TP BDD Merlino 2c2024

Integrantes

- Julián Gorge - 104286
- Martin Pata Fraile de Manterola - 106226
- Joaquín Velurtas - 109655
- Rodrigo Souto - 97649

Elección de las Tecnologías

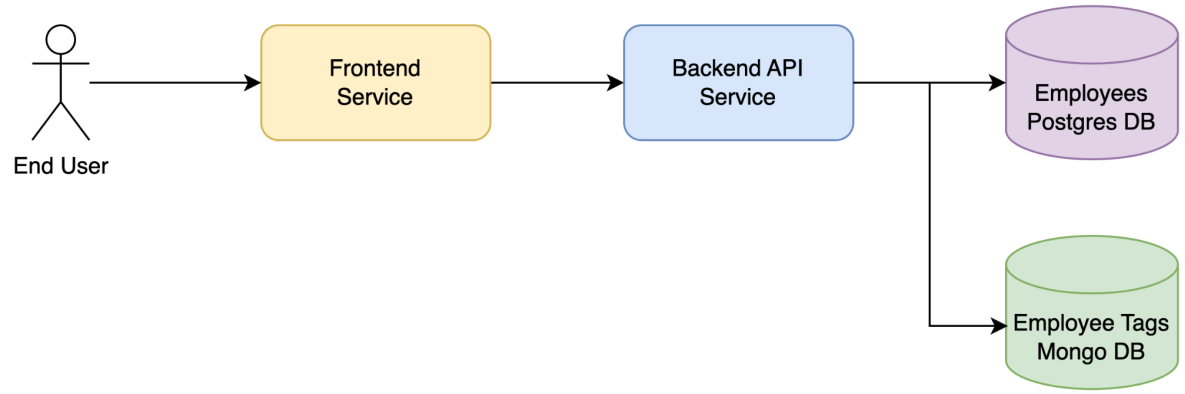
Para el frontend se definió trabajar con el framework NextJS, escrito en React. Junto con la conocida librería de componentes Shadcn basada en Tailwind nos ayudó a tener una interfaz de usuario moderna y elegante. Esta se conectaba con la API del backend utilizando pocas líneas de código.

Para el servicio Backend API nos inclinamos por Golang, porque ser uno de los lenguajes relativamente emergentes más prometedores que en la actualidad ya ha llegado a consagrarse en la industria. Lo elegimos porque algunos miembros del equipo trabajamos con este lenguaje y otros estábamos interesados en adquirir experiencia con él, por fuera de esta preferencia llegamos al consenso que cualquiera de los otros lenguajes backend de gran adopción por la industria que barajamos hubiesen sido igual de proficientes para los objetivos de este TP (Python, NodeJS, Java, Scala).

Como base de datos NoSQL elegimos MongoDB por su versatilidad y conocimiento previo de la herramienta. Para la conexión utilizamos la librería MongoDB Go Driver debido a que tiene soporte directo de MongoDB. Como herramienta para las DB migrations de Mongo usamos Golang Migrate (única opción que encontramos).

En la elección de base de datos SQL fuimos por PostgreSQL, por ser en nuestra opinión el motor de base de datos open source más confiable y con el que todos los miembros del equipo ya teníamos experiencia previa. Como librería de conexión a la DB usamos pq, y para las migrations usamos goose.

Diagrama de Arquitectura



Configuración y Conexión a Bases de Datos

Ambas DBs, PostgreSQL y MongoDB, fueron levantadas con Docker Compose, usando las imágenes estándar `postgres` y `mongodb` respectivamente, que permiten una rápida y simple configuración utilizando solamente variables de entorno (como se acostumbra en el estado del arte de la configuración de servicios actual) para definir las credenciales. Para los puertos utilizamos los estándares de cada DB.

Gracias al feature de Docker Compose poder crear una network bridge interna entre los diferentes servicios pudimos conectar fácilmente el servicio backend a ambas DBs.

Para conectarnos PostgreSQL para las operaciones CRUD usamos la librería pq, pero apenas arranca el servicio backend Golang se ejecuta una migración con la librería goose que crea la tabla con los esquemas y agrega mock data para fácil testeo del programa. En la URI tuvimos que poner el protocolo `postgres`, agregar el host (al que puede acceder gracias a que comparten la Docker Compose network), el puerto 5432 (default de PostgreSQL), las credenciales que proveímos por variable de entorno y la tabla que creamos en la migración.

