Click on the parent folder and venture inside it. Once within the directory, attempt to open a command prompt/ any terminal whose root should be the abovementioned folder.

Now, Install the requirements.txt file using the command -

```
$ pip install -r requirements.txt
```

Since we will be defining various schemas & models for our database and also will leverage the power of the inbuilt db module, changes will have to be made iteratively, building on top of one another. After a new model/schema is added, or tweaks are done to existing ones, one has to make migrations in Django for these changes to be reflected. The commands to enter in sequence are-

```
$ python manage.py makemigrations
```

```
$ python manage.py migrate
```

The above commands help in making the proposed changes permanent.

Lastly, to kickstart the Django application, we need to launch the development server from the terminal by using the following command

```
$ python manage.py runserver
```

I have taken the liberty of assuming certain logical and logistical assumptions while preparing the schemas of the models present in API. Real-life scenarios may greatly vary depending upon the vision of the company and the needs of its users.

As Django can closely mimic an MVC design pattern, choosing it naturally became my first choice as I wanted to follow an MVC approach in the development of this API. Additionally, MVC allows for the logical grouping of similar actions on a controller. A given model's views are also clustered together. Multiple views of a model are possible. All of these abstractions have been duly taken care of and have been kept in mind while writing code.

Broadly 4 models have been defined:-

1. ***CartItem***
2. ***Users***
3. ***Sellers***
4. ***Warehouses***

These models have been further subdivided into a grand total of 8 API endpoints, adhering to the guidelines prescribed in the take-home assessment question. These are

1. ***ShoppingCart***
2. ***ShoppingCartUpdate***
3. ***UserDetails***
4. ***UserDetailsUpdate***
5. ***SellerDetails***
6. ***SellerDetailsUpdate***
7. ***WarehouseDetails***
8. ***WarehouseDetailsUpdate***

Each model has two endpoints. One deals with the POST & GET requests, (e.g ***UserDetails***)while the other handles the PATCH & DELETE requests(e.g ***UserDetailsUpdate***). These HTTP requests help us in establishing whether the API is functioning correctly. Furthermore, it also assists in identifying if the Django application is qualified enough to handle all sorts of CRUD operations.

One has a choice to perform CRUD operations either through cURL or the plain trustworthy Django administration webpage (Link to the admin page - http://127.0.0.1:8000/admin/). I’ve also included a text file containing examples of cURL commands that can be used for API testing purposes, some of which, have already been used up.

	POST	GET	PATCH	DELETE
CartItems	✓✓	✓✓	✓✓	✓✓
Users	✓✓	✓✓	✓✓	✓✓
Sellers	✓✓	✓✓	✓✓	✓✓
Warehouses	✓✓	✓✓	✓✓	✓✓

“Checklist for CRUD Operations”

Note: Double ticks indicate that the CRUD operation for its corresponding model can be seen in both cURL and the Django admin page. Therefore it works both ways and is functioning as expected.