

به نام خدا

پروژه

داکرایز سازی پروژه CRUD با فریم ورک Django

موضوع پروژه

کتاب

نام درس

رایانش ابری

استاد مربوطه

سرکار خانم دکتر هدی طاهری

دانشجو

عرفان حسنی – ۴۰۰۱۲۷۷۳۵۲

فهرست مطالب

۳ Docker-compose فایل‌های
۵ Docker-compose فایل‌های نحوه اجرای
۷ Postman با استفاده از CRUD عملیات
۱۲ نحوه توقف کانتینر های اجرایی

درباب محتوای فایل‌های Docker-compose

در این پروژه دو فایل مجزای docker-compose وجود دارد که یکی از آنها برای دیتابیس و دیگری برای پروژه می‌باشد. پس از ایجاد و بالا آوردن کانتینر های هر دو ، در قالب یک شبکه به یکدیگر متصل شده و عملیات CRUD را به درستی انجام می‌دهند.

در ادامه محتوای فایل های docker-compose نشان داده شده است.

.env

```
1 POSTGRES_USER=book_user
2 POSTGRES_PASSWORD=your_password
3 POSTGRES_DB=book_manager
4 DATABASE_HOST=postgres_db
5 DATABASE_PORT=5432
6
```

عکس بالا مربوط به فایل env می‌باشد که مقادیر را به صورت کلید و مقدار در خود نگه داشته و از آن‌ها در docker-compose ها استفاده می‌شود. به این صورت که کلید آن را استفاده می‌کنیم و مقدار آن خود به جای کلید در زمان اجرا استفاده خواهد شد.

Docker-compose.db.yml

```
1 version: '3.8'
2
3 services:
4   db:
5     image: postgres:15
6     container_name: postgres_db
7     environment:
8       POSTGRES_USER: ${POSTGRES_USER}
9       POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
10      POSTGRES_DB: ${POSTGRES_DB}
11     ports:
12       - "5433:5432"
13     volumes:
14       - postgres_data:/var/lib/postgresql/data
15     networks:
16       - app_network
17
18 volumes:
19   postgres_data:
20
21 networks:
22   app_network:
23     name: app_network
24     driver: bridge
25
```

این فایل مربوط به دیتابیس می‌باشد. از دیتابیس postgresdb استفاده شده است. به عنوان مقادیری که برای ایجاد و اتصال به دیتابیس نیاز داریم ، کلیدهای مشخص شده در فایل env قرار داده شده است.

اسم کانتینر مربوط به این فایل ، نیز در خط ۶ مشخص شده است.

نکته قابل توجه دیگر، بخش net-work می‌باشد که چون قرار است با فایل

docker-compose مربوط به پروژه متصل شود، یک شبکه ایجاد می‌کند به نام app-network و در آن با موفقیت اتصال دو کانتینر ایجاد می‌شود.

Docker-compose.app.yml

```
1 version: '3.8'
2
3 services:
4   web:
5     build:
6       context: .
7       dockerfile: Dockerfile
8     container_name: django_app
9     command: python manage.py runserver 0.0.0.0:8000
10    volumes:
11      - ./app
12    ports:
13      - "8000:8000"
14    environment:
15      DATABASE_NAME: ${POSTGRES_DB}
16      DATABASE_USER: ${POSTGRES_USER}
17      DATABASE_PASSWORD: ${POSTGRES_PASSWORD}
18      DATABASE_HOST: ${DATABASE_HOST}
19      DATABASE_PORT: ${DATABASE_PORT}
20    networks:
21      - app_network
22
23 networks:
24   app_network:
25     external: true
26
```

این فایل مربوط به پروژه می‌باشد.
Volumes آن نیز از پوشه محلی ./app گرفته شده که در آن پروژه ایجاد شده است.

مقادیر مورد نیاز برای اتصال به پایگاه داده در بخش envirement آمده است. از مقادیر مربوط در env استفاده شده است. نکته قابل ذکر این است که مقدار مربوط به Host برابر با نام کانتینر دیتابیس می‌باشد.

در بخش port مشخص شده است که بر روی چه port ای این کانتینر اجرا شود.

و دقیقاً در بخش مشابه ینی network

مشخص شده است که به شبکه app-network وارد شود و این باید اتصال دو کانتینر به یکدیگر می‌شود.

نحوه اجرای فایل‌های Docker-compose

برای ساخت و اجرای کانتینر های مربوط به docker-compose ها ، نیاز است که اول آنها را به اصطلاح build کنیم و سپس بالا بیاوریم (Up).

