Introduction

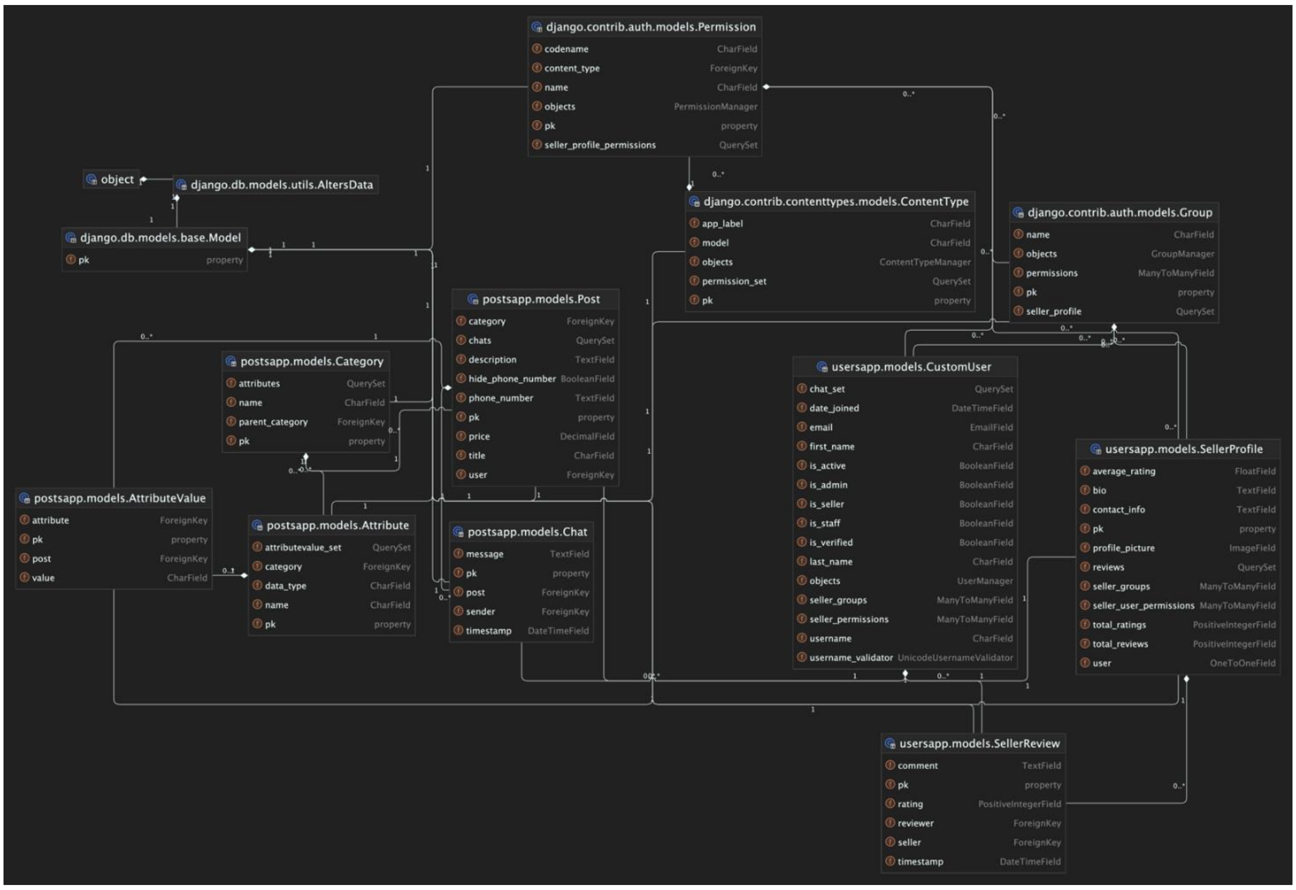
Welcome to the documentation for the e-commerce application, showcasing the intricate design and functionality of the project. This document provides an in-depth exploration of the **postsapp** and **usersapp** modules, offering insights into the database models, API endpoints, and authentication mechanisms employed within the application.

Project Overview
The e-commerce application represents the basic fundamentals of a comprehensive platform for buying and selling goods and services, designed to streamline the online shopping experience for users. Built upon the Django framework and RESTful principles, this project harnesses the power of modern web technologies to deliver a robust and scalable solution.

usersapp: User Authentication and Profiles
Complementing the functionality of the **postsapp**, the **usersapp** module handles user authentication, profile management, and seller verification. By documenting the authentication mechanisms, user models, and profile customization features, this documentation empowers developers to implement secure and personalized user experiences within the e-commerce platform.
Whether you're a developer delving into the intricacies of database design, an API consumer seeking to leverage the application's endpoints, or a contributor looking to enhance its features, this documentation serves as your comprehensive guide to understanding and extending the e-commerce application.
Let's embark on a journey through the intricacies of the **postsapp** and **usersapp**, unraveling the architecture and functionality that powers this dynamic e-commerce platform.

postsapp: Product Management and Messaging
The **postsapp** module serves as the backbone of the e-commerce platform, facilitating product management, category organization, and private messaging between users. Through detailed documentation of its database models and API endpoints, developers gain a thorough understanding of the underlying data architecture and communication channels within the application.

Here is a visual look at the project structure:



Usersapp

usersapp/models.py

This file contains the database models for user-related functionality within the e-commerce application.

CustomUser

The **CustomUser** model extends Django's built-in **AbstractUser** model to incorporate custom user fields and permissions. It includes the following fields:

- **username**: The unique username for each user.
- **password**: Stored securely using Django's password hashing.
- **email**: The email address associated with the user.
- **is_admin**: Boolean field indicating if the user is an administrator.
- **is_seller**: Boolean field indicating if the user is a seller.
- **is_verified**: Boolean field indicating if the user's account has been verified.

SellerProfile

The **SellerProfile** model represents additional information specific to sellers. It is linked to a **CustomUser** instance via a one-to-one relationship. The fields of this model are:

- **user**: A one-to-one relationship with a **CustomUser**.
- **contact_info**: Text field to store seller's contact information.
- **profile_picture**: Image field for the seller's profile picture or their company logo.
- **bio**: Text field for the seller's biography, service explanation or business introduction.
- **total_reviews**: Integer field to store the total number of reviews for the seller.
- **total_ratings**: Integer field to store the total sum of ratings given to the seller.
- **average_rating**: Float field to calculate and store the average rating of the seller based on reviews and ratings.

SellerReview

The **SellerReview** model represents reviews left by users for sellers. It includes the following fields:

- **seller**: ForeignKey linking to the **SellerProfile** being reviewed.
- **reviewer**: ForeignKey linking to the user leaving the review.
- **rating**: Integer field storing the rating given by the reviewer.
- **comment**: Text field allowing the reviewer to provide additional comments.
- **timestamp**: DateTime field indicating when the review was created.

These models collectively manage user authentication, seller profiles, and seller reviews within the e-commerce platform.

usersapp/serializers.py

This file contains serializer classes responsible for translating Django models into JSON representations and vice versa, facilitating data exchange between the client and the server.

ObtainAuthTokenSerializer

The **ObtainAuthTokenSerializer** class handles user authentication by validating username and password credentials. It includes the following fields:

- **username**: CharField for the username.
- **password**: CharField for the password (write-only).

CustomUserSerializer

The **CustomUserSerializer** class, serializes **CustomUser** objects, providing JSON representations for user data. It includes the following fields:

- **id**: Unique identifier for the user.
- **username**: Username of the user.
- **password**: Password field (write-only).
- **email**: Email address of the user.
- **is_seller**: Boolean field indicating if the user is a seller.

SellerReviewSerializer

The **SellerReviewSerializer** class serializes **SellerReview** objects, converting review data into JSON format. It includes the following fields:

- **rating**: Integer field representing the rating given by the reviewer.
- **comment**: Text field allowing the reviewer to provide additional comments.

SellerProfileSerializer

The **SellerProfileSerializer** class serializes **SellerProfile** objects, transforming seller profile data into JSON format. It includes the following fields:

- **reviews**: Serializer field representing the reviews associated with the seller profile.

CreateSellerProfileSerializer

The **CreateSellerProfileSerializer** class facilitates the creation of new seller profiles. It includes the following fields:

- **contact_info**: Text field for the seller's contact information.
- **bio**: Text field for the seller's biography.

These serializers ensure smooth data exchange between the client-side and server-side components of the e-commerce application, handling authentication, user profiles, and seller reviews.

usersapp/views.py

This file contains view classes responsible for handling incoming HTTP requests and generating appropriate responses. The views define the logic for user authentication, user registration, seller profile management, and seller reviews.

LoginView
The **LoginView** class handles user authentication by allowing users to log in with their credentials. It includes a POST method that validates the provided username and password, returning a token upon successful authentication.

RegisterUserView
The **RegisterUserView** class, facilitates user registration by allowing new users to create accounts. It includes a POST method that validates and saves user registration data, generating a token for the newly registered user.

SellerProfileDetailView
The **SellerProfileDetailView** class, retrieves detailed information about a specific seller profile. It includes a GET method that returns the serialized representation of the requested seller profile.

SellerReviewCreateView
The **SellerReviewCreateView** class, allows authenticated users to submit reviews for sellers. It includes a POST method that validates and saves review data, updating the seller's profile with the submitted review after all.

CreateSellerProfileView
The **CreateSellerProfileView** class enables authenticated users to create seller profiles. It includes a POST method that validates and saves seller profile data, ensuring that each user can have only one seller profile.

These views implement the business logic necessary for user authentication, registration, seller profile management, and seller reviews within the e-commerce application, providing endpoints for interacting with these functionalities.

usersapp/urls.py

This file contains URL patterns for routing incoming HTTP requests to the appropriate views in the **usersapp**.

API Token Authentication
The **api-token-auth/** endpoint is associated with the **LoginView** class, allowing users to authenticate and obtain an authentication token by providing their username and password.

User Registration
The **register/** endpoint is linked to the **RegisterUserView** class, enabling new users to register by providing their registration details such as username, password, and email.

Seller Profile Detail
The **seller-profile/<int:pk>/** endpoint is mapped to the **SellerProfileDetailView** class, allowing users to retrieve detailed information about a specific seller profile by providing its primary key.
Seller Review Creation
The **seller-reviews/** endpoint corresponds to the **SellerReviewCreateView** class, enabling authenticated users to submit reviews for sellers by providing their ratings and comments.

Create Seller Profile
The **create-seller-profile/** endpoint is associated with the **CreateSellerProfileView** class, allowing authenticated users to create seller profiles by providing their contact information and biography.

These URL patterns define the routes through which clients can access the various functionalities provided by the **usersapp**, including user authentication, registration, seller profile management, and seller reviews.

Postsapp

postsapp/models.py

This file defines the database models for managing posts, categories, attributes, and chats.

Category Model

The Category model represents a category for posts. It includes the following fields:

- name: CharField for the name of the category.
- parent_category: ForeignKey to itself, allowing for hierarchical categorization.

Attribute Model

The Attribute model represents attributes associated with categories. It includes the following fields:

- name: CharField for the name of the attribute.
- category: ForeignKey to Category, linking the attribute to its associated category.
- data_type: CharField indicating the data type of the attribute (e.g., String, Integer, Boolean).

Post Model

The Post model represents a product post within the platform. It includes the following fields:

- title: CharField for the title of the post.
- description: TextField for the description of the post.
- price: DecimalField for the price of the product.
- category: ForeignKey to Category, indicating the category of the post.
- user: ForeignKey to the CustomUser model, representing the user who created the post.
- hide_phone_number: BooleanField indicating whether the phone number associated with the user should be hidden in the post.
- phone_number: TextField for the phone number associated with the post.

AttributeValue Model

The AttributeValue model represents values for attributes associated with posts. It includes the following fields:

- post: ForeignKey to Post, linking the attribute value to its associated post.
- attribute: ForeignKey to Attribute, indicating the attribute to which the value belongs.
- value: CharField for the value of the attribute.

Chat Model

The Chat model represents conversations between users regarding a specific post. It includes the following fields:

- post: ForeignKey to Post, linking the chat to its associated post.
- sender: ForeignKey to CustomUser, representing the user who sent the message.
- message: TextField for the content of the chat message.
- timestamp: DateTimeField indicating when the chat message was sent.

These models collectively manage the data related to posts, categories, attributes, and chats within the e-commerce platform.

postsapp/permissions.py

This file contains custom permission classes used to control access to resources within the **postsapp** module.

