

Introduction

Docs that describe all endpoints in restify. This doc is split into sections based on where the api endpoint lies in the code. That is, if an api endpoint's view is defined in the 'properties' sub folder in the code, then it will be under the properties heading.

Properties

Models

The following screenshot is taken from the models.py file taken from the properties subfolder.

```

restify > properties > models.py > Availability > prop
You, 16 hours ago | 1 author (You)
1  from django.db import models
2  from django.utils.html import mark_safe
3  from user.models import User
4  from django.core.validators import RegexValidator
5  # Create your models here.
You, 2 days ago | 1 author (You)
6  class Property(models.Model):
7      city = models.CharField(max_length=100)
8      province = models.CharField(max_length=100)
9      country = models.CharField(max_length=100)
10     guests_cap = models.PositiveIntegerField(null=False, blank=False)
11     owner = models.ForeignKey(to=User, on_delete=models.CASCADE)
12     beds = models.PositiveIntegerField(null = False, blank = False)
13     baths = models.PositiveIntegerField(null = False, blank = False)
14     unit_num = models.PositiveIntegerField(null = False, blank = False)
15     street = models.CharField(max_length=100)
16     desc = models.CharField(max_length=200)
17
You, 16 hours ago | 1 author (You)
18 class Availability(models.Model):
19     arrive_date = models.DateField()
20     depart_date = models.DateField()
21     total_price = models.PositiveIntegerField()
22     prop = models.ForeignKey(Property, on_delete=models.CASCADE)
23
You, last week | 1 author (You)
24 class Amenity(models.Model):
25     amenity = models.CharField(max_length=20)
26     prop = models.ForeignKey(Property, on_delete=models.CASCADE)
27
You, last week | 1 author (You)
28 class Pictures(models.Model):
29     picture = models.ImageField()
30     prop = models.ForeignKey(Property, on_delete=models.CASCADE)
31     def img_preview(self): #new
32         return mark_safe(f'<img src = "{self.picture.url}" width = "300"/>')
33
34
35
36

```

Note that availabilities are treated as ranges. That is, a user must book a property that starts exactly at the `arrive_date` and must leave exactly on the `depart_date`. A user cannot book an interval in between those 2 dates.

URLS

Create Property

The url is localhost:8000/properties/create/

The method supported is PUT

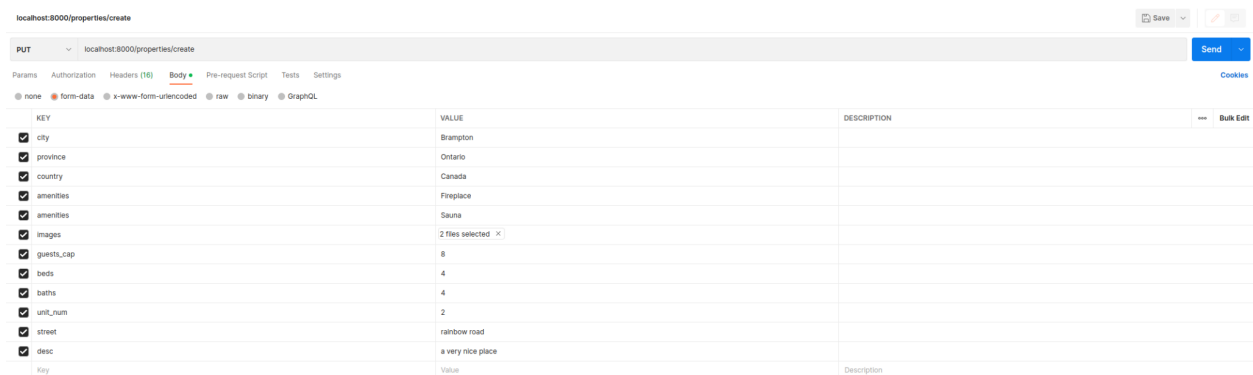
This endpoint requires the url to be authorized. That is, the token must be in the header.

This url creates a property where the user in the request body is assigned as the host. The server expects the response to be in form-data format and expects all the following parameters:

- city (A string)
- province (A string)
- country (A string)
- amenities (A list of strings)
 - Note that each string in the list can only be one of the following: 'Fireplace', 'Sauna', 'Tub', 'Kitchen', 'Tv', 'Sofa'.
- Images (A list of images)
- guests_cap (A positive integer)
- beds (A positive integer)
- baths (A positive integer)
- unit_num (A positive integer)
- street (A string)
- desc (A string)

Note that desc stands for description.