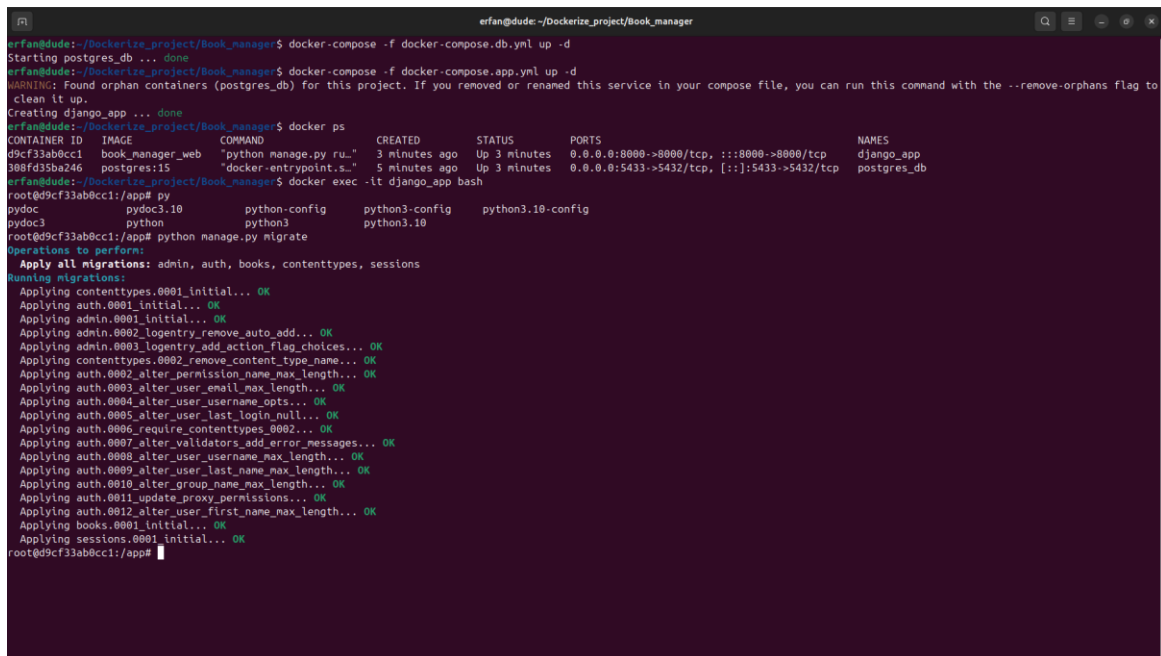
اما می‌شود با یک خط کد هر دور این کار ها را انجام داد

`Docker-compose up --build`

اما با توجه با اینکه اینجا ما با دو تا فایل docker-compose وجود دارد باید آنها را با نام فایل خطاب قرار دهیم که برای اینکار از -f استفاده می‌کنیم، همچنین برای اینکه بتوانیم اجرای کانتینر ها را به background منتقل کنیم ، از -d استفاده می‌شود. که در نتیجه به عنوان مثال برای ایجاد و اجرای کانتینر مربوط به دیتا بیس باید چنین کدی را در ترمینال وارد کنیم.

`Docker-compose -f docker-compose.db.yml up --build -d`

عکسی که در ادامه قرار داده شده ، با توجه به اینکه در مرحله بعد از ایجاد کانتینر ها گرفته شده است، از build در دستور زده شده استفاده نشده است.



```
erfan@dude:~/Dockerize_project/Book_manager$ docker-compose -f docker-compose.db.yml up -d
Starting postgres_db ... done
erfan@dude:~/Dockerize_project/Book_manager$ docker-compose -f docker-compose.app.yml up -d
WARNING: Found orphan containers (postgres_db) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to
clean it up.
Creating django_app ... done
erfan@dude:~/Dockerize_project/Book_manager$ docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED        STATUS        PORTS                               NAMES
d9cf33ab0cc1   book_manager_web    "python manage.py ru..." 3 minutes ago  Up 3 minutes  0.0.0.0:8000->8000/tcp, :::8000->8000/tcp  django_app
308fd35ba246   postgres:15         "docker-entrypoint.s..." 5 minutes ago  Up 3 minutes  0.0.0.0:5433->5432/tcp, [::]:5433->5432/tcp  postgres_db
erfan@dude:~/Dockerize_project/Book_manager$ docker exec -it django_app bash
root@d9cf33ab0cc1:/app# py
pydoc3          pydoc3.10          python-config      python3-config     python3.10-config
pydoc3          python             python3            python3.10         python3.10-config
root@d9cf33ab0cc1:/app# python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, books, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying books.0001_initial... OK
  Applying sessions.0001_initial... OK
root@d9cf33ab0cc1:/app#
```

برای اینکه بدانیم چه کانتینرهایی در حال اجرا هستند، از دستور `docker ps` استفاده می‌کنیم. و مشاهده می‌شود که هر دو کانتینر در حالت اجرا قرار دارند.

