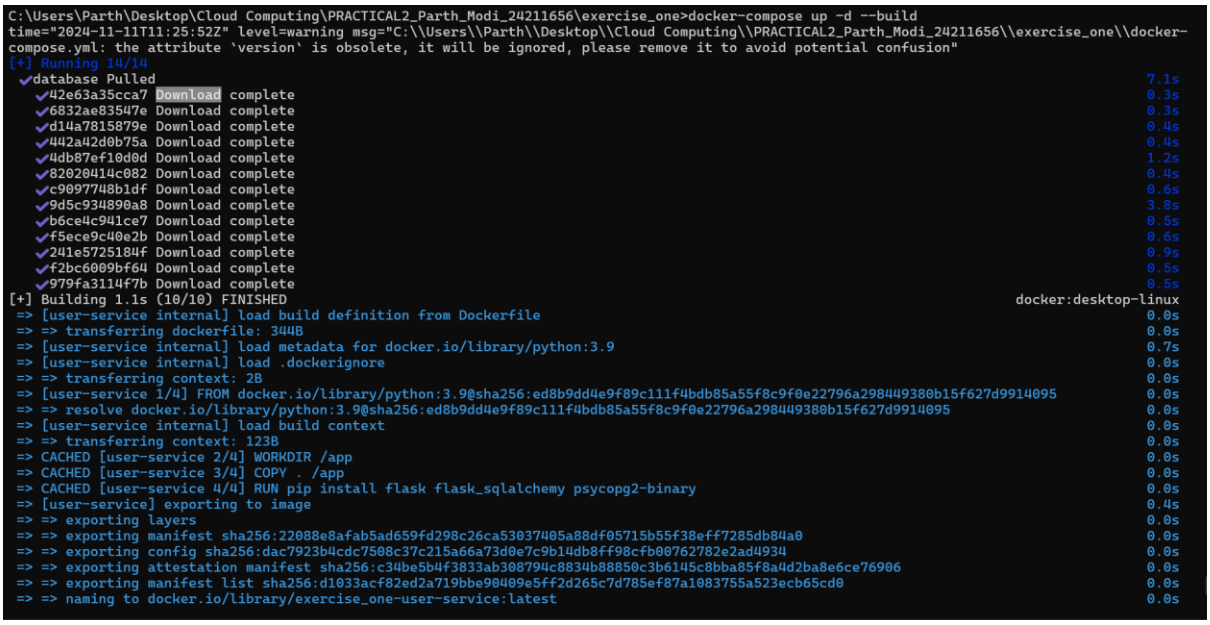
CLOUD COMPUTING

EXERCISE ONE

Step 1) The first thing after setting up folder structure and downloading the file is to create a docker file for running User Service in port 5002.

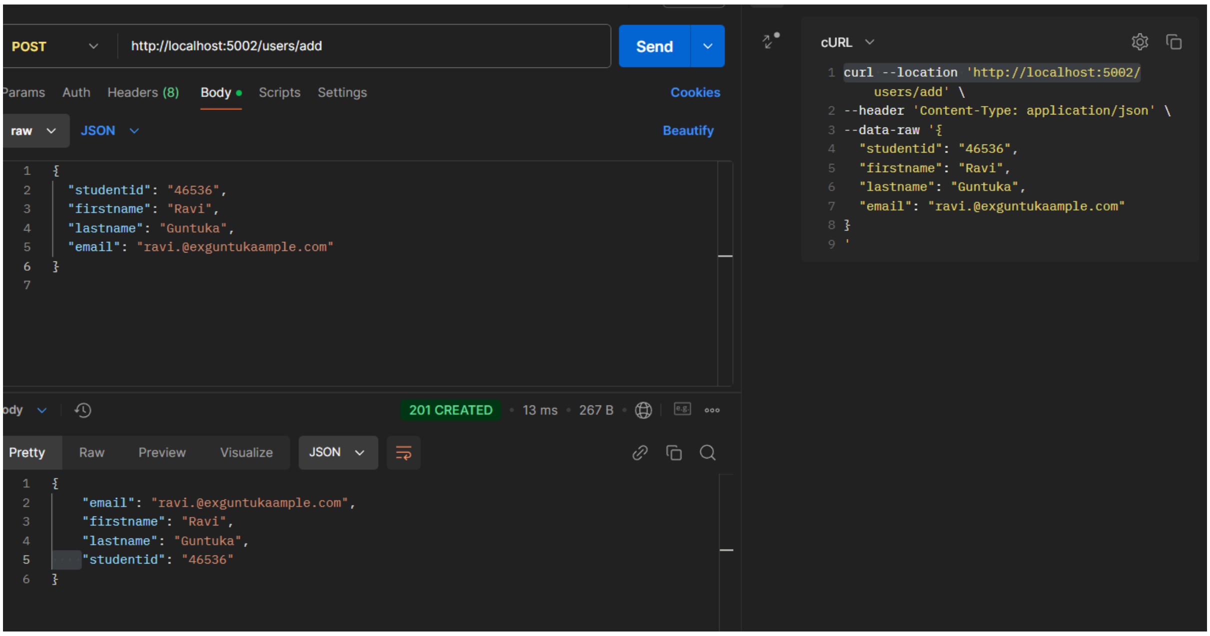
Step 2) We need to create docker compose file as well for the User Service and postgres db so that we can test few api's for user Service. We did a setup for network and volumes for efficient communication and persistent storage.

Step 3) After the setup we can run the docker compose build command to test if everything is working fine.



```
C:\Users\Parth\Desktop\Cloud Computing\PRACTICAL2_Parth_Modi_24211656\exercise_one>docker-compose up -d --build
time="2024-11-11T11:25:52Z" level=warning msg="C:\\Users\\Parth\\Desktop\\Cloud Computing\\PRACTICAL2_Parth_Modi_24211656\\exercise_one\\docker-
compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 14/14
 ✓database Pulled                                                                                                                          7.1s
   ✓42e63a35cca7 Download complete                                                                                                         0.3s
   ✓6832ae83547e Download complete                                                                                                         0.3s
   ✓d14a7815879e Download complete                                                                                                         0.4s
   ✓442a42d0b75a Download complete                                                                                                         0.4s
   ✓4db87ef10d0d Download complete                                                                                                         1.2s
   ✓82020414c082 Download complete                                                                                                         0.4s
   ✓c9097748b1df Download complete                                                                                                         0.6s
   ✓9d5c934890a8 Download complete                                                                                                         3.8s
   ✓b6ce4c941ce7 Download complete                                                                                                         0.5s
   ✓f5ece9c40e2b Download complete                                                                                                         0.6s
   ✓241e5725184f Download complete                                                                                                         0.9s
   ✓f2bc6009bf64 Download complete                                                                                                         0.5s
   ✓979fa3114f7b Download complete                                                                                                         0.5s
[+] Building 1.1s (10/10) FINISHED                                                                                         docker:desktop-linux
 => [user-service internal] load build definition from Dockerfile                                                                          0.0s
 => => transferring dockerfile: 344B                                                                                                       0.0s
 => [user-service internal] load metadata for docker.io/library/python:3.9                                                                 0.7s
 => [user-service internal] load .dockerignore                                                                                             0.0s
 => => transferring context: 2B                                                                                                            0.0s
 => [user-service 1/4] FROM docker.io/library/python:3.9@sha256:ed8b9dd4e9f89c111f4bdb85a55f8c9f0e22796a298449380b15f627d9914095           0.0s
 => => resolve docker.io/library/python:3.9@sha256:ed8b9dd4e9f89c111f4bdb85a55f8c9f0e22796a298449380b15f627d9914095                        0.0s
 => [user-service internal] load build context                                                                                             0.0s
 => => transferring context: 123B                                                                                                          0.0s
 => CACHED [user-service 2/4] WORKDIR /app                                                                                                 0.0s
 => CACHED [user-service 3/4] COPY . /app                                                                                                  0.0s
 => CACHED [user-service 4/4] RUN pip install flask flask_sqlalchemy psycopg2-binary                                                       0.0s
 => [user-service] exporting to image                                                                                                      0.4s
 => => exporting layers                                                                                                                    0.0s
 => => exporting manifest sha256:22088e8afab5ad659fd298c26ca53037405a88df05715b55f38eff7285db84a0                                          0.0s
 => => exporting config sha256:dac7923b4cdc7508c37c215a66a73d0e7c9b14db8ff98cfb00762782e2ad4934                                            0.0s
 => => exporting attestation manifest sha256:c34be5b4f3833ab308794c8834b88850c3b6145c8bba85f8a4d2ba8e6ce76906                              0.0s
 => => exporting manifest list sha256:d1033acf82ed2a719bbe90409e5ff2d265c7d785ef87a1083755a523ecb65cd0                                     0.0s
 => => naming to docker.io/library/exercise_one-user-service:latest                                                                        0.0s
```

Step 4) Let’s add the user’s using the post api



POST http://localhost:5002/users/add

Send

```
{
  "studentid": "46536",
  "firstname": "Ravi",
  "lastname": "Guntuka",
  "email": "ravi.@exguntukaample.com"
}
```

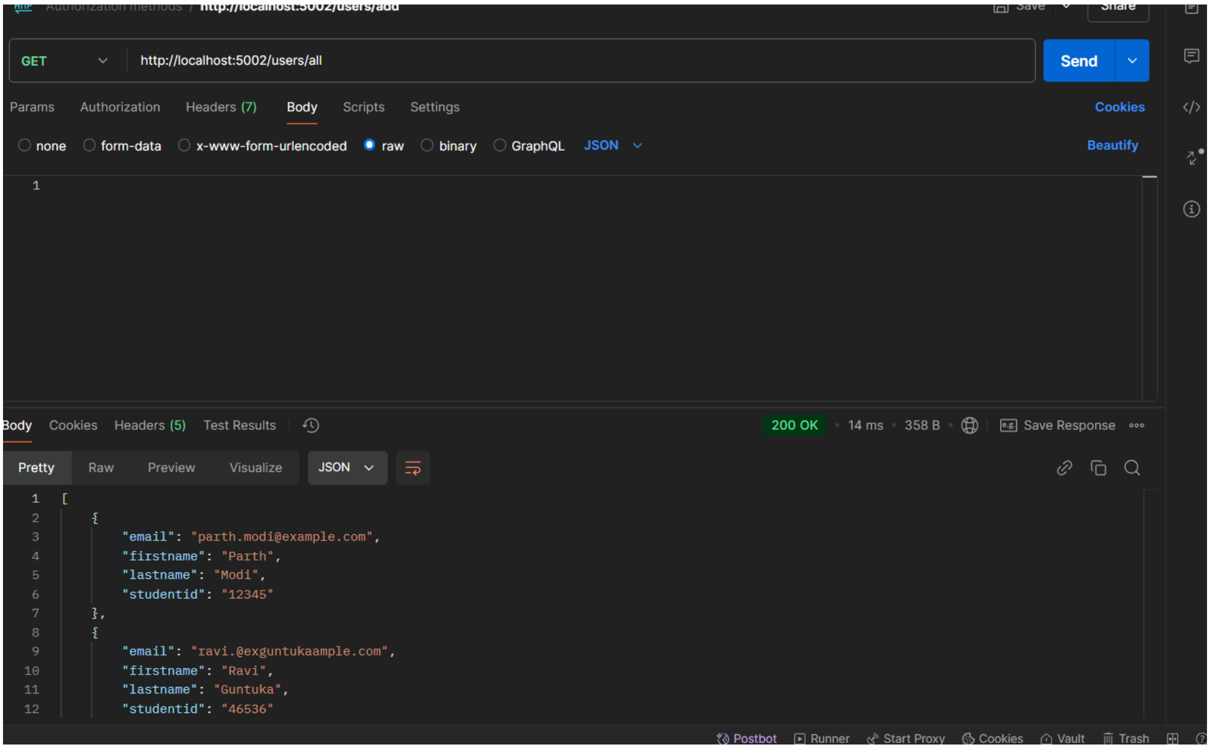
201 CREATED • 13 ms • 267 B

```
{
    "email": "ravi.@exguntukaample.com",
    "firstname": "Ravi",
    "lastname": "Guntuka",
    "studentid": "46536"
}
```

cURL

```
curl --location 'http://localhost:5002/
    users/add' \
--header 'Content-Type: application/json' \
--data-raw '{
  "studentid": "46536",
  "firstname": "Ravi",
  "lastname": "Guntuka",
  "email": "ravi.@exguntukaample.com"
}
'
```