IsOwnerOrReadOnly

The **IsOwnerOrReadOnly** permission class allows only the owner of an object to modify it, while allowing read-only access to others. It includes the following methods:

- **has_permission(self, request, view)**: Overrides the base method to allow read-only access for all users.
- **has_object_permission(self, request, view, obj)**: Overrides the base method to check if the requesting user is the owner of the object.

IsAdminUser

The **IsAdminUser** permission class restricts access to users with administrative privileges. It includes the following method:

- **has_permission(self, request, view)**: Overrides the base method to check if the requesting user is authenticated and is an administrator.

These permission classes help enforce access control policies within the **postsapp** module, ensuring that users can only perform actions they are authorized to perform.

postsapp/serializers.py

This file contains serializer classes responsible for converting complex data types, such as querysets and model instances, into native Python datatypes that can then be easily rendered into JSON, XML, or other content types.

ChatSerializer

The **ChatSerializer** class serializes chat objects to and from JSON representations. It includes the following fields:

- **post**: Serializer for the related post.
- **sender**: Serializer for the sender of the message.
- **message**: Serializer for the content of the message.
- **timestamp**: Serializer for the timestamp of the message.

PostSerializer

The **PostSerializer** class serializes post objects to and from JSON representations. It includes the following fields:

- **title**: Serializer for the title of the post.
- **description**: Serializer for the description of the post.
- **price**: Serializer for the price of the post.
- **category**: Serializer for the category of the post.
- **user**: Serializer for the user who created the post.
- **hide_phone_number**: Serializer for the flag indicating whether the phone number should be hidden.
- **phone_number**: Serializer for the phone number associated with the post.
- **chats**: Serializer for the related chat messages.

AttributeSerializer

The **AttributeSerializer** class serializes attribute objects to and from JSON representations. It includes the following fields:

- **name**: Serializer for the name of the attribute.
- **category**: Serializer for the category to which the attribute belongs.
- **data_type**: Serializer for the data type of the attribute.

CategorySerializer

The **CategorySerializer** class serializes category objects to and from JSON representations. It includes the following fields:

- **name**: Serializer for the name of the category.
- **attributes**: Serializer for the attributes associated with the category.

These serializer classes facilitate the conversion of complex data structures into a format that can be easily consumed by the frontend or other parts of the application.

postsapp/views.py

This file contains the view classes responsible for handling HTTP requests and returning appropriate responses. It defines various views for interacting with posts, categories, and chats within the e-commerce application.

CategoryListCreateView

The **CategoryListCreateView** class is a generic view that handles listing and creating categories. It includes the following methods:

- **get**: Retrieves a list of all categories.
- **post**: Creates a new category.

PostListCreateView

The **PostListCreateView** class is a generic view that handles listing and creating posts. It includes the following methods:

- **get_queryset**: Retrieves a queryset of all posts based on search queries and filters.
- **perform_create**: Creates a new post, sets the user field, and handles phone number visibility.
- **get_category_attributes**: Retrieves attributes associated with a category.

PostRetrieveUpdateDeleteView

The **PostRetrieveUpdateDeleteView** class is a generic view that handles retrieving, updating, and deleting posts. It includes the following methods:

- **get_queryset**: Retrieves a queryset of all posts.
- **perform_create**: Creates a new post, sets the user field, and handles phone number visibility.
- **get_category_attributes**: Retrieves attributes associated with a category.

ChatListCreateView

The **ChatListCreateView** class is a generic view that handles listing and creating chat messages. It includes the following methods:

- **perform_create**: Creates a new chat message and sets the sender field.
- **list**: Retrieves a list of all chat messages and includes chats in the response.

These views define the logic for interacting with posts, categories, and chat messages, ensuring proper handling of HTTP requests and responses within the e-commerce application.

postsapp/urls.py

This file contains the URL patterns for routing HTTP requests to the appropriate view functions within the postsapp application of the e-commerce project.

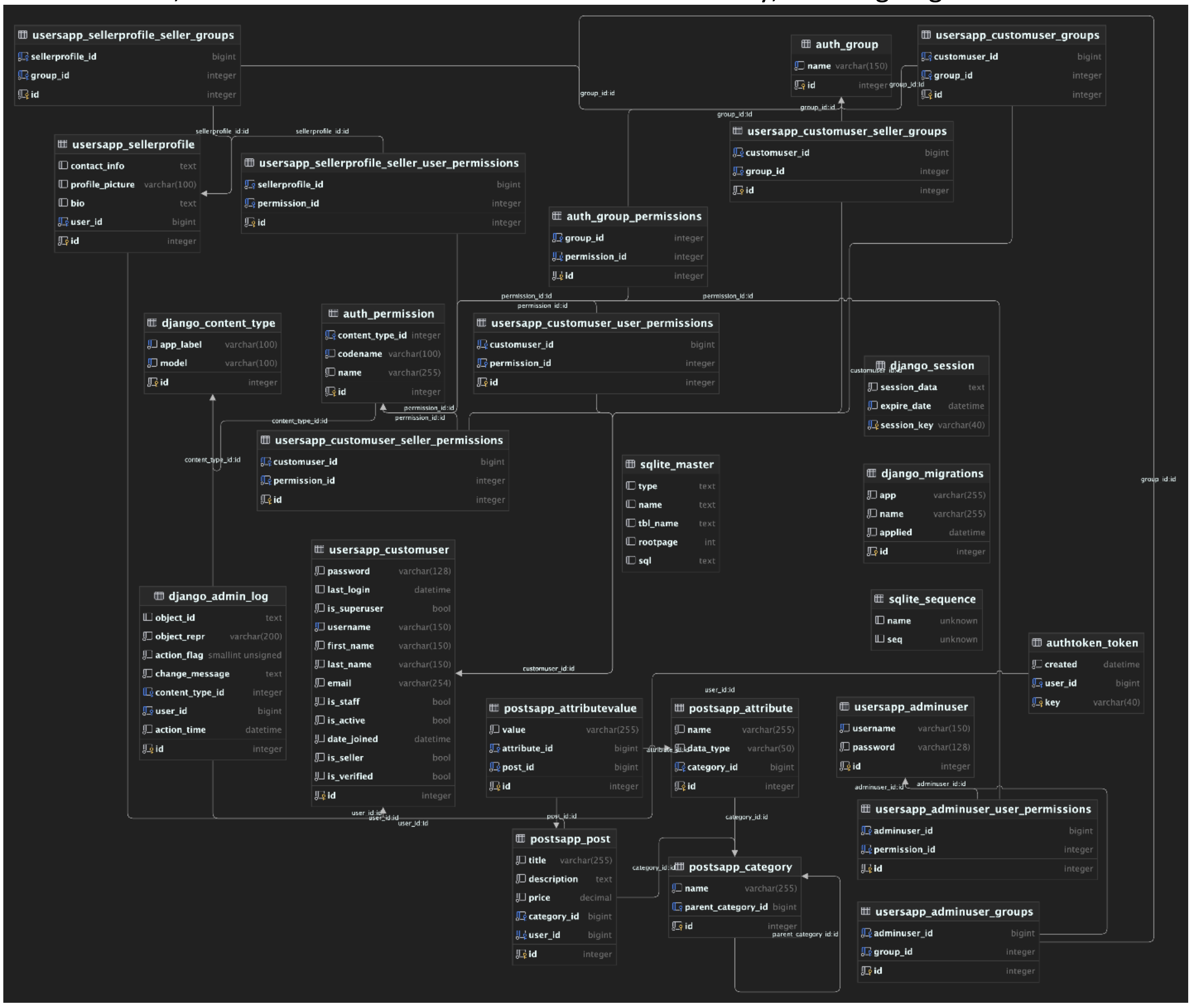
urlpatterns

The **urlpatterns** list defines the mapping between URL patterns and view functions. It includes the following patterns:

- **/posts/**: Maps to **PostListCreateView** for listing and creating posts.
- **/posts/<int:pk>/**: Maps to **PostRetrieveUpdateDeleteView** for retrieving, updating, and deleting a specific post.
- **/categories/**: Maps to **CategoryListCreateView** for listing and creating categories.

These URL patterns ensure that HTTP requests to specific endpoints are routed to the corresponding view functions, allowing for the proper handling of user interactions with posts and categories within the e-commerce application.

With all these, it's better to take look at the Data-Base visually, before going for tests:


usersapp_sellerprofile_seller_groups: sellerprofile_id bigint, group_id integer, id integer
auth_group: name varchar(150), id integer
usersapp_customuser_groups: customuser_id bigint, group_id integer, id integer
usersapp_sellerprofile: contact_info text, profile_picture varchar(100), bio text, user_id bigint, id integer
usersapp_sellerprofile_seller_user_permissions: sellerprofile_id bigint, permission_id integer, id integer
usersapp_customuser_seller_groups: customuser_id bigint, group_id integer, id integer
auth_group_permissions: group_id integer, permission_id integer, id integer
django_content_type: app_label varchar(100), model varchar(100), id integer
auth_permission: content_type_id integer, codename varchar(100), name varchar(255), id integer
usersapp_customuser_user_permissions: customuser_id bigint, permission_id integer, id integer
django_session: session_data text, expire_date datetime, session_key varchar(40)
usersapp_customuser_seller_permissions: customuser_id bigint, permission_id integer, id integer
sqlite_master: type text, name text, tbl_name text, rootpage int, sql text
django_migrations: app varchar(255), name varchar(255), applied datetime, id integer
usersapp_customuser: password varchar(128), last_login datetime, is_superuser bool, username varchar(150), first_name varchar(150), last_name varchar(150), email varchar(254), is_staff bool, is_active bool, date_joined datetime, is_seller bool, is_verified bool, id integer
django_admin_log: object_id text, object_repr varchar(200), action_flag smallint unsigned, change_message text, content_type_id integer, user_id bigint, action_time datetime, id integer
sqlite_sequence: name unknown, seq unknown
authtoken_token: created datetime, user_id bigint, key varchar(40)
postsapp_attributevalue: value varchar(255), attribute_id bigint, post_id bigint, id integer
postsapp_attribute: name varchar(255), data_type varchar(50), category_id bigint, id integer
usersapp_adminuser: username varchar(150), password varchar(128), id integer
usersapp_adminuser_user_permissions: adminuser_id bigint, permission_id integer, id integer
postsapp_post: title varchar(255), description text, price decimal, category_id bigint, user_id bigint, id integer
postsapp_category: name varchar(255), parent_category_id bigint, id integer
usersapp_adminuser_groups: adminuser_id bigint, group_id integer, id integer
group_id:id
sellerprofile_id:id
permission_id:id
content_type_id:id
customuser_id:id
user_id:id
post_id:id
attribute_id:id
category_id:id
parent_category_id:id
adminuser_id:id

Tests

The e-commerce project includes comprehensive testing to ensure the reliability and functionality of its features. Tests are available within the **test.py** file of each Django app in the project files. These tests cover various aspects of the application, including API endpoints, database interactions, and user authentication.