برای ایجاد جدول های مورد نیاز در پایگاه داده پروژه نیاز است که به کانتینر پروژه وارد شویم و دستور زیر را بزنیم. تا جداول ایجاد شوند و برای استفاده آماده شوند.

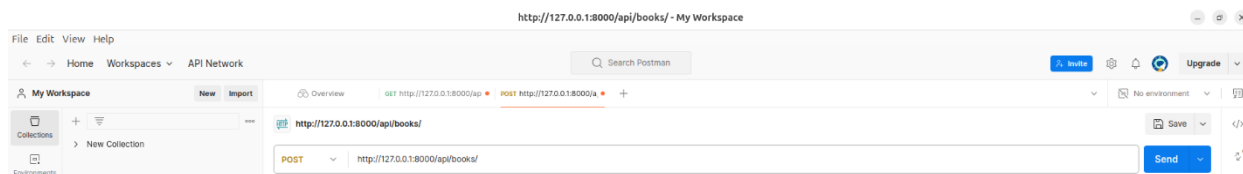
```
Python manage.py migrate
```

و اما برای اینکه بتوانیم به داخل کانتینر پروژه که در حال اجرا است، وارد شویم باید از دستور زیر استفاده کرد.

```
Docker exec -it django_app bash
```

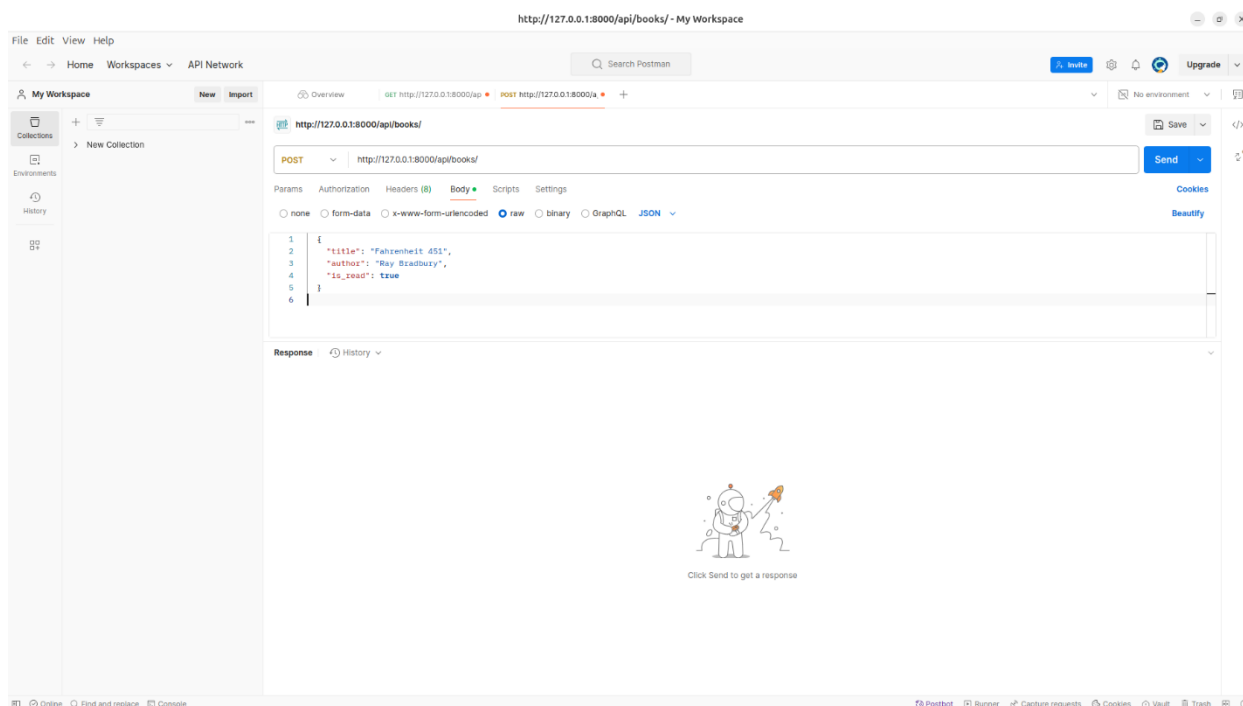
در حال حاضر، کانتینر ها با هم دیگر شبکه شده اند و آماده استفاده شدن هستند. در مرحله بعد ، تست و اجرای چهار دستور CRUD با استفاده از postman بر روی پروژه انجام خواهد شد.