Step 5) Let’s get the details of all the users we added using the get api



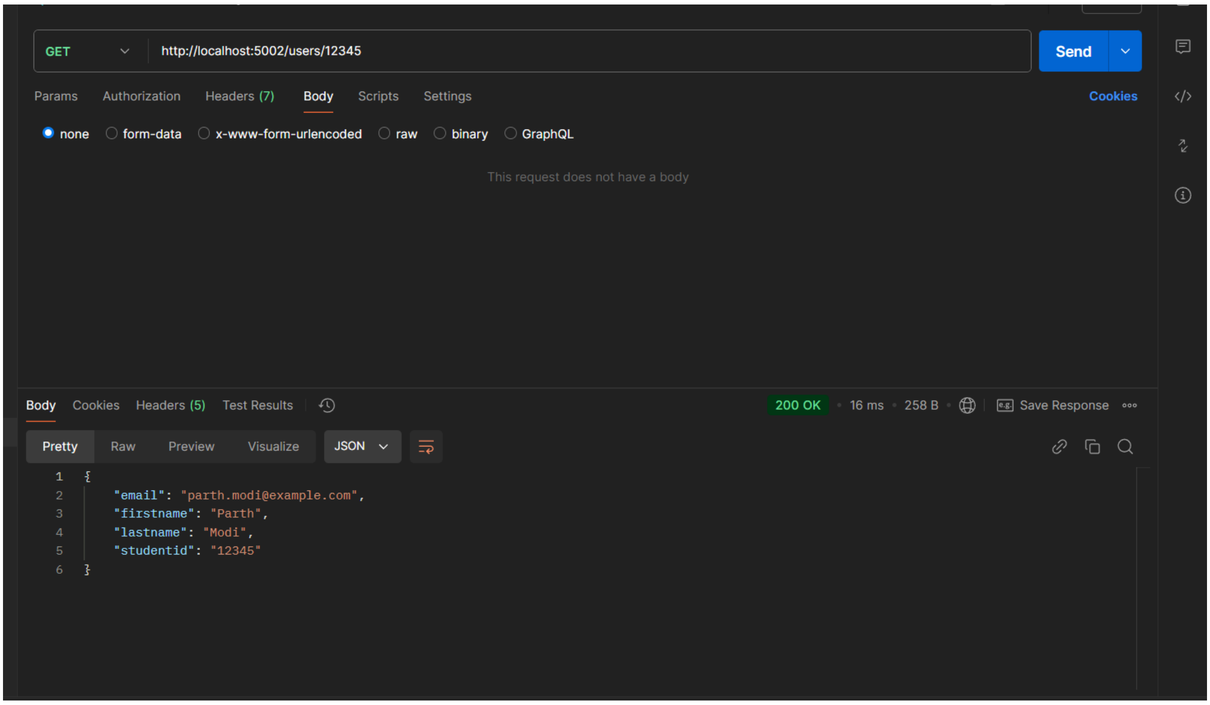
GET http://localhost:5002/users/all

Send

200 OK • 14 ms • 358 B

```
[
    {
        "email": "parth.modi@example.com",
        "firstname": "Parth",
        "lastname": "Modi",
        "studentid": "12345"
    },
    {
        "email": "ravi.@exguntukaample.com",
        "firstname": "Ravi",
        "lastname": "Guntuka",
        "studentid": "46536"
```

Step 6) Let’s get details of single user using the get request using params



http://localhost:5002/users/12345

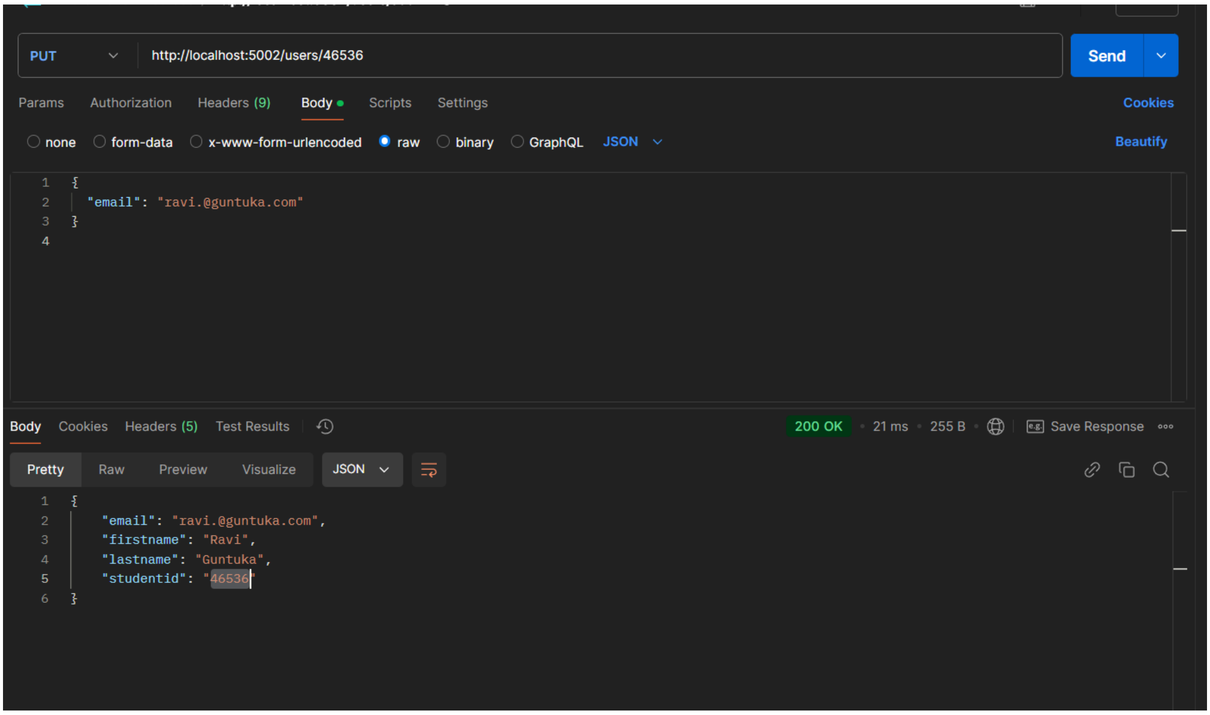
○ none ○ form-data ○ x-www-form-urlencoded ○ raw ○ binary ○ GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results — 200 OK • 16 ms • 258 B

```
{
    "email": "parth.modi@example.com",
    "firstname": "Parth",
    "lastname": "Modi",
    "studentid": "12345"
}
```

Step 7) Let’s update the data of a particular user using the put request



http://localhost:5002/users/46536

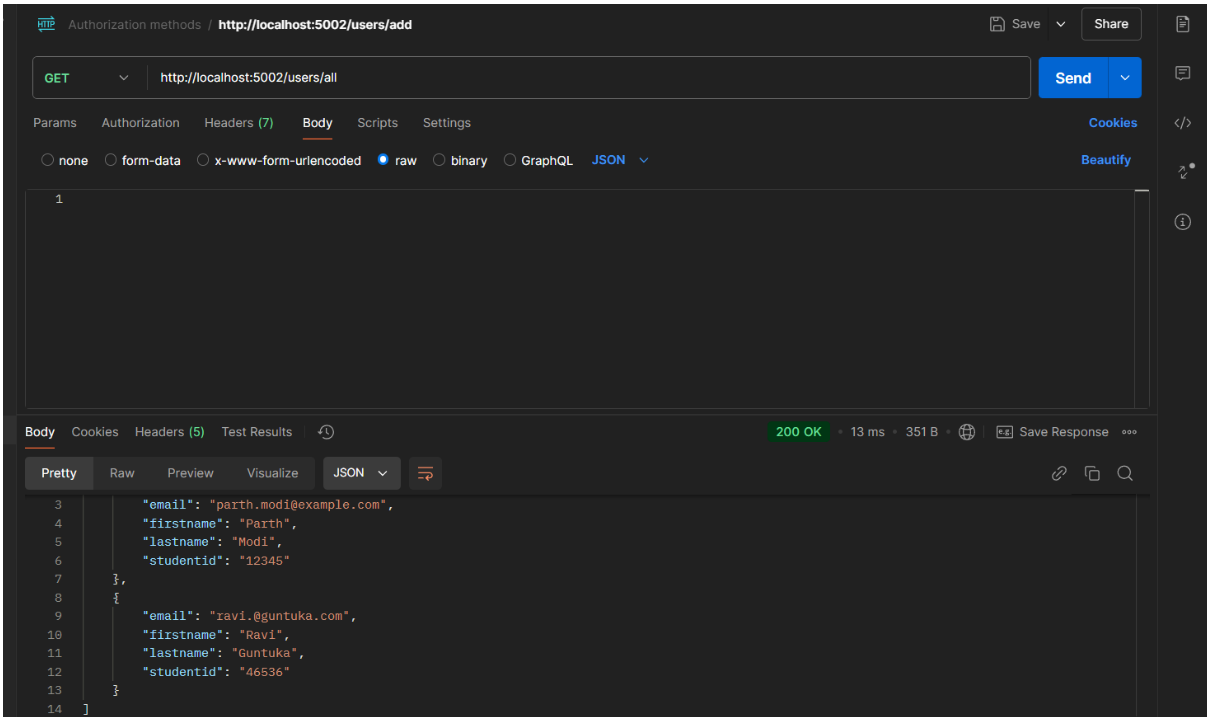
○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
  "email": "ravi.@guntuka.com"
}
```

Body Cookies Headers (5) Test Results — 200 OK • 21 ms • 255 B

