Relatório de Modelagem de Banco de Dados do Projeto Rede Social Connect

- Integrantes: Fellipe Araújo, Isadora Matsunaga, Erick Godinho, Daniel Amorim
- Universidade Católica de Brasília - UCB

- Disciplina: Desenvolvimento Web
- Função deste relatório: Análise de requisitos, documentação de decisões de modelagem de dados e de design da API

- ***Link do repositório da parte de Lab. de Dados no Github:*** Projeto Rede Social

1. Introdução

Este relatório apresenta a **modelagem de banco de dados** e as **decisões de design da API** para um sistema de **rede social aberta**, desenvolvido como projeto final da disciplina. Serão descritos os requisitos, o processo de análise e justificativas para as escolhas de entidades, atributos, relacionamentos e endpoints.

2. Análise de Requisitos

Com base no enunciado da atividade e nas discussões de sala, identificamos as seguintes funcionalidades principais:

Requisito	Descrição
Cadastro de Usuários	Registro seguro, autenticação JWT e perfil público.
Postagens e Interações	Criação de posts, comentários, respostas e avaliações.
Grupos e Comunidades	Criação/participação em grupos, roles admin/member.
Mensagens Privadas	Envio e recepção de mensagens com status (sent/received/read).
Tags e Interesses	Associação de até 5 tags, busca de usuários por tag.

3. Modelo Conceitual (Contextualização)

No modelo conceitual, o foco é mapear as entidades principais (`Usuário`, `Postagem`, `Comentário`, `Grupo`, `Mensagem`, `Tag`) e seus relacionamentos sem detalhar atributos ou tipos, garantindo que todas as funcionalidades do sistema estejam contempladas.

Exemplo:

- `Usuário` relacionado a `Postagem` (1:N)
- `Usuário` relacionado a `Grupo` (N:N)
- `Postagem` relacionada a `Comentário` (1:N)
- `Usuário` relacionado a `Tag` (N:N)

Esse modelo garante a visão geral dos dados, assegurando que o sistema suporte múltiplas interações sociais, compartilhamento e comunicação.

4. Modelo Lógico (Contextualização)

No modelo lógico, já definimos os atributos, as chaves primárias e estrangeiras, as tabelas associativas para N:N e aplicamos regras para garantir integridade referencial.

- **Definição de PKs e FKs:**
  Cada entidade tem chave primária única (normalmente `id` auto-incrementável). Tabelas associativas têm PK composta pelas duas FKs para garantir unicidade.
- **Tipos de dados:**
  São definidos conforme o SGBD (MySQL), por exemplo, `INT` para IDs, `VARCHAR` para textos curtos, `TEXT` para conteúdo maior, `DATETIME` para datas.
- **Normalização:**
  O modelo é normalizado para evitar redundância e inconsistência, permitindo manutenção e evolução facilitadas.

5. Modelo Físico (Contextualização)

O modelo físico é a implementação concreta no banco MySQL:

- Criação das tabelas com colunas e tipos específicos.
- Definição das chaves primárias (`PRIMARY KEY`).
- Definição das chaves estrangeiras (`FOREIGN KEY`) para garantir integridade referencial e evitar exclusões ou atualizações que quebrariam vínculos importantes.
- Índices para melhorar desempenho em consultas frequentes, especialmente nas tabelas associativas e colunas de busca.
- Uso de restrições (por exemplo, `NOT NULL`) para garantir qualidade dos dados.

6. Justificativas de Design de Banco

1. **Chave Surrogada (`id`)**: Facilita joins e mantém consistência.
2. **Enums**: Em `role` e `status` para controlar valores no nível do schema.
3. **Tabelas de Associação (M:N)**: `GroupMember`, `UserTag` para relacionamentos flexíveis.
4. **Integridade Referencial**: FKs com ações `ON DELETE SET NULL` ou `CASCADE` conforme necessidade.

7. Decisões de Design da API e Controllers

Para cada conjunto de endpoints, a implementação nos controllers reflete decisões de negócio e de modelagem:

7.1 Grupos (`groupController.js`)

- **createGroup**: Cria `Group` e registra o criador como `admin` via `GroupMember` (role fixo);
- **getAllGroups**: Ordenação por data para feed de grupos recentes;
- **joinGroup**: Valida existência do grupo e membership, evita duplicidade (HTTP 409) e protege recursos (somente membros podem acessar posts).