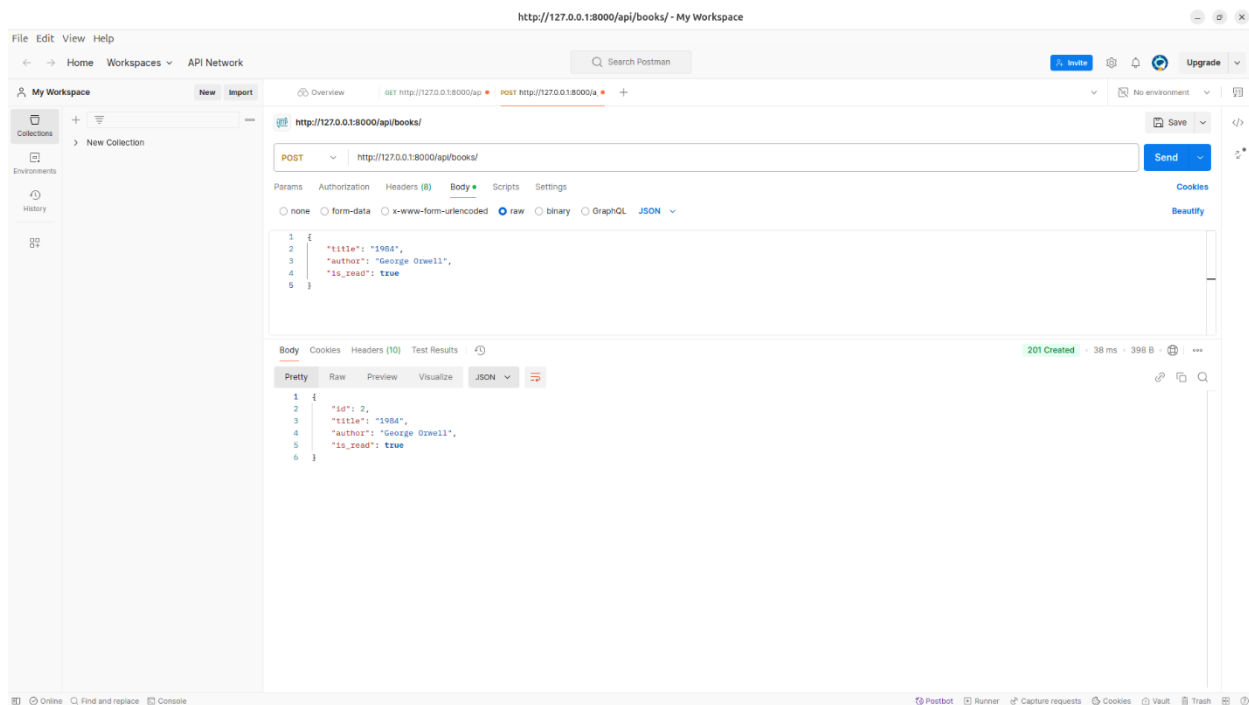
عملیات CRUD با استفاده از Postman



برای استفاده از postman باید اول لینک اجرایی‌ای که پروژه داره انجام میشه رو بهش داد (در بخشی که در بالا دیده می‌شود باید قرار داده شود)