As an example, a request made using postman would look like the following:



localhost:8000/properties/create

PUT localhost:8000/properties/create

Params Authorization Headers (16) Body **form-data** Pre-request Script Tests Settings

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

| KEY | VALUE | DESCRIPTION |
|--|-------------------|-------------|
| <input checked="" type="checkbox"/> city | Brampton | |
| <input checked="" type="checkbox"/> province | Ontario | |
| <input checked="" type="checkbox"/> country | Canada | |
| <input checked="" type="checkbox"/> amenities | Fireplace | |
| <input checked="" type="checkbox"/> amenities | Sauna | |
| <input checked="" type="checkbox"/> images | 2 files selected | |
| <input checked="" type="checkbox"/> guests_cap | 8 | |
| <input checked="" type="checkbox"/> beds | 4 | |
| <input checked="" type="checkbox"/> baths | 4 | |
| <input checked="" type="checkbox"/> unit_num | 2 | |
| <input checked="" type="checkbox"/> street | rainbow road | |
| <input checked="" type="checkbox"/> desc | a very nice place | |
| Key | Value | Description |

The request will return a message and status code of 200 if successful. Otherwise it will return an error message.

Search

The url is localhost:8000/properties/search/

The method supported is GET

This endpoint does not require the user to be authorized.

This url will search all available properties and return properties that satisfy the criteria given in the request's body.

Note all of these filters are exact, so it will only find properties that match all of the criteria. If nothing is specified in the request body, then it returns all properties.

The request body is expected to be in form-data format.

The parameters of the request are as follows:

- city (A string)
- province (A string)
- country (A string)
- arrive_date (A string representing date in YYYY/MM/DD, like 2023/03/13)
- depart_date (A string representing date in YYYY/MM/DD, like 2023/03/13)
- order_by (A string that can be either be 'beds', 'baths')
- guests_cap (An integer)
- beds (An integer)
- baths (An integer)

Note that an example request look like the following using postman with all fields given

The screenshot shows a Postman interface for a GET request to `localhost:8000/properties/search`. The request body is set to 'form-data' and contains the following parameters:

| KEY | VALUE | DESCRIPTION |
|---|------------|-------------|
| <input checked="" type="checkbox"/> city | Brampton | |
| <input checked="" type="checkbox"/> province | Ontario | |
| <input checked="" type="checkbox"/> country | Canada | |
| <input checked="" type="checkbox"/> arrive_date | 2023/06/01 | |
| <input checked="" type="checkbox"/> depart_date | 2023/07/01 | |
| <input checked="" type="checkbox"/> order_by | beds | |
| <input checked="" type="checkbox"/> guests_cap | 6 | |
| <input checked="" type="checkbox"/> beds | 2 | |
| <input checked="" type="checkbox"/> baths | 2 | |
| Key | Value | Description |

Note:

- All of (city, province, country) are required if filtering on location. If not all 3 are given, the search will not perform.
- All of (arrive_date, depart_date) are required if filtering on availability. If not all given, search will not perform
-

The request will return a paginated data containing properties that satisfy the constraints and status code of 200 if successful. Otherwise it will return an error message.

Update Property

The url is `localhost:8000/properties/update/<int>/`

where <int> represents the primary id created by the sqlite db.

The method supported is POST

The endpoint requires the user to be authenticated and also must be the owner of the property.

Also the body should use form-data

This url allows the owner to update certain aspects of the property. Namely, you can use this endpoint to add an availability, overwrite the amenities, or change guests_cap, number of beds, or number of baths.

Namely, the url parameters are as follows:

- arrive_date (A string representing date in YYYY/MM/DD, like 2023/03/13)
- depart_date (A string representing date in YYYY/MM/DD, like 2023/03/13)
- amenities (A list of strings)
 - Note that each string in the list can only be one of the following: 'Fireplace', 'Sauna', 'Tub', 'Kitchen', 'Tv', 'Sofa'.
- Images (A list of images)
- guests_cap (An integer)
- beds (An integer)
- baths (An integer)
- price (An integer)

Note that all of (arrive_date, depart_date, price) must be given if updating availability. Otherwise, at least 1 parameter must be given for the request to be valid. That is, either all of (arrive_date, depart_date, price) is given or amenities or images or at least one of guests, beds, and baths.