```
{
    "email": "ravi.@guntuka.com",
    "firstname": "Ravi",
    "lastname": "Guntuka",
    "studentid": "46536"
}
```

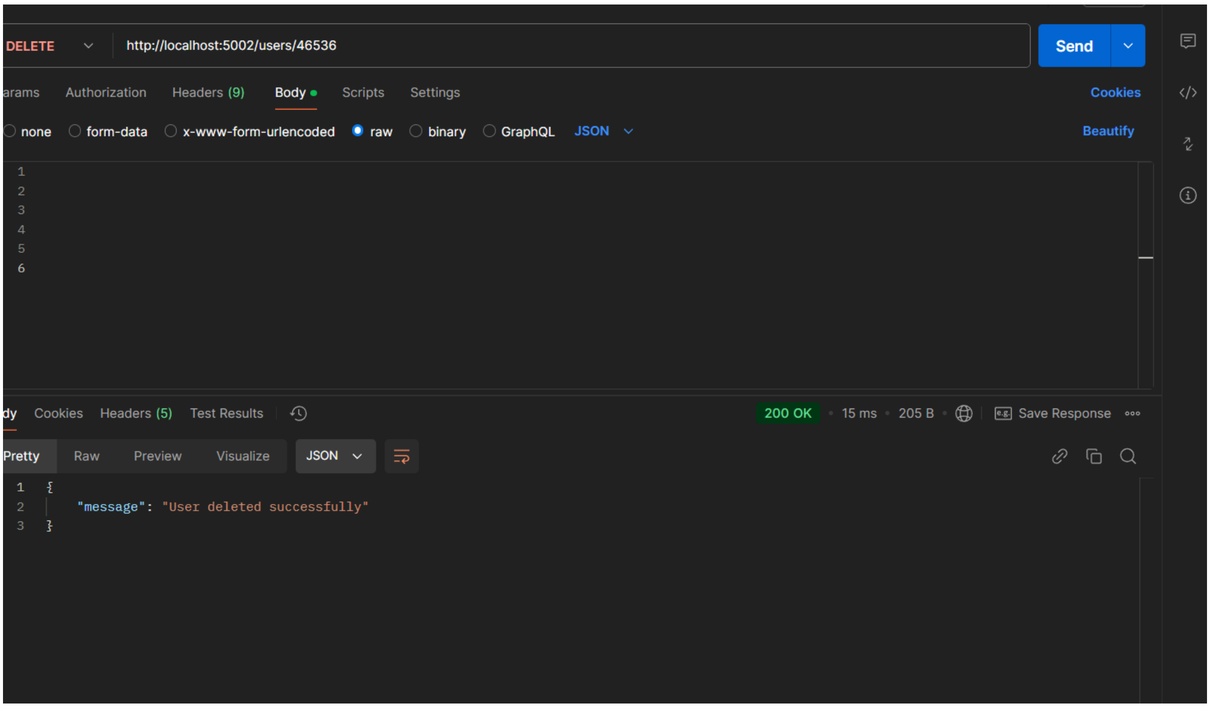
Step 8) Let’s see if the data is updated in the database using the get request



GET http://localhost:5002/users/all

```
        "email": "parth.modi@example.com",
        "firstname": "Parth",
        "lastname": "Modi",
        "studentid": "12345"
    },
    {
        "email": "ravi.@guntuka.com",
        "firstname": "Ravi",
        "lastname": "Guntuka",
        "studentid": "46536"
    }
]
```

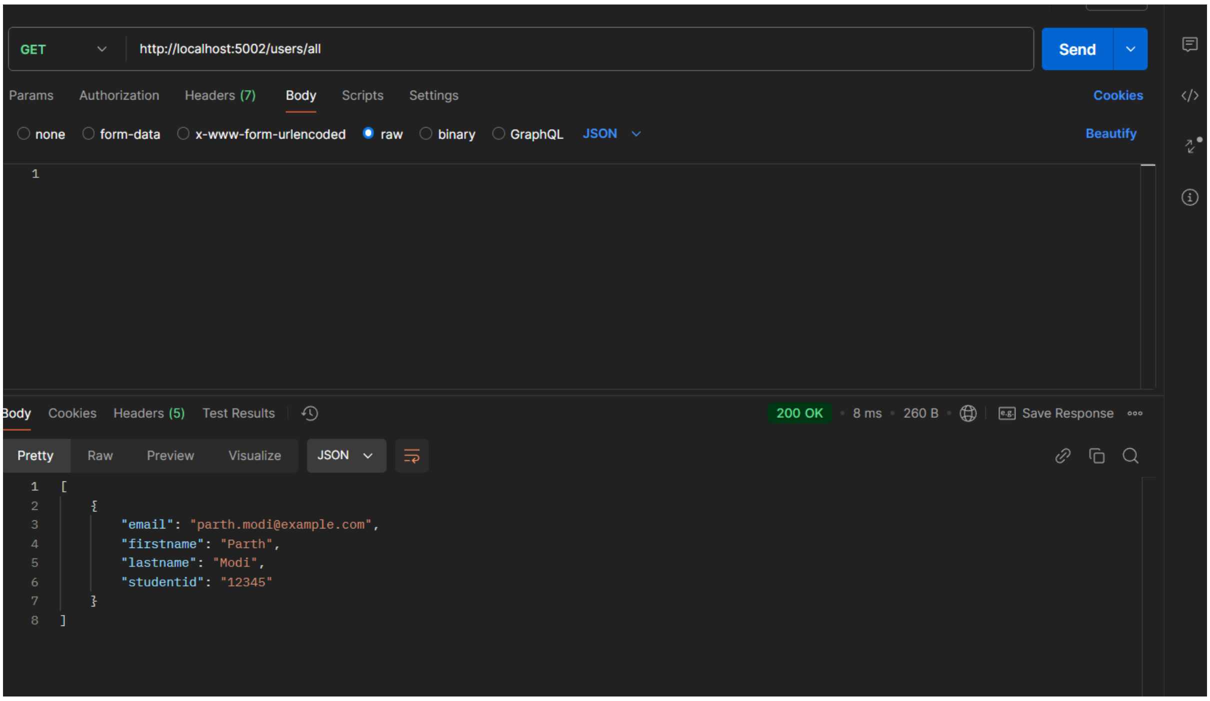
Step 9) Let’s delete one record for the db, using the userid.



DELETE http://localhost:5002/users/46536

```
{
    "message": "User deleted successfully"
}
```

Step 10) Let’s check if the record is being deleted or not



GET http://localhost:5002/users/all

200 OK • 8 ms • 260 B

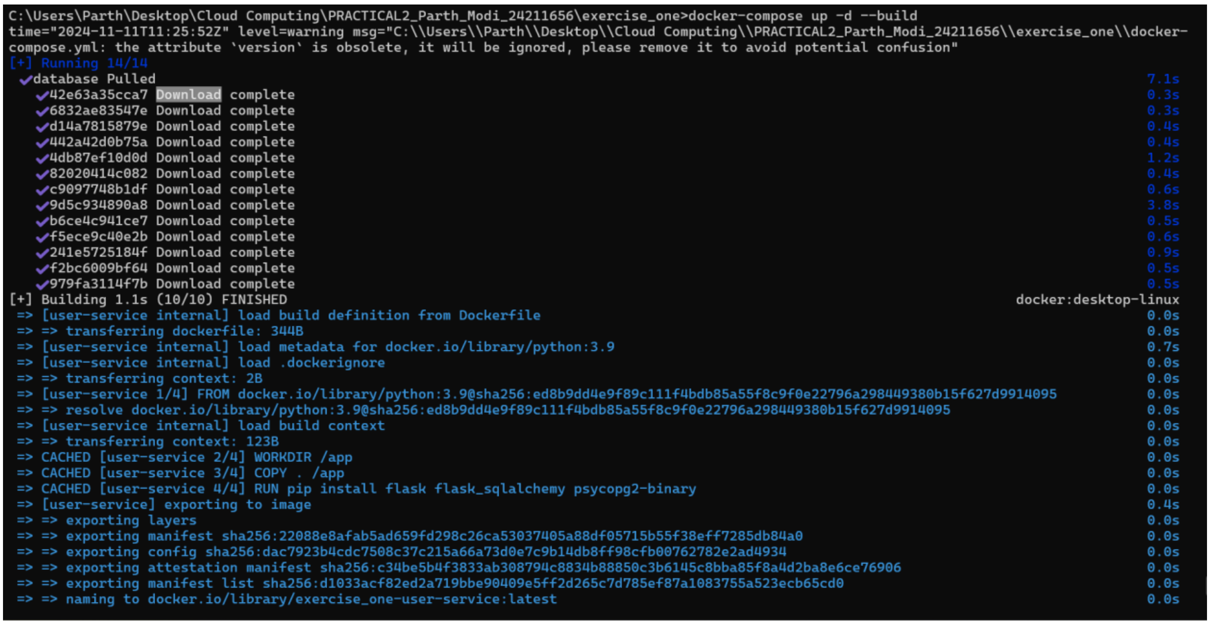
```
[
    {
        "email": "parth.modi@example.com",
        "firstname": "Parth",
        "lastname": "Modi",
        "studentid": "12345"
    }
]
```

Now lets replicate the same thing for book Service

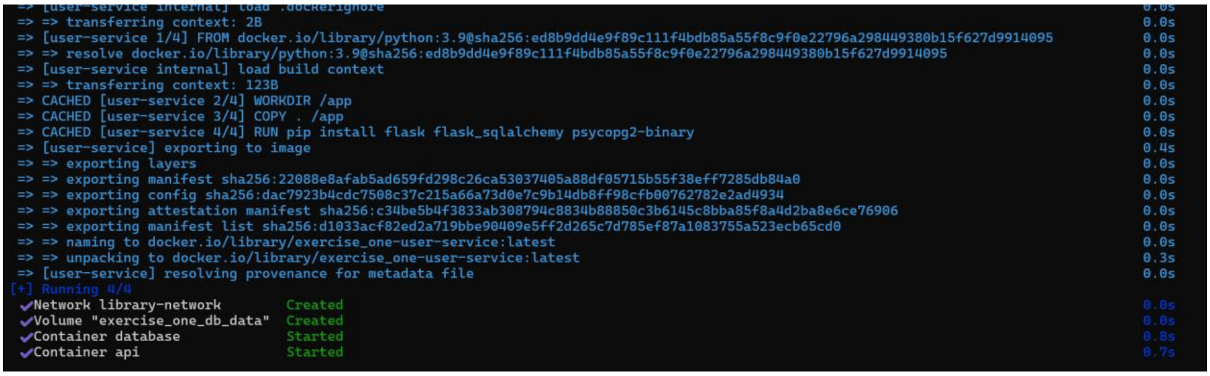
Step 1) The first thing after setting up folder structure and downloading the file is to create a docker file for running Book Service in port 5006.

Step 2) We need to create docker compose file as well for the Book Service and postgres db so that we can test few api’s for Book Service. We did a setup for network and volumes for efficient communication and persistent storage.

Step 3) After the setup we can run the docker compose build command to test if everything is working fine.

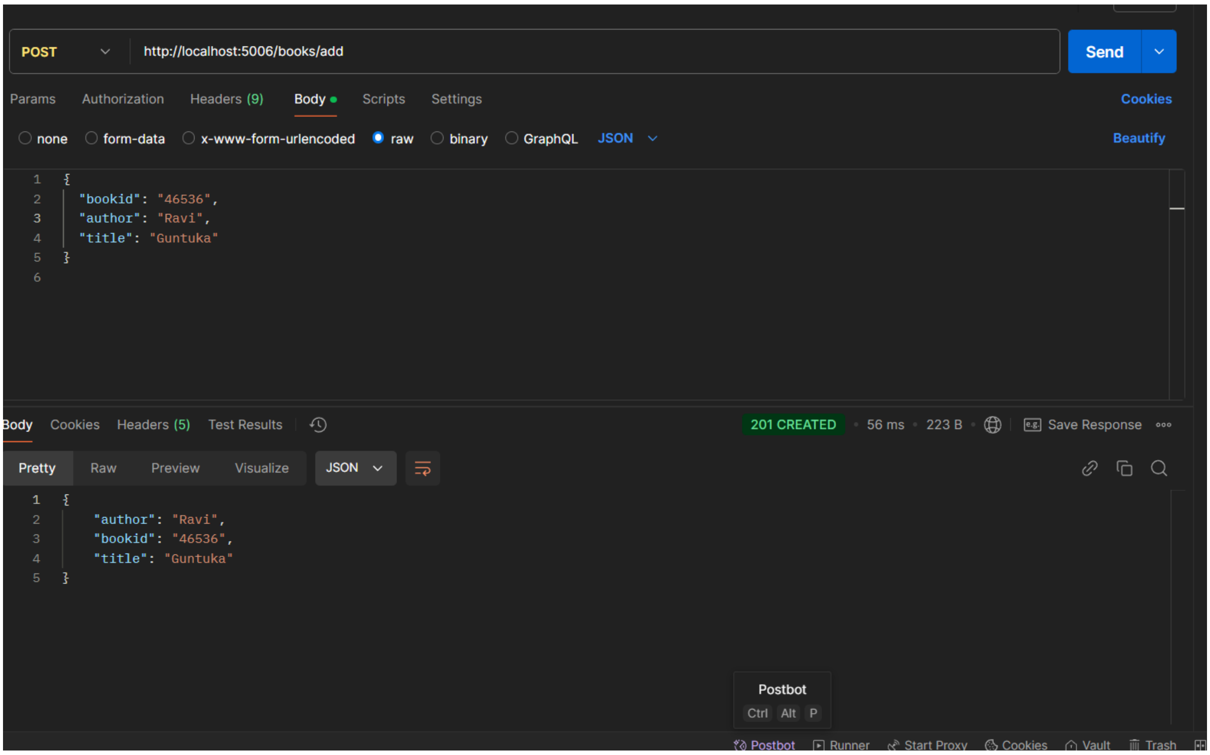


