حتما! در اینجا متن توضیحات پروژه **Task Management API به صورت آماده برای کپی‌کردن در فایل متنی (مانند Word یا Google Docs) آورده شده است. شما می‌توانید آن را کپی کرده و در فایل دلخواه خود قرار دهید و سپس به فرمت PDF تبدیل کنید.**

 # **Task Management API با **NestJS****

1. **نصب پیش‌نیازها**
برای شروع، باید محیط توسعه را آماده کنیم.

1.1 نصب Node.js
ابتدا باید **Node.js را روی سیستم نصب کنید. برای نصب آن، از دستورات زیر در ترمینال استفاده کنید:**
```

```
**sudo apt update**
**sudo apt install nodejs**
**sudo apt install npm**
```

```

**### 1.2 نصب NestJS CLI**
**برای راحت‌تر کردن فرآیند ایجاد و مدیریت پروژه‌ها با استفاده از **NestJS**، باید **NestJS CLI** را نصب کنید:**
```

```
**npm i -g @nestjs/cli**
```

```

**### 1.3 ایجاد پروژه جدید NestJS**
**از **NestJS CLI** برای ایجاد پروژه جدید استفاده می‌کنیم:**
```

```
**nest new task-management-api**
```

```

**این دستور یک پروژه جدید به نام `task-management-api` ایجاد می‌کند که شامل ساختار اولیه پروژه و فایل‌های مورد نیاز است.**

**## 2. **نصب وابستگی‌ها****
**در فایل `package.json`، وابستگی‌هایی را برای پروژه معرفی می‌کنیم که شامل **JWT**، **SQLite**، **TypeORM**، **NestJS** و سایر وابستگی‌های مورد نیاز هستند.**

**برای نصب این وابستگی‌ها دستور زیر را اجرا کنید:**
```
```

```
npm install @nestjs/jwt passport passport-jwt bcryptjs reflect-metadata rxjs sqlite3 typeorm
npm install --save-dev @nestjs/cli @nestjs/schematics @nestjs/testing eslint prettier jest ts-jest supertest
```

این وابستگی‌ها به شرح زیر هستند:

- **@nestjs/common, @nestjs/core**: اجزای اصلی NestJS برای ساخت API.
- **typeorm, sqlite3**: برای مدیریت پایگاه داده SQLite.
- **@nestjs/jwt, passport, passport-jwt, bcryptjs**: برای احراز هویت و استفاده از JWT.
- **reflect-metadata**: برای پشتیبانی از دکوریتورها در TypeScript.
- **rxjs**: برای کار با عملیات غیرهمزمان.
- **jest, supertest**: برای نوشتن تست‌های واحد و تست‌های API.

3. **تنظیمات پایگاه داده**
در ابتدا، باید اتصال به پایگاه داده را تنظیم کنیم. در این پروژه از **SQLite** به عنوان پایگاه داده استفاده کرده‌ایم و از **TypeORM** برای ارتباط با پایگاه داده استفاده می‌کنیم.

در فایل `app.module.ts`، کد زیر را برای اتصال به پایگاه داده SQLite اضافه می‌کنیم:

```typescript
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { TaskModule } from './task/task.module';
import { Task } from './task/task.entity';

@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'sqlite',
      database: 'tasks.db', // فایل پایگاه داده SQLite
      entities: [Task], // مدل تسک‌ها
      synchronize: true, // همگام‌سازی خودکار پایگاه داده
    }),
    TaskModule, // ماژول تسک‌ها
  ],
})
```

```
export class AppModule {}
```

4. **تعریف مدل Task**
مدل `Task` برای ذخیره و مدیریت تسک‌ها در پایگاه داده است. این مدل در فایل `task.entity.ts` تعریف می‌شود.

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

@Entity() // این کلاس را به یک موجودیت (Entity) در TypeORM تبدیل می‌کند.
export class Task {
  @PrimaryGeneratedColumn() // ایجاد یک ستون ID به صورت خودکار و با مقدار افزایشی
  id: number;

  @Column() // ستون عنوان تسک
  title: string;

  @Column() // ستون توضیحات تسک
  description: string;

  @Column({ default: false }) // ستون وضعیت تکمیل تسک، پیش‌فرض false
  completed: boolean;
}
```

5. **ساخت ماژول، سرویس و کنترلر Task**

5.1 **ماژول Task**
ماژول `Task` برای مدیریت تسک‌ها در پروژه استفاده می‌شود. در فایل `task.module.ts`، ماژول را تعریف می‌کنیم که شامل سرویس و کنترلر `Task` است.

```typescript
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Task } from './task.entity';
import { TaskService } from './task.service';
import { TaskController } from './task.controller';
```

```
@Module({
  imports: [TypeOrmModule.forFeature([Task])], // افزودن مدل Task به ماژول
  providers: [TaskService], // افزودن سرویس به ماژول
  controllers: [TaskController], // افزودن کنترلر به ماژول
})
export class TaskModule {}
```

5.2 **سرویس Task**

سرویس `TaskService` شامل منطق اصلی برنامه برای مدیریت تسک‌ها است، از جمله ایجاد، خواندن، به‌روزرسانی و حذف تسک‌ها. این سرویس از **TypeORM** برای تعامل با پایگاه داده استفاده می‌کند.