7.2 Mensagens Privadas (`messageController.js`)

- **sendMessage**: Verifica existência do destinatário e grava com status padrão 'sent';
- **getConversation**: Busca bidirecional com `Op.or` e inclui metadados de sender/recipient para UI clara;
- Ordenação ascendente para histórico completo.

7.3 Posts e Interações (`postController.js`)

- **createPost**: Permite posts públicos ou em grupos (valida `GroupMember` para acesso);
- **getAllPosts**: Árvore de inclusão de `User` e `Group` para feed com autor e contexto;
- **addComment**: Cria `Comment` simples com FK para `Post` e `User`;
- **ratePost**: Lógica para toggling de avaliações: remover voto igual, alterar ou criar novo, garantindo idempotência.

7.4 Tags e Interesses (`tagController.js`)

- **addTagsToUser**: Uso de transação para apagar antigas e inserir novas tags em lote;
- `findOrCreate` para criar tags dinamicamente e manter unicidade; • Limitação de até 5 itens e rollback em caso de falha para consistência.

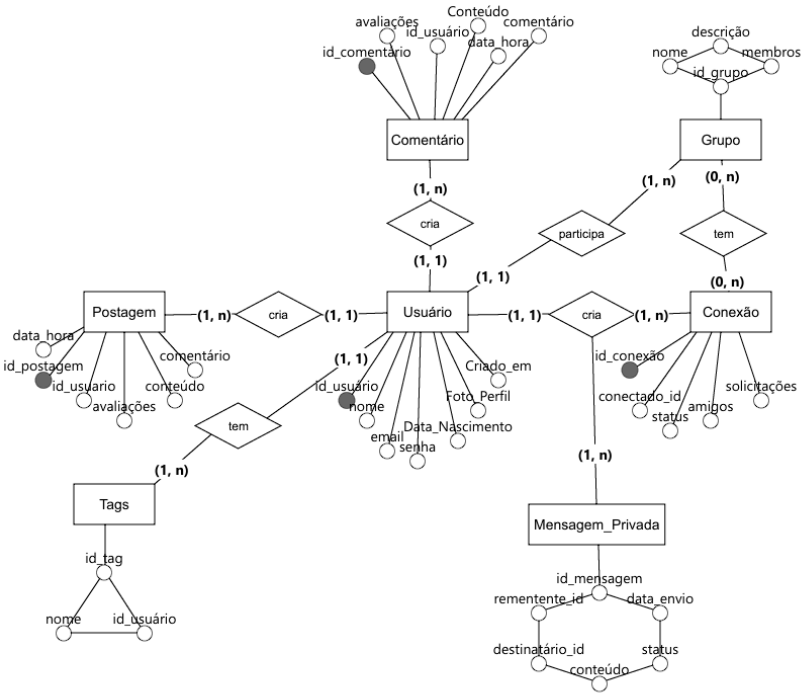
7.5 Usuários e Autenticação (`userController.js`)

- **register**: Hash da senha com `bcrypt` e omissão de `password_hash` na resposta;
- **login**: Validação de credenciais e geração de JWT com `expiresIn` balanceado (8h);
- **getProfile**: Exclui campos sensíveis (`password_hash`);
- **findUsersByTag**: Join M:N via `include` do Sequelize para busca por tag, retornando apenas atributos públicos.