```
C:\Users\Parth\Desktop\Cloud Computing\PRACTICAL2_Parth_Modi_24211656\exercise_one>docker-compose up -d --build
time="2024-11-11T11:25:52Z" level=warning msg="C:\\Users\\Parth\\Desktop\\Cloud Computing\\PRACTICAL2_Parth_Modi_24211656\\exercise_one\\docker-
compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 14/14
 ✓database Pulled                                                                                                                              7.1s
   ✓42e63a35cca7 Download complete                                                                                                             0.3s
   ✓6832ae83547e Download complete                                                                                                             0.3s
   ✓d14a7815879e Download complete                                                                                                             0.4s
   ✓442a42d0b75a Download complete                                                                                                             0.4s
   ✓4db87ef10d0d Download complete                                                                                                             1.2s
   ✓82020414c082 Download complete                                                                                                             0.4s
   ✓c9097748b1df Download complete                                                                                                             0.6s
   ✓9d5c934890a8 Download complete                                                                                                             3.8s
   ✓b6ce4c941ce7 Download complete                                                                                                             0.5s
   ✓f5ece9c40e2b Download complete                                                                                                             0.6s
   ✓241e5725184f Download complete                                                                                                             0.9s
   ✓f2bc6009bf64 Download complete                                                                                                             0.5s
   ✓979fa3114f7b Download complete                                                                                                             0.5s
[+] Building 1.1s (10/10) FINISHED                                                                                         docker:desktop-linux
 => [user-service internal] load build definition from Dockerfile                                                                              0.0s
 => => transferring dockerfile: 344B                                                                                                           0.0s
 => [user-service internal] load metadata for docker.io/library/python:3.9                                                                     0.7s
 => [user-service internal] load .dockerignore                                                                                                 0.0s
 => => transferring context: 2B                                                                                                                0.0s
 => [user-service 1/4] FROM docker.io/library/python:3.9@sha256:ed8b9dd4e9f89c111f4bdb85a55f8c9f0e22796a298449380b15f627d9914095                0.0s
 => => resolve docker.io/library/python:3.9@sha256:ed8b9dd4e9f89c111f4bdb85a55f8c9f0e22796a298449380b15f627d9914095                            0.0s
 => [user-service internal] load build context                                                                                                 0.0s
 => => transferring context: 123B                                                                                                              0.0s
 => CACHED [user-service 2/4] WORKDIR /app                                                                                                     0.0s
 => CACHED [user-service 3/4] COPY . /app                                                                                                      0.0s
 => CACHED [user-service 4/4] RUN pip install flask flask_sqlalchemy psycopg2-binary                                                           0.0s
 => [user-service] exporting to image                                                                                                          0.4s
 => => exporting layers                                                                                                                        0.0s
 => => exporting manifest sha256:22088e8afab5ad659fd298c26ca53037405a88df05715b55f38eff7285db84a0                                              0.0s
 => => exporting config sha256:dac7923b4cdc7508c37c215a66a73d0e7c9b14db8ff98cfb00762782e2ad4934                                                0.0s
 => => exporting attestation manifest sha256:c34be5b4f3833ab308794c8834b88850c3b6145c8bba85f8a4d2ba8e6ce76906                                  0.0s
 => => exporting manifest list sha256:d1033acf82ed2a719bbe90409e5ff2d265c7d785ef87a1083755a523ecb65cd0                                         0.0s
 => => naming to docker.io/library/exercise_one-user-service:latest                                                                            0.0s
```



```
 => [user-service internal] load .dockerignore                                                                                                 0.0s
 => => transferring context: 2B                                                                                                                0.0s
 => [user-service 1/4] FROM docker.io/library/python:3.9@sha256:ed8b9dd4e9f89c111f4bdb85a55f8c9f0e22796a298449380b15f627d9914095                0.0s
 => => resolve docker.io/library/python:3.9@sha256:ed8b9dd4e9f89c111f4bdb85a55f8c9f0e22796a298449380b15f627d9914095                            0.0s
 => [user-service internal] load build context                                                                                                 0.0s
 => => transferring context: 123B                                                                                                              0.0s
 => CACHED [user-service 2/4] WORKDIR /app                                                                                                     0.0s
 => CACHED [user-service 3/4] COPY . /app                                                                                                      0.0s
 => CACHED [user-service 4/4] RUN pip install flask flask_sqlalchemy psycopg2-binary                                                           0.0s
 => [user-service] exporting to image                                                                                                          0.4s
 => => exporting layers                                                                                                                        0.0s
 => => exporting manifest sha256:22088e8afab5ad659fd298c26ca53037405a88df05715b55f38eff7285db84a0                                              0.0s
 => => exporting config sha256:dac7923b4cdc7508c37c215a66a73d0e7c9b14db8ff98cfb00762782e2ad4934                                                0.0s
 => => exporting attestation manifest sha256:c34be5b4f3833ab308794c8834b88850c3b6145c8bba85f8a4d2ba8e6ce76906                                  0.0s
 => => exporting manifest list sha256:d1033acf82ed2a719bbe90409e5ff2d265c7d785ef87a1083755a523ecb65cd0                                         0.0s
 => => naming to docker.io/library/exercise_one-user-service:latest                                                                            0.0s
 => => unpacking to docker.io/library/exercise_one-user-service:latest                                                                         0.3s
 => [user-service] resolving provenance for metadata file                                                                                      0.0s
[+] Running 4/4
 ✓Network library-network          Created                                                                                                     0.0s
 ✓Volume "exercise_one_db_data"    Created                                                                                                     0.0s
 ✓Container database               Started                                                                                                     0.8s
 ✓Container api                    Started                                                                                                     0.7s
```

Step 4) Let’s add the book using the post api



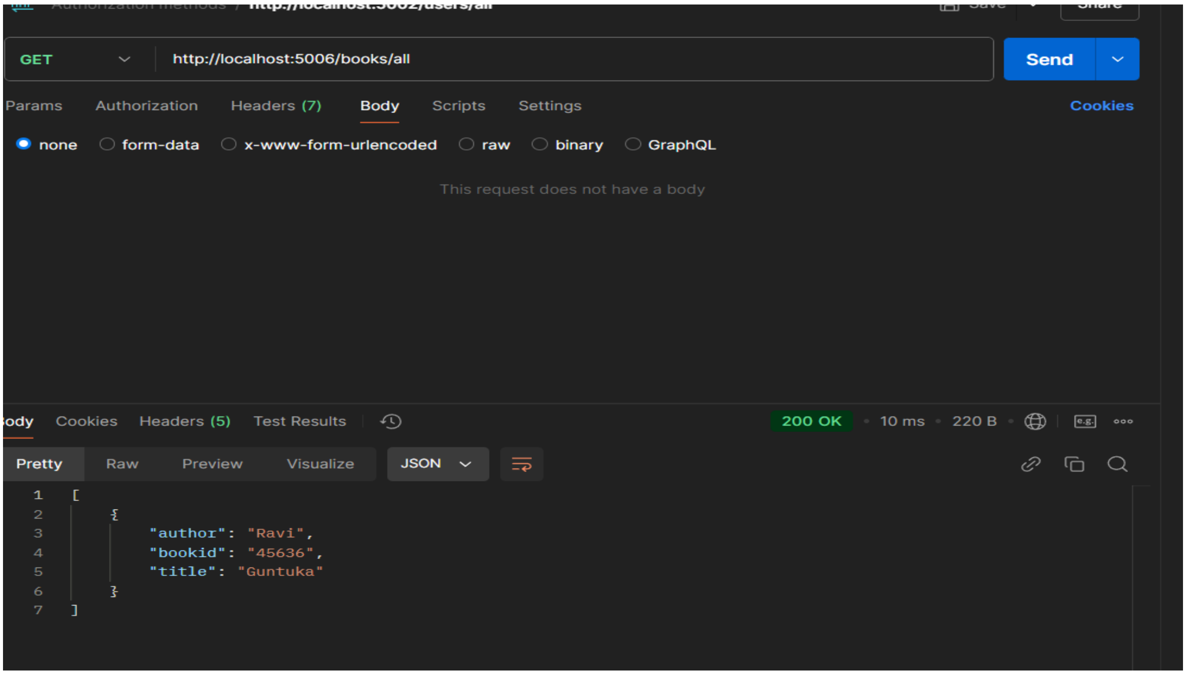
POST http://localhost:5006/books/add