Also **note**

- The amenities as well as the images given in the request, if given at all, will overwrite what is currently stored in the db
- If there already exists an availability that overlaps with a given availability in the request, the availability will not be added, and the user should delete the old availability first before updating

An example request using postman would be as follows:

The screenshot shows a Postman interface for a POST request to the endpoint `localhost:8000/properties/update/1`. The 'Body' tab is selected, and the data type is set to 'form-data'. The request body contains the following key-value pairs:

| KEY | VALUE | DESCRIPTION |
|---|---|-------------|
| <input checked="" type="checkbox"/> arrive_date | 2023/07/01 | |
| <input checked="" type="checkbox"/> depart_date | 2023/08/01 | |
| <input checked="" type="checkbox"/> price | 500 | |
| <input checked="" type="checkbox"/> amenities | Kitchen | |
| <input checked="" type="checkbox"/> guests_cap | 4 | |
| <input checked="" type="checkbox"/> beds | 5 | |
| <input checked="" type="checkbox"/> baths | 2 | |
| <input checked="" type="checkbox"/> images | Screenshot from 2021-10-16 23-00-17.png | |
| Key | Value | Description |

Delete Property

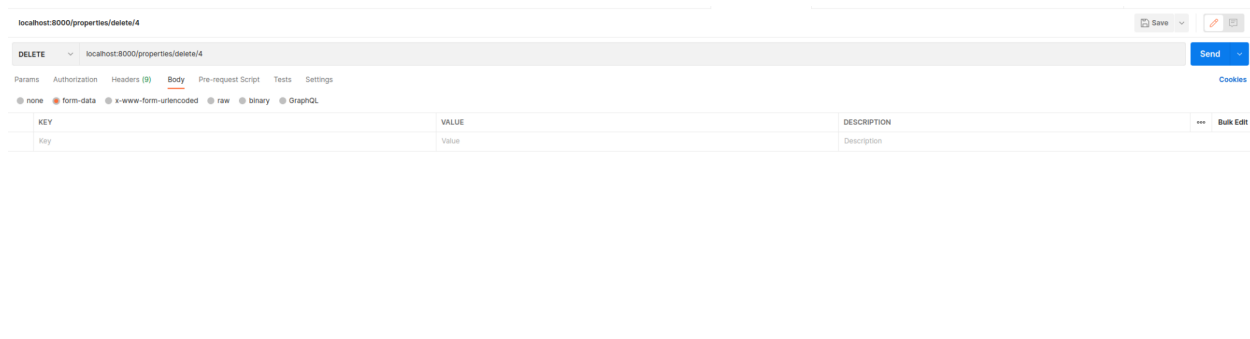
The url is localhost:8000/properties/delete/<int>/ where <int> represents the id of the property to delete.

The method supported is DELETE

This request must have an authenticated user and must be the owner of the property.

The request's body is empty.

An example request would be:



The method will return a message and status code of 200 if successful. Otherwise it will return an error message.

Delete Availability

The url is localhost:8000/properties/delete/availability/<int>/ where <int> represents the id of the property to delete.

The method supported is DELETE

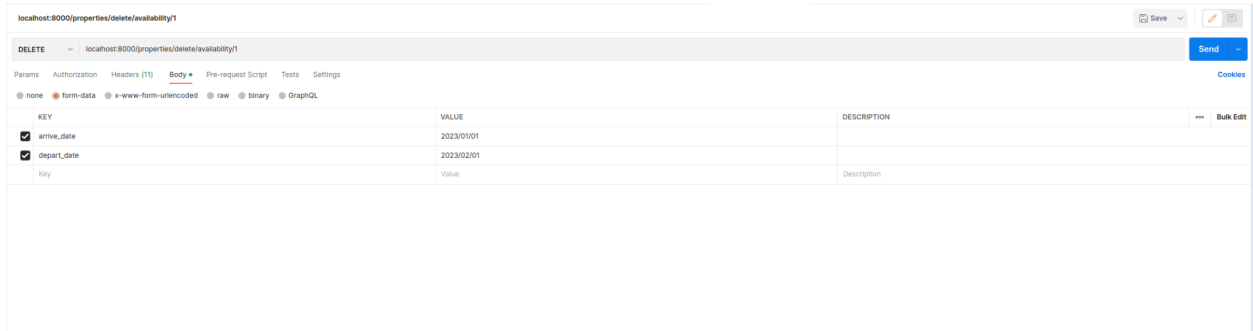
This request must have an authenticated user and must be the owner of the property.

