Loyihaning tavsifi:

Siz onlayn kitob do'koni uchun backend API yaratishingiz kerak. Ushbu API orqali foydalanuvchilar kitoblarni ko'rish, qidirish, savatga qo'shish va buyurtma berish imkoniyatiga ega bo'ladi. Bunda foydalanuvchilar oddiy ro'yxatdan o'tish va login qilish jarayonlarini o'tkazishlari kerak. Har bir foydalanuvchi o'zining savatchasini boshqarishi, buyurtmalarini kuzatishi mumkin bo'ladi. Shuningdek, admin foydalanuvchilar kitob qo'shishi, yangilashi va o'chirishi mumkin.

Talablar:

1. **Foydalanuvchilar**:
    - Ro'yxatdan o'tish va login qilish.
    - JWT orqali autentifikatsiya qilish.
    - Foydalanuvchi o'z profilini ko'rishi va yangilashi mumkin.
2. **Kitoblar**:
    - Foydalanuvchilar kitoblar ro'yxatini ko'rishi, qidirish va filtr bo'yicha kitoblarni ajratib ko'rishlari mumkin (masalan, narx bo'yicha, muallif bo'yicha).
    - Har bir kitob haqida batafsil ma'lumot olish mumkin (nomi, muallifi, narxi, tavsifi, mavjudligi va hokazo).
    - Faqat admin foydalanuvchilar yangi kitob qo'shishi, o'zgartirishi va o'chirishi mumkin.
3. **Savat**:
    - Har bir foydalanuvchi o'z savatchasiga kitoblar qo'shishi va ularni ko'rib chiqishi mumkin.
    - Savatchadan kitoblarni o'chirish yoki ularning sonini o'zgartirish imkoniyati mavjud.
4. **Buyurtmalar**:
    - Foydalanuvchilar savatchadagi kitoblarini buyurtma qilishlari mumkin.
    - Har bir buyurtma haqida ma'lumot (buyurtma raqami, sana, kitoblar ro'yxati, umumiy narx) saqlanadi.
    - Foydalanuvchilar o'z buyurtmalarini ko'rishlari mumkin.
5. **Admin panel**:
    - Faqat admin foydalanuvchilar kirishi mumkin.
    - Adminlar kitoblar bilan ishlashlari mumkin (qo'shish, yangilash, o'chirish).

Funksional talablarga qo'shimchalar:

- Foydalanuvchilar JWT token orqali himoyalangan resurslarga kirishi kerak.
- Kitoblar qidiruvida paginatsiya bo'lishi kerak.
- API uchun Swagger hujjatlari tayyorlangan bo'lishi kerak.
- Kitob va foydalanuvchi ma'lumotlari uchun validatsiya qoidalari va xatoliklarni qaytarish mexanizmini joriy etish.

Texnik talablar:

- **NestJS**: Framework sifatida foydalaning.
- **JWT**: Foydalanuvchi autentifikatsiyasi uchun.
- **TypeORM yoki Mongoose**: Ma'lumotlar bazasi bilan ishlash uchun.
- **PostgreSQL yoki MongoDB**: Ma'lumotlar bazasi sifatida.
- **Swagger**: API hujjatlarini yaratish uchun.

API-ning asosiy endpointlari:

- **/auth/register** – Foydalanuvchi ro'yxatdan o'tkazish.
- **/auth/login** – Foydalanuvchi login qilish.
- **/books** – Kitoblar ro'yxati va qidiruv.
- **/books/**
  – Kitob haqida batafsil ma'lumot olish.
- **/cart** – Foydalanuvchi savatchasi.
- **/cart/add** – Savatchaga kitob qo'shish.
- **/cart/remove/**
  – Savatchadan kitobni o'chirish.
- **/order** – Buyurtma berish.
- **/order/history** – Foydalanuvchi buyurtma tarixi.
- **/admin/books** – Kitoblar qo'shish, yangilash, o'chirish (faqat adminlar uchun).

1. Foydalanuvchi (User) Entity

typescript
Copy code

```
import { Entity, Column,
PrimaryGeneratedColumn, OneToMany } from
'typeorm';
import { Order } from './order.entity';
import { Cart } from './cart.entity';

@Entity()
export class User {
```

```
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @Column({ unique: true })
  email: string;

  @Column()
  password: string;

  @Column({ default: 'user' })
  role: string; // 'user' yoki 'admin' roli

  @OneToMany(() => Order, (order) =>
order.user)
  orders: Order[];

  @OneToMany(() => Cart, (cart) =>
cart.user)
  cart: Cart[];
}
```

2. Kitob (Book) Entity

typescript

Copy code

```
import { Entity, Column,
PrimaryGeneratedColumn, ManyToOne } from
'typeorm';

@Entity()
export class Book {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  title: string;

  @Column()
  author: string;

  @Column('decimal')
  price: number;

  @Column({ default: true })
```

```
  inStock: boolean;

  @Column({ type: 'text' })
  description: string;

  @Column()
  category: string;
}
```

3. Savatcha (Cart) Entity

typescript
Copy code

```
import { Entity, PrimaryGeneratedColumn,
ManyToOne, ManyToMany, JoinTable } from
'typeorm';
import { User } from './user.entity';
import { Book } from './book.entity';

@Entity()
export class Cart {
  @PrimaryGeneratedColumn()
  id: number;
```

```
  @ManyToOne(() => User, (user) =>
user.cart)
  user: User;

  @ManyToMany(() => Book)
  @JoinTable()
  books: Book[];
}
```

4. Buyurtma (Order) Entity

typescript
Copy code

```
import { Entity, PrimaryGeneratedColumn,
ManyToOne, ManyToMany, JoinTable, Column }
from 'typeorm';
import { User } from './user.entity';
import { Book } from './book.entity';

@Entity()
export class Order {
  @PrimaryGeneratedColumn()
  id: number;
```

```
  @ManyToOne(() => User, (user) =>
user.orders)
  user: User;

  @ManyToMany(() => Book)
  @JoinTable()
  books: Book[];

  @Column()
  totalPrice: number;

  @Column({ default: 'pending' })
  status: string; // 'pending', 'completed',
'canceled'
}
```