TASK 1

This section of this documentation outlines the implementation of authenticating and signing up users. This provides an overview of custom user models, serializers, and endpoints used.

1) **Custom User Model and Manager:**
    - The CustomUser model inherits from AbstractBaseUser provided by Django, allowing customization of the user model.
    - It includes fields such as user_id, username, email, password, name, and phone.
    - The UserManager class provides methods for creating and managing users. The create_user method creates a user with a given email and password.

2) **Serializers:**

    UserSerializer:
    - Serializes user data for retrieval.
    - Includes fields such as user_id, username, email, password, name, and phone.

    CreateUserSerializer:
    - Serializes data for creating new users.
    - Includes validation for email, password, phone number, and name.
    - Overrides to_representation method to customize representation of user data.
    - Overrides the validate method for data validation.
    - Overrides create method to create a new user with validated data.

3) **Endpoints:**

    There are three endpoints used for register, signup and for testing the token.

User Registration API:

- Endpoint: /register/
- Method: POST
- Registers a new user with provided data.
- Validates user input and creates a new user if valid.
- Returns a token for user authentication.

For calling this API use POST http://127.0.0.1:8000/accounts/register/

Request body:

```
{
    "email": "ald@gmail.com",
    "password": "al123456",
    "name": "aldi",
    "phone": 1234567891,
    "username": "al28"
}
```

Response from the server:

{"token":"3569395d49414b9d5c19366345b507696264d6b8","user":{"user_id":1,"username":"al28","email":"ald@gmail.com","password":"pbkdf2_sha256$390000$iy1UTTQ9ZTZArVZD90zSuW$vykDOxL81nRM4c7xsLbwJeUyHh1FvtjTMCMDK7qtjEw=","name":"Aldi","phone":1234578691},"message":"User registered successfully"}

User Login API:

- Endpoint: /login/
- Method: POST
- Logs in an existing user with provided credentials.
- Validates user input and returns an authentication token if successful.

For calling this API use POST http://127.0.0.1:8000/accounts/login/

Request body:

```
{
    "username": "al28",
    "password": "al123456"

}
```

Response from the server

{"token":"3569395d49414b9d5c19366345b507696264d6b8","user":{"user_id":1,"username":"al28","email":"ald@gmail.com","password":"pbkdf2_sha256$390000$L4izk8qtzjPxSI6fAzBS8I$Xs3CeHKfGqiAkCQoT2rsif8Hnbpge6UrhxrcCHeAA4I=","name":"aldi","phone":1234578691},"message":"User is logged in successfully"}

Token Testing:

- Endpoint: /test_token/
- Method: GET
- Tests the authentication token.
- The user must include a valid token in the request headers.

For calling this API use GET http://127.0.0.1:8000/accounts/test_token/

Add a new header with the key Authorization and the value Token <your_token>, where <your_token> is the authentication token you want to test.

Request:

Key :Authorization

Value: Token 3569395d49414b9d5c19366345b507696264d6b8

Response from server

"passed!"

TASK 2

This section of this documentation outlines the implementation of crud operations to add a toy,update the toy details,get the toy details and delete the toy data using the class based api views.

Model is designed with the following fields:

- id (BigAutoField): Primary key for the toy.
- name (CharField): Name of the toy.
- model (CharField): Unique model identifier for the toy.
- price (DecimalField): Price of the toy.

The ToyModelSerializer does the validation of the data and serializes the Toy model.

There are totally 5 endpoints for create, update, delete, retrieval of details of all toys or a specific toy and another endpoint for retrieving details of a specific toy by its model.

1) **ToyCreateAPI:**

   Method: POST
   URL: /toy/create/
   For calling this API use POST http://127.0.0.1:8000/toy_create/
   Request body:

```
{
    "name":"Cat stuffed toy",
    "model":"CA05",
    "price":200
}
```

   Response from the server
   {"id":6,"name":"Cat stuffed toy","model":"CA05","price":"200.00"}

2) **ToyUpdateAPI:**

Method: PUT

URL: /toy/update/

For calling this API use PUT http://127.0.0.1:8000/toy_update/

Request body

```
{
    "name":"Cat toy",
    "model":"CA05",
    "price":200
}
```

Response from the server

{"id":6,"name":"Cat toy","model":"CA05","price":"200.00"}

3) **ToyDeleteAPI**

Method: DELETE

URL: /toy/delete/

For calling this API use DELETE http://127.0.0.1:8000/toy_delete/

Request body

```
{
    "model":"CA05"
}
```

Response from the server

{"msg":"Toy deleted successfully"}

4) **AllToyDetailsAPI**

Method: GET

URL: /toy/details/

For calling this API use GET http://127.0.0.1:8000/toy_details/

Request body:

```
{


}
```

Response from server

[{"id":1,"name":"Dino stuffed toy","model":"DI28","price":"500.00"},{"id":2,"name":"Dragon stuffed toy","model":"DR20","price":"500.00"},{"id":3,"name":"Elephant stuffed toy","model":"EL20","price":"800.00"}]

Request body:

```
{
    "model":"EL20"
}
```

Response from server

{"id":3,"name":"Elephant stuffed toy","model":"EL20","price":"800.00"}

5) ToyDetailsAPI

Method: GET

URL: /toy/details/<model>/

For calling this API use GET http://127.0.0.1:8000/toy_details/<model name>

Request body

http://127.0.0.1:8000/toy_details/DI28

```
{


}
```

Response from the server

{"id":1,"name":"Dino stuffed toy","model":"DI28","price":"500.00"}