The request's body contains the following:

- arrive_date (A string representing date in YYYY/MM/DD, like 2023/03/13)
- depart_date (A string representing date in YYYY/MM/DD, like 2023/03/13)

This method will delete all availabilities that overlap with the range given.

An example request would be:



The method will return a message and status code of 200 if successful. Otherwise it will return an error message.

User

Packages installed: phonenumbers, django-phonenummer-field

Models

The following screenshot is taken from the models.py file taken from the properties subfolder.

```

restify > user > models.py > ...
1  from django.db import models
2  from django.contrib.auth.models import AbstractUser
3  from phonenumber_field.modelfields import PhoneNumberField
4  from django.conf import settings
5
6
7  class User(AbstractUser):
8      username = models.EmailField(unique=True)
9      first_name = models.CharField(max_length=100)
10     last_name = models.CharField(max_length=100)
11     password = models.CharField(max_length=100)
12     phone = PhoneNumberField(unique=True)
13     profile_pic = models.ImageField(blank=True)
14

```

Signup

The url for signup is localhost:8000/user/signup/

The method supported is POST

The endpoint does not require the user to be authorized

The url will allow a new user to signup and return a response with status code 201 and user's signup information if the signup is successful, otherwise it will raise a 400 bad request error if:

Username is not a valid email address, or it's already taken;

Phone number already exists in our database;

Fields except profile_pic is missing.

The request body is expected to be in form-data format and expects all the following parameters except profile_pic:

First_name: A string

Last_name: A string

Username: An email address

Password: Could be anything less than 100 characters

Phone: phone number in E.164 format (e.g. +14378723333)

Profile_pic: An image. If there is no profile_pic in the requested data, by default it will be a default image located in media/.

An example request may look like this:

http://localhost:8000/user/signup/

POST http://localhost:8000/user/signup/ Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

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

| | KEY | VALUE | DESCRIPTION | ... | Bulk Edit |
|-------------------------------------|-------------|-----------------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> | first_name | jack | | | |
| <input checked="" type="checkbox"/> | last_name | sun | | | |
| <input checked="" type="checkbox"/> | username | jacksun@ut.com1 | | | |
| <input checked="" type="checkbox"/> | password | 123456 | | | |
| <input checked="" type="checkbox"/> | phone | +16478710622 | | | |
| <input type="checkbox"/> | profile_pic | Select Files | | | |

Body Cookies (1) Headers (10) Test Results Status: 201 Created Time: 160 ms Size: 545 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "first_name": "jack",
3   "last_name": "sun",
4   "username": "jacksun@ut.com1",
5   "password": "pbkdf2_sha256$390000$8JcBwVaRD3krc0aVbTuHZN$6S8PALbKe2XH8iMfsRxdXqrZjovzgrfQTGcYmSkvvvw=",
6   "phone": "+16478710622",
7   "profile_pic": "/media/default.jpeg"
8 }
```

Login

The url for signup is localhost:8000/user/login/

The method supported is POST

The endpoint does not require the user to be authorized

The url will allow a registered user to login and return a response with a refresh token and an access token that lasts for a day if the login is successful, otherwise it will raise a 401 unauthorized error.

The request body is expected to be in form-data format and expects all the following parameters

Username: The email address the user signs up with

Password: The password the user signs up with

An example request may look like this:

POST http://localhost:8000/user/login/ Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

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

| | KEY | VALUE | DESCRIPTION | ... | Bulk Edit |
|-------------------------------------|-------------|----------------|-------------|-----|-----------|
| <input type="checkbox"/> | first_name | jack | | | |
| <input type="checkbox"/> | last_name | sun | | | |
| <input checked="" type="checkbox"/> | username | jacksun@ut.com | | | |
| <input checked="" type="checkbox"/> | password | 123456 | | | |
| <input type="checkbox"/> | phone | +16478710622 | | | |
| <input type="checkbox"/> | profile_pic | Select File | | | |

body Cookies (1) Headers (10) Test Results Status: 200 OK Time: 150 ms Size: 797 B Save Response

Pretty Raw Preview Visualize JSON ⌵