```
{
  "bookid": "46536",
  "author": "Ravi",
  "title": "Guntuka"
}

```

201 CREATED • 56 ms • 223 B

```
{
    "author": "Ravi",
    "bookid": "46536",
    "title": "Guntuka"
}
```

Step 5) Let’s get the details of all the books we added using the get api



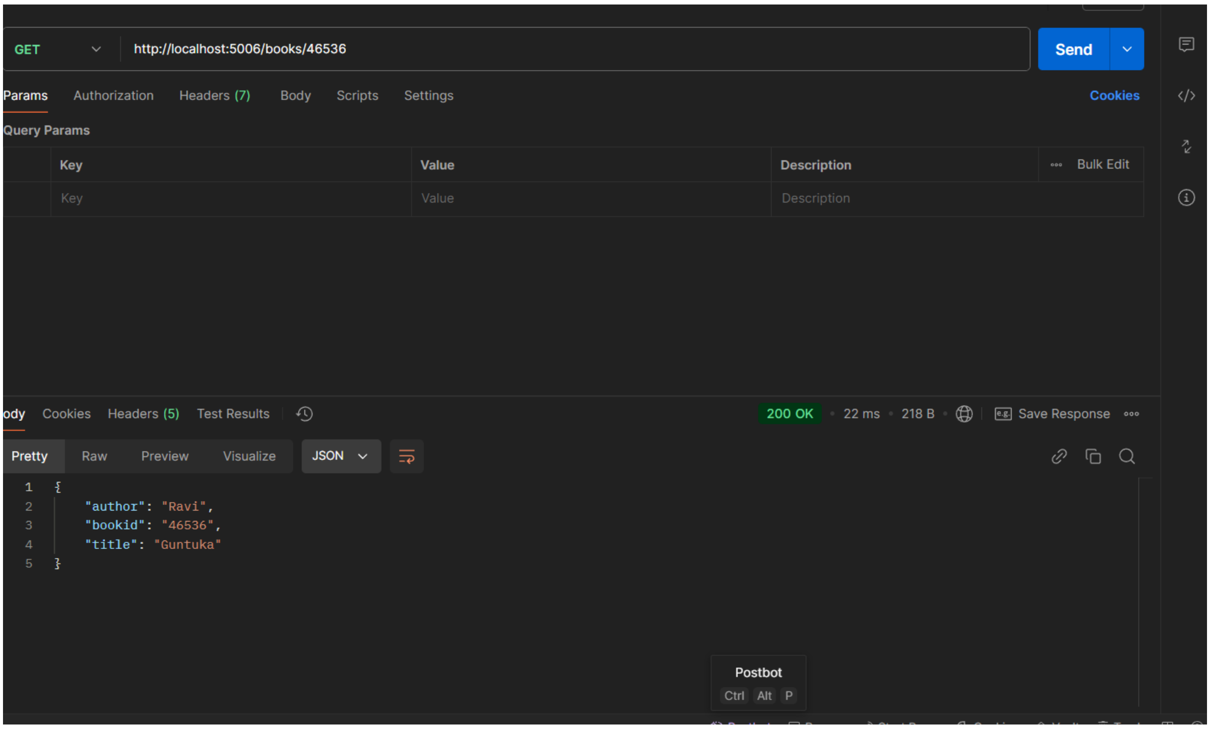
GET http://localhost:5006/books/all

This request does not have a body

200 OK • 10 ms • 220 B

```
[
    {
        "author": "Ravi",
        "bookid": "45636",
        "title": "Guntuka"
    }
]
```

Step 6) Let's get details of single book using the get request using params



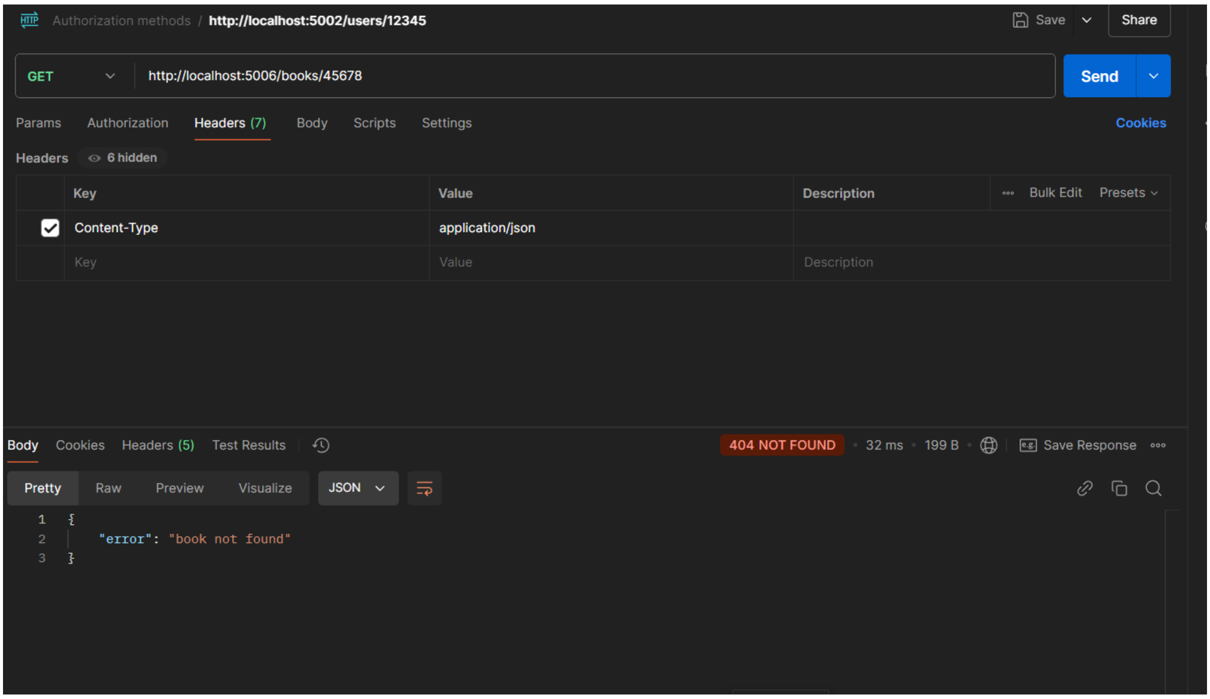
http://localhost:5006/books/46536

Key	Value	Description
Key	Value	Description

200 OK • 22 ms • 218 B

```
{
    "author": "Ravi",
    "bookid": "46536",
    "title": "Guntuka"
}
```

If no book with a particular id is found it return the below message



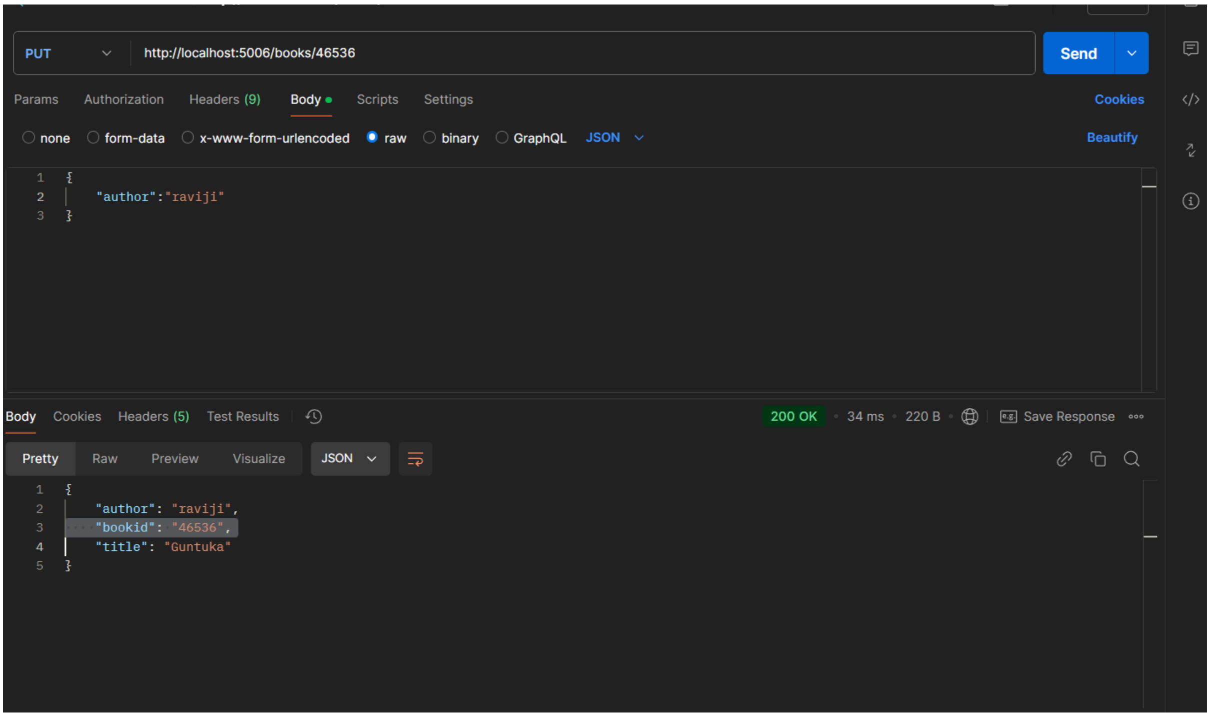
http://localhost:5006/books/45678

	Key	Value	Description
☑	Content-Type	application/json	
	Key	Value	Description

404 NOT FOUND • 32 ms • 199 B