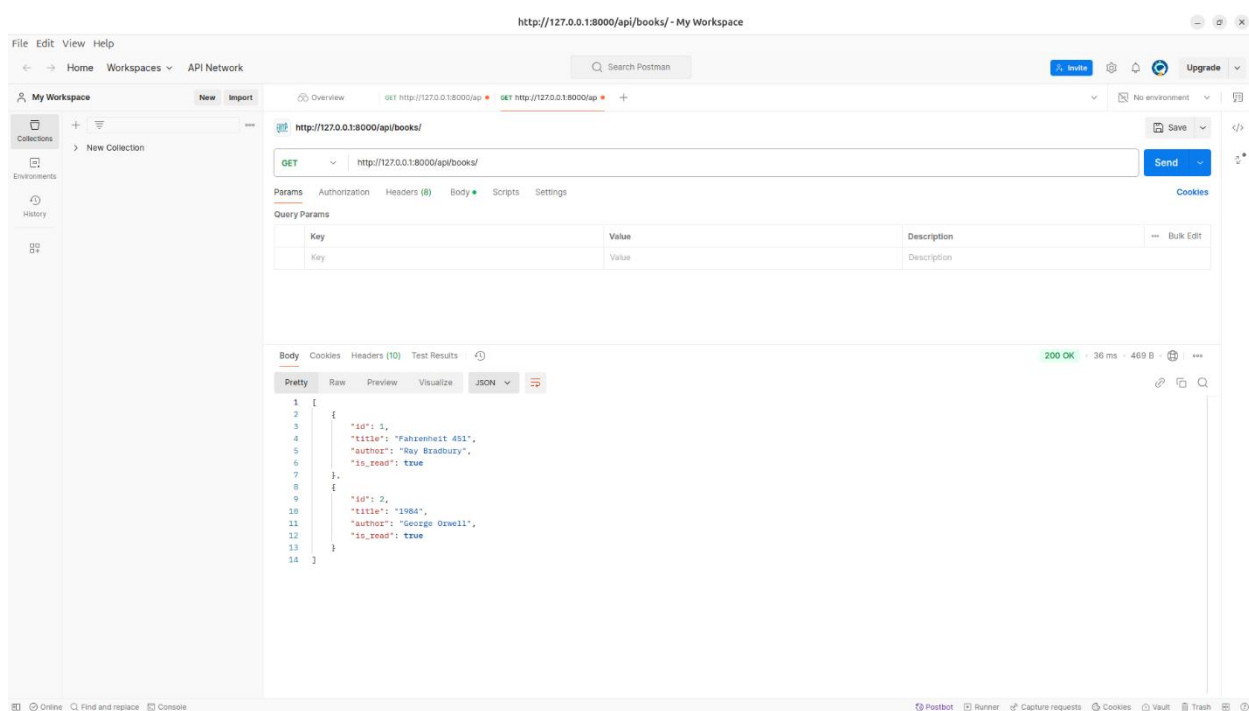
عملیات Create در postman همان عملیات Post می‌باشد. برای عملیات post باید از بخش عملیات ها (سمت چپ لینک اجرایی) آن را انتخاب کرده از بخش Body حالت raw را انتخاب و در آن داده‌ای که می‌خواهیم به پایگاه‌داده ارسال کنیم، به صورت فایل json بنویسیم و در نهایت روی گزینه Send کلیک کنیم !!





همانطور که دیده می‌شود؛ پس از ارسال داده به پایگاه داده ، پیام 201 created به ما نشان داده می‌شود که نشان دهنده موفقیت آمیز بودن عملیات است.

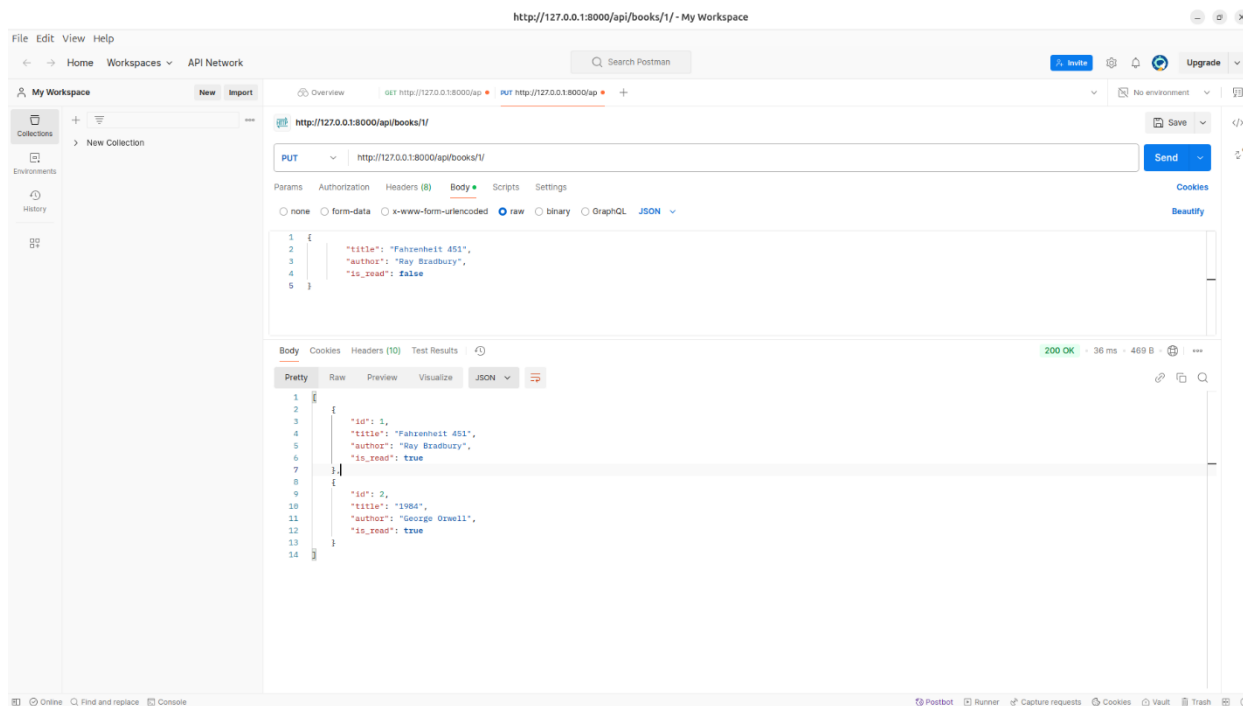
برای عملیات Read در postman از عملیات get استفاده می‌کنیم



