Backend System Documentation

GitHub Repository URL

GitHub Repository: **https://github.com/vashikovich/dealls-test**

Functional & Non-functional Requirements

Functional Requirements:

1. Users can sign up and log in to the system.

2. Users can swipe left (pass) or swipe right (like) on profiles.

3. Users have a daily swipe quota (with a premium feature to remove the limit).

4. Users can view a list of candidate profiles, filtered by their preferences.

5. Users can update their profiles with name, birth date, bio, and gender.

Non-functional Requirements:

1. The system should be highly available with minimal downtime.

2. The system should be scalable to support a growing number of users.

3. The system should be secure, especially when handling user data and payments.

Tech Stack & Reasoning

1. NestJS - For building scalable and maintainable APIs.

2. TypeORM - For easy database interaction with an ORM.

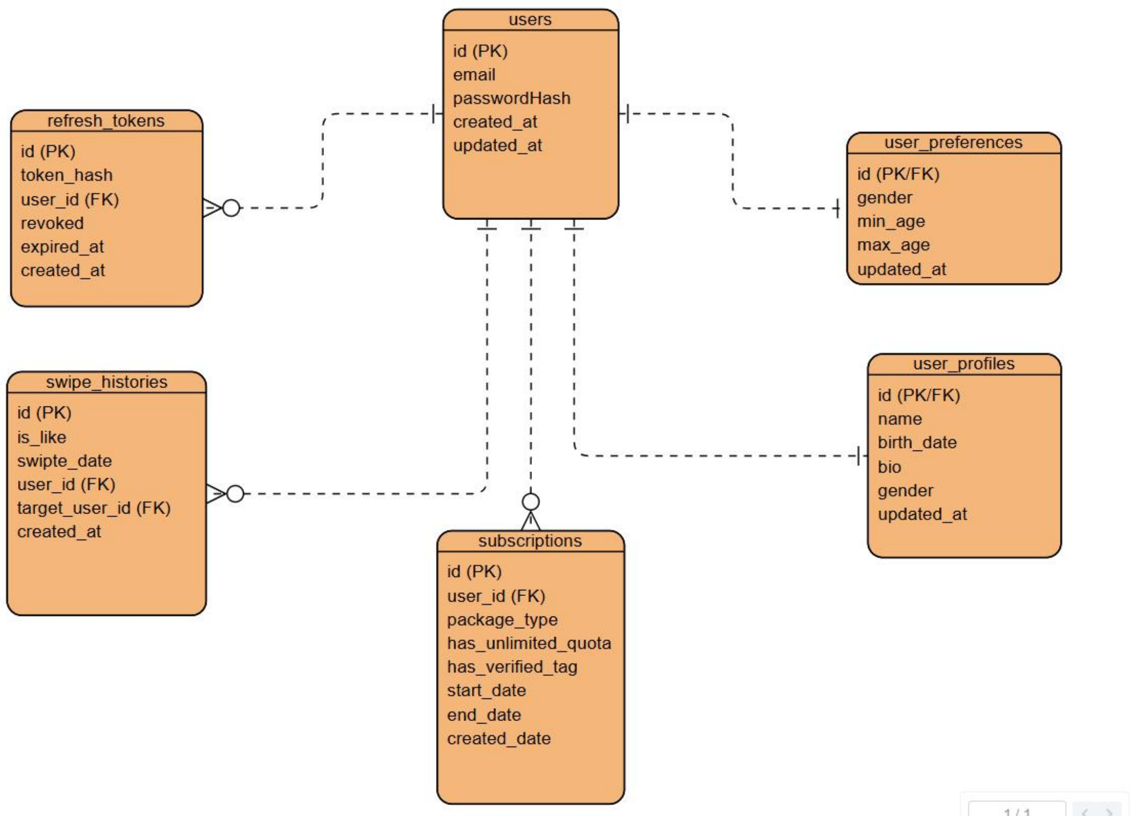
3. PostgreSQL - For a robust, production-ready database solution.

4. JWT - For stateless user authentication and secure token management.

6. Jest - For unit and integration testing of the backend.

System Design

ERD Diagram



List of Test Cases

Auth Module

`/auth/signup(POST)`

- should create a new user and return tokens
- should throw a conflict error for duplicate email

`/auth/login(POST)`

- should log in an existing user and return tokens
- should throw an unauthorized error for invalid credentials

`/auth/refresh(POST)`

- should generate new tokens using a valid refresh token
- should throw an unauthorized error for an invalid refresh token
- should return UnauthorizedException for expired refresh token

Auth DTO (Validations)

`LoginDto`

1. should validate successfully with valid data
2. should fail if email is undefined
3. should fail if email is null
4. should fail if email is an empty string
5. should fail if email is invalid
6. should fail if password is undefined
7. should fail if password is null
8. should fail if password is an empty string
9. should fail if email and password are both missing

`RefreshTokenDto`

1. should validate successfully with valid data
2. should fail if refreshToken is missing
3. should fail if refreshToken is an empty string

`SignupDto`

1. should validate successfully with valid data
2. should fail if email is undefined
3. should fail if email is null
4. should fail if email is an empty string
5. should fail if password is undefined
6. should fail if password is null
7. should fail if email is invalid
8. should fail if password is less than 8 characters
9. should fail if password does not contain at least one uppercase letter
10. should fail if password does not contain at least one lowercase letter
11. should fail if password does not contain at least one number or special character
12. should fail if email and password are both missing

Matcher Module

`/matcher/profile (PUT)`

1. should update user profile successfully
2. should return 400 for invalid input data
3. should return 404 if user does not exist
4. should return 401 if user is not authenticated

`/matcher/candidates (GET)`

1. should return a list of candidates based on user preferences
2. should exclude users specified in skipUserIds
3. should return an empty list if no candidates match preferences
4. should return 401 if user is not authenticated
5. should handle limiting result

`/matcher/remaining-quota (GET)`

1. should return the correct remaining swipe quota
2. should return 0 if the quota is exhausted
3. should return 401 if user is not authenticated

`/matcher/swipe (POST)`

1. should record a swipe successfully
2. should return a match if both users like each other
3. should not allow swiping the same user twice in a day
4. should return 400 for invalid input data
5. should return 401 if user is not authenticated
6. should return 403 if swipe quota is exhausted

`/matcher/match-history (GET)`

1. should return the match history of the user
2. should return an empty list if no matches exist
3. should return 401 if user is not authenticated

`/matcher/swipe-history (GET)`

1. should return the swipe history of the user
2. should filter history by date range if provided
3. should return an empty list if no history exists
4. should return 401 if user is not authenticated

Matcher DTO (Validations)

`GetCandidatesDto`

1. should allow valid data
2. should allow undefined skipUserIds
3. should fail when skipUserIds is not an array

4. should fail when skipUserIds contains non-string elements
5. should fail when skipUserIds contains invalid UUIDs

`SwipeDto`

1. should allow valid data
2. should fail when targetUserId is missing
3. should fail when targetUserId is null
4. should fail when targetUserId is not a UUID
5. should fail when isLike is missing
6. should fail when isLike is null
7. should fail when isLike is not a boolean

`UpdateProfileDto`

1. should allow valid data
2. should fail when name is missing
3. should fail when name is empty
4. should fail when birthDate is missing
5. should fail when birthDate is not a valid date
6. should allow bio to be optional
7. should fail when bio is not a string
8. should fail when gender is missing
9. should fail when gender is not a valid enum value