```
{
    "error": "book not found"
}
```

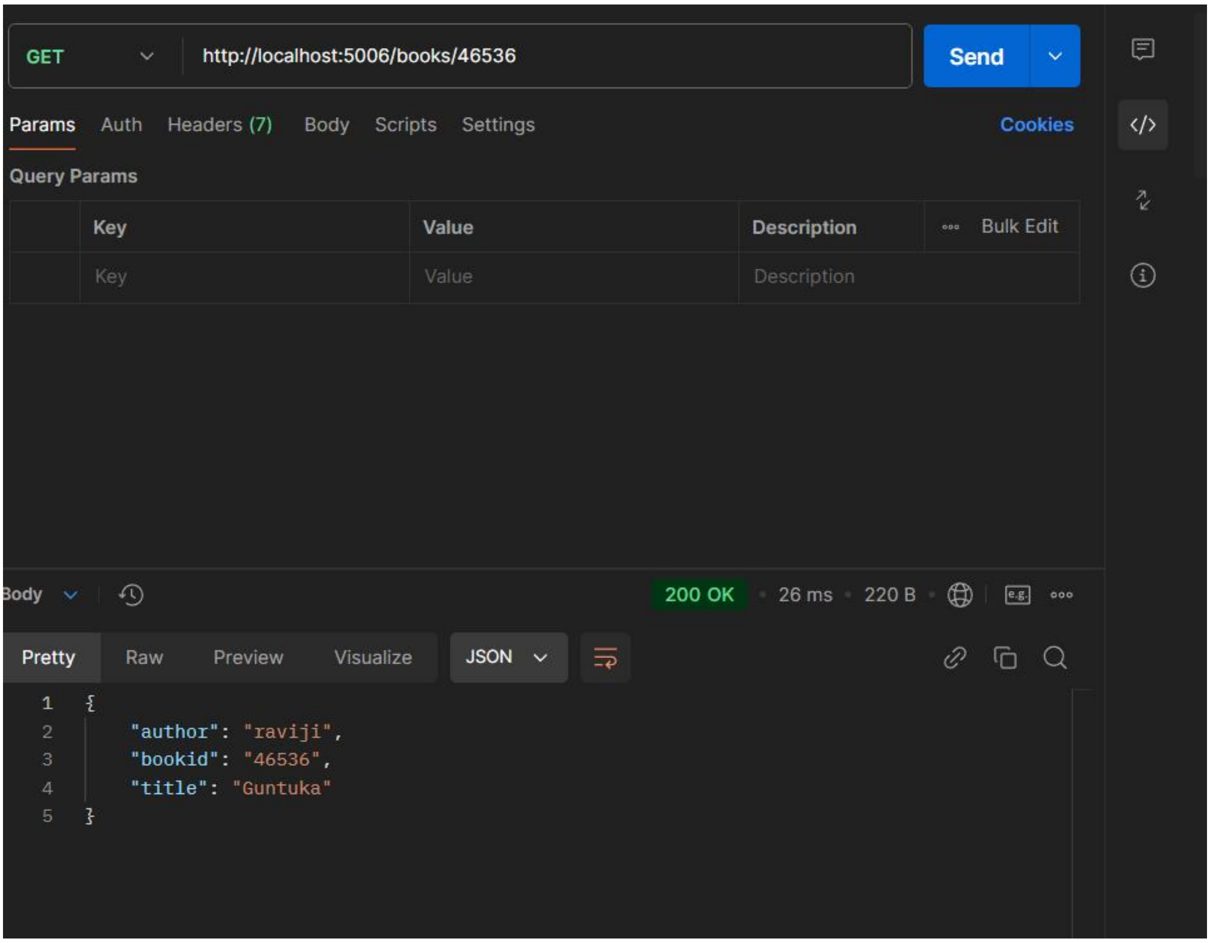
Step 7) Let's update the data of a particular user using the put request



```
{
    "author":"raviji"
}
```

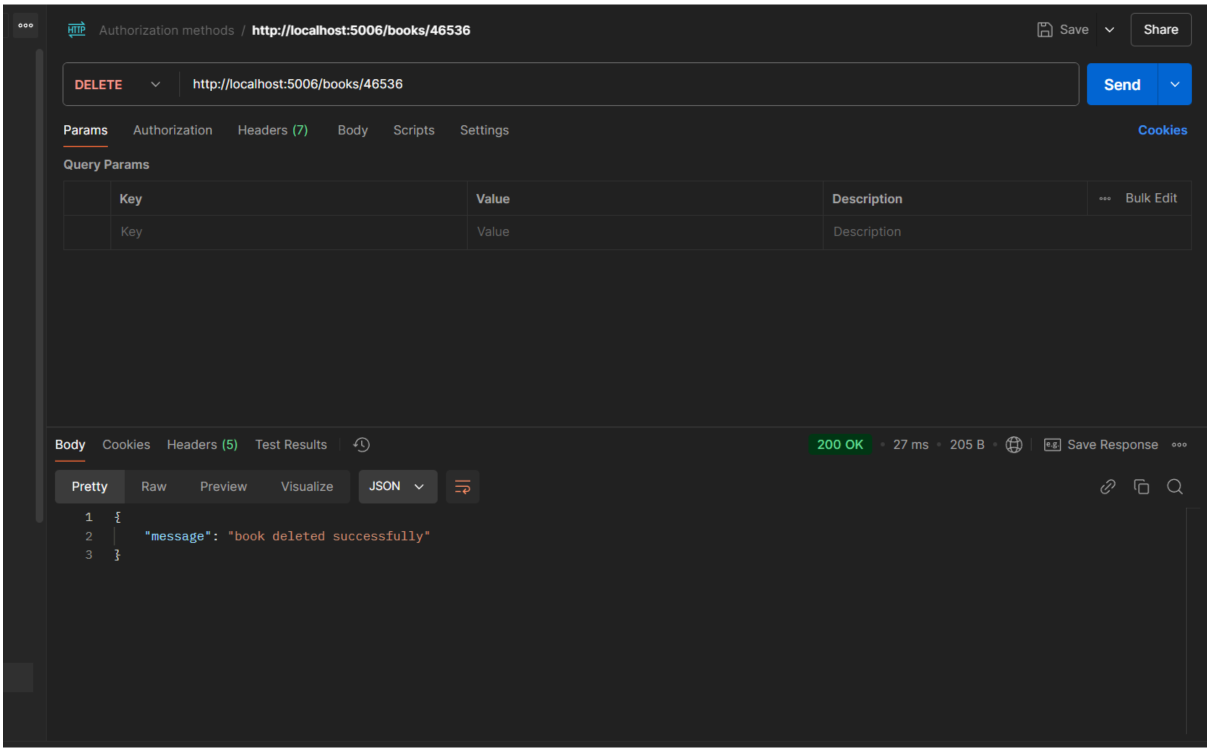
```
{
    "author": "raviji",
    "bookid": "46536",
    "title": "Guntuka"
}
```

Step 8) Let's see if the data is updated in the database using the get request



```
{
    "author": "raviji",
    "bookid": "46536",
    "title": "Guntuka"
}
```

Step 9) Let's delete one record for the db

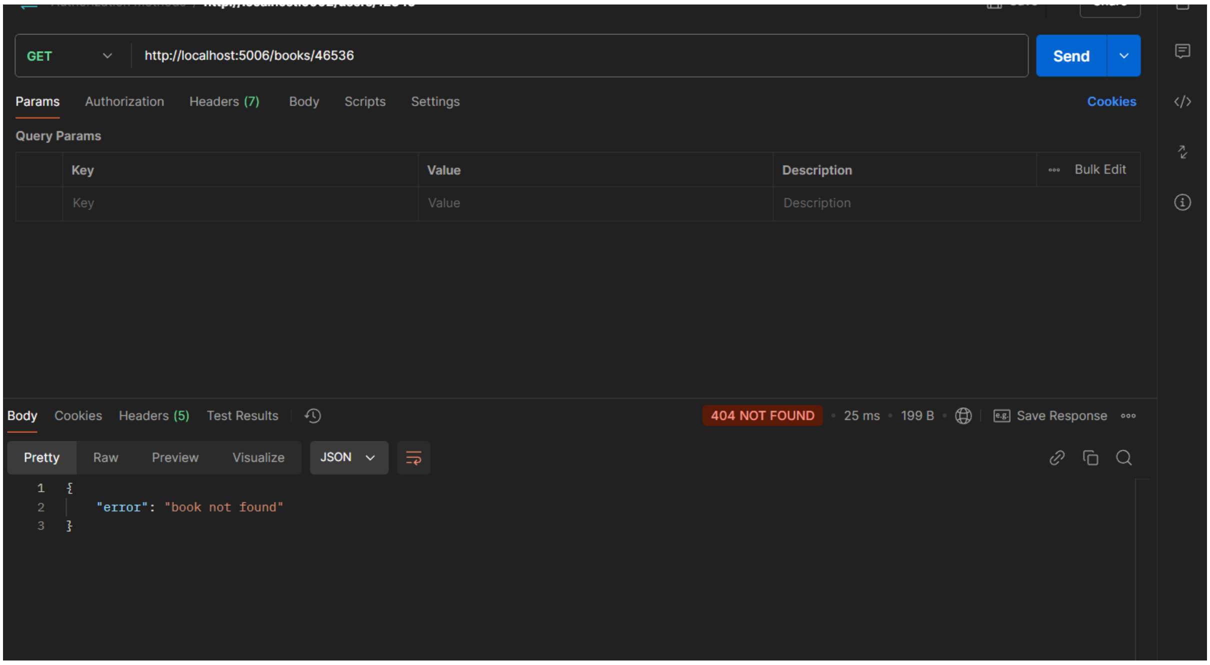


DELETE http://localhost:5006/books/46536

200 OK • 27 ms • 205 B

```
{
    "message": "book deleted successfully"
}
```

Step 10) Let's check if the record is being deleted or not



GET http://localhost:5006/books/46536

404 NOT FOUND • 25 ms • 199 B