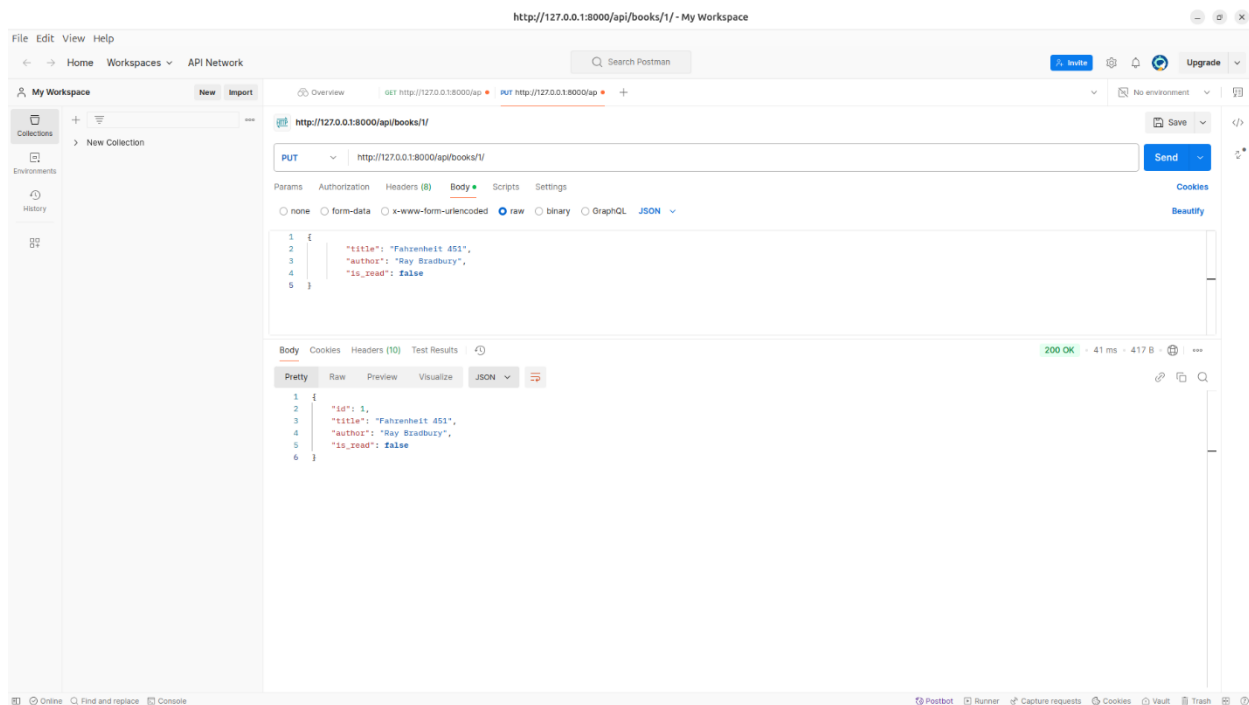
همانطور که در تصویر بالا مشخص است ، پس از درخواست عملیات get به ما پیام 200 OK نمایش داده می‌شود که نشان دهنده موفقیت آمیز بودن عملیات است و به ما مقادیر داخل پایگاه داده برگشت داده می‌شود.

