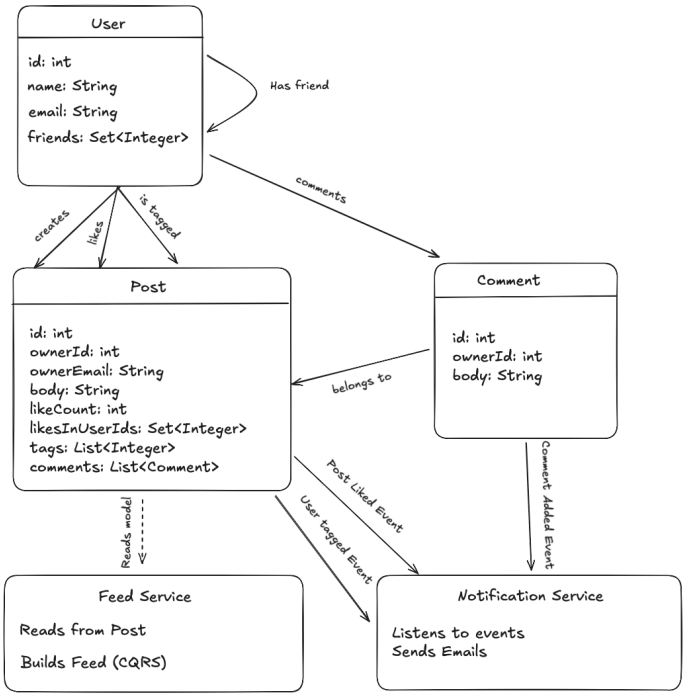
Design: Social Network API using Microservices

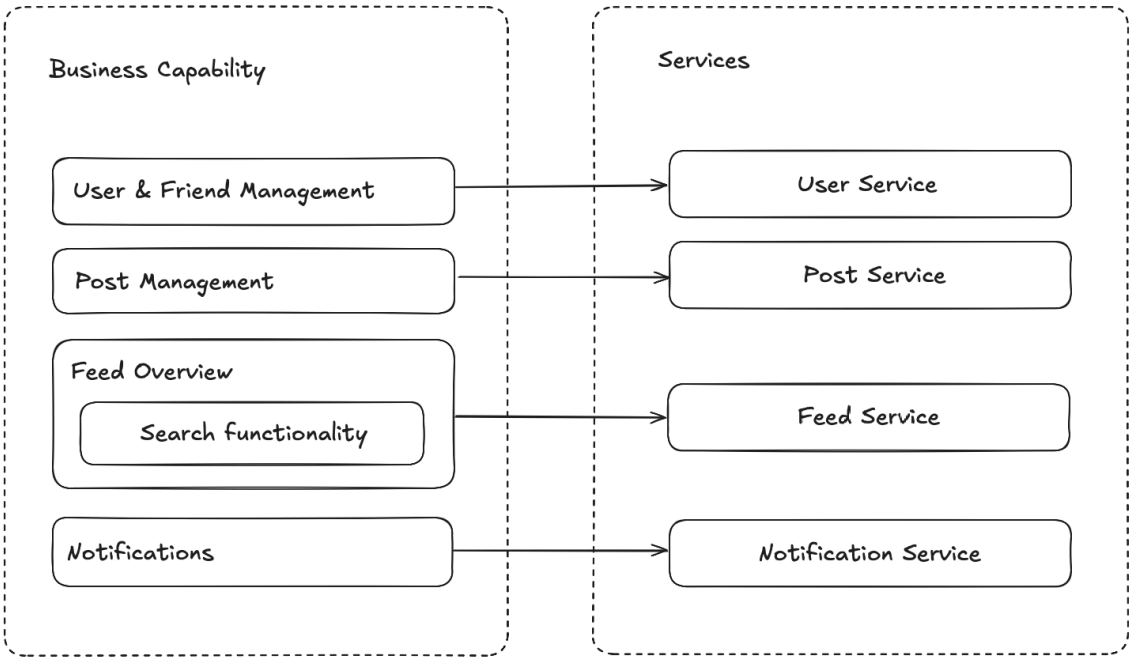
High Level Domain Model



System Operations

Actor	Story	Command/Query	Description
User	Get User Feed	getUserFeed()	Fetches a list of posts for the user's feed.
User	Search User Feed	searchUserFeed()	Fetches a filtered list of posts from the user's feed based on a search query.
Guest	Create User	createUser()	Creates a new user in the system.
User	Add Friend	addFriend()	Adds a friendship between two users.
User	Create Post	createPost()	Creates a new post and optionally tags others. Tagged users receive an email.
User	Remove Post	removePost()	Deletes an existing post created by the user.
User	Like Post	likePost()	Adds a like to a specific post. Creator of post gets notified by email.
User	Remove Like	removeLike()	Removes a like from a specific post.
User	Comment On Post	commentOnPost()	Adds a comment to a specific post. Creator of post gets notified by email.
User	Remove Comment	removeComment()	Deletes a specific comment made by the user on a post.