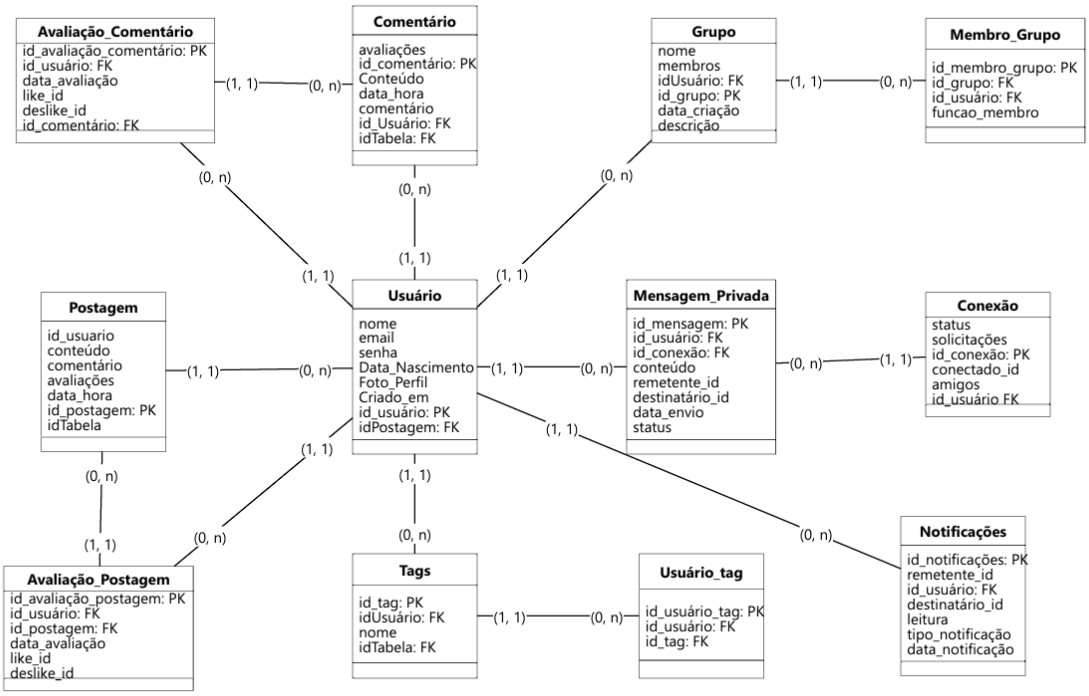
8. Conexão entre Modelagem e API

- **Consistência**: Cada relação no banco (FK) é refletida em `include` nos controllers para retorno de dados completos;
- **Segurança**: Middleware de autenticação extrai `req.userId` de JWT, validando acesso;
- **Escalabilidade**: Padrões M:N via tabelas de junção facilitam consultas e futura indexação.

9. Modelo Conceitual



10. Modelo lógico



11. Modelo Físico

- ***Script SQL:***

-- Tabela de usuários

CREATE TABLE `users` (`id` int NOT NULL AUTO_INCREMENT, `username` varchar(255) NOT NULL, `email` varchar(255) NOT NULL, `password_hash` varchar(255) NOT NULL, `birth_date` date NOT NULL, `profile_picture_url` varchar(255) DEFAULT NULL, `createdAt` datetime NOT NULL, `updatedAt` datetime NOT NULL, PRIMARY KEY (`id`), UNIQUE KEY `username` (`username`), UNIQUE KEY `email` (`email`)) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela de grupos

CREATE TABLE `groups` (`id` int NOT NULL AUTO_INCREMENT, `name` varchar(255) NOT NULL, `description` text, `createdAt` datetime NOT NULL, `updatedAt` datetime NOT NULL, PRIMARY KEY (`id`), UNIQUE KEY `name` (`name`)) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela de membros de grupo

CREATE TABLE `group_members` (`role` enum('member','admin') NOT NULL DEFAULT 'member', `groupId` int NOT NULL, `userId` int NOT NULL, PRIMARY KEY (`groupId`,`userId`), KEY `userId` (`userId`), CONSTRAINT `group_members_ibfk_1` FOREIGN KEY (`groupId`) REFERENCES `groups` (`id`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `group_members_ibfk_2` FOREIGN KEY (`userId`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela de postagens

CREATE TABLE `posts` (`id` int NOT NULL AUTO_INCREMENT, `content_type` enum('text','image','video') NOT NULL, `content` text, `media_url` varchar(255) DEFAULT NULL, `createdAt` datetime NOT NULL, `updatedAt` datetime NOT NULL, `groupId` int DEFAULT NULL, `userId` int DEFAULT NULL, PRIMARY KEY (`id`), KEY `groupId` (`groupId`), KEY `userId` (`userId`), CONSTRAINT `posts_ibfk_1` FOREIGN KEY (`groupId`) REFERENCES `groups` (`id`) ON DELETE SET NULL ON UPDATE CASCADE, CONSTRAINT `posts_ibfk_2` FOREIGN KEY (`userId`) REFERENCES

users (id) ON DELETE SET NULL ON UPDATE CASCADE) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela de comentários