عملیات update در postman برابر با عملیات put می‌باشد.

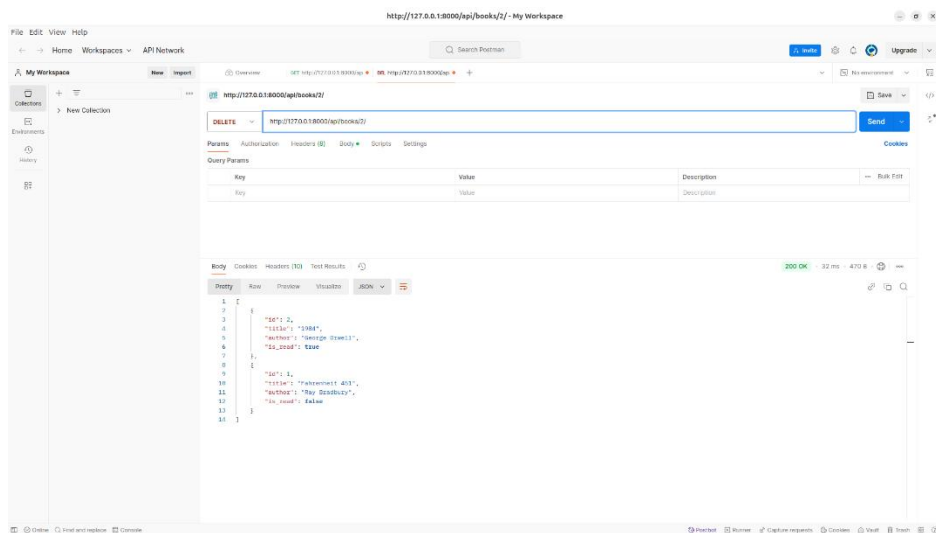
این عملیات به این صورت انجام می‌شود که در کنار لینک اجرا باید id مقداری که مد نظرمان هست و می‌خواهیم تغییری در آن اجرا کنیم را وارد می‌کنیم که تغییرات بر روی آن اعمال شود و به اصطلاح به‌روز شود.



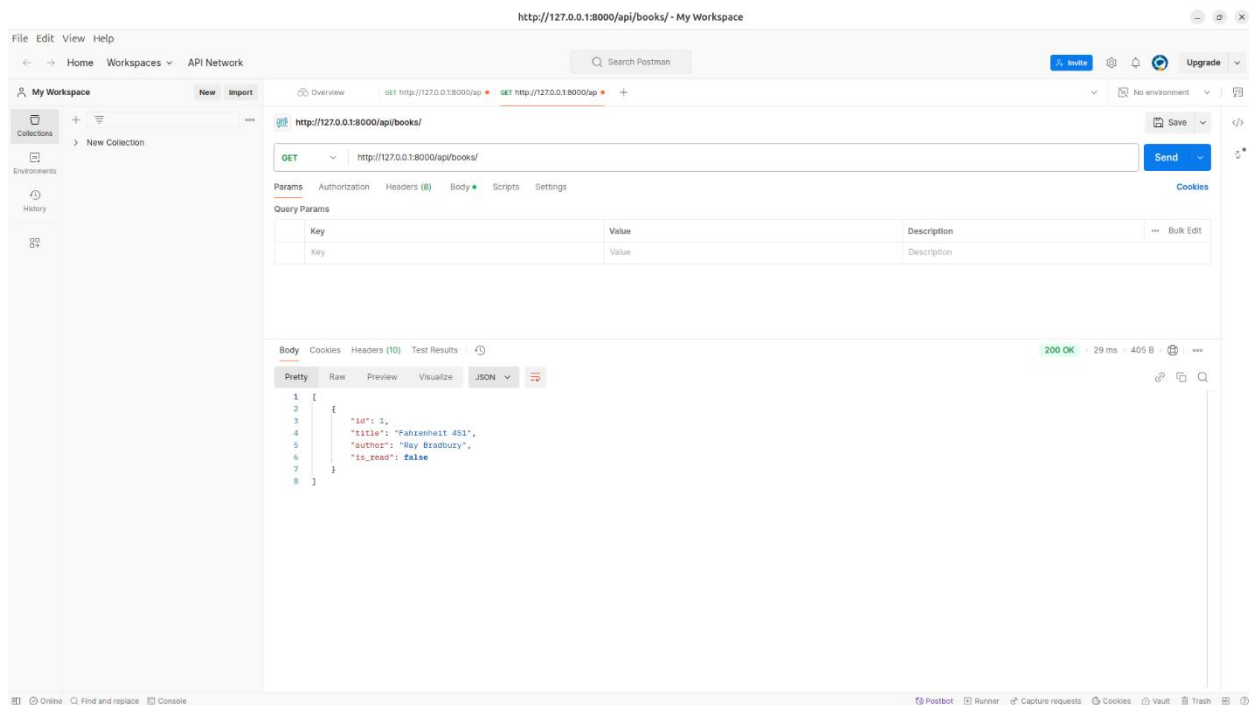
در عکس بالا ، id مشخص شده برای تغییر «۱» می‌باشد. و چیزی که قرار است در آن تغییر کند، در حالت raw از بخش Body قابل مشاهده هست و می‌توان آنجا مقادیری که می‌خواهیم ویرایش کنیم وارد کنیم. (در این مثال مقدار is_read تغییر یافته است). عکس بعدی مربوط به بعد از زدن دکمه send و اعمال تغییرات است.



