Documentação de Projeto

para o sistema

Ajunta

Versão 1.0

Projeto de sistema elaborado pelo aluno Gabriel Reis Lebron de Oliveira
como parte da disciplina **Projeto de Software**.

15/11/2025

Tabela de Conteúdo

Histórico de Revisões

Nome	Data	Razões para Mudança	Versão

1. Introdução

Este documento agrega: 1) a elaboração e revisão de modelos de domínio e 2) modelos de projeto para o sistema Ajunta. A referência principal para a descrição geral do problema, domínio e requisitos do sistema é o documento de especificação que descreve a visão de domínio do sistema.

2. Modelos de Usuário e Requisitos

2.1 Descrição de Atores

Usuário (Ator Principal)

O **Usuário** é o ator principal do sistema. Do ponto de vista técnico (UML), todos os casos de uso, como Autenticar-se, Gerenciar Perfil, Gerenciar Postagens e Interagir com Usuários, são iniciados por este ator.

No entanto, com base no domínio de negócio do Ajunta, o ator Usuário pode ser categorizado em três papéis ou perfis principais, que definem suas motivações e como eles utilizam a plataforma:

1.1. Artista

É o foco central da rede social. O Artista utiliza o sistema para:

- **Divulgar Trabalhos:** Publica postagens, gerencia seu portfólio e compartilha seu processo criativo.
- **Networking Profissional:** Conecta-se com outros artistas, segue-os, envia mensagens e interage com postagens (curtindo, comentando).
- **Buscar Oportunidades:** Procura ativamente e candidata-se a vagas, editais e convites publicados por Recrutadores.

1.2. Recrutador

Representa produtores culturais, empresas ou qualquer pessoa que procure contratar talentos artísticos. O Recrutador utiliza o sistema para:

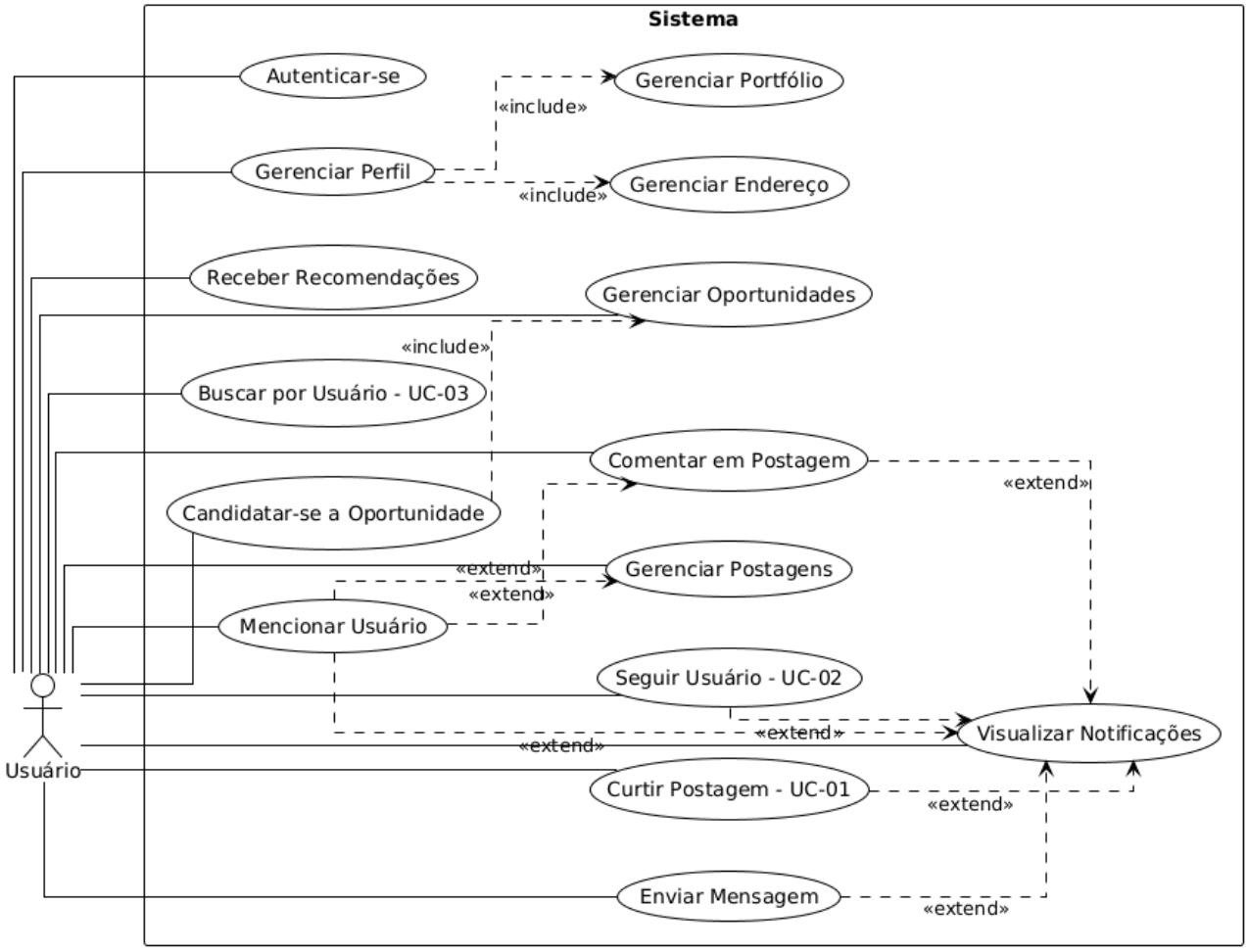
- **Descobrir Talentos:** Busca por artistas, filtrar por habilidades e analisar portfólios.
- **Publicar Oportunidades:** Criar postagens de vagas, eventos, editais ou audições.
- **Networking:** Conecta-se com artistas de interesse.

1.3. Público

Representa o consumidor de cultura local, entusiastas de arte e a comunidade em geral. O Público utiliza o sistema para:

- **Descoberta Cultural:** Segue artistas locais, acompanha seus trabalhos e descobre eventos na região.
- **Interação:** Interage com postagens e compartilha o trabalho dos artistas, ajudando a fortalecer a cena cultural.

