Trabajo Final

Desarrollo de un Sistema Web con Microservicios

El objetivo del Trabajo Final es desarrollar un sistema web completo, basado en microservicios y utilizando las tecnologías vistas durante la materia. El dominio del sistema es a elección (por ejemplo: e-commerce, reservas, plataforma educativa, actividades deportivas, fintech, etc.). El mismo debe incluir al menos los siguientes componentes:

Backend,

desarrollando en Go, subdividido en:

- API de usuarios (users-api): permite crear usuarios, obtenerlos por ID y realizar login. Este último debe implementarse con validación de token JWT. Debe utilizar un mecanismo de hashing para no almacenar las contraseñas en texto plano. Utiliza MySQL como storage y se conecta utilizando GORM. Soporta 2 tipos de usuarios: normal, admin. Este último tiene permisos de administrar el sitio y acceso a las operaciones de escritura además del owner (dueño) de la entidad.

- API de la entidad principal (products-api, booking-api, activities-api, courses-api, etc): soporta las operaciones de creación, actualización, obtención y borrado por ID de la entidad principal. Por ej. si el sitio es reserva de hoteles, opera contra esta entidad. Utiliza MongoDB como storage y al menos una colección. Para las operaciones de escritura (create, update, delete) valida la existencia del owner (user ID) contra la API de usuarios invocando al endpoint de obtención por ID mediante HTTP. Además, envía una notificación con la operación y el ID de la entidad a un tópico de RabbitMQ, encargado de comunicar las novedades. Implementa un endpoint de acción en el caso de que no sea suficiente con el de creación. La operación principal (creación o acción) debe implementar un proceso de cálculo concurrente, que subdivide la ejecución en Go Routines y comunica el resultado utilizando un Channel. Las mismas deben sincronizarse utilizando un Wait Group.

- API de búsqueda (search-api): soporta la búsqueda paginada sobre la entidad principal. Por ej. si la entidad principal es hoteles, entonces search-api indexa los hoteles. Permite filtrar las búsquedas y ordenar los resultados por uno o más campos presentes en la entidad. Utiliza SolR como motor de búsqueda. Define la entidad en un índice. Además, implementa un consumidor de RabbitMQ que propaga (sincroniza) las operaciones de escrituras (create, update, delete de la entidad principal) al índice. Para completar el ID y garantizar consistencia, invoca al endpoint de obtención por ID de la API principal. Implementa una doble capa de caché (local con CCache y distribuída con Memcached) para aquellas queries de búsqueda más repetidas.

Frontend,

desarrollado con React, con las siguientes pantallas:

- Login: con campos para usuario o email, y contraseña. El mismo genera una llamada al endpoint de login de API de usuarios.
- Registro: con campos para usuario, email, nombre, apellido, y contraseña. El mismo genera una llamada al endpoint de creación de usuario normal en API de usuarios.
- Home / Búsqueda: compuesta al menos de:
    - Barra de búsqueda con resultados para empty query, que genera una llamada al endpoint de listado paginado de API de búsqueda
    - Vista mínima de los resultados filtrados luego de la búsqueda, con botón de Ver Detalles que redirecciona a la vista de Detalles.
- Detalles: vista completa de un objeto de la entidad con toda su información y un botón para accionar sobre la entidad (crear la reserva, comprar el producto, inscribirse a la clase, etc.). Genera una llamada al endpoint de acción (o creación) de la entidad principal.
- Congrats: no es necesariamente una vista, pero debe implementar un mensaje que comunique al usuario si la operación se realizó de manera exitosa.
- Mis acciones: vista de las acciones generadas por el usuario. Por ej. si el sitio es de reserva, entonces esta vista se llama “Mis reservas” y muestra las reservas del usuario.
- Pantalla de administración: sólo visible para usuarios administradores, permite:
    - Listar todas los objetos de la entidad principal (reservas, cursos, etc.).
    - Operar sobre las mismas (actualizar o eliminar).

Consideraciones

- Todos los microservicios de backend deben estar implementados siguiendo el patrón MVC (controllers, domain, services) y deben implementar los repositorios necesarios para su funcionamiento. Los mismos deben implementar la interfaz requerida en cada capa superior.
- Las comunicaciones necesarias entre los microservicios del backend y entre frontend-backend deben realizarse mediante el protocolo HTTP y utilizar formato JSON en los request bodies y en las responses.
- Todos los microservicios de backend deben validar los errores en todas las capas (repositorios, servicios y controladores) utilizando los códigos de estado correctos. En el caso que algún paso falle, el mismo debe mostrarse y propagarse.
- Al menos un microservicio de backend debe implementar tests para los casos de uso definidos en el service. Esto se implementar en un archivo llamado {entidad}_service_test.go
- Todo el código debe subirse a Github y estar versionado por todos los integrantes del grupo. Si bien cada uno puede enfocarse en algún componente en particular, todos deben entender el funcionamiento del sistema y las tecnologías de manera global.
- Todos los microservicios deben correr en contenedores de Docker y gestionado por Docker Compose, herramienta con la que se debe ejecutar el proyecto completo.

Primera Entrega *(Semana del 7/11 al 14/11)*

No es necesario implementar el proyecto completo para la nota parcial (primera entrega). Para esta fecha, deben estar implementados todos los componentes utilizando Docker, pero sólo es necesario soportar el flujo normal del usuario: Login → Búsqueda → Detalle → Acción → Congrats, tanto a nivel de frontend como de backend. No es necesario implementar usuarios administradores ni pantalla de administración. Tampoco es necesario tener la Vista de Registro, la Vista de Mis Acciones o el cálculo concurrente para la acción de la entidad principal.

Diagrama Orientativo