و در نهایت عملیات Delete در postman همان عملیات Delete می‌باشد.
در این عملیات هم همانند عملیات put باید id داده مورد نظر را انتخاب کرد.



عکس بعدی ، مربوط به گرفتن دستور Get (یا همون Read) بعد از حذف مقداری‌ست که در عکس بالا انتخاب شده بود.



در حال حاضر پس از انجام و تست چهار عملیات CRUD در پایگاه داده ، تنها مقداری که وجود دارد همین مقداری است که در عکس بالا وجود دارد. برای اطمینان از این قضیه و اینکه بدانیم داده ها در کجا ذخیره می شود. باید کانتینر مربوط به پایگاه داده را بررسی کرد.

برای اینکار ، میتوان سلسله دستورات عکس زیر را در ترمینال اجرا کرد و مشاهده کرد که تنها داده موجود در پایگاه داده همان داده ای هست که پس از تست چهار عملیات CRUD در postman به جا مانده بود.

```

[1] mfan@node: /dockerwsl_project/book_manager$ docker exec -it postgres_db psql -U book_user -d book_manager
psql (15.10 (Debian 15.10-1.pgdg12b1))
Type "help" for help.

book_manager=# \dt
          List of relations
Schema | Name          | Type  | Owner
-----|-----|-----|-----
public | auth_group    | table | book_user
public | auth_group_permissions | table | book_user
public | auth_permission | table | book_user
public | auth_user     | table | book_user
public | auth_user_groups | table | book_user
public | auth_user_user_permissions | table | book_user
public | books_book    | table | book_user
public | django_admin_log | table | book_user
public | django_content_type | table | book_user
public | django_migrations | table | book_user
public | django_session | table | book_user
(11 rows)

book_manager=# SELECT * FROM books_book;
 id | title          | author      | is_read
-----|-----|-----|-----
  1 | Fahrenheit 451 | Ray Bradbury | f
(1 row)

book_manager=#

```

نحوه توقف کانتینر های اجرایی

برای توقف اجرای کانتینر ها تنها کافیست طبق دستورات عکس پایین عمل کنید.

```
erfan@dude: ~/Dockerize_project/Book_manager$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
d9cf33ab8cc1   book_manager_web  "python manage.py ru..." About an hour ago Up About an hour 0.0.0.0:8000->8000/tcp, :::8000->8000/tcp  django_app
308fd35ba246   postgres:15      "docker-entrypoint.s..." About an hour ago Up About an hour 0.0.0.0:5433->5432/tcp, [::]:5433->5432/tcp  postgres_db

erfan@dude: ~/Dockerize_project/Book_manager$ docker-compose -f docker-compose.app.yml down
Stopping django_app ... done
WARNING: Found orphan containers (postgres_db) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to
clean it up.
Removing django_app ... done
Network app_network is external, skipping
erfan@dude: ~/Dockerize_project/Book_manager$
erfan@dude: ~/Dockerize_project/Book_manager$ docker-compose -f docker-compose.db.yml down
Stopping postgres_db ... done
Removing postgres_db ... done
Removing network app_network
erfan@dude: ~/Dockerize_project/Book_manager$ docker ps
CONTAINER ID   IMAGE          COMMAND                  STATUS        PORTS        NAMES
erfan@dude: ~/Dockerize_project/Book_manager$
```

در نهایت میتوان مشاهده کرد که با زدن دستور docker ps هیچ کانتینر فعالی وجود ندارد.

پایان !!!