Documentación sobre la página web

1. Diagrama de arquitectura:

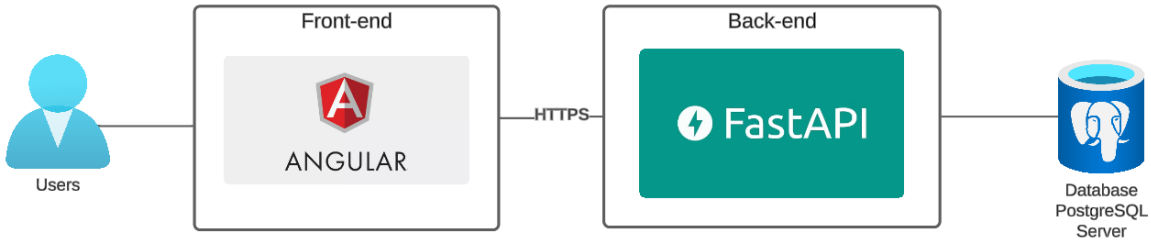

Figura 1. Diagrama arquitectura general

Patrones utilizados:
Back-end: Arquitectura basada en DDD.

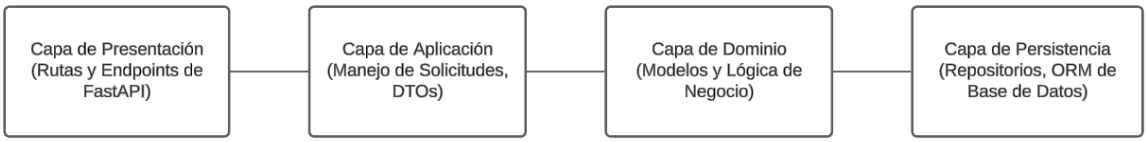

Figura 2. Diagrama arquitectura back-end

Front-end: Arquitectura modular y basada en componentes.

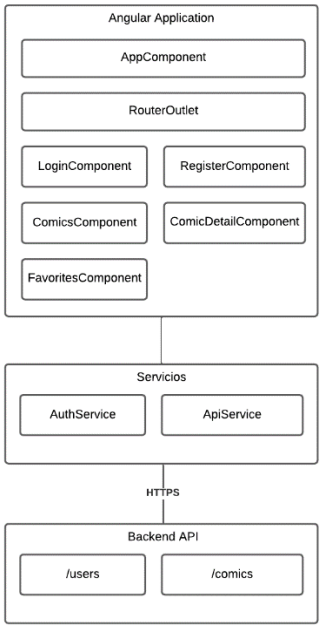

Figura 3. Diagrama arquitectura front-end

2. Base de datos:

   Se trabajó en una base de datos SQL y relacional, mediante el uso de PostgreSQL y la estructura utilizada fue la siguiente, se dejaron como valores únicos los ids, la identificación y el email del usuario:

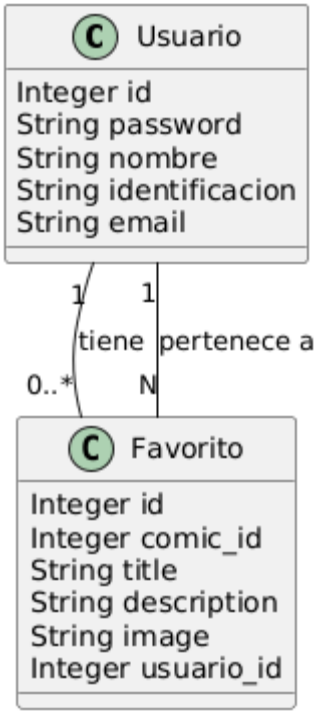

Figura 4. Diagrama UML

3. Back-end:
   - Arquitectura basada en DDD (Domain Driven Design):
     Domain Driven Design (DDD) se implementa estructurando el código en módulos claramente definidos, como models, schemas, repositories, y routers, cada uno representando contextos delimitados del dominio. Los modelos de dominio (Usuario, Favorito) y los repositorios encapsulan la lógica de acceso a datos, facilitando cambios en la persistencia sin afectar la lógica de negocio.

   - Patrones de Diseño Empleados

   a. Repository Pattern:
      Uso de repositorios (repositories/user_repository.py, repositories/comic_repository.py) para abstraer la lógica de acceso a datos.

Facilita el manejo y prueba de las operaciones de datos, separando la lógica de negocio de la lógica de acceso a datos.

b. Dependency Injection:
   Uso de la inyección de dependencias en FastAPI (Depends) para gestionar las dependencias de manera explícita y facilitar la prueba y mantenibilidad del código.
c. DTO (Data Transfer Object):
   Los modelos de Pydantic actúan como DTOs, asegurando que los datos sean validados y estructurados correctamente antes de ser procesados o enviados como respuesta.

4. Front-end:
   - Sigue una arquitectura modular y basada en componentes (se separa la página en módulos), con un enfoque en la separación de concerns y la reutilización de código. Se apoya en Angular Material para la UI, utiliza Reactive Forms para la gestión de formularios, y se comunica con el backend mediante servicios inyectados y HttpClient.

   - Patrones de Diseño Empleados:
     a. Component Pattern: Encapsulación y reutilización de la lógica de presentación.
     b. Service Pattern: Encapsulación de la lógica de negocio y la interacción con APIs externas.
     c. Dependency Injection: Gestión de dependencias y mejora de la testabilidad.
     d. Observer Pattern: Manejo de operaciones asincrónicas y reactivas con Observables.
     e. Mediator Pattern: Centralización de la lógica de enrutamiento.
     f. Facade Pattern: Interacción simplificada con el backend.
     g. Strategy Pattern: Aplicación de diferentes estrategias de validación.
     h. Singleton Pattern: Uso de servicios singleton para consistencia y eficiencia.