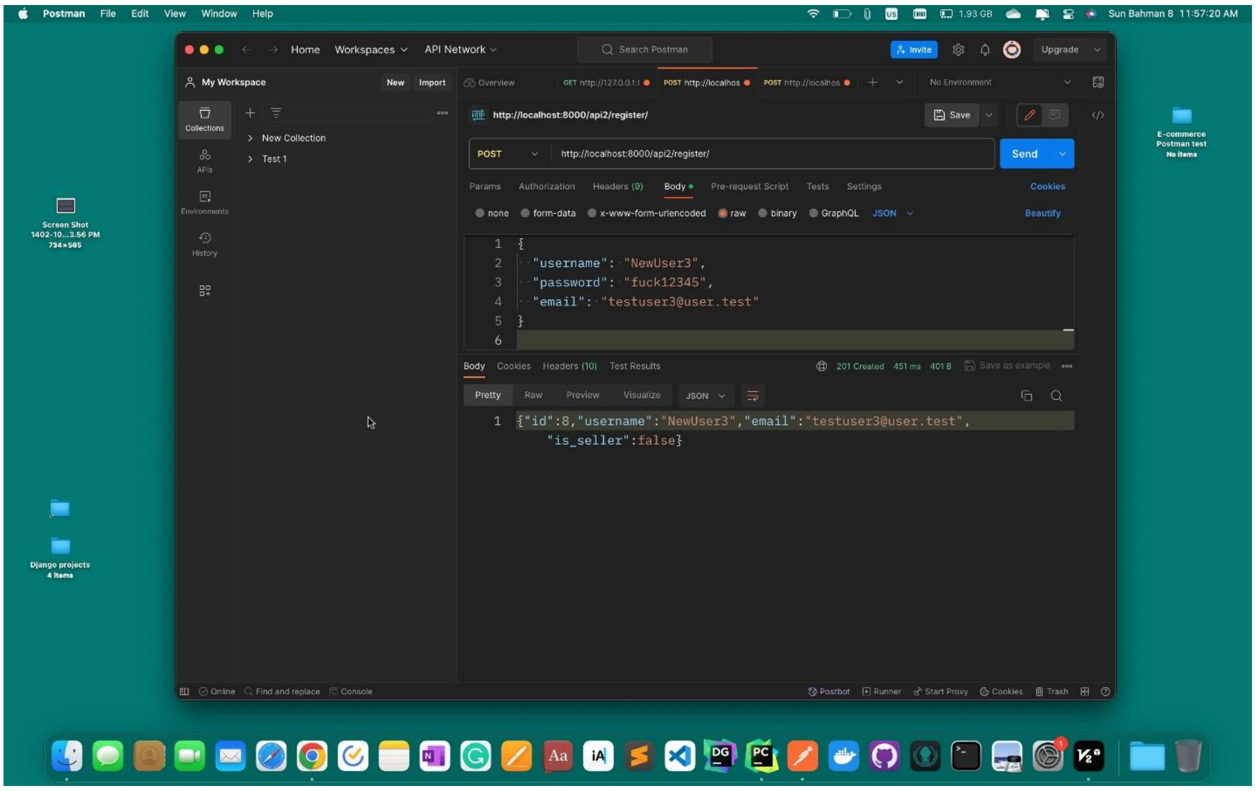
although it was possible to use DRF browsable api interface, I preferred to use Postman to be more practical in terms of being able to archive tests in python syntax at test.py file of each Django app.

Each test within the **test.py** files is designed to run independently, allowing developers to execute specific tests as needed without running the entire test suite. This modular approach to testing enhances efficiency during development and debugging processes.

Additionally, screenshots of tests conducted using Postman are provided below. These screenshots offer visual verification of the application's behavior and responses to HTTP requests, further validating its functionality and adherence to requirements.

User Registration

Created:



POST http://localhost:8000/api2/register/

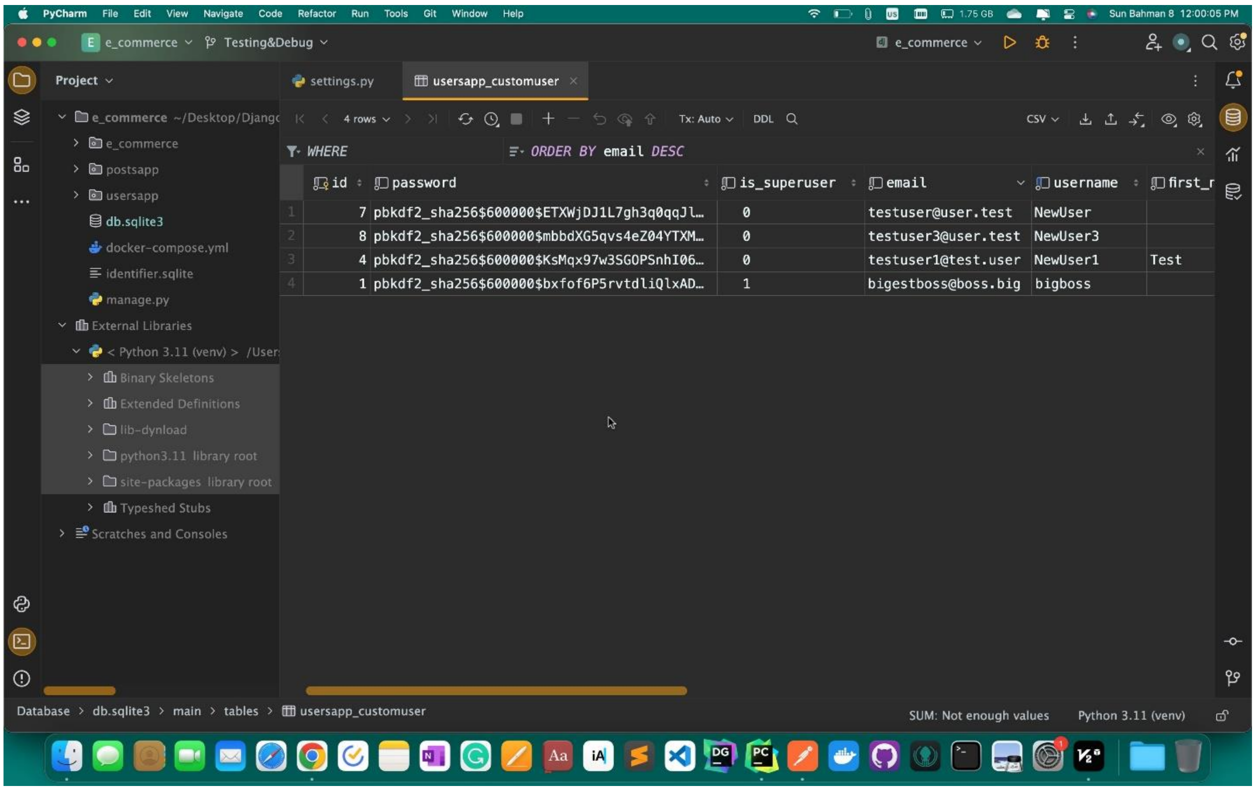
Body — raw — JSON

```
{
  "username": "NewUser3",
  "password": "fuck12345",
  "email": "testuser3@user.test"
}
```

201 Created 451 ms 401 B

```
{"id":8,"username":"NewUser3","email":"testuser3@user.test",
    "is_seller":false}
```

In DataBase with hashed Password:

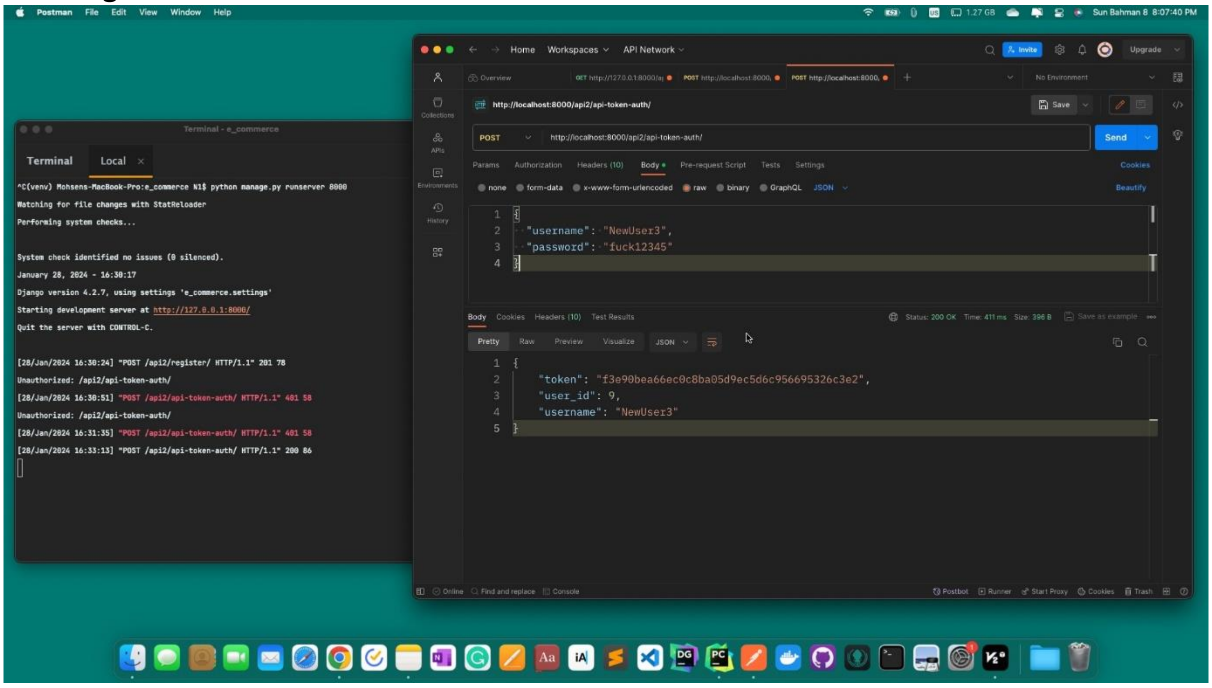


usersapp_customuser — 4 rows

WHERE ORDER BY email DESC

	id	password	is_superuser	email	username	first_r
1	7	pbkdf2_sha256$600000$ETXWjDJ1L7gh3q0qqJl…	0	testuser@user.test	NewUser	
2	8	pbkdf2_sha256$600000$mbbdXG5qvs4eZ04YTXM…	0	testuser3@user.test	NewUser3	
3	4	pbkdf2_sha256$600000$KsMqx97w3SGOPSnhI06…	0	testuser1@test.user	NewUser1	Test
4	1	pbkdf2_sha256$600000$bxfof6P5rvtdliQlxAD…	1	bigestboss@boss.big	bigboss	

User Login:



```
^C(venv) Mohsens-MacBook-Pro:e_commerce N1$ python manage.py runserver 8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 28, 2024 - 16:30:17
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[28/Jan/2024 16:30:24] "POST /api2/register/ HTTP/1.1" 201 78
Unauthorized: /api2/api-token-auth/
[28/Jan/2024 16:30:51] "POST /api2/api-token-auth/ HTTP/1.1" 401 58
Unauthorized: /api2/api-token-auth/
[28/Jan/2024 16:31:35] "POST /api2/api-token-auth/ HTTP/1.1" 401 58
[28/Jan/2024 16:33:13] "POST /api2/api-token-auth/ HTTP/1.1" 200 86
```

POST http://localhost:8000/api2/api-token-auth/

Body: raw JSON

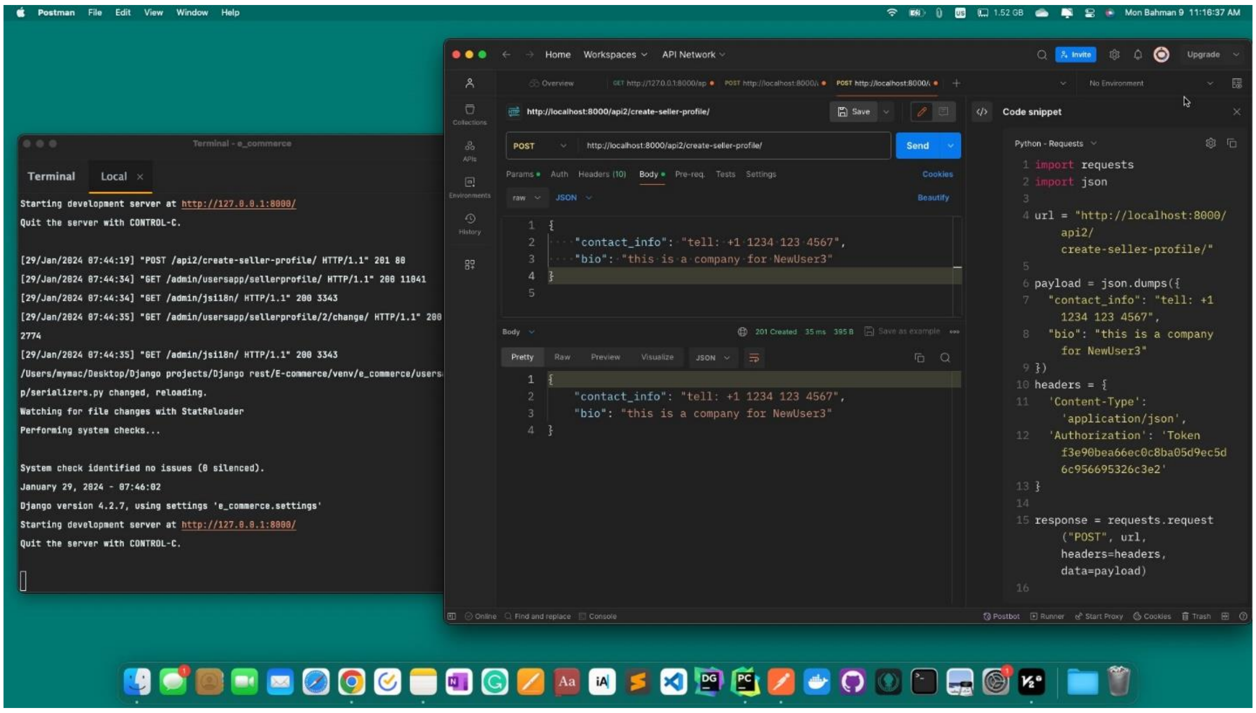
```
{
  "username": "NewUser3",
  "password": "fuck12345"
}
```