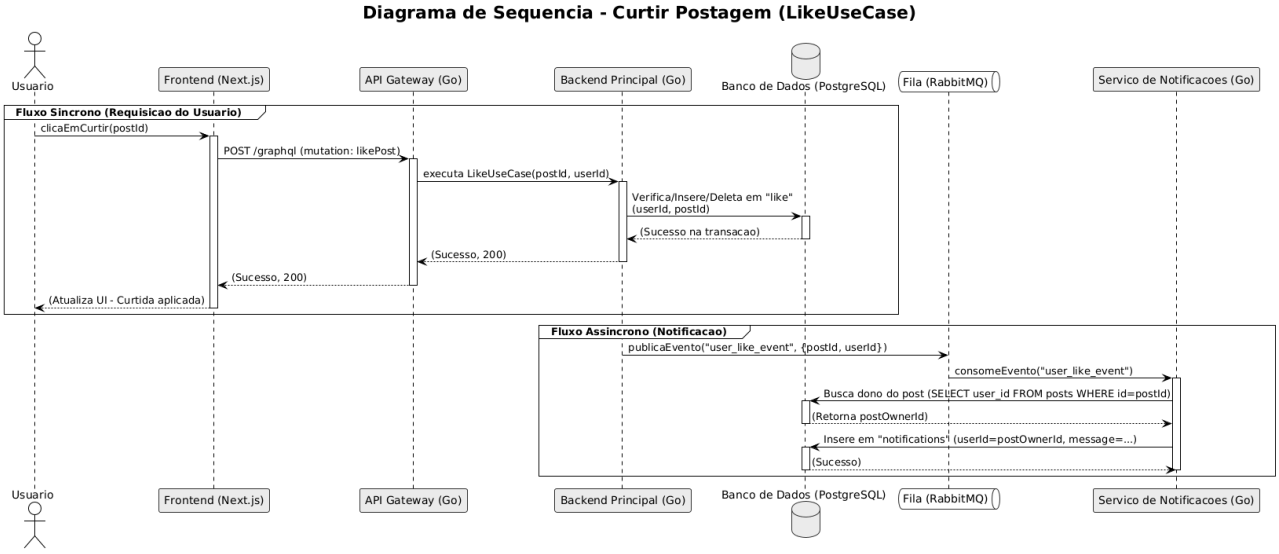
2.2 Modelo de Casos de Uso



UC-01 - Curtir Postagem **/ UC-02 -** Seguir Usuário **/ UC-03 -** Buscar por Usuário

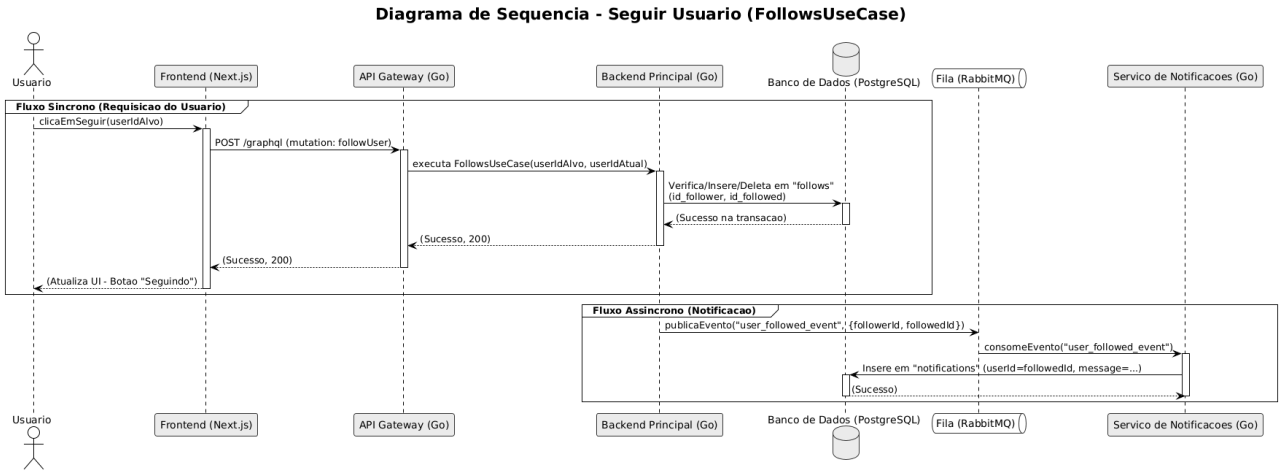
2.3 Diagrama de Sequência do Sistema

UC-01



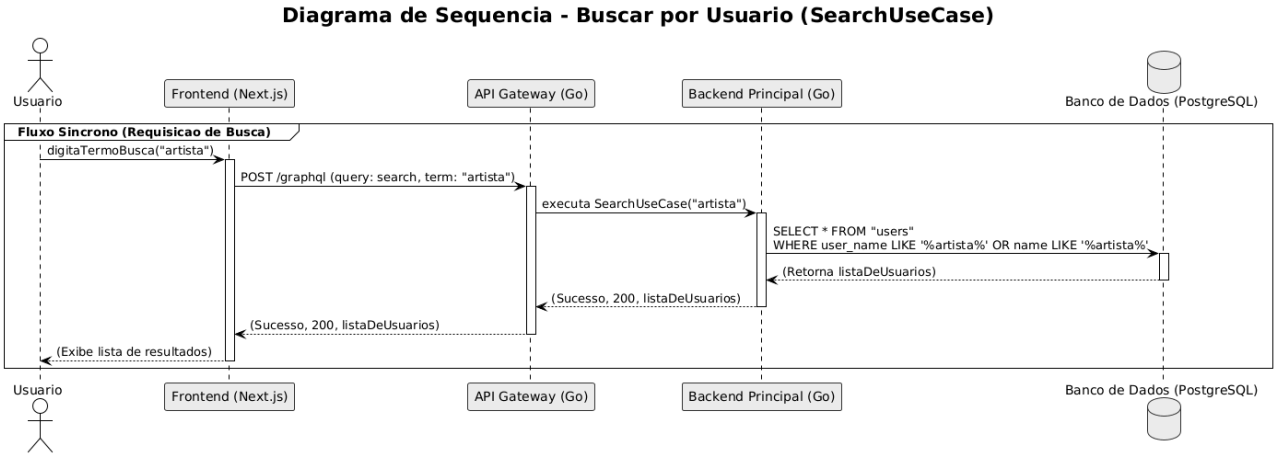
Contrato	Curtir Postagem
Operação	likePost(postId: UUID, userId: UUID)
Referências cruzadas	1- História de Usuário: Como usuário, eu quero poder curtir uma postagem para demonstrar apreciação. 2- Diagrama: Diagrama de Sequência - UC-01 3- UC Relacionado: Receber Notificação (A curtida dispara uma notificação)
Pré-condições	1- O usuário deve estar autenticado no sistema (validado pelo Gateway). 2- A postagem deve existir na tabela posts do banco de dados.
Pós-condições	1- Se a curtida não existia (Ação: Curtir): - Uma nova linha é criada na tabela like associando o userId ao postId. - O contador likes na tabela post é incrementado em 1. - Um evento user_like_event é publicado na fila (RabbitMQ) para o serviço de notificações processar. - O sistema retorna uma resposta de sucesso (HTTP 200) ao usuário. 2- Se a curtida já existia (Ação: Descurtir): - A linha correspondente na tabela like (com userId e postId) é removida. - O contador likes na tabela post é decrementado em 1. - O sistema retorna uma resposta de sucesso (HTTP 200) ao usuário.