```
{
    "error": "book not found"
}
```

So, this means if the bookid is not found, then the book is deleted successfully.

Exercise Two

Step 1) We need to duplicate exercise one to exercise two

Step 2)We need to add rabbitmq to the Docker compose.For rabbitmq we define two ports in docker-compose file.One is default port(5672) and one is default management

Step 3) We need to modify the .env file for rabbitmq credentials

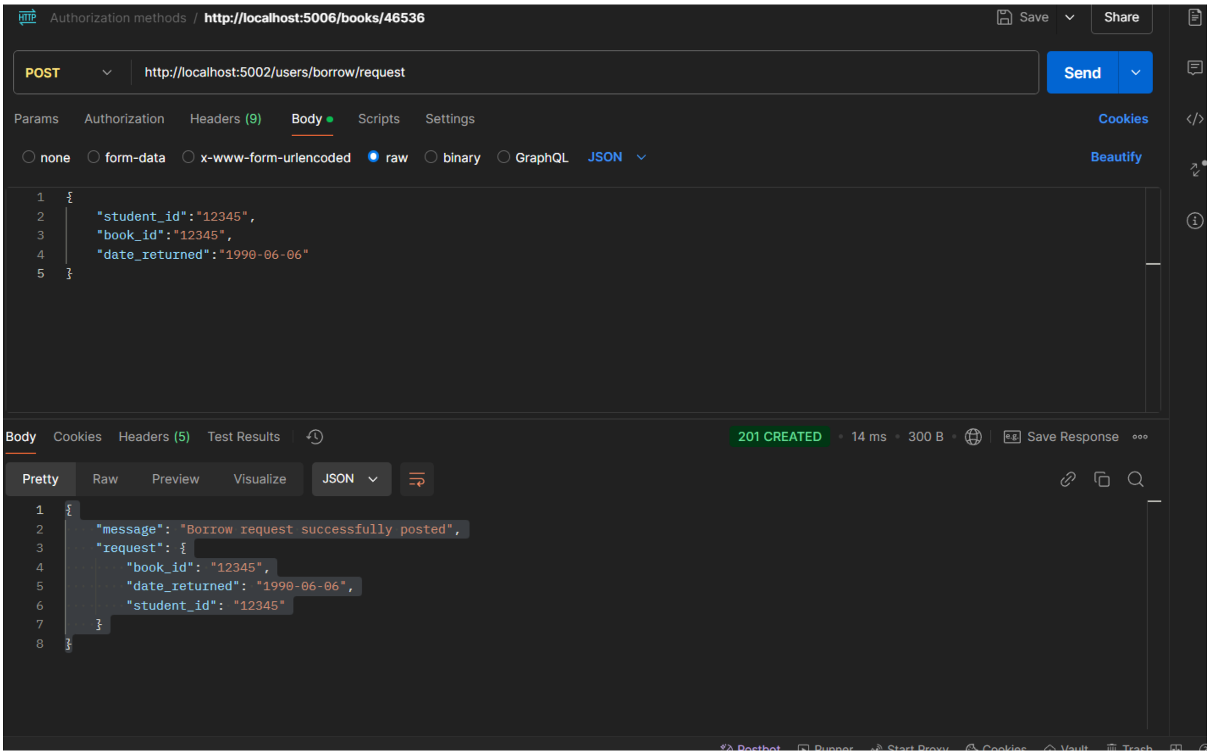
Step 4) We need to modify the user Service for it to connect with rabbitmq.We need to update main.py code for this.

Step 5) We need to create new Service i.e borrow Service that will listen to messages from user service and process them asynchronously

Step 6) We need to update docker-compose.yml to include BorrowService

Step 7) Build and Run the Services:

Now lets test the new endpoint which we have added in borrow Service which is /users/borrow/request



POST http://localhost:5002/users/borrow/request

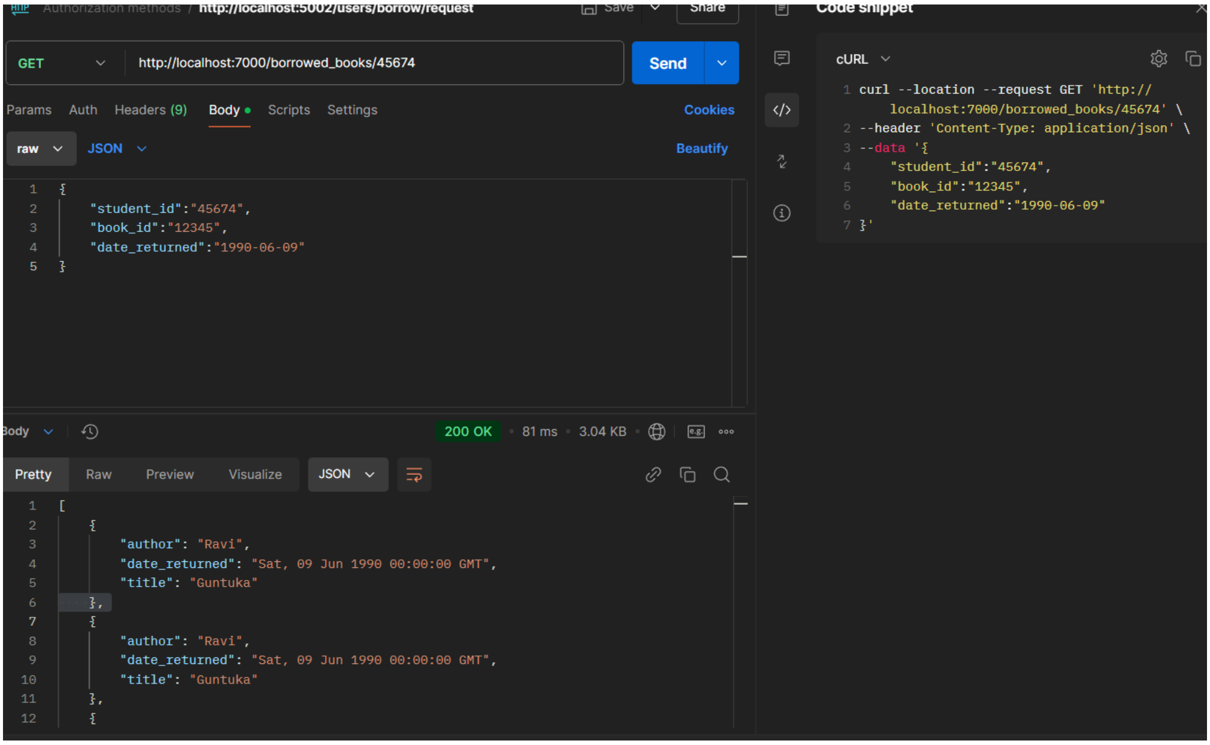
```
{
    "student_id":"12345",
    "book_id":"12345",
    "date_returned":"1990-06-06"
}
```

201 CREATED · 14 ms · 300 B

```
{
    "message": "Borrow request successfully posted",
    "request": {
        "book_id": "12345",
        "date_returned": "1990-06-06",
        "student_id": "12345"
    }
}
```

We did hit this api multiple times to test some conditions in future.

Step 8) We have created a api for list of books for a particular student id
if we hit that api, we can get list of books for a particular student id that logic is wrriten in the borrow service and its an get api with params



```
http://localhost:7000/borrowed_books/45674
```

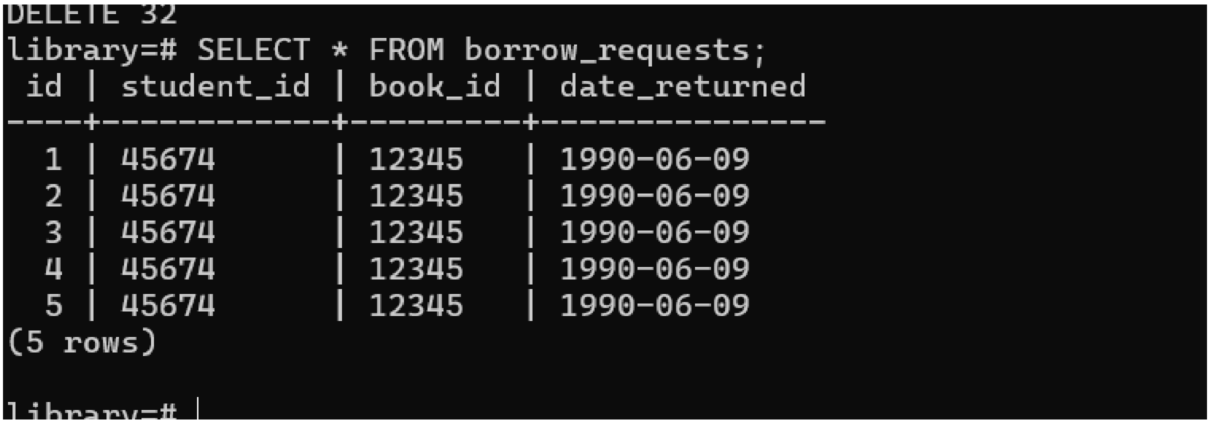
```
{
    "student_id":"45674",
    "book_id":"12345",
    "date_returned":"1990-06-09"
}
```

```
curl --location --request GET 'http://
    localhost:7000/borrowed_books/45674' \
--header 'Content-Type: application/json' \
--data '{
    "student_id":"45674",
    "book_id":"12345",
    "date_returned":"1990-06-09"
}'
```

```
[
    {
        "author": "Ravi",
        "date_returned": "Sat, 09 Jun 1990 00:00:00 GMT",
        "title": "Guntuka"
    },
    {
        "author": "Ravi",
        "date_returned": "Sat, 09 Jun 1990 00:00:00 GMT",
        "title": "Guntuka"
    },
    {
```

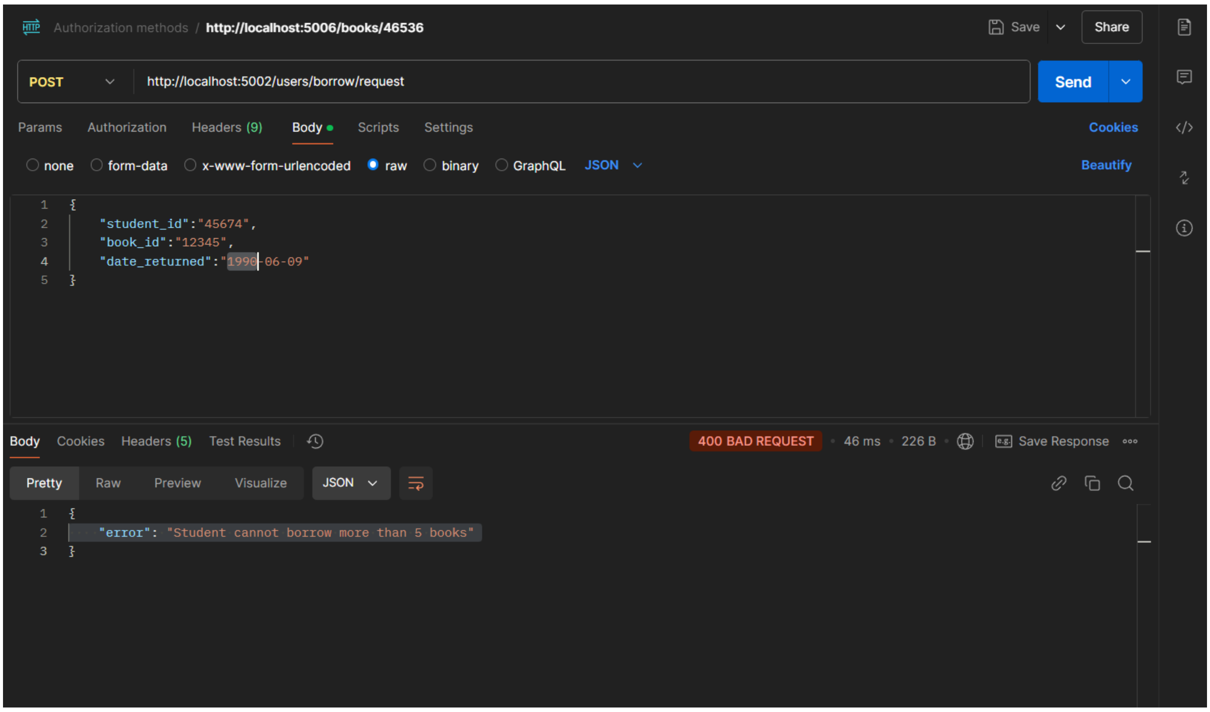
So here we get list of books borrowed by a user.So as we can see we get this list in the response.

Step 9) Let’s check the database, and to see how many records.