Status: 200 OK Time: 411 ms Size: 396 B

```
{
    "token": "f3e90bea66ec0c8ba05d9ec5d6c956695326c3e2",
    "user_id": 9,
    "username": "NewUser3"
}
```

Seller Profile:

created



```
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[29/Jan/2024 07:44:19] "POST /api2/create-seller-profile/ HTTP/1.1" 201 80
[29/Jan/2024 07:44:34] "GET /admin/usersapp/sellerprofile/ HTTP/1.1" 200 11041
[29/Jan/2024 07:44:34] "GET /admin/jsi18n/ HTTP/1.1" 200 3343
[29/Jan/2024 07:44:35] "GET /admin/usersapp/sellerprofile/2/change/ HTTP/1.1" 200
2774
[29/Jan/2024 07:44:35] "GET /admin/jsi18n/ HTTP/1.1" 200 3343
/Users/mymac/Desktop/Django projects/Django rest/E-commerce/venv/e_commerce/users
p/serializers.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 29, 2024 - 07:46:02
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

POST http://localhost:8000/api2/create-seller-profile/

Body: raw JSON

```
{
    "contact_info": "tell: +1 1234 123 4567",
    "bio": "this is a company for NewUser3"
}
```

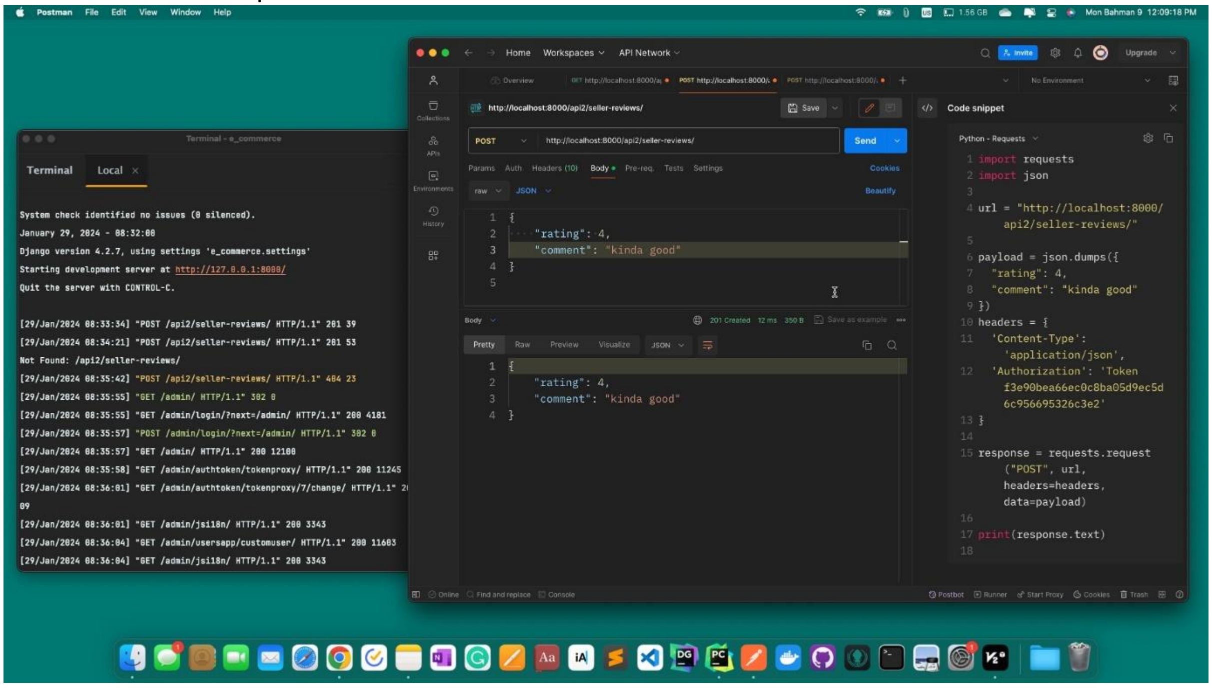
201 Created 35 ms 395 B

```
{
    "contact_info": "tell: +1 1234 123 4567",
    "bio": "this is a company for NewUser3"
}
```

Code snippet — Python - Requests

```
import requests
import json

url = "http://localhost:8000/
    api2/
    create-seller-profile/"

payload = json.dumps({
  "contact_info": "tell: +1
    1234 123 4567",
  "bio": "this is a company
    for NewUser3"
})
headers = {
  'Content-Type':
    'application/json',
  'Authorization': 'Token
    f3e90bea66ec0c8ba05d9ec5d
    6c956695326c3e2'
}

response = requests.request
    ("POST", url,
    headers=headers,
    data=payload)
```

Comment and rate performed:



```
System check identified no issues (0 silenced).
January 29, 2024 - 08:32:00
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[29/Jan/2024 08:33:34] "POST /api2/seller-reviews/ HTTP/1.1" 201 39
[29/Jan/2024 08:34:21] "POST /api2/seller-reviews/ HTTP/1.1" 201 53
Not Found: /api2/seller-reviews/
[29/Jan/2024 08:35:42] "POST /api2/seller-reviews/ HTTP/1.1" 404 23
[29/Jan/2024 08:35:55] "GET /admin/ HTTP/1.1" 302 0
[29/Jan/2024 08:35:55] "GET /admin/login/?next=/admin/ HTTP/1.1" 200 4181
[29/Jan/2024 08:35:57] "POST /admin/login/?next=/admin/ HTTP/1.1" 302 0
[29/Jan/2024 08:35:57] "GET /admin/ HTTP/1.1" 200 12100
[29/Jan/2024 08:35:58] "GET /admin/authtoken/tokenproxy/ HTTP/1.1" 200 11245
[29/Jan/2024 08:36:01] "GET /admin/authtoken/tokenproxy/7/change/ HTTP/1.1" 2
09
[29/Jan/2024 08:36:01] "GET /admin/jsi18n/ HTTP/1.1" 200 3343
[29/Jan/2024 08:36:04] "GET /admin/usersapp/customuser/ HTTP/1.1" 200 11603
[29/Jan/2024 08:36:04] "GET /admin/jsi18n/ HTTP/1.1" 200 3343
```

POST http://localhost:8000/api2/seller-reviews/

```
{
    "rating": 4,
    "comment": "kinda good"
}
```

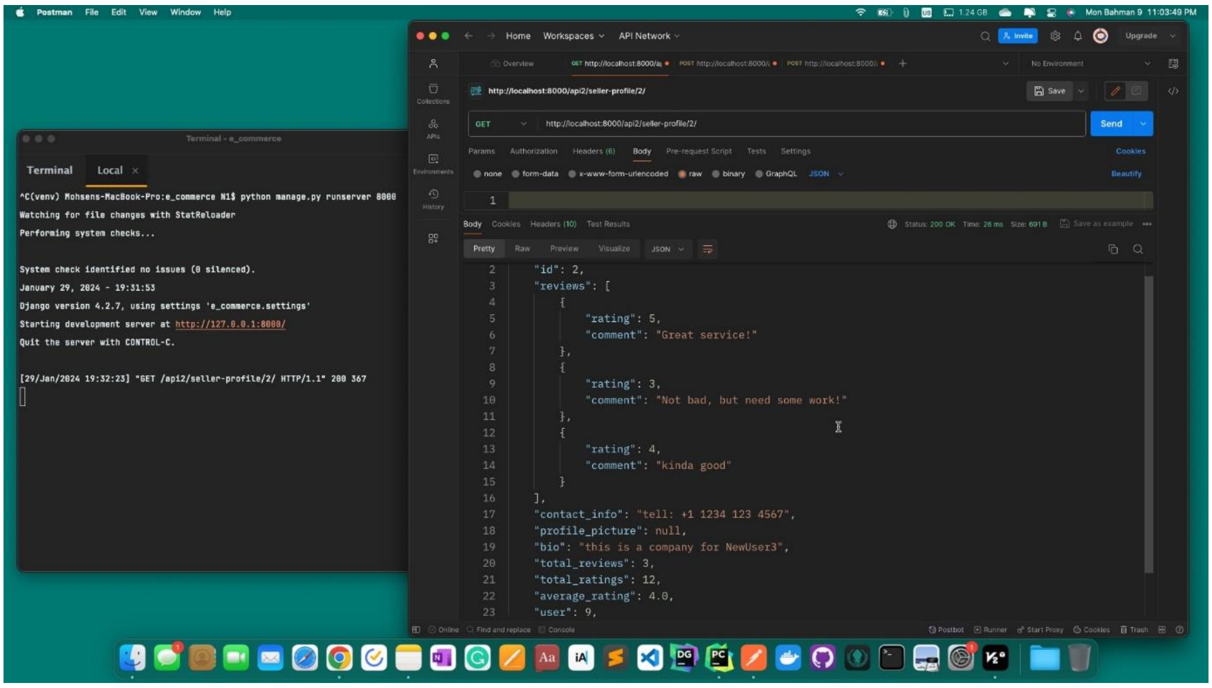
201 Created 12 ms 350 B

```
{
    "rating": 4,
    "comment": "kinda good"
}
```

Code snippet — Python - Requests

```
import requests
import json

url = "http://localhost:8000/
    api2/seller-reviews/"

payload = json.dumps({
  "rating": 4,
  "comment": "kinda good"
})
headers = {
  'Content-Type':
    'application/json',
  'Authorization': 'Token
    f3e90bea66ec0c8ba05d9ec5d
    6c956695326c3e2'
}

response = requests.request
    ("POST", url,
    headers=headers,
    data=payload)

print(response.text)
```

then fetched with details:



```
^C(venv) Mohsens-MacBook-Pro:e_commerce N1$ python manage.py runserver 8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 29, 2024 - 19:31:53
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[29/Jan/2024 19:32:23] "GET /api2/seller-profile/2/ HTTP/1.1" 200 367
```

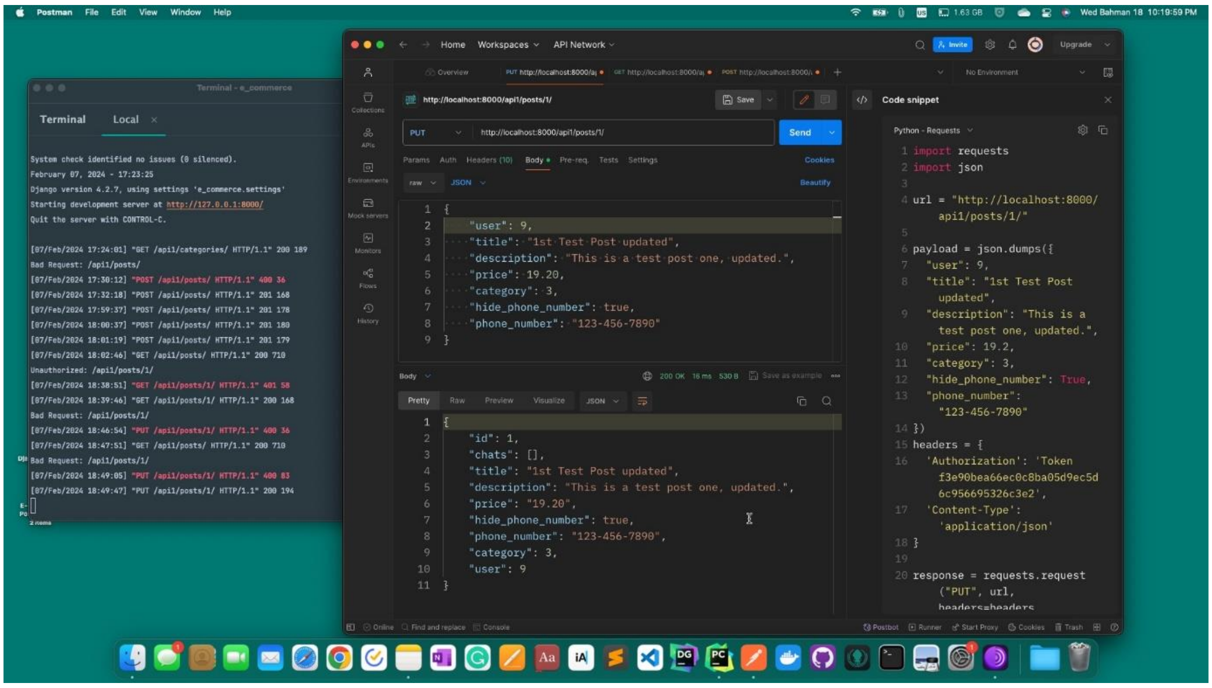
GET http://localhost:8000/api2/seller-profile/2/

Status: 200 OK Time: 26 ms Size: 691 B