En la conexión a MongoDB con el backend Golang usamos la librería estándar de MongoDB (MongoDB Go Driver) y para la migración de la data inicial usamos Golang Migrate, asegurándonos que la mock data agregada coincida con la insertada previamente en la migración de la PostgreSQL. Al igual que en el caso anterior tuvimos que proveer la URI de conexión, pero en este caso el protocolo a usar es `mongodb`, agregar el host (al que también puede acceder gracias a que comparten la Docker Compose network), el puerto 27017 (default de MongoDB), las credenciales que proveímos por variable de entorno y la tabla que creamos en la migración.

Descripción de Funcionalidades CRUD

- `GET /employees`
  - Es el primer llamado que hace el frontend para traer la lista de todos los empleados de la SQL DB.
- `POST /employees`
  - Usado por el botón “Add Employee” para crear nuevos registros de empleados en la SQL DB.
- `PUT /employees/{id}`
  - Dentro del modal que se crea al apretar el botón de “Edit”, se permite cambiar algunos de los valores del empleado, y los cambios se realizan luego de clickear el botón “Update Employee”.
- `DELETE /employees/{id}`
  - Tras clickear el botón “Delete” de un empleado, saltará un modal para una doble confirmación que luego de aceptarla, eliminará por completo el registro en la SQL DB.
- `GET /employee_tags/{employee_id}`
  - Para cada empleado, se traen las tags asociadas de la NoSQL DB.
- `POST /employee_tags/{employee_id}/tags/{tag_name}`
  - Agrega una nueva tag para un empleado. Las tags son entidades que no pueden modificarse. Para una misma tag solo puede haber un único registro, por lo que esta es una operación idempotente.
- `DELETE /employee_tags/{employee_id}/tags/{tag_name}`
  - Elimina una nueva tag para un empleado. Para una misma tag solo puede haber un único registro, por lo que esta es una operación idempotente.

Comparación entre Bases de Datos Relacionales y NoSQL

Utilizar PostgreSQL como motor de base de datos relacional nos pareció la mejor opción para persistir los registros de los empleados, al ser una herramienta confiable ya probada durante muchos años en la industria y por la naturaleza relacional de los datos de los empleados que vamos a guardar.
Por otro lado para los datos sobre los tags buscamos una herramienta más flexible, que nos permita mantener data no estructurada y con features como “sets” de valores únicos, y con la posibilidad de extender fácilmente la funcionalidad de las tags de los empleados con features como “no eliminación permanente” (para tener un registro histórico de todas las tags que tuvo).