```
DELETE 32
library=# SELECT * FROM borrow_requests;
 id | student_id | book_id | date_returned
----+------------+---------+---------------
  1 | 45674      | 12345   | 1990-06-09
  2 | 45674      | 12345   | 1990-06-09
  3 | 45674      | 12345   | 1990-06-09
  4 | 45674      | 12345   | 1990-06-09
  5 | 45674      | 12345   | 1990-06-09
(5 rows)

library=#
```

Step 10) As we know that a particular Student cannot borrow more than 5 books at a time,so we need fetch list of books borrowed by a given user and then add condition that if he has borrowed 5 books, it would throw an error when he is borrowing the 6th book.



http://localhost:5002/users/borrow/request

```
{
    "student_id":"45674",
    "book_id":"12345",
    "date_returned":"1990-06-09"
}
```

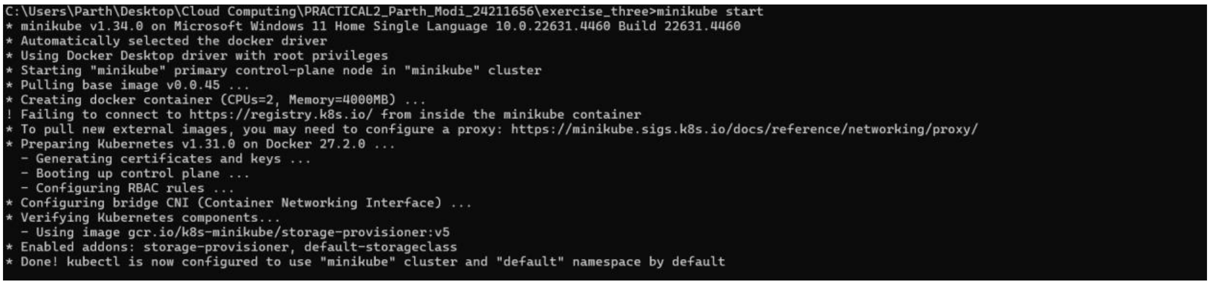
```
{
    "error": "Student cannot borrow more than 5 books"
}
```

As we saw in the early api response, this student has borrowed 5 books and when he borrows the 6th book it gives and error like this.

Exercise three

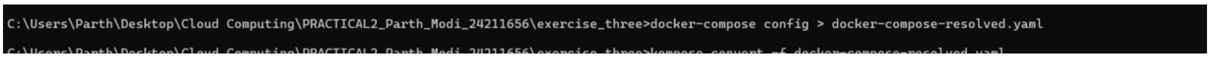
Step 1) Copy the exercise two and make it exercise three

Step 2) Start minikube cluster



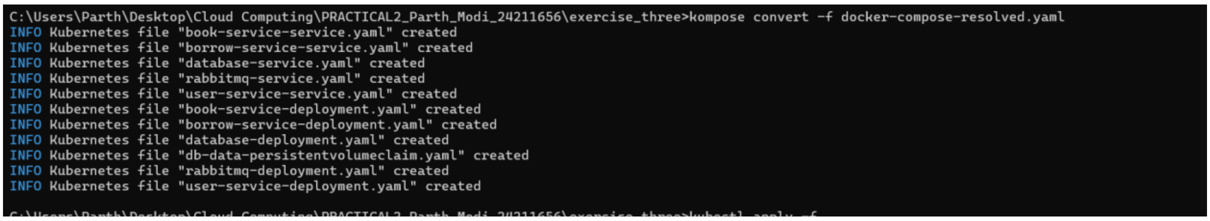
```
C:\Users\Parth\Desktop\Cloud Computing\PRACTICAL2_Parth_Modi_24211656\exercise_three>minikube start
* minikube v1.34.0 on Microsoft Windows 11 Home Single Language 10.0.22631.4460 Build 22631.4460
* Automatically selected the docker driver
* Using Docker Desktop driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.45 ...
* Creating docker container (CPUs=2, Memory=4000MB) ...
! Failing to connect to https://registry.k8s.io/ from inside the minikube container
* To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
* Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Step 3) Then we run the following command so that it validates and merges the configuration in a docker-compose.yml file, resolving variables and defaults, and then outputs the resulting configuration into a new file named docker-compose-resolved.yaml.



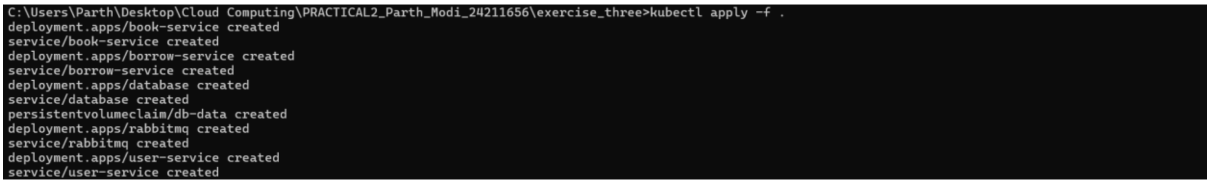
```
C:\Users\Parth\Desktop\Cloud Computing\PRACTICAL2_Parth_Modi_24211656\exercise_three>docker-compose config > docker-compose-resolved.yaml

C:\Users\Parth\Desktop\Cloud Computing\PRACTICAL2_Parth_Modi_24211656\exercise_three>kompose convert -f docker-compose-resolved.yaml
```

Step 4) The following command converts the Docker Compose configuration specified in docker-compose-resolved.yaml into Kubernetes resource files, enabling deployment to a Kubernetes cluster.



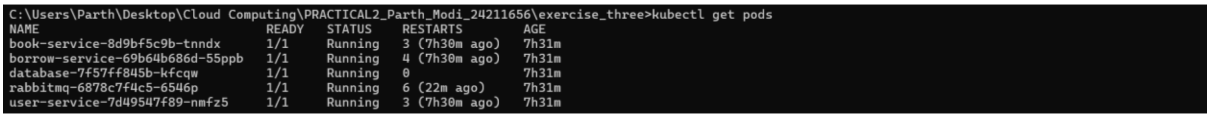
```
C:\Users\Parth\Desktop\Cloud Computing\PRACTICAL2_Parth_Modi_24211656\exercise_three>kompose convert -f docker-compose-resolved.yaml
INFO Kubernetes file "book-service-service.yaml" created
INFO Kubernetes file "borrow-service-service.yaml" created
INFO Kubernetes file "database-service.yaml" created
INFO Kubernetes file "rabbitmq-service.yaml" created
INFO Kubernetes file "user-service-service.yaml" created
INFO Kubernetes file "book-service-deployment.yaml" created
INFO Kubernetes file "borrow-service-deployment.yaml" created
INFO Kubernetes file "database-deployment.yaml" created
INFO Kubernetes file "db-data-persistentvolumeclaim.yaml" created
INFO Kubernetes file "rabbitmq-deployment.yaml" created
INFO Kubernetes file "user-service-deployment.yaml" created

C:\Users\Parth\Desktop\Cloud Computing\PRACTICAL2_Parth_Modi_24211656\exercise_three>kubectl apply -f
```

Step 5) The following command applies all Kubernetes configuration files in the current directory, creating or updating the resources (like pods, services, deployments) defined in them in the Kubernetes cluster.



```
C:\Users\Parth\Desktop\Cloud Computing\PRACTICAL2_Parth_Modi_24211656\exercise_three>kubectl apply -f .
deployment.apps/book-service created
service/book-service created
deployment.apps/borrow-service created
service/borrow-service created
deployment.apps/database created
service/database created
persistentvolumeclaim/db-data created
deployment.apps/rabbitmq created
service/rabbitmq created
deployment.apps/user-service created
service/user-service created
```

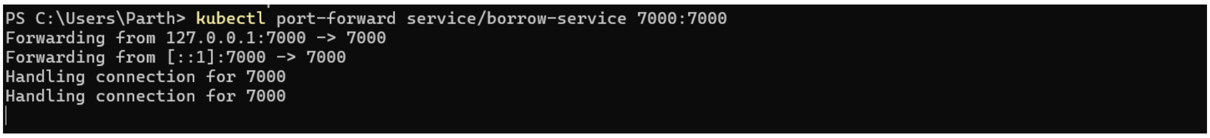
Step 6) Just check the status of the pods which you have created



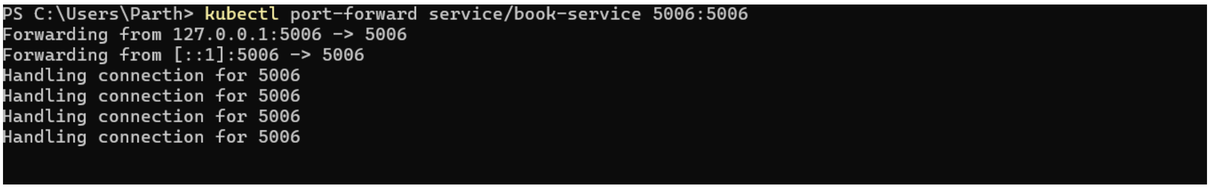
```
C:\Users\Parth\Desktop\Cloud Computing\PRACTICAL2_Parth_Modi_24211656\exercise_three>kubectl get pods
NAME                               READY   STATUS    RESTARTS        AGE
book-service-8d9bf5c9b-tnndx       1/1     Running   3 (7h30m ago)   7h31m
borrow-service-69b64b686d-55ppb    1/1     Running   4 (7h30m ago)   7h31m
database-7f57ff845b-kfcqw          1/1     Running   0               7h31m
rabbitmq-6878c7f4c5-6546p          1/1     Running   6 (22m ago)     7h31m
user-service-7d49547f89-nmfz5      1/1     Running   3 (7h30m ago)   7h31m
```

Everything is running fine

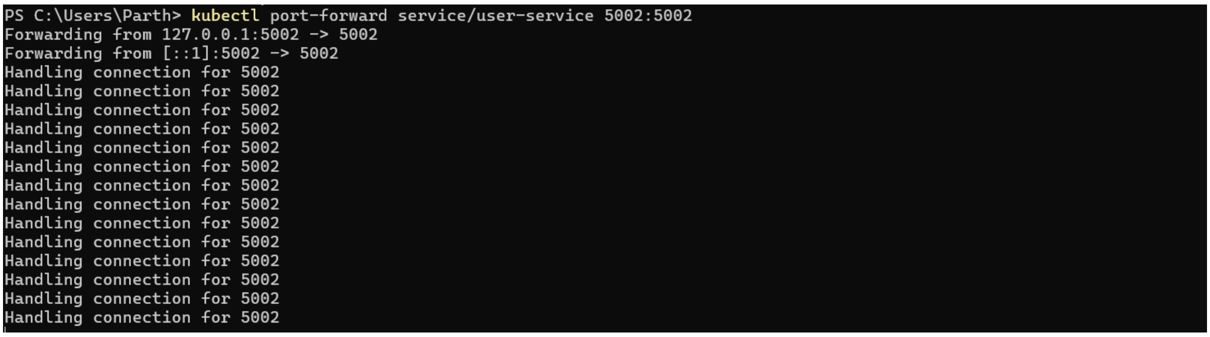
Step 7) After that, we just need to do the port forwarding, for user, borrow and book service.



```
PS C:\Users\Parth> kubectl port-forward service/borrow-service 7000:7000
Forwarding from 127.0.0.1:7000 -> 7000
Forwarding from [::1]:7000 -> 7000
Handling connection for 7000
Handling connection for 7000
```



```
PS C:\Users\Parth> kubectl port-forward service/book-service 5006:5006
Forwarding from 127.0.0.1:5006 -> 5006
Forwarding from [::1]:5006 -> 5006
Handling connection for 5006
Handling connection for 5006
Handling connection for 5006
Handling connection for 5006
```



```
PS C:\Users\Parth> kubectl port-forward service/user-service 5002:5002
Forwarding from 127.0.0.1:5002 -> 5002
Forwarding from [::1]:5002 -> 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
Handling connection for 5002
```

Conclusion: We have successfully completed the deployment of whole services on docker-compose and Kubernetes.