```
    "id": 2,
    "reviews": [
        {
            "rating": 5,
            "comment": "Great service!"
        },
        {
            "rating": 3,
            "comment": "Not bad, but need some work!"
        },
        {
            "rating": 4,
            "comment": "kinda good"
        }
    ],
    "contact_info": "tell: +1 1234 123 4567",
    "profile_picture": null,
    "bio": "this is a company for NewUser3",
    "total_reviews": 3,
    "total_ratings": 12,
    "average_rating": 4.0,
    "user": 9,
```

Posts:

First to fourth Post, created



```
System check identified no issues (0 silenced).
February 07, 2024 - 17:23:25
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[07/Feb/2024 17:24:01] "GET /api1/categories/ HTTP/1.1" 200 189
Bad Request: /api1/posts/
[07/Feb/2024 17:30:12] "POST /api1/posts/ HTTP/1.1" 400 36
[07/Feb/2024 17:32:18] "POST /api1/posts/ HTTP/1.1" 201 168
[07/Feb/2024 17:59:37] "POST /api1/posts/ HTTP/1.1" 201 178
[07/Feb/2024 18:00:37] "POST /api1/posts/ HTTP/1.1" 201 180
[07/Feb/2024 18:01:19] "POST /api1/posts/ HTTP/1.1" 201 179
[07/Feb/2024 18:02:46] "GET /api1/posts/ HTTP/1.1" 200 710
Unauthorized: /api1/posts/1/
[07/Feb/2024 18:38:51] "GET /api1/posts/1/ HTTP/1.1" 401 58
[07/Feb/2024 18:39:46] "GET /api1/posts/1/ HTTP/1.1" 200 168
Bad Request: /api1/posts/1/
[07/Feb/2024 18:46:54] "PUT /api1/posts/1/ HTTP/1.1" 400 36
[07/Feb/2024 18:47:51] "GET /api1/posts/ HTTP/1.1" 200 710
Bad Request: /api1/posts/1/
[07/Feb/2024 18:49:05] "PUT /api1/posts/1/ HTTP/1.1" 400 83
[07/Feb/2024 18:49:47] "PUT /api1/posts/1/ HTTP/1.1" 200 194
```

PUT http://localhost:8000/api1/posts/1/

```
{
    "user": 9,
    "title": "1st Test Post updated",
    "description": "This is a test post one, updated.",
    "price": 19.20,
    "category": 3,
    "hide_phone_number": true,
    "phone_number": "123-456-7890"
}
```

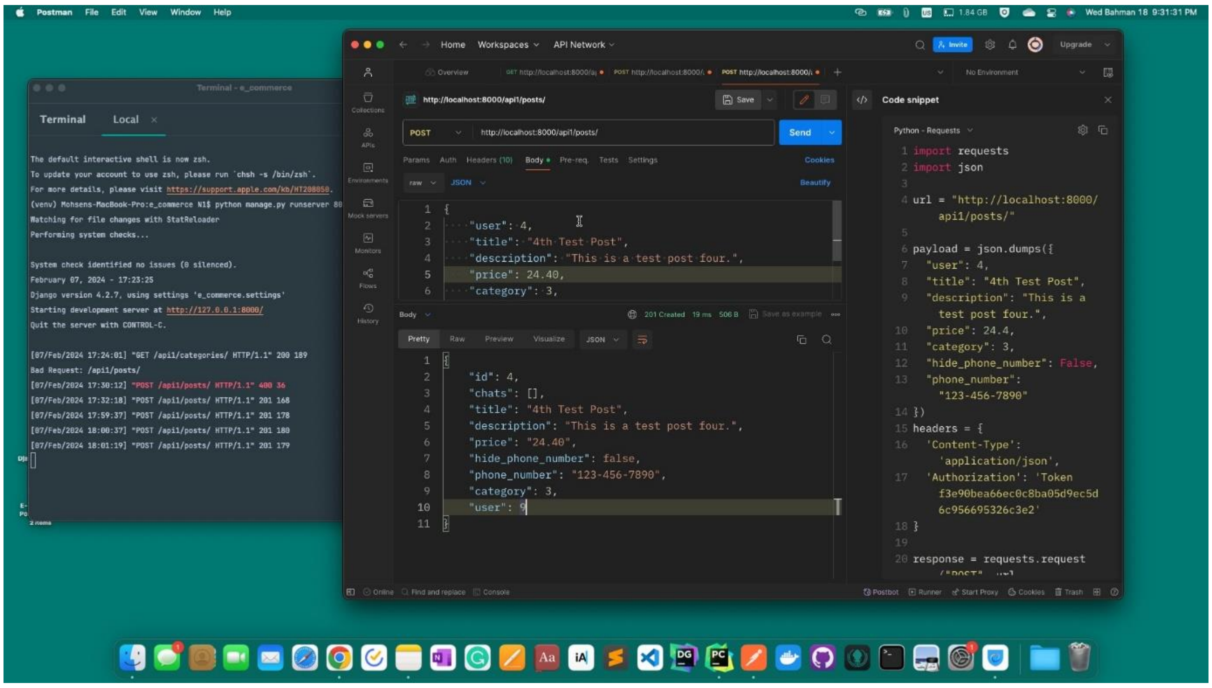
200 OK 16 ms 530 B

```
{
    "id": 1,
    "chats": [],
    "title": "1st Test Post updated",
    "description": "This is a test post one, updated.",
    "price": "19.20",
    "hide_phone_number": true,
    "phone_number": "123-456-7890",
    "category": 3,
    "user": 9
}
```

Code snippet — Python - Requests

```
import requests
import json

url = "http://localhost:8000/
    api1/posts/1/"

payload = json.dumps({
  "user": 9,
  "title": "1st Test Post
    updated",
  "description": "This is a
    test post one, updated.",
  "price": 19.2,
  "category": 3,
  "hide_phone_number": True,
  "phone_number":
    "123-456-7890"
})
headers = {
  'Authorization': 'Token
    f3e90bea66ec0c8ba05d9ec5d
    6c956695326c3e2',
  'Content-Type':
    'application/json'
}

response = requests.request
    ("PUT", url,
    headers=headers
```

Postman File Edit View Window Help
Wed Bahman 18 10:19:59 PM
Terminal - e_commerce
Terminal Local
Home Workspaces API Network
Send Save
Params Auth Headers (10) Body Pre-req. Tests Settings Cookies
raw JSON Beautify
Pretty Raw Preview Visualize JSON



```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(venv) Mohsens-MacBook-Pro:e_commerce N1$ python manage.py runserver 80
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 07, 2024 - 17:23:25
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[07/Feb/2024 17:24:01] "GET /api1/categories/ HTTP/1.1" 200 189
Bad Request: /api1/posts/
[07/Feb/2024 17:30:12] "POST /api1/posts/ HTTP/1.1" 400 36
[07/Feb/2024 17:32:18] "POST /api1/posts/ HTTP/1.1" 201 168
[07/Feb/2024 17:59:37] "POST /api1/posts/ HTTP/1.1" 201 178
[07/Feb/2024 18:00:37] "POST /api1/posts/ HTTP/1.1" 201 180
[07/Feb/2024 18:01:19] "POST /api1/posts/ HTTP/1.1" 201 179
```

POST http://localhost:8000/api1/posts/

```
{
    "user": 4,
    "title": "4th Test Post",
    "description": "This is a test post four.",
    "price": 24.40,
    "category": 3,
```

201 Created 19 ms 506 B

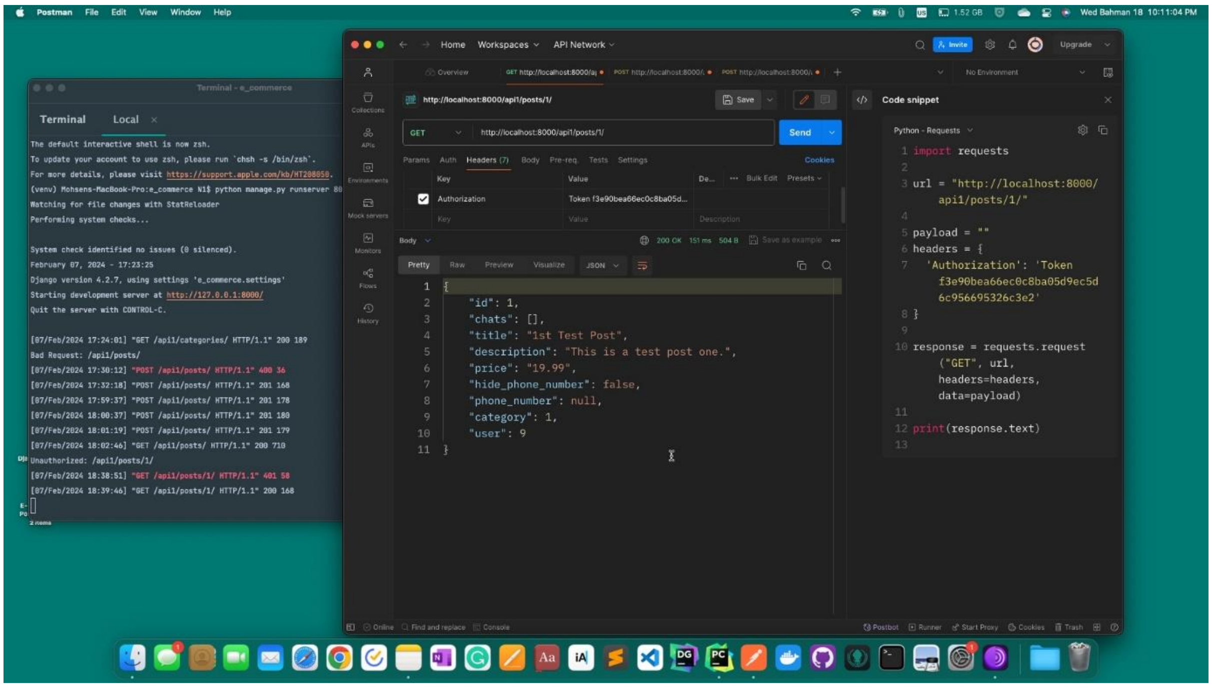
```
{
    "id": 4,
    "chats": [],
    "title": "4th Test Post",
    "description": "This is a test post four.",
    "price": "24.40",
    "hide_phone_number": false,
    "phone_number": "123-456-7890",
    "category": 3,
    "user": 9
}
```

Code snippet — Python - Requests

```
import requests
import json

url = "http://localhost:8000/
    api1/posts/"

payload = json.dumps({
  "user": 4,
  "title": "4th Test Post",
  "description": "This is a
    test post four.",
  "price": 24.4,
  "category": 3,
  "hide_phone_number": False,
  "phone_number":
    "123-456-7890"
})
headers = {
  'Content-Type':
    'application/json',
  'Authorization': 'Token
    f3e90bea66ec0c8ba05d9ec5d
    6c956695326c3e2'
}

response = requests.request
```

Postman File Edit View Window Help
Wed Bahman 18 9:31:31 PM
Terminal - e_commerce
Terminal Local
Home Workspaces API Network
Send Save
Params Auth Headers (10) Body Pre-req. Tests Settings Cookies
raw JSON Beautify
Pretty Raw Preview Visualize JSON

First post (as an example) retrieved and updated