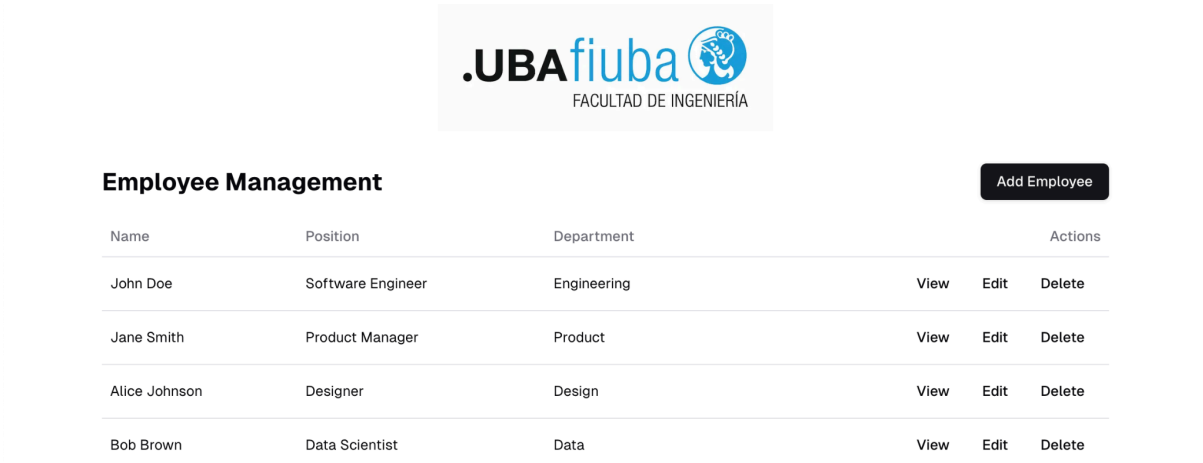
Dificultades y Aprendizajes

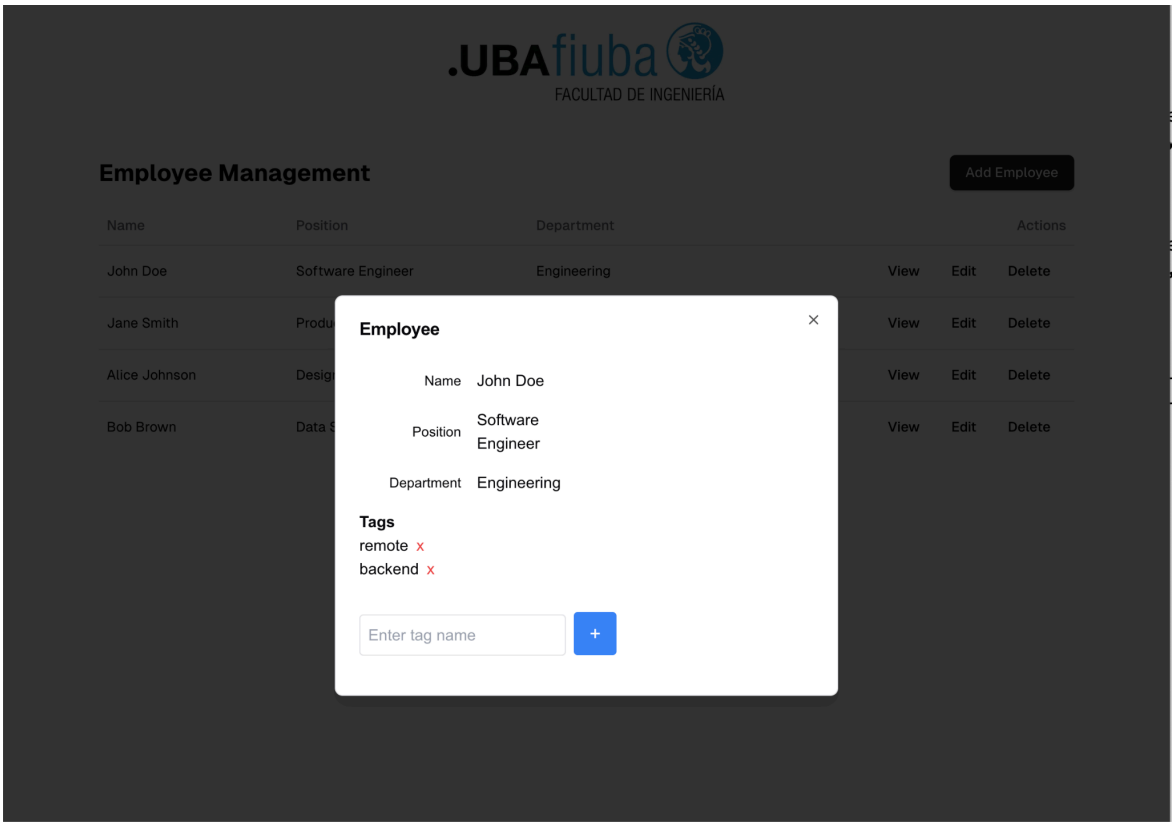
Una dificultad que nos encontramos fue la poca variedad y documentación de migration tools en Go para MongoDB. Prácticamente Golang Migrate es la única alternativa para utilizar y en todos los tutoriales que encontramos mostraban prácticamente el mismo ejemplo.

