UsersService

September 4, 2026

UsersService

Kind: Service

Source: `atloria-monorepo/apps/parser-orchestrator/test/fixtures/sample-projects/fullstack-nextjs-nestjs/backend/src/users/users.service.ts`

`UsersService` encapsulates user-related backend operations for the NestJS application. It provides methods for retrieving, creating, updating, and deleting users, as well as accessing user relationships such as posts and followers.

Methods

Method	Signature	Returns
`findAll`	`findAll(paginationQuery: PaginationQueryDto)`	`unknown`
`findOne`	`findOne(id: string)`	`unknown`
`findByEmail`	`findByEmail(email: string)`	`unknown`
`create`	`create(createUserDto: CreateUserDto)`	`unknown`
`update`	`update(id: string, updateUserDto: UpdateUserDto)`	`unknown`
`remove`	`remove(id: string)`	`unknown`
`getUserPosts`	`getUserPosts(userId: string)`	`unknown`
`getFollowers`	`getFollowers(userId: string)`	`unknown`

Dependencies

- `PrismaService`
- `EventEmitter2`

Where it refuses work

- `UsersService` stops the work with `NotFoundException` when `!user`.
- `UsersService` stops the work with `ConflictException` when `existingUser` — “User with this email already exists”.

Diagram

```
sequenceDiagram
    participant Controller as UsersController
    participant Service as UsersService
    participant Database as Data Layer

    Controller->>Service: findOne(userId)
    Service->>Database: Query user by ID
    Database-->>Service: User record
    Service-->>Controller: User response

    Controller->>Service: getUserPosts(userId)
    Service->>Database: Query user's posts
    Database-->>Service: Posts collection
    Service-->>Controller: Posts response
```

Usage

```
import { Injectable } from '@nestjs/common';
import { UsersService } from './users.service';

@Injectable()
export class ProfileService {
  constructor(private readonly usersService: UsersService) {}

  async getProfile(email: string) {
    const user = await this.usersService.findByEmail(email);

    if (!user) {
      return null;
    }

    const [posts, followers] = await Promise.all([
      this.usersService.getUserPosts(user.id),
      this.usersService.getFollowers(user.id),
    ]);

    return {
      user,
      posts,
      followers,
    };
```

```
  }
}
```

AI Coding Instructions

- Keep user persistence and relationship queries inside `UsersService`; controllers should delegate business operations to the service.
- Use `findByEmail()` before user creation when email uniqueness must be enforced.
- Ensure `findOne()` and `findByEmail()` handle missing users consistently with the application's error-handling conventions.
- Prefer `getUserPosts()` and `getFollowers()` over duplicating relation-query logic in other services.
- Validate and sanitize create/update payloads through DTOs before calling `create()` or `update()`.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `eventemitter2`

Referenced By

- `UsersController` (DEPENDS_ON)
- `UsersModule` (MODULE_PROVIDES)
- `UsersModule` (MODULE_EXPORTS)