```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(venv) Mohsens-MacBook-Pro:e_commerce N1$ python manage.py runserver 80
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 07, 2024 - 17:23:25
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[07/Feb/2024 17:24:01] "GET /api1/categories/ HTTP/1.1" 200 189
Bad Request: /api1/posts/
[07/Feb/2024 17:30:12] "POST /api1/posts/ HTTP/1.1" 400 36
[07/Feb/2024 17:32:18] "POST /api1/posts/ HTTP/1.1" 201 168
[07/Feb/2024 17:59:37] "POST /api1/posts/ HTTP/1.1" 201 178
[07/Feb/2024 18:00:37] "POST /api1/posts/ HTTP/1.1" 201 180
[07/Feb/2024 18:01:19] "POST /api1/posts/ HTTP/1.1" 201 179
[07/Feb/2024 18:02:46] "GET /api1/posts/ HTTP/1.1" 200 710
Unauthorized: /api1/posts/1/
[07/Feb/2024 18:38:51] "GET /api1/posts/1/ HTTP/1.1" 401 58
[07/Feb/2024 18:39:46] "GET /api1/posts/1/ HTTP/1.1" 200 168
```

GET http://localhost:8000/api1/posts/1/

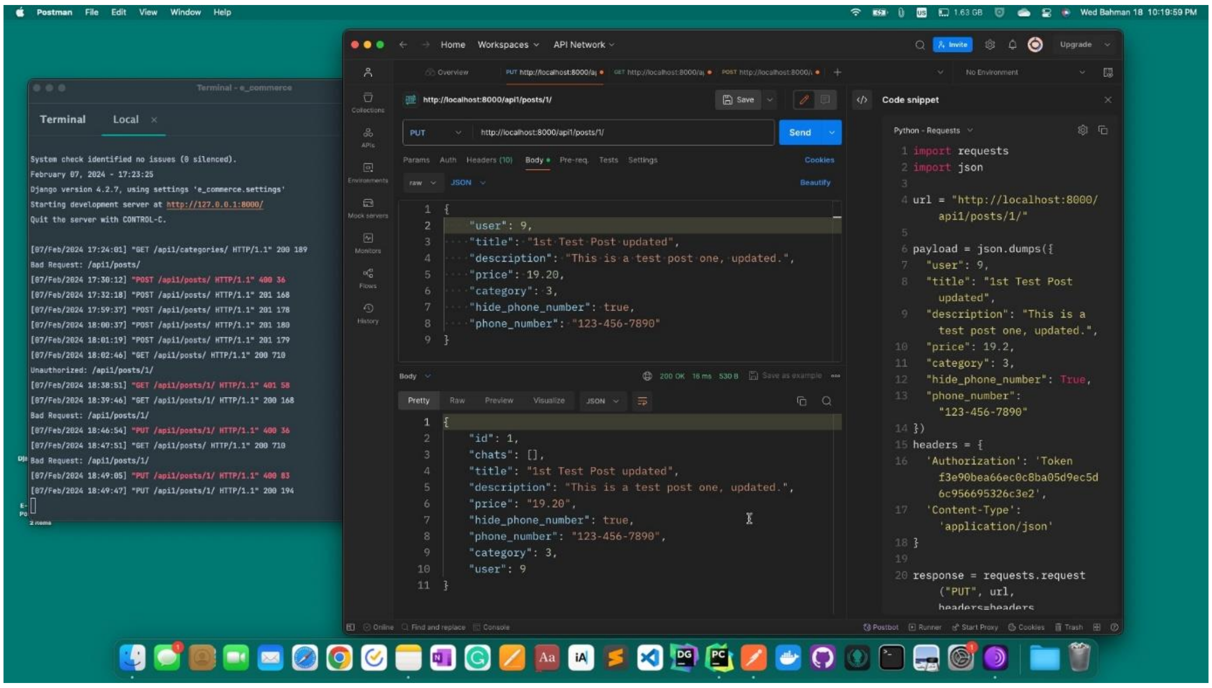
Headers (7): Authorization: Token f3e90bea66ec0c8ba05d...

200 OK 151 ms 504 B

```
{
    "id": 1,
    "chats": [],
    "title": "1st Test Post",
    "description": "This is a test post one.",
    "price": "19.99",
    "hide_phone_number": false,
    "phone_number": null,
    "category": 1,
    "user": 9
}
```

Code snippet — Python - Requests

```python
import requests

url = "http://localhost:8000/api1/posts/1/"

payload = ""
headers = {
  'Authorization': 'Token f3e90bea66ec0c8ba05d9ec5d6c956695326c3e2'
}

response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)
```



```
System check identified no issues (0 silenced).
February 07, 2024 - 17:23:25
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[07/Feb/2024 17:24:01] "GET /api1/categories/ HTTP/1.1" 200 189
Bad Request: /api1/posts/
[07/Feb/2024 17:30:12] "POST /api1/posts/ HTTP/1.1" 400 36
[07/Feb/2024 17:32:18] "POST /api1/posts/ HTTP/1.1" 201 168
[07/Feb/2024 17:59:37] "POST /api1/posts/ HTTP/1.1" 201 178
[07/Feb/2024 18:00:37] "POST /api1/posts/ HTTP/1.1" 201 180
[07/Feb/2024 18:01:19] "POST /api1/posts/ HTTP/1.1" 201 179
[07/Feb/2024 18:02:46] "GET /api1/posts/ HTTP/1.1" 200 710
Unauthorized: /api1/posts/1/
[07/Feb/2024 18:38:51] "GET /api1/posts/1/ HTTP/1.1" 401 58
[07/Feb/2024 18:39:46] "GET /api1/posts/1/ HTTP/1.1" 200 168
Bad Request: /api1/posts/1/
[07/Feb/2024 18:46:54] "PUT /api1/posts/1/ HTTP/1.1" 400 36
[07/Feb/2024 18:47:51] "GET /api1/posts/ HTTP/1.1" 200 710
Bad Request: /api1/posts/1/
[07/Feb/2024 18:49:05] "PUT /api1/posts/1/ HTTP/1.1" 400 83
[07/Feb/2024 18:49:47] "PUT /api1/posts/1/ HTTP/1.1" 200 194
```

PUT http://localhost:8000/api1/posts/1/

Body — raw — JSON

```
{
    "user": 9,
    "title": "1st Test Post updated",
    "description": "This is a test post one, updated.",
    "price": 19.20,
    "category": 3,
    "hide_phone_number": true,
    "phone_number": "123-456-7890"
}
```

200 OK 16 ms 530 B

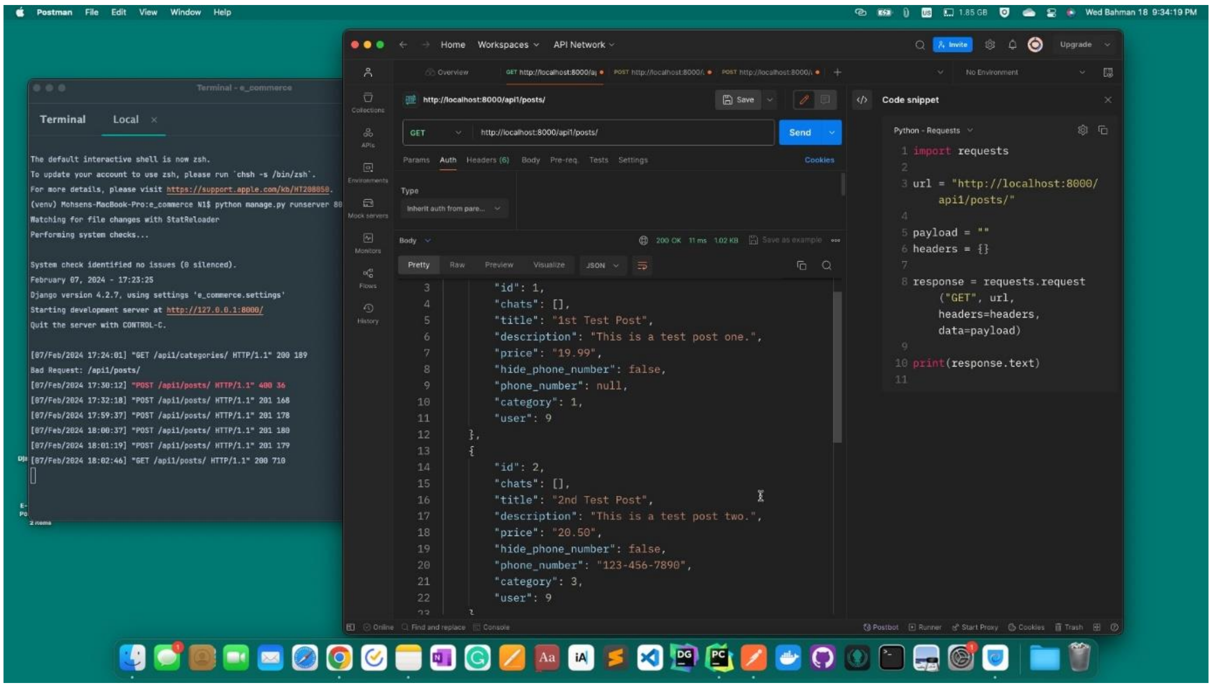
```
{
    "id": 1,
    "chats": [],
    "title": "1st Test Post updated",
    "description": "This is a test post one, updated.",
    "price": "19.20",
    "hide_phone_number": true,
    "phone_number": "123-456-7890",
    "category": 3,
    "user": 9
}
```

Code snippet — Python - Requests

```python
import requests
import json

url = "http://localhost:8000/api1/posts/1/"

payload = json.dumps({
  "user": 9,
  "title": "1st Test Post updated",
  "description": "This is a test post one, updated.",
  "price": 19.2,
  "category": 3,
  "hide_phone_number": True,
  "phone_number": "123-456-7890"
})
headers = {
  'Authorization': 'Token f3e90bea66ec0c8ba05d9ec5d6c956695326c3e2',
  'Content-Type': 'application/json'
}

response = requests.request("PUT", url, headers=headers
```

Here in second screen-shot the retrieved post (first one) is updated.

All posts listed



```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(venv) Mohsens-MacBook-Pro:e_commerce N1$ python manage.py runserver 80
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 07, 2024 - 17:23:25
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[07/Feb/2024 17:24:01] "GET /api1/categories/ HTTP/1.1" 200 189
Bad Request: /api1/posts/
[07/Feb/2024 17:30:12] "POST /api1/posts/ HTTP/1.1" 400 36
[07/Feb/2024 17:32:18] "POST /api1/posts/ HTTP/1.1" 201 168
[07/Feb/2024 17:59:37] "POST /api1/posts/ HTTP/1.1" 201 178
[07/Feb/2024 18:00:37] "POST /api1/posts/ HTTP/1.1" 201 180
[07/Feb/2024 18:01:19] "POST /api1/posts/ HTTP/1.1" 201 179
[07/Feb/2024 18:02:46] "GET /api1/posts/ HTTP/1.1" 200 710
```

GET http://localhost:8000/api1/posts/

200 OK 11 ms 1.02 KB

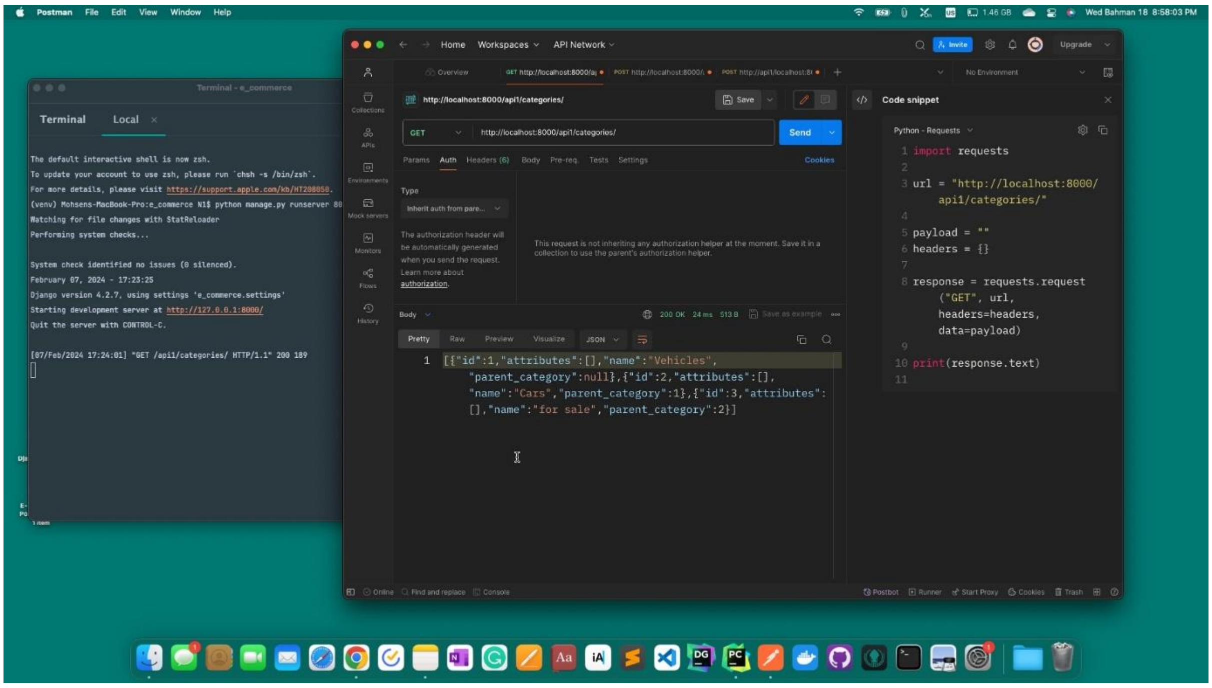
```
        "id": 1,
        "chats": [],
        "title": "1st Test Post",
        "description": "This is a test post one.",
        "price": "19.99",
        "hide_phone_number": false,
        "phone_number": null,
        "category": 1,
        "user": 9
    },
    {
        "id": 2,
        "chats": [],
        "title": "2nd Test Post",
        "description": "This is a test post two.",
        "price": "20.50",
        "hide_phone_number": false,
        "phone_number": "123-456-7890",
        "category": 3,
        "user": 9
```

Code snippet — Python - Requests