```

1
2  "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2t1b190eXB1IjoicmVmcmVzaCIsImV4cCI6MTY3ODgzMjg4MywiaWF0IjoxNjc4NzQ2NDgzLCJqdGkiOiIyNWl5ZmM4NmAyNmU0ZGI2YTY1N2MyMmRmNmZlYThmZCIsInVzZXJfaWQiOiJlZWfQ.EzATv3S2pn2Dxkv1PpscDWu8wnkRluoy0QaSDaFemt8",
3  "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2t1b190eXB1IjoieYWNjZXRhIiwiaXNjaW90dGZLCjYXQ10jE2Nzg3NDY0ODMsImp0aSI6IjFkNDA5NjQ1YzQxYjQ5ZTA4MzkyZDNmYWU1YzcyYTk2IiwidXN1c19pZCI6MTB9.LtzS21LeD8b9-9mFeSnmDEZM2V-DfJH39gMvUPBZQUE"
4

```

Edit

The url for sign up is localhost:8000/user/edit/

The method supported is GET and PUT

The endpoint requires the user to be authorized

The url will allow an authorized user to view and edit profile. For the GET method, it will return a response with the user's information except the password. For the PUT method, it will update the field the user wants to change, and the rest of the fields remain the same. If the user is not authorized, it will raise a 401 unauthorized error for both GET and PUT.

The request body is expected to be in form-data format and only expects the parameter that the user wants to update. The user can change the username, first and last name, phone number and profile_pic. Note that the updated username and phone number still have to be unique in the database.

An example request may look like this:

PUT http://localhost:8000/user/edit/ Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

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

| | KEY | VALUE | DESCRIPTION | ... | Bulk Edit |
|-------------------------------------|-------------|-------------------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> | first_name | jack111 | | | |
| <input type="checkbox"/> | last_name | sun | | | |
| <input type="checkbox"/> | username | jacksun@ut.com | | | |
| <input type="checkbox"/> | password | 123456 | | | |
| <input checked="" type="checkbox"/> | phone | Text +16478710621 | | | |
| <input type="checkbox"/> | profile_pic | Select File | | | |

Body Cookies (1) Headers (10) Test Results Status: 200 OK Time: 9 ms Size: 478 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "first_name": "jack111",
3   "last_name": "sun",
4   "username": "jacksun@ut.com",
5   "phone": "+16478710621",
6   "profile_pic": "http://localhost:8000/media/default.jpeg"
7 }
```

Logout

We will implement logout in P3 by blacklisting the token.

Comment

Models

The following screenshot is taken from the models.py file taken from the properties subfolder.

```

from django.db import models
from reservations.models import Reservation
from django.conf import settings

class CommentOnProperty(models.Model):
    reservation = models.ForeignKey(Reservation, on_delete=models.CASCADE)
    content = models.TextField()
    rating = models.IntegerField(
        choices=[(1, "1"), (2, "2"), (3, "3"), (4, "4"), (5, "5")]
    )
    created_at = models.DateField(auto_now_add=True)

class HostReply(models.Model):
    comment = models.OneToOneField(CommentOnProperty, on_delete=models.CASCADE)
    content = models.TextField()
    created_at = models.DateField(auto_now_add=True)

class CommentOnReply(models.Model):
    reply = models.OneToOneField(HostReply, on_delete=models.CASCADE)
    content = models.TextField()
    created_at = models.DateField(auto_now_add=True)

class CommentOnUser(models.Model):
    reservation = models.ForeignKey(Reservation, on_delete=models.CASCADE)
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name="reviewer"
    )
    content = models.TextField()
    rating = models.IntegerField(
        choices=[(1, "1"), (2, "2"), (3, "3"), (4, "4"), (5, "5")]
    )
    created_at = models.DateField(auto_now_add=True)

```

Comment on property

The url for creating comment on a property is

localhost:8000/comment/createcomment/<reservation_id>/<property_id>/

The method supported is POST

The endpoint requires the user to be authorized

The url will allow the user of the reservation with reservation_id to leave a comment and rating on the property with property_id. The user can only comment on the property once per reservation. Only the user of the reservation with reservation_id can leave a comment.