UC-02



Contrato	Seguir Usuário
Operação	followUser(targetUserId: UUID, currentUserId: UUID)
Referências cruzadas	1- História de Usuário: Como usuário, eu quero poder seguir outros usuários para ver suas futuras postagens no meu feed. 2- Diagrama: Diagrama de Sequência - UC-02 3- UC Relacionado: Buscar por Usuário 4- UC Relacionado: Receber Notificação (A ação de seguir dispara uma notificação)
Pré-condições	1- O Usuário (currentUserId) deve estar autenticado. 2- O Usuário Alvo (targetUserId) deve existir na tabela users. 3- currentUserId e targetUserId não devem ser iguais.
Pós-condições	1- Se a relação não existia (Ação: Seguir): - Uma nova linha é criada na tabela follows (id_follower = currentUserId, id_followed = targetUserId). - O contador followingCount (seguindo) do currentUserId é incrementado. - O contador followersCount (seguidores) do targetUserId é incrementado. - Um evento user_followed_event é publicado na fila (RabbitMQ). - O sistema retorna uma resposta de sucesso (HTTP 200). 2- Se a relação já existia (Ação: Deixar de Seguir): - A linha correspondente na tabela follows é removida. - O contador followingCount do currentUserId é decrementado. - O contador followersCount do targetUserId é decrementado. - O sistema retorna uma resposta de sucesso (HTTP 200).

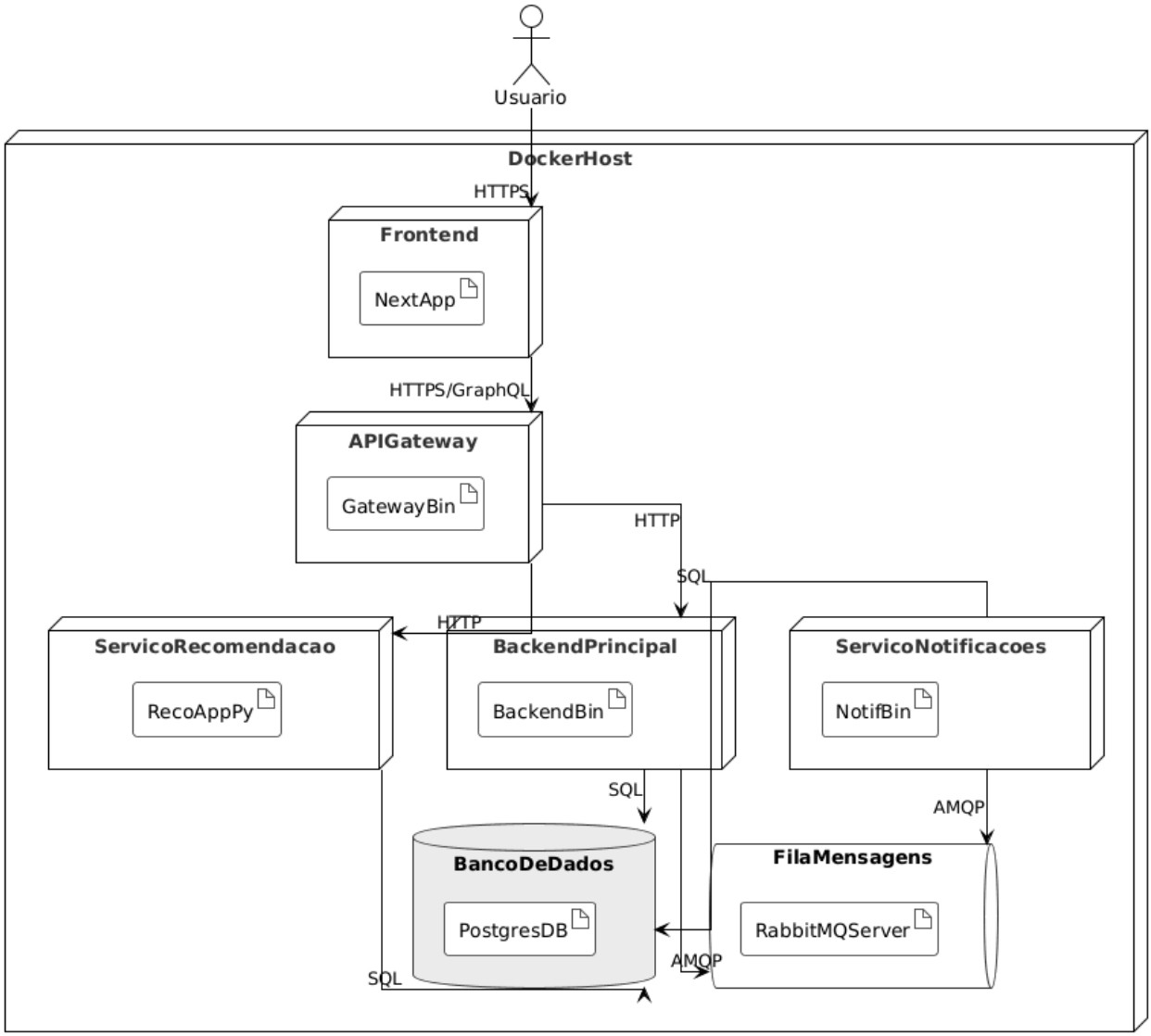
UC-03



Contrato	Buscar por Usuário
Operação	search(term: String, type: String)
Referências cruzadas	1- História de Usuário: Como usuário, eu quero buscar outros usuários para poder visitar seus perfis. 2- Diagrama: Diagrama de Sequência - UC-03 3- UC Relacionado: Visualizar Perfil de Usuário
Pré-condições	1- O Usuário deve estar autenticado. 2- O termo de busca é uma string válida e não-vazia.
Pós-condições	1- O sistema executa uma consulta (SELECT) no Banco de Dados. 2- Uma lista de resultados (podendo ser vazia) é retornada ao usuário. 3- Nenhum dado no banco de dados é alterado (operação de leitura). 4- O sistema retorna uma resposta de sucesso (HTTP 200) contendo a lista.