Decomposition



Service	Operators	Collaborators
User Service	createUser() addFriend()	- -
Post Service	createPost() removePost() commentOnPost() removeComment() likePost() removeLike()	User Service, Notification Service - User Service, Notification Service - User Service, Notification Service -
Feed Service	getUserFeed() searchUserFeed()	- -

API Definitions

API Gateway

Serves as the entry point for all client requests, handling routing and load balancing.

- **Routing**
  - Directs requests to appropriate microservices (e.g., Post Service, User Service, Feed Service).
  - Example: /users → User Service; /posts → Post Service.

User Service

Manages users and friendships.

- **Endpoints**
  - Create User
    - POST /users
    - Creates a user in the system.
  - Get All Users (extra)
    - GET /users
    - Retrieves all users.
  - Get User By ID (extra)
    - GET /users/{id}
    - Retrieves a user by ID.
  - Add Friend
    - POST /users/addFriend
    - Adds a friend to a user.

Post Service

Manages posts, likes, comments, using saga patterns

- **Endpoints**
  - Get All Posts (extra)
    - GET /posts
    - Retrieves all posts.
  - Get Posts by User (extra)
    - GET /posts/user/{userId}
    - Retrieves posts created by a specific user.
  - Create Post
    - POST /posts/saga
    - Creates a new post and tags users using the saga pattern.
  - Remove Post
    - POST /posts/remove/{postId}/user/{userId}
    - Deletes a user's post.
  - Like Post
    - POST /posts/like
    - Adds a like to a post.
  - Unlike Post
    - POST /posts/unlike
    - Removes a like from a post.
  - Comment On Post
    - POST /posts/comment/{postId}
    - Adds a comment to a post
  - Remove Comment
    - POST /posts/removeComment
    - Removes a comment from a post.

Feed Service

Manages user feeds and provides posts tailored to users, including search functionality.

- **Endpoints**
    - Get User Feed
        - GET /feed/{userId}
        - Retrieves a list of posts for a specific user consisting of own posts, friends posts and post in which you are tagged.
    - Search User Feed
        - GET /feed/{userId}/search
        - Retrieves posts for a user that match a search query than can consist of the text body of a post, a friend, tags.

Notification Service

Responsible for sending email notifications based on system events. It is event-driven and reacts to actions from other services, such as the Post Service.

- **Listens to events like:**
    - Post Liked
    - User Tagged in Post
    - Comment Added
- **After receiving events:**
    - Sends personalized emails to users about relevant activities in the system (e.g., "You were tagged in a post" or "Your post received a like").

Service Registry

Enables service discovery in my system. It keeps track of all available microservices and their locations ensuring they can communicate effectively.

Sagas

Create Post Saga (post-service)

1. **Start**:
    - Sends a ValidateOwnerCommand to the **user-Service**.
2. **Owner Validation**:
    - Listens for OwnerValidatedEvent.
    - If valid:
        - Updates the post status to VALIDATING_TAGGED_USERS and stores the owner's email.
        - Sends a ValidateTaggedUsersCommand to the **user-Service**.
    - If invalid, rolls back the post.
3. **Tagged Users Validation**:
    - Listens for TaggedUsersValidatedEvent.
    - If valid:
        - Updates the post status to ACCEPTED.
        - Adds valid tagged user IDs to the post.
        - Sends email notifications.
        - Emits a PostCreatedEvent for Feed-Service.
    - If invalid, rolls back the post.
4. **Rollback**:
    - Deletes the post if validation fails and the post is not ACCEPTED.

Comment on Post Saga (post-service)

1. **Start**:
    - Sends a ValidatePostCommentUserCommand to the **User-Service** to validate the commenting user.
2. **User Validation**:
    - Listens for the PostCommentUserValidatedEvent.
    - If valid:
        - Adds the comment to the post.
        - Sends an email notification to the post owner.
        - Emits a PostUpdatedEvent for the Feed-Service (CQRS).
    - If invalid:
        - Logs a warning that the user does not exist and doesn't add the comment.

Like Post Saga (post-service)

1. **Start**:
    - Sends a ValidatePostLikeUserCommand to the **User-Service** to validate the user liking the post.
2. **User Validation**:
    - Listens for the PostLikeUserValidatedEvent.
    - If valid:
        - Adds a like to the post.
        - Emits a PostUpdatedEvent to update the Feed-Service (CQRS).
        - Sends an email notification to the post owner.
    - If invalid:
        - Logs a warning that the user does not exist and doesn't add the like.