User

September 4, 2026

User

Kind: Database Model

Source: `atloria-monorepo/apps/parser-orchestrator/test/fixtures/sample-projects/fullstack-nextjs-nestjs/backend/prisma/schema.prisma`

User model - represents registered users in the system

The `User` Prisma model represents registered users in the application. It stores identity, authentication, profile, authorization, and lifecycle fields used by the NestJS backend and its Prisma data-access layer. User roles are defined by the `Role` enum and should be used to enforce authorization consistently.

Fields

Field	Type	Required	Key
`id`	`String`	✓	PK
`email`	`String`	✓	unique
`password`	`String`	✓	
`name`	`String`	✓	
`role`	`Role`	✓	
`createdAt`	`DateTime`	✓	
`updatedAt`	`DateTime`	✓	

Diagram

```
erDiagram
    USER {
        String id PK
        String email UK
        String password
```

```
        String name
        Role role
        DateTime createdAt
        DateTime updatedAt
    }
```

Usage

```typescript
import { PrismaClient, Role } from '@prisma/client';
import * as bcrypt from 'bcrypt';

const prisma = new PrismaClient();

async function createUser() {
  const hashedPassword = await bcrypt.hash('secure-password', 12);

  const user = await prisma.user.create({
    data: {
      email: 'jane@example.com',
      name: 'Jane Doe',
      password: hashedPassword,
      role: Role.USER,
    },
    select: {
      id: true,
      email: true,
      name: true,
      role: true,
      createdAt: true,
    },
  });

  return user;
}

async function findUserByEmail(email: string) {
  return prisma.user.findUnique({
    where: { email },
  });
}
```

AI Coding Instructions

- Hash passwords before writing them to `User.password`; never store or log plaintext passwords.
- Query users by `email` with `findUnique` when authenticating or checking for an existing account.

- Use the generated `Role` enum rather than hard-coded role strings when creating or updating users.
- Exclude the `password` field from API responses by using Prisma `select` clauses or DTO serialization.
- Treat `createdAt` and `updatedAt` as system-managed lifecycle fields; do not manually overwrite them unless performing a controlled migration.

Relationships

- HAS_ONE → `Profile`
- HAS_MANY → `Post`
- HAS_MANY → `Comment`
- HAS_MANY → `Like`
- HAS_MANY → `Follow`
- HAS_MANY → `Follow`

Referenced By

- `Profile` (BELONGS_TO)
- `Post` (BELONGS_TO)
- `Comment` (BELONGS_TO)
- `Like` (BELONGS_TO)
- `Follow` (BELONGS_TO)
- `Follow` (BELONGS_TO)