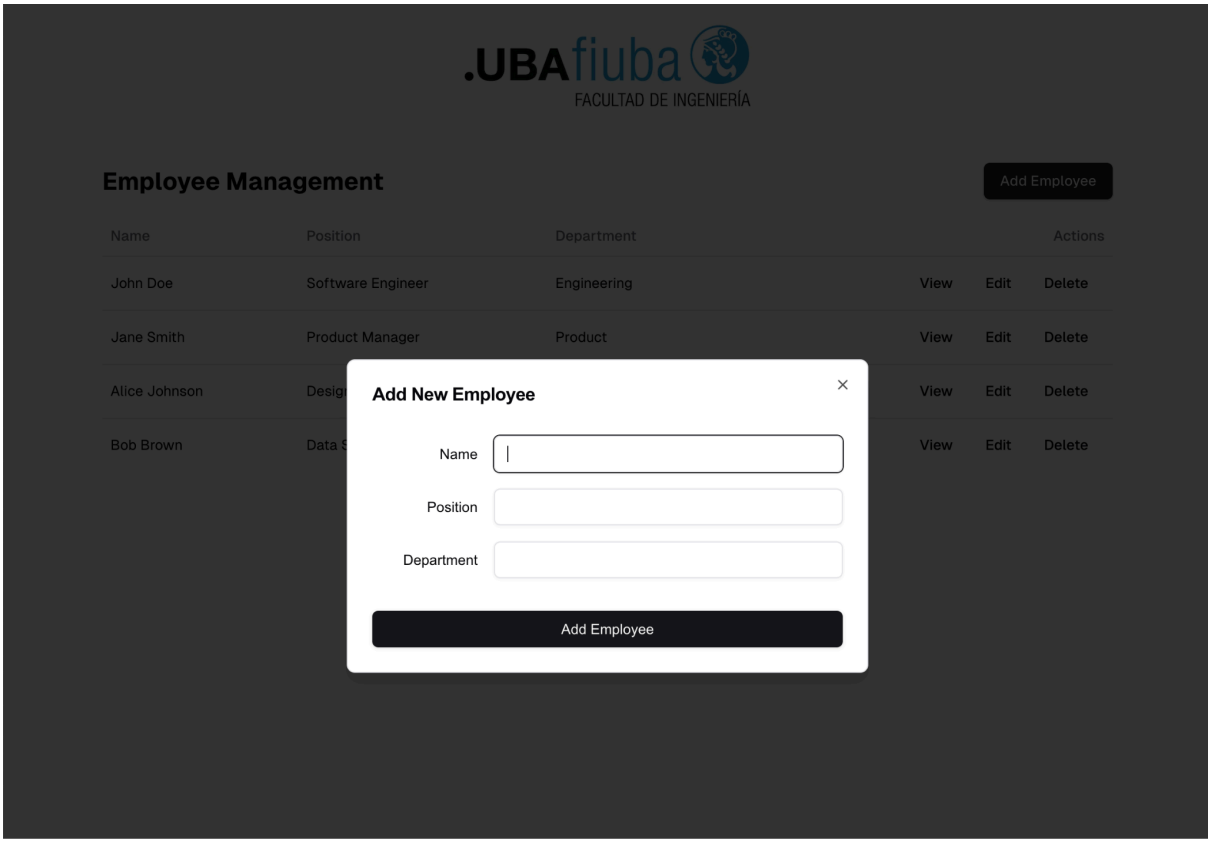
Confirmamos una vez más el enorme poder de desarrollo que provee la herramienta Docker Compose para estos tipos de trabajos, donde la portabilidad del proyecto entre integrantes con que poseen computadoras con diferentes sistemas operativos y distintos backgrounds técnicos se vuelve imprescindible para el óptimo desarrollo del proyecto sin percances innecesarios de configuración.
Aparte de los servicios, nos permitió setupear y levantar las bases de datos rápidamente incrementando la rapidez de las iteraciones de desarrollo.

Si bien configurar herramientas de migración para ambas DBs no fue tan simple por el ramp up de conocimiento que requirió, también nos ayudó mucho a que todos los miembros del equipo manejemos los mismos datos y podamos visualizar rápidamente las tablas, estructura y naturaleza de la data que fueron aportando los otros integrantes en los features que trabajaron.

Algunas Vistas del Frontend






.UBAfiuba
FACULTAD DE INGENIERÍA
Employee Management
Add Employee
Name
Position
Department
Actions
John Doe
Software Engineer
Engineering
View
Edit
Delete
Jane Smith
Product Manager
Product
View
Edit
Delete
Alice Johnson
View
Edit
Delete
Bob Brown
View
Edit
Delete
Add New Employee
Name
Position
Department
Add Employee