The request body is expected to be in form-data format and expects all the following parameters:

Content: A textarea that contains the content of the comment

Rating: An integer in [1,2,3,4,5]

An example request may look like this:

The screenshot shows a REST client interface with a POST request to `http://localhost:8000/comment/createcomment/5/1/`. The request body is a JSON object with the following fields:

| KEY | VALUE | DESCRIPTION |
|---|-----------------|-------------|
| <input checked="" type="checkbox"/> content | not bad | |
| <input checked="" type="checkbox"/> rating | 2 | |
| <input type="checkbox"/> username | test3@email.com | |
| <input type="checkbox"/> password | 123 | |
| <input type="checkbox"/> first_name | jack | |
| <input type="checkbox"/> last_name | sun | |

The response is a JSON object with the following fields:

```
1 {
2   "id": 2,
3   "content": "not bad",
4   "rating": 2,
5   "created_at": "2023-03-14",
6   "reservation": 5
7 }
```

View comment on property

The url for viewing comments on a property is

`localhost:8000/comment/commentonprop/<property_id>/`

The method supported is GET

The endpoint does not require the user to be authorized

The url will allow the user to see comments on a property with `property_id`.

An example request may look like this:

GET http://localhost:8000/comment/commentonprop/1/?page=2

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

Query Params

| | KEY | VALUE | DESCRIPTION | ... | Bulk Edit |
|-------------------------------------|-----------|-------|-------------|-----|-----------|
| <input type="checkbox"/> | user_type | host | | | |
| <input type="checkbox"/> | state | Cm | | | |
| <input checked="" type="checkbox"/> | page | 2 | | | |
| | Key | Value | Description | | |

Body Cookies (1) Headers (10) Test Results

Status: 200 OK Time: 7 ms Size: 398 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "content": "not bad",
4   "rating": 2,
5   "created_at": "2023-03-14",
6   "reservation": 5
7 }
8
9
```

Reply to the comment

The url for creating reply on a comment is

localhost:8000/comment/replycomment/<comment_id>/

The method supported is POST

The endpoint requires the user to be authorized

The url will allow the host of the reservation with reservation_id to reply to a comment with comment_id. The host can only reply to the comment once per reservation. Only the host of the reservation with reservation_id can reply to a comment.

The request body is expected to be in form-data format and expects all the following parameters:

Content: A textarea that contains the content of the comment

An example request may look like this:

The screenshot shows a REST client interface with a POST request to `http://localhost:8000/comment/replycomment/1/`. The request body is set to `form-data` and contains the following fields:

| Field | Value |
|---|-----------------|
| username | another@one.com |
| password | 123 |
| phone | +14378783377 |
| profile_pic | flag.jpeg |
| <input checked="" type="checkbox"/> content | thanks |
| <input type="checkbox"/> rating | 4 |
| <input type="checkbox"/> id | 7 |

The response status is 201 Created, with a time of 17 ms and a size of 380 B. The response body is shown in JSON format:

```
1 {
2   "id": 1,
3   "content": "thanks",
4   "created_at": "2023-03-14",
5   "comment": 1
6 }
```

View reply to the comment

The url for viewing reply to a comment is

`localhost:8000/comment/commentonprop/<porperty_id>/<comment_id>/`

The method supported is GET

The endpoint does not require the user to be authorized

The url will allow the user to see the host's reply to the comment with `comment_id` on a property with `property_id`.

An example request may look like this:

GET http://localhost:8000/comment/commentonprop/1/1/ Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

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

| | | |
|-------------------------------------|-------------|-----------------|
| <input type="checkbox"/> | username | another@one.com |
| <input type="checkbox"/> | password | 123 |
| <input type="checkbox"/> | phone | +14378783377 |
| <input type="checkbox"/> | profile_pic | flag.jpeg X |
| <input checked="" type="checkbox"/> | content | thanks |
| <input type="checkbox"/> | rating | 4 |
| <input type="checkbox"/> | id | 7 |

Body Cookies (1) Headers (10) Test Results Status: 200 OK Time: 6 ms Size: 382 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "content": "thanks",
4   "created_at": "2023-03-14",
5   "comment": 1
6 }
7
8
```

Reply to the host's reply

The url for creating a reply back to a host's reply is
localhost:8000/comment/commentonreply/<reply_id>/

The method supported is POST

The endpoint requires the user to be authorized

The url will allow the author of the original comment to reply back to a host's reply

The request body is expected to be in form-data format and expects all the following parameters:

Content: A textarea that contains the content of the comment

An example request may look like this:

POST ▼ http://localhