateRoute asegura que solo los usuarios autenticados puedan acceder a ciertas páginas.
- **services**: Centraliza toda la comunicación con el backend. Aquí se definen las funciones para hacer login, registrarse y gestionar los cumpleaños.
- **utils**: Incluye funciones de ayuda reutilizables, como la que muestra notificaciones (Toast).

3. Deploy



1. **Contenerización (Build):** El código fuente del **Frontend (React)** y del **Backend (Node.js)** se empaqueta en imágenes de Docker. Este proceso crea entornos aislados y portátiles (Front-Image y Back-Image).
2. **Registro de Imágenes (Push):** Las imágenes de Docker generadas se suben a **Docker Hub**, que funciona como un registro centralizado.



3. **Creación de máquina virtual:** Una vez subidas las imágenes al Docker Hub, el despliegue se realiza en una Máquina Virtual (VM) en Microsoft Azure, configurada con el sistema operativo Ubuntu. A la misma hay que habilitarle los puertos 80, 22 y 443. El 22 nos permite conectarnos a través de ssh, y acceder a la terminal bash de la máquina. Mientras que los puertos 80 y 443 nos permiten comunicarnos via HHTP/S.
4. **Entorno de Producción (Pull):** A través de ssh y con las keys descargadas, le instalamos a la VM todo lo necesario: Docker, Nginx, Docker Compose. Para luego poder descargar (pull) las imágenes del frontend y backend desde Docker Hub y orquestar los contenedores.

5. **Orquestación (Compose):** Se utiliza **Docker Compose** dentro de la VM para orquestar los contenedores. A través de un archivo de configuración, Docker Compose levanta y conecta los servicios de la aplicación: el contenedor del frontend, el del backend y el de la base de datos **MySQL**. Para la base de datos se corre un init.sql que crea todas las tablas declaradas en el punto 3.1 (DER).
6. **Acceso y Enrutamiento:**

- Un **usuario** accede a la aplicación a través de un nombre de dominio público (**DNS**), que resuelve a la IP de la máquina virtual en Azure.
- Las solicitudes llegan al puerto 80 y son recibidas por **NGINX**, que actúa como un **proxy inverso**.
- NGINX se encarga de redirigir el tráfico: las solicitudes a la interfaz de usuario se envían al contenedor del frontend, mientras que las llamadas a la API se dirigen al contenedor del backend, permitiendo una comunicación fluida y segura entre los componentes.

4. Principales dificultades

1. Configuración del Proxy Inverso (Nginx)
2. Comunicación entre Contenedores (Networking)
3. Inconsistencias en Puertos e Imágenes
4. Inicialización de la Base de Datos