CREATE TABLE comments (id int NOT NULL AUTO_INCREMENT, content text NOT NULL, parentId int DEFAULT NULL, createdAt datetime NOT NULL, updatedAt datetime NOT NULL, postId int DEFAULT NULL, userId int DEFAULT NULL, PRIMARY KEY (id), KEY parentId (parentId), KEY postId (postId), KEY userId (userId), CONSTRAINT comments_ibfk_1 FOREIGN KEY (parentId) REFERENCES comments (id) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT comments_ibfk_2 FOREIGN KEY (postId) REFERENCES posts (id) ON DELETE SET NULL ON UPDATE CASCADE, CONSTRAINT comments_ibfk_3 FOREIGN KEY (userId) REFERENCES users (id) ON DELETE SET NULL ON UPDATE CASCADE) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela de conexões (seguidores e seguindo)

CREATE TABLE connections (id int NOT NULL AUTO_INCREMENT, followerId int DEFAULT NULL, followingId int DEFAULT NULL, PRIMARY KEY (id), UNIQUE KEY connections_followingId_followerId_unique (followerId,followingId), KEY followingId (followingId), CONSTRAINT connections_ibfk_1 FOREIGN KEY (followerId) REFERENCES users (id) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT connections_ibfk_2 FOREIGN KEY (followingId) REFERENCES users (id) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela de mensagens privadas

CREATE TABLE private_messages (id int NOT NULL AUTO_INCREMENT, content text NOT NULL, status enum('sent','received','read') NOT NULL DEFAULT 'sent', createdAt datetime NOT NULL, updatedAt datetime NOT NULL, sender_id int DEFAULT NULL, recipient_id int DEFAULT NULL, PRIMARY KEY (id), KEY sender_id (sender_id), KEY recipient_id (recipient_id), CONSTRAINT private_messages_ibfk_1 FOREIGN KEY (sender_id) REFERENCES users (id) ON DELETE SET NULL ON UPDATE CASCADE, CONSTRAINT private_messages_ibfk_2 FOREIGN KEY (recipient_id) REFERENCES users (id) ON DELETE SET NULL ON UPDATE CASCADE) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela de notificações

CREATE TABLE notifications (id int NOT NULL AUTO_INCREMENT, recipientId int NOT NULL, senderId int NOT NULL, type enum('follow','like','comment','message') NOT NULL, targetId int DEFAULT NULL, targetType varchar(255) DEFAULT NULL, read tinyint(1) NOT NULL DEFAULT '0', createdAt datetime NOT NULL, updatedAt datetime NOT NULL, PRIMARY KEY (id), KEY recipientId (recipientId), KEY senderId (senderId), CONSTRAINT notifications_ibfk_1 FOREIGN KEY (recipientId) REFERENCES users (id) ON UPDATE CASCADE, CONSTRAINT notifications_ibfk_2 FOREIGN KEY (senderId) REFERENCES users (id) ON UPDATE CASCADE) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela de tags

CREATE TABLE tags (id int NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL, createdAt datetime NOT NULL, updatedAt datetime NOT NULL, PRIMARY KEY (id), UNIQUE KEY name (name)) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela associativa user_tags

CREATE TABLE user_tags (tagId int NOT NULL, userId int NOT NULL, PRIMARY KEY (tagId,userId), KEY userId (userId), CONSTRAINT user_tags_ibfk_1 FOREIGN KEY (tagId) REFERENCES tags (id) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT user_tags_ibfk_2 FOREIGN KEY (userId) REFERENCES users (id) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Tabela de avaliações (likes e dislikes) para posts e comentários

CREATE TABLE ratings (id int NOT NULL AUTO_INCREMENT, rating_type enum('positive','negative') NOT NULL, createdAt datetime NOT NULL, updatedAt datetime NOT NULL, commentId int DEFAULT NULL, postId int DEFAULT NULL, userId int DEFAULT NULL, PRIMARY KEY (id), KEY commentId (commentId), KEY postId (postId), KEY userId (userId), CONSTRAINT ratings_ibfk_1 FOREIGN KEY (commentId) REFERENCES comments (id) ON DELETE SET NULL ON UPDATE CASCADE, CONSTRAINT ratings_ibfk_2 FOREIGN KEY (postId) REFERENCES posts (id) ON DELETE SET NULL ON UPDATE CASCADE, CONSTRAINT ratings_ibfk_3 FOREIGN KEY (userId) REFERENCES users (id) ON DELETE SET NULL ON UPDATE CASCADE) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;