```python
import requests

url = "http://localhost:8000/api1/posts/"

payload = ""
headers = {}

response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)
```

Here all posts are retrieved, and this shot is taken before first post was updated.

Categories:
Listed for front to show them in front-end



```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(venv) Mohsens-MacBook-Pro:e_commerce N1$ python manage.py runserver 80
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 07, 2024 - 17:23:25
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[07/Feb/2024 17:24:01] "GET /api1/categories/ HTTP/1.1" 200 189
```

GET http://localhost:8000/api1/categories/

The authorization header will be automatically generated when you send the request. Learn more about authorization.

This request is not inheriting any authorization helper at the moment. Save it in a collection to use the parent's authorization helper.

200 OK 24 ms 513 B

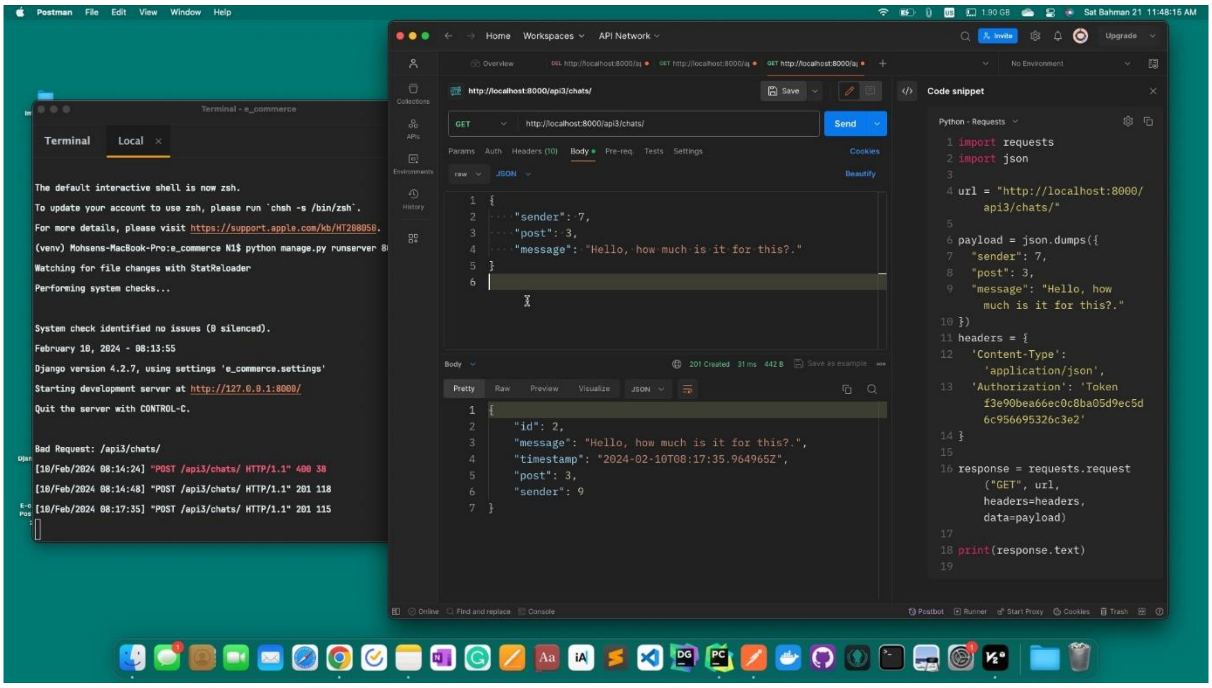
```
[{"id":1,"attributes":[],"name":"Vehicles",
    "parent_category":null},{"id":2,"attributes":[],
    "name":"Cars","parent_category":1},{"id":3,"attributes":
    [],"name":"for sale","parent_category":2}]
```

Code snippet — Python - Requests

```python
import requests

url = "http://localhost:8000/api1/categories/"

payload = ""
headers = {}

response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)
```

Chats:

Created



```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(venv) Mohsens-MacBook-Pro:e_commerce N1$ python manage.py runserver 8
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 10, 2024 - 08:13:55
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

Bad Request: /api3/chats/
[10/Feb/2024 08:14:24] "POST /api3/chats/ HTTP/1.1" 400 38
[10/Feb/2024 08:14:48] "POST /api3/chats/ HTTP/1.1" 201 118
[10/Feb/2024 08:17:35] "POST /api3/chats/ HTTP/1.1" 201 115
```

GET http://localhost:8000/api3/chats/

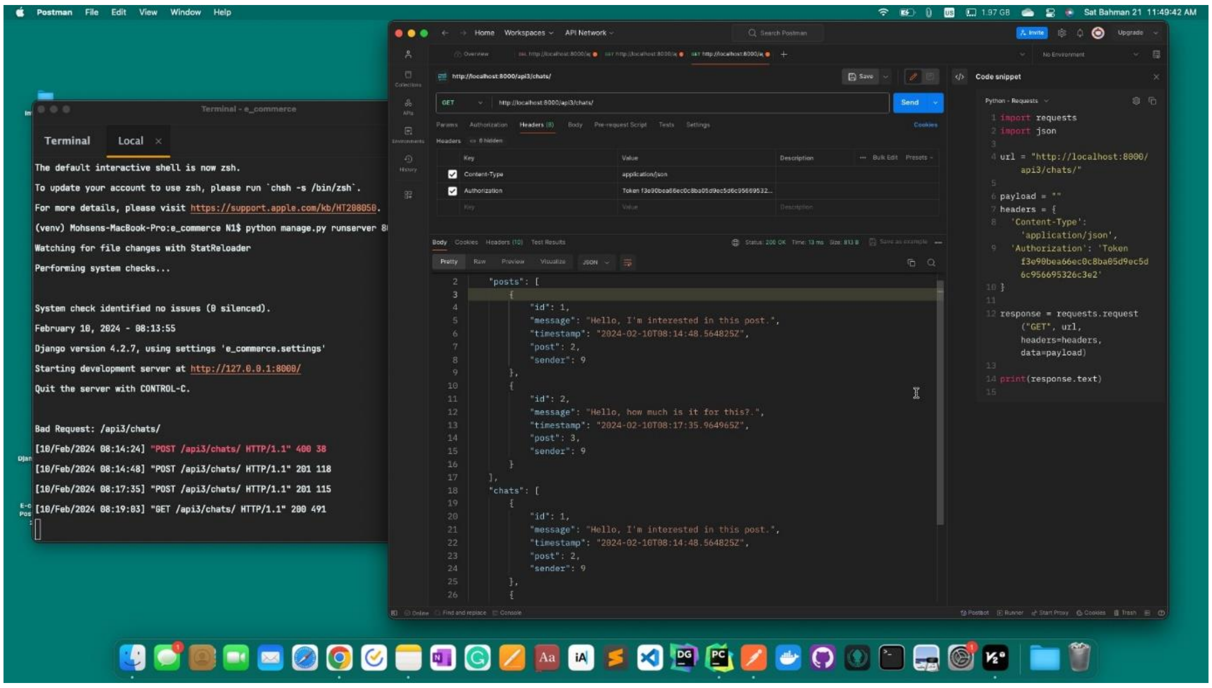
```
{
    "sender": 7,
    "post": 3,
    "message": "Hello, how much is it for this?."
}
```

201 Created 31 ms 442 B

```
{
    "id": 2,
    "message": "Hello, how much is it for this?.",
    "timestamp": "2024-02-10T08:17:35.964965Z",
    "post": 3,
    "sender": 9
}
```

```python
import requests
import json

url = "http://localhost:8000/api3/chats/"

payload = json.dumps({
  "sender": 7,
  "post": 3,
  "message": "Hello, how much is it for this?."
})
headers = {
  'Content-Type': 'application/json',
  'Authorization': 'Token f3e90bea66ec0c8ba05d9ec5d6c956695326c3e2'
}

response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)
```

Listed



```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(venv) Mohsens-MacBook-Pro:e_commerce N1$ python manage.py runserver 8
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 10, 2024 - 08:13:55
Django version 4.2.7, using settings 'e_commerce.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

Bad Request: /api3/chats/
[10/Feb/2024 08:14:24] "POST /api3/chats/ HTTP/1.1" 400 38
[10/Feb/2024 08:14:48] "POST /api3/chats/ HTTP/1.1" 201 118
[10/Feb/2024 08:17:35] "POST /api3/chats/ HTTP/1.1" 201 115
[10/Feb/2024 08:19:03] "GET /api3/chats/ HTTP/1.1" 200 491
```

GET http://localhost:8000/api3/chats/

Key	Value	Description
Content-Type	application/json	
Authorization	Token f3e90bea66ec0c8ba05d9ec5d6c95669532…	

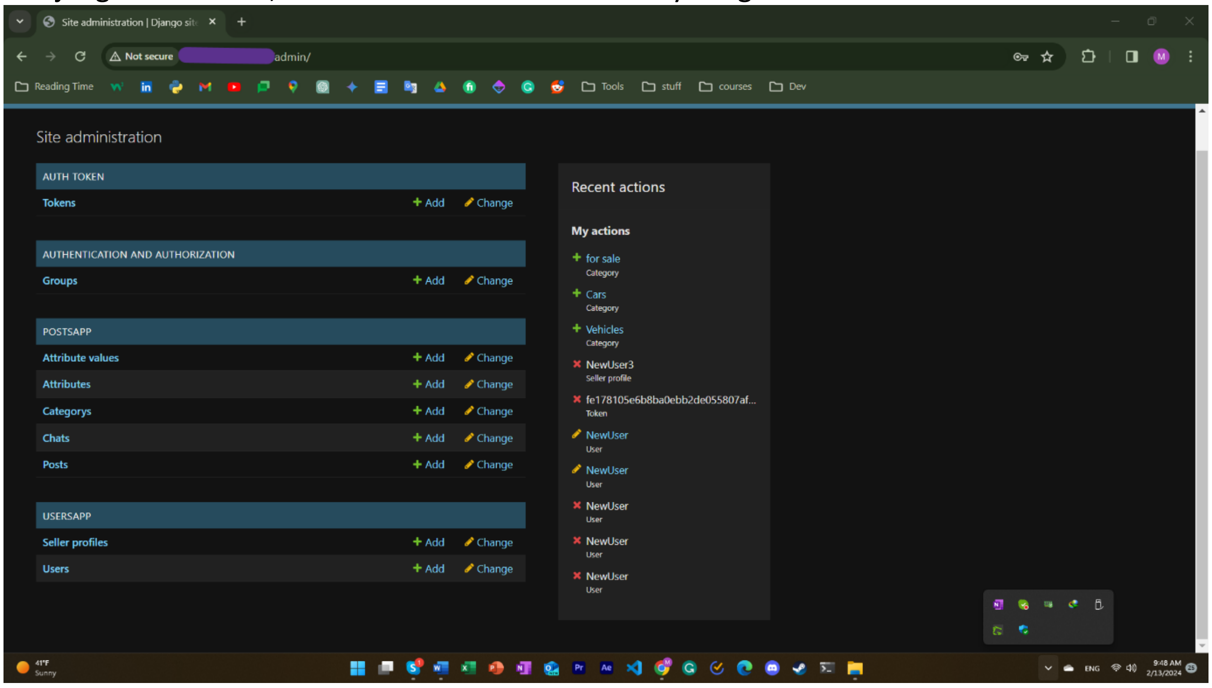
Status: 200 OK Time: 13 ms Size: 813 B