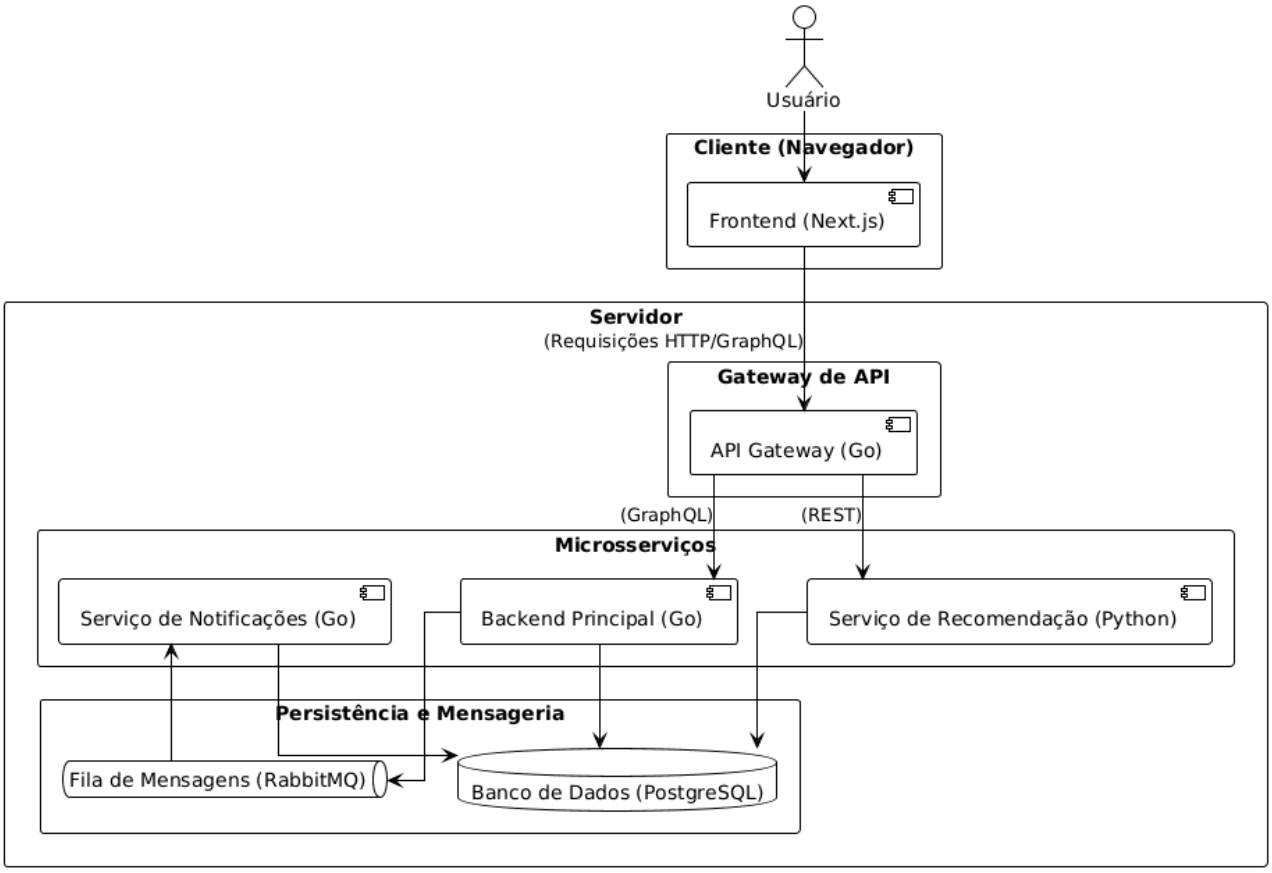
3. Modelos de Projeto

3.1 Arquitetura

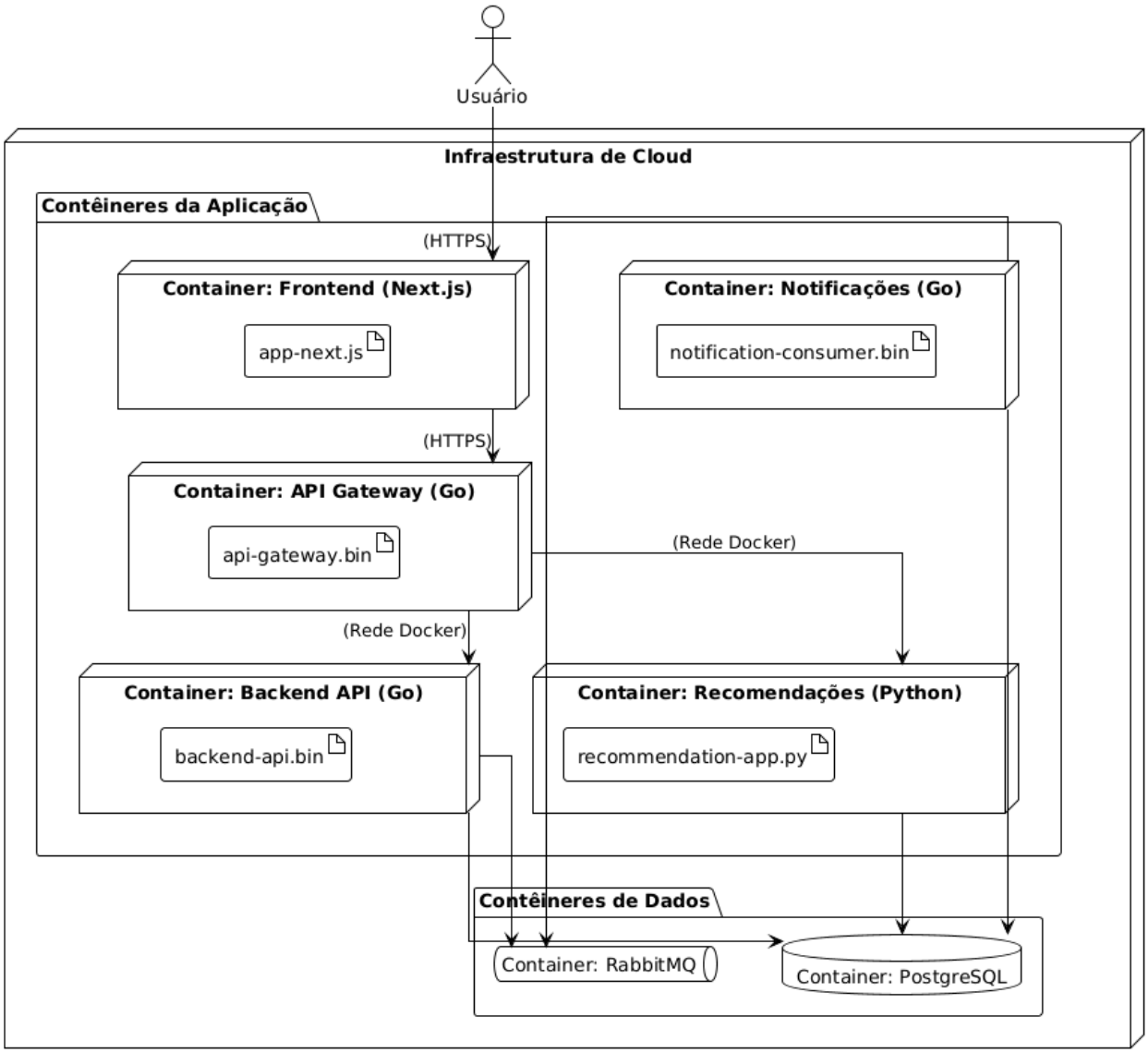


3.2 Diagrama de Componentes e Implantação.

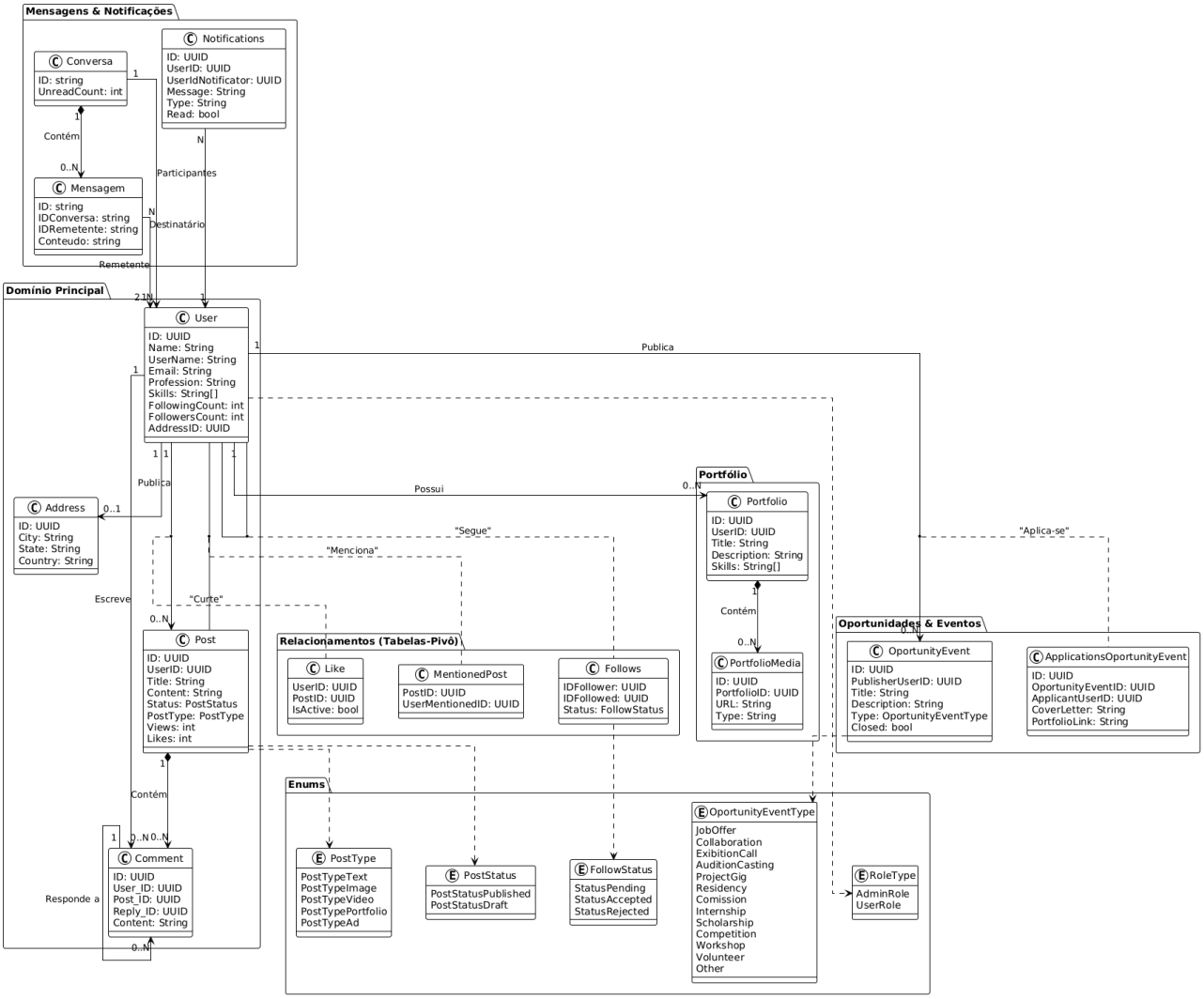
3.2.1 Diagrama de Componentes



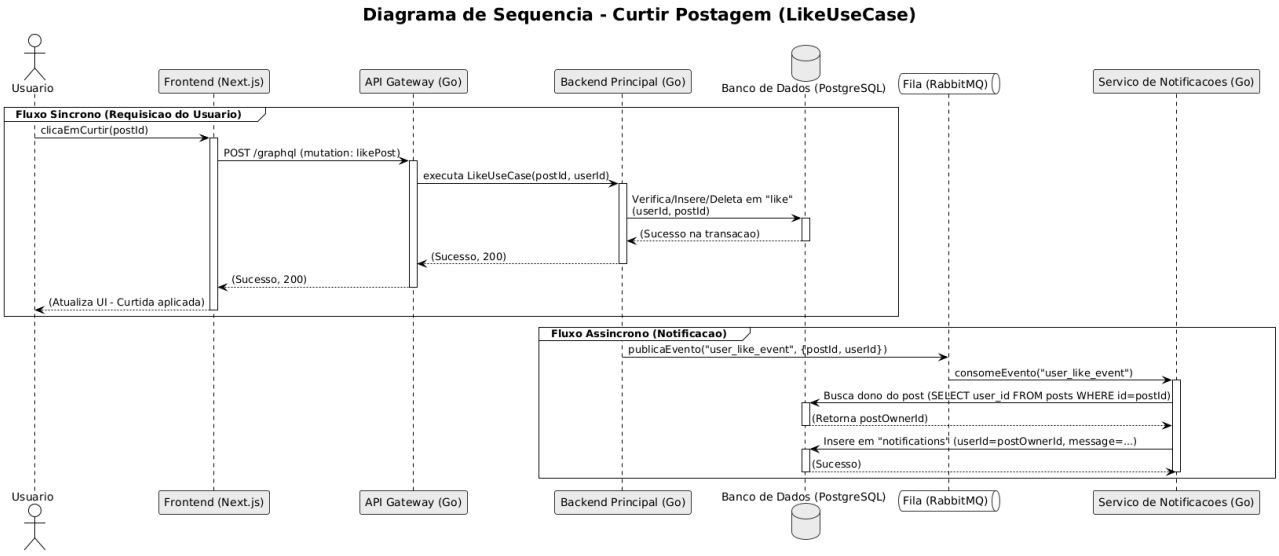
3.2.2 Diagrama de Implantação

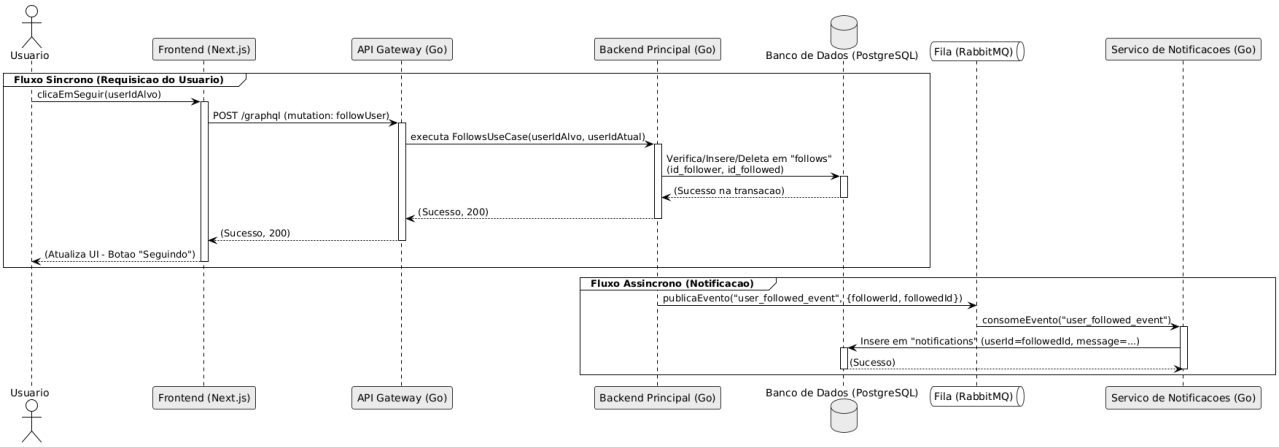


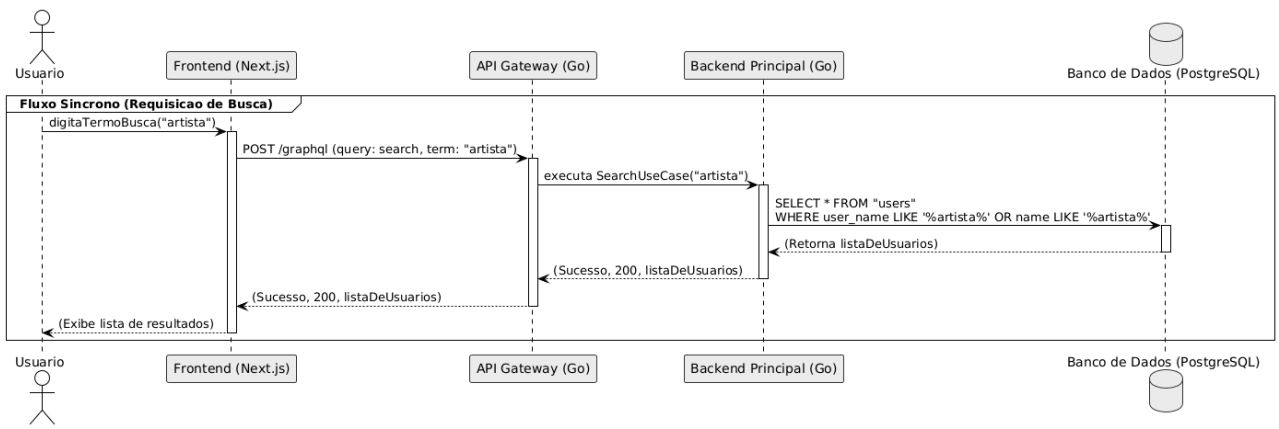
3.3 Diagrama de Classes



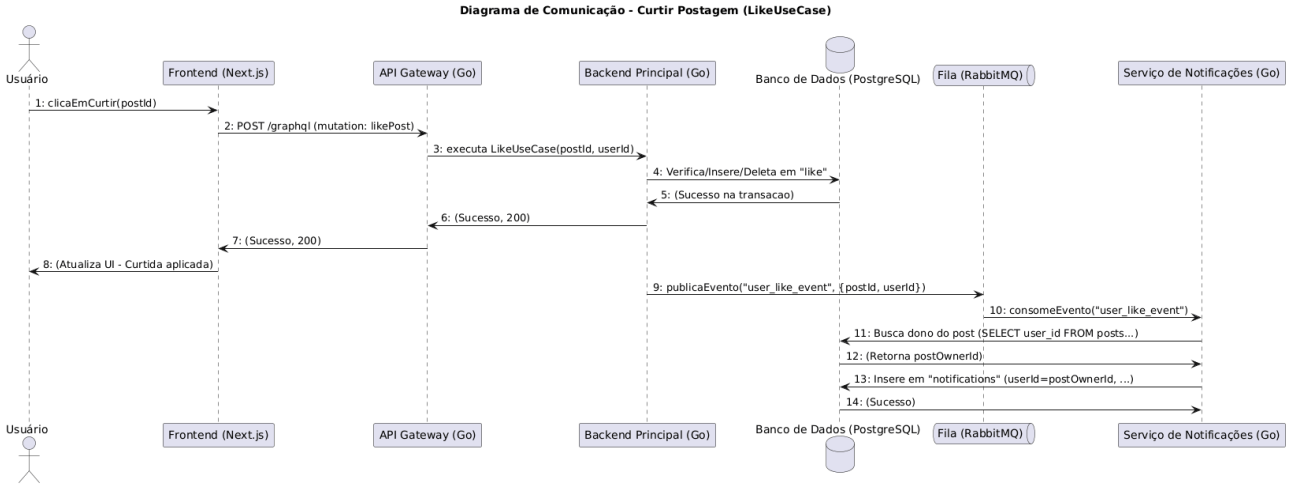
3.3 Diagramas de Sequência