```typescript
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Task } from './task.entity';

@Injectable()
export class TaskService {
  constructor(
    @InjectRepository(Task) // این دکوریتور برای تزریق مخزن Task استفاده می‌شود.
    private taskRepository: Repository<Task>,
  ) {}

  // ایجاد تسک جدید
  createTask(title: string, description: string): Promise<Task> {
    const task = this.taskRepository.create({ title, description });
    return this.taskRepository.save(task); // ذخیره تسک در پایگاه داده
  }

  // دریافت همه تسک‌ها
  getAllTasks(): Promise<Task[]> {
    return this.taskRepository.find(); // بازیابی تمام تسک‌ها
  }

  // دریافت تسک بر اساس ID
```

```
  getTaskById(id: number): Promise<Task> {
    return this.taskRepository.findOneBy({ id }); // جستجوی تسک با ID مشخص
  }

  // به‌روزرسانی وضعیت تسک
  updateTask(id: number, completed: boolean): Promise<Task> {
    return this.taskRepository.save({ id, completed }); // به‌روزرسانی وضعیت تکمیل تسک
  }

  // حذف تسک
  deleteTask(id: number): Promise<void> {
    return this.taskRepository.delete(id).then(() => {}); // حذف تسک از پایگاه داده
  }
}
```

5.3 **کنترلر Task**

کنترلر `TaskController` مسئول مدیریت درخواست‌های HTTP است. این کنترلر با استفاده از سرویس `TaskService` درخواست‌های CRUD (ایجاد، خواندن، به‌روزرسانی، حذف) را مدیریت می‌کند.

```typescript
import { Controller, Get, Post, Body, Param, Put, Delete } from '@nestjs/common';
import { TaskService } from './task.service';
import { Task } from './task.entity';

@Controller('tasks') // مسیر روت برای تسک‌ها
export class TaskController {
  constructor(private readonly taskService: TaskService) {}

  // ایجاد تسک جدید
  @Post()
  create(@Body() body: { title: string; description: string }): Promise<Task> {
    return this.taskService.createTask(body.title, body.description);
  }

  // دریافت تمام تسک‌ها
  @Get()
```

```
  getAll(): Promise<Task[]> {
    return this.taskService.getAllTasks();
  }

  // دریافت تسک خاص بر اساس ID
  @Get(':id')
  getOne(@Param('id') id: number): Promise<Task> {
    return this.taskService.getTaskById(id);
  }

  // به‌روزرسانی وضعیت تسک
  @Put(':id')
  update(
    @Param('id') id: number,
    @Body() body: { completed: boolean },
  ): Promise<Task> {
    return this.taskService.updateTask(id, body.completed);
  }

  // حذف تسک
  @Delete(':id')
  delete(@Param('id') id: number): Promise<void> {
    return this.taskService.deleteTask(id);
  }
}
```

6. **اجرای پروژه**

پس از تنظیمات و کدنویسی، برای اجرای پروژه دستور زیر را در ترمینال وارد کنید:

```
npm run start
```

این دستور سرور را راه‌اندازی کرده و API در `http://localhost:3000` در دسترس خواهد بود.

7. **تست‌ها**

برای نوشتن تست‌های واحد از **Jest** استفاده می‌کنیم.

7.1 **نوشتن تست‌ها برای سرویس**
به عنوان مثال، یک تست برای `TaskService` که بررسی می‌کند آیا تسک جدید به درستی ایجاد می‌شود:

```typescript
import { Test, TestingModule } from '@nestjs/testing';
import { TaskService } from './task.service';
import { getRepositoryToken } from '@nestjs/typeorm';
import { Task } from './task.entity';
import { Repository } from 'typeorm';

describe('TaskService', () => {
  let service: TaskService;
  let repository: Repository<Task>;

  beforeEach(async () => {


 const module: TestingModule = await Test.createTestingModule({
     providers: [
       TaskService,
       {
         provide: getRepositoryToken(Task),
         useClass: Repository,
       },
     ],
   }).compile();

   service = module.get<TaskService>(TaskService);
   repository = module.get<Repository<Task>>(getRepositoryToken(Task));
  });

  it('should create a new task', async () => {
   const task = await service.createTask('Test Task', 'Test Description');
   expect(task).toHaveProperty('id');
   expect(task.title).toBe('Test Task');
```

```
  });
});
```

7.2 **تست‌های E2E (انتها به انتها)**
از **Supertest** برای ارسال درخواست‌ها به **API** استفاده می‌کنیم و پاسخ‌ها را بررسی می‌کنیم. این تست‌ها اطمینان می‌دهند که **API** درست کار می‌کند.

8. **مستندات API با Swagger**
برای تولید مستندات **API** از **Swagger** استفاده می‌کنیم. در فایل `main.ts`، کد زیر را اضافه کنید تا مستندات به صورت خودکار ایجاد شود:

```typescript
import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';

const config = new DocumentBuilder()
  .setTitle('Task Management API')
  .setDescription('API for managing tasks')
  .setVersion('1.0')
  .build();
const document = SwaggerModule.createDocument(app, config);
SwaggerModule.setup('api', app, document);
```

با این کار، مستندات **API** در مسیر `/api` در دسترس خواهد بود.