```
    "posts": [
        {
            "id": 1,
            "message": "Hello, I'm interested in this post.",
            "timestamp": "2024-02-10T08:14:48.564825Z",
            "post": 2,
            "sender": 9
        },
        {
            "id": 2,
            "message": "Hello, how much is it for this?.",
            "timestamp": "2024-02-10T08:17:35.964965Z",
            "post": 3,
            "sender": 9
        }
    ],
    "chats": [
        {
            "id": 1,
            "message": "Hello, I'm interested in this post.",
            "timestamp": "2024-02-10T08:14:48.564825Z",
            "post": 2,
            "sender": 9
        },
        {
```

```python
import requests
import json

url = "http://localhost:8000/api3/chats/"

payload = ""
headers = {
  'Content-Type': 'application/json',
  'Authorization': 'Token f3e90bea66ec0c8ba05d9ec5d6c956695326c3e2'
}

response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)
```

Administration:

In Django admin site, this is what full access to everything looks like:


Site administration | Django site

Not secure [redacted]admin/

Site administration

AUTH TOKEN
- Tokens + Add ✎ Change

AUTHENTICATION AND AUTHORIZATION
- Groups + Add ✎ Change

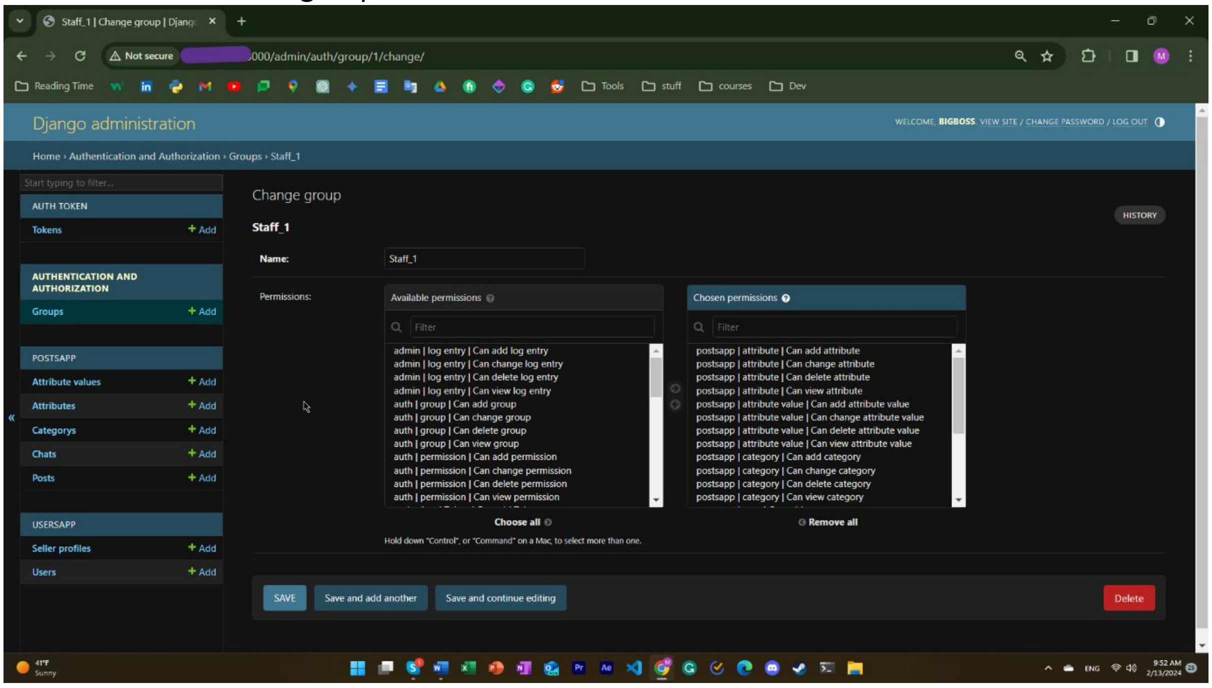
POSTSAPP
- Attribute values + Add ✎ Change
- Attributes + Add ✎ Change
- Categorys + Add ✎ Change
- Chats + Add ✎ Change
- Posts + Add ✎ Change

USERSAPP
- Seller profiles + Add ✎ Change
- Users + Add ✎ Change

Recent actions

My actions
- + for sale — Category
- + Cars — Category
- + Vehicles — Category
- ✗ NewUser3 — Seller profile
- ✗ fe178105e6b8ba0ebb2de055807af... — Token
- ✎ NewUser — User
- ✎ NewUser — User
- ✗ NewUser — User
- ✗ NewUser — User
- ✗ NewUser — User

But of course we have groups defined to limit that access like this:


Staff_1 | Change group | Django

Not secure [redacted]000/admin/auth/group/1/change/

Django administration

WELCOME, **BIGBOSS**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Authentication and Authorization › Groups › Staff_1

Change group

Staff_1

Name: Staff_1

Permissions:

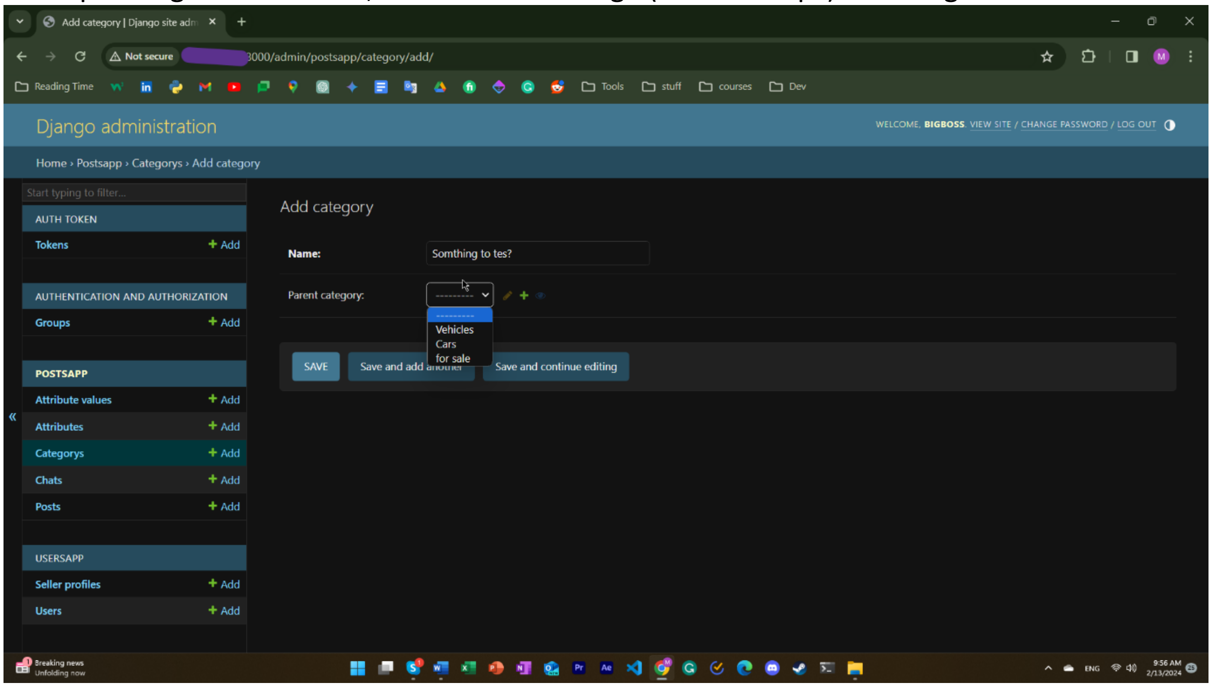
Available permissions
```
admin | log entry | Can add log entry
admin | log entry | Can change log entry
admin | log entry | Can delete log entry
admin | log entry | Can view log entry
auth | group | Can add group
auth | group | Can change group
auth | group | Can delete group
auth | group | Can view group
auth | permission | Can add permission
auth | permission | Can change permission
auth | permission | Can delete permission
auth | permission | Can view permission
```

Chosen permissions
```
postsapp | attribute | Can add attribute
postsapp | attribute | Can change attribute
postsapp | attribute | Can delete attribute
postsapp | attribute | Can view attribute
postsapp | attribute value | Can add attribute value
postsapp | attribute value | Can change attribute value
postsapp | attribute value | Can delete attribute value
postsapp | attribute value | Can view attribute value
postsapp | category | Can add category
postsapp | category | Can change category
postsapp | category | Can delete category
postsapp | category | Can view category
```

Hold down "Control", or "Command" on a Mac, to select more than one.

Using this Django Admin site, we (as the super-user, Full-access admin or BigBoss) can define different admin rules with different levels of access to moderate the service easily, without source code or Data-base access.

Ok depending on the access, admins can manage (do CRUD ops) on Categories:



[redacted]3000/admin/postsapp/category/add/

Home › Postsapp › Categorys › Add category

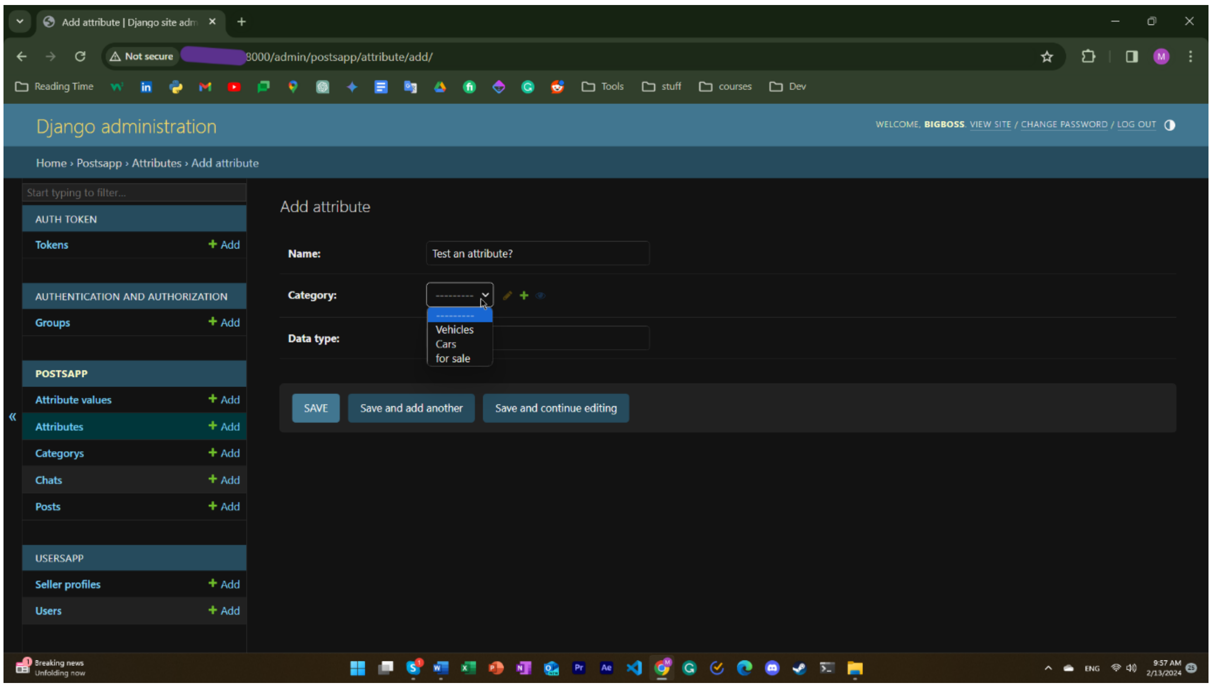
Add category

Name: Somthing to tes?

Parent category: ---------

Vehicles
Cars
for sale

And attributes:



[redacted]8000/admin/postsapp/attribute/add/

Home › Postsapp › Attributes › Add attribute

Add attribute

Name: Test an attribute?

Category: ---------

Vehicles
Cars
for sale

Data type:

As you can see each category has a parent one, so a tree of categories and sub-categories can be built, with related attributes for each level of the categories trees and all these can be fetched and be sent to the front-end, as been show-cased in the Postman tests above (Categories listed test).

Conclusion

Thank you for reviewing this documentation.
This project was meticulously designed, structured, and developed by me, serving as a comprehensive example of my capabilities with Django Rest Framework. I hope you found it insightful in understanding the functionalities of my e-commerce platform.
Should you have any questions, feedback, or suggestions for improvement, please don't hesitate to reach out.

Contact me

For any further inquiries or assistance, feel free to contact me at goldfighter368@gmail.com or join me on GitHub & Linkedin.
Thank you once again for your interest and support.

Mohsen Fato;