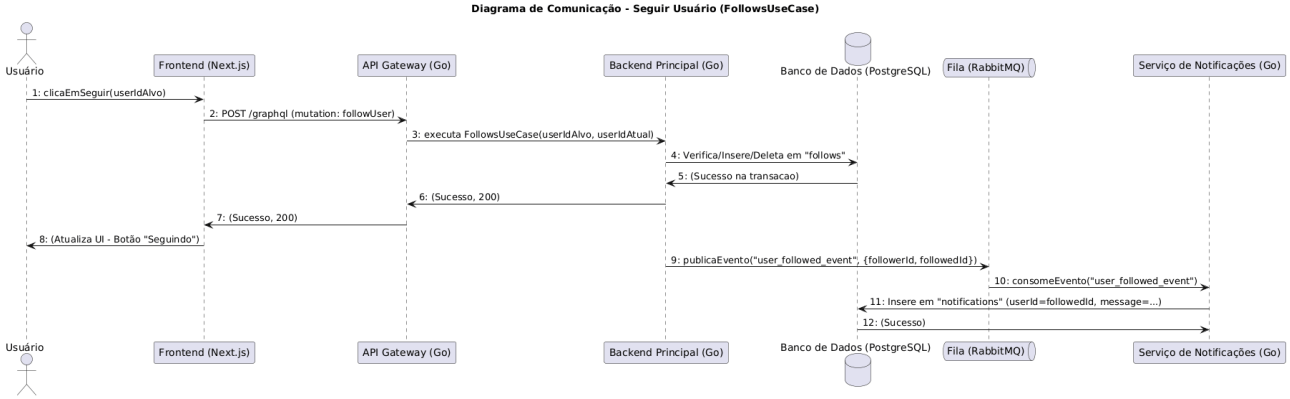


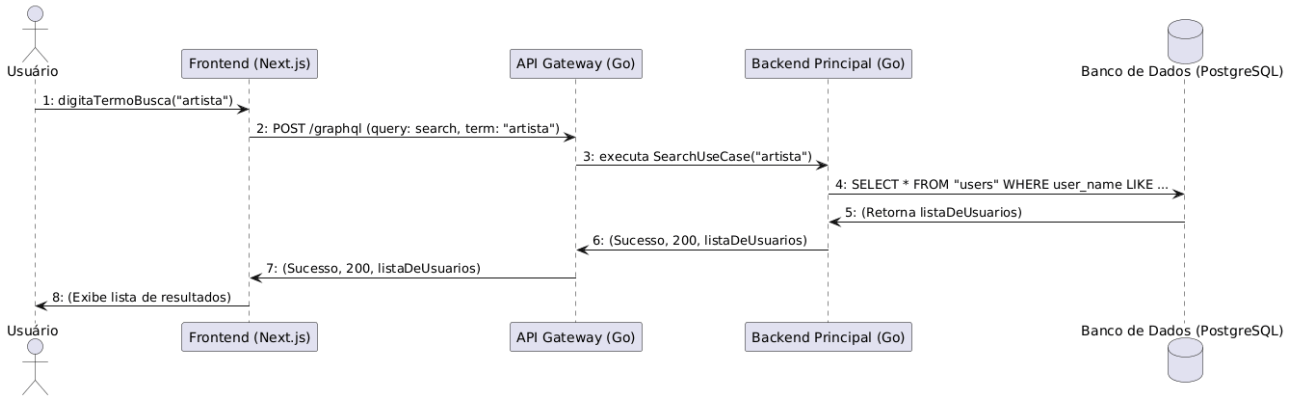




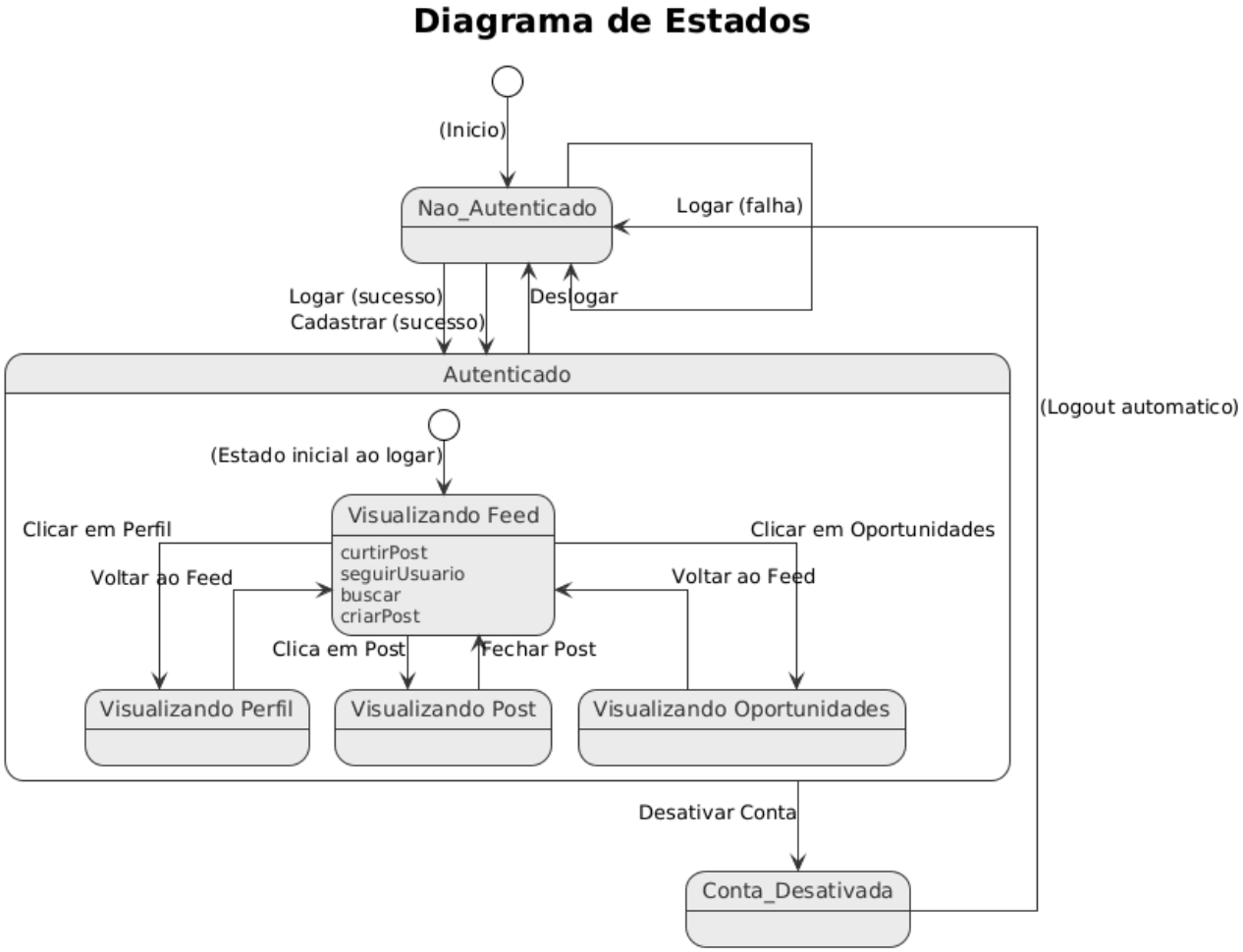
3.4 Diagramas de Comunicação



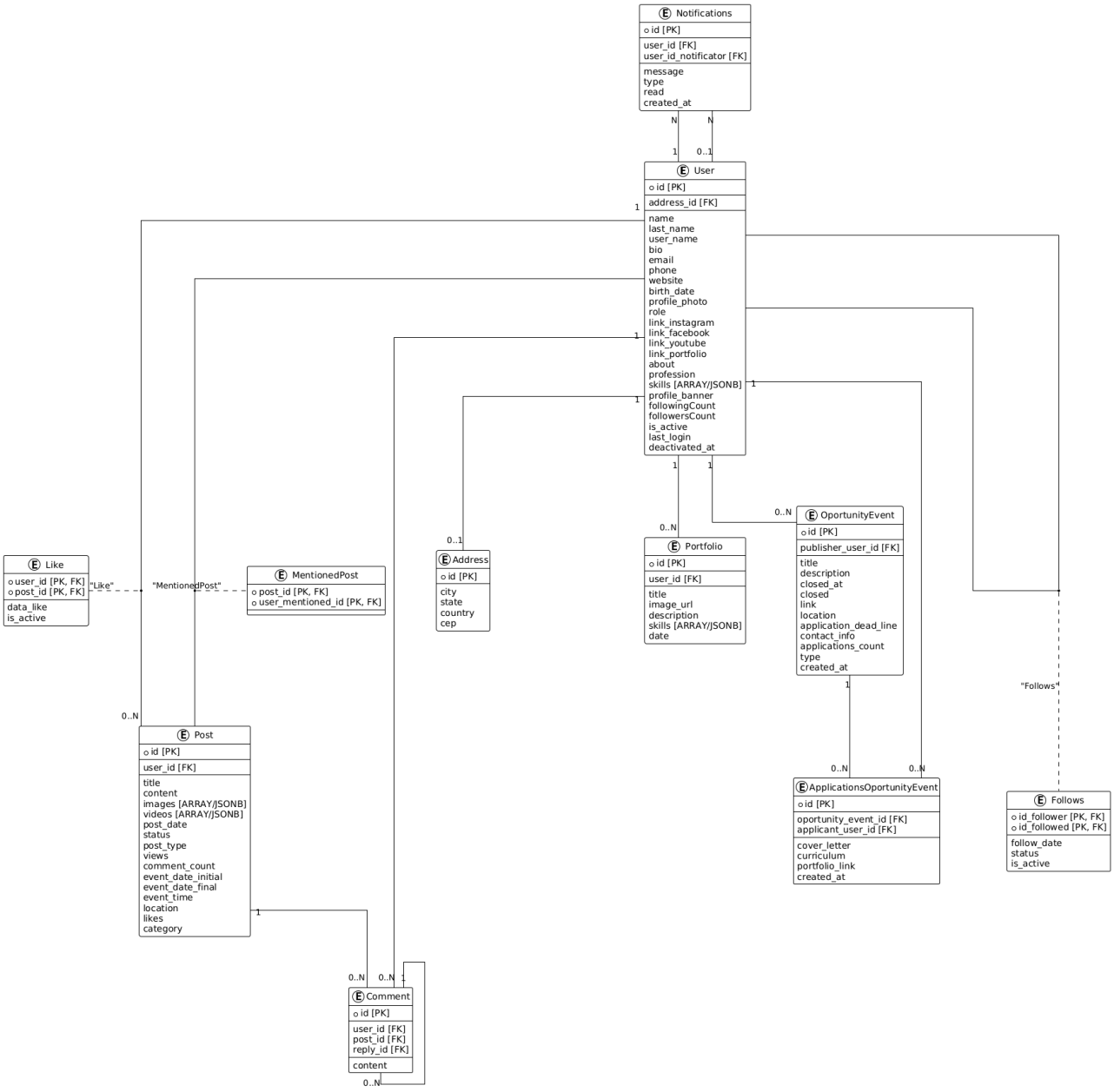




3.5 Diagrama de Estados



4. Modelos de Dados



Estratégia de Mapeamento Objeto-Relacional (ORM)

Este documento explica como o código Go (objetos/structs) se conecta e salva informações em um banco de dados PostgreSQL (tabelas relacionais). Esse processo é conhecido como Mapeamento Objeto-Relacional (ORM).

Existem quatro estratégias principais usadas:

1. Mapeamento Básico (1-para-1)

É a regra mais simples: cada struct principal no Go vira uma tabela no banco de dados.

- struct User (Go) ➔ TABLE users (SQL)
- struct Post (Go) ➔ TABLE posts (SQL)

2. Relacionamentos "Um-para-Muitos" (1-N)

Usado quando um item "possui" vários outros (ex: 1 Usuário tem muitos Posts).

- Como funciona: A tabela "filha" (ex: posts) recebe uma Chave Estrangeira (FK) que aponta para o "pai" (ex: user_id na tabela posts aponta para a id na tabela users).

3. Relacionamentos "Muitos-para-Muitos" (N-N)

Usado quando muitos itens se relacionam com muitos outros (ex: 1 Usuário segue muitos Usuários).

- Como funciona: É criada uma Tabela de Junção (ou "tabela pivô") no meio. Por exemplo, uma tabela follows é criada apenas para conectar o id_follower (ID do seguidor) ao id_followed (ID do seguido).

4. Mapeamento de Arrays/Slices (Tipos Especiais)

Campos em Go que são listas/arrays (como Skills []string ou Images []string) não existem no SQL tradicional. No PostgreSQL, isso é resolvido de duas formas:

1. Usando o tipo ARRAY (Recomendado): O PostgreSQL permite criar colunas que são listas (ex: skills TEXT[]). Isso é limpo, eficiente e permite fazer buscas dentro da lista.
2. Usando o tipo JSONB: O array do Go é convertido em um texto JSON (ex: ["design", "go"]) e salvo em uma coluna JSONB. Isso é extremamente flexível se a estrutura desses dados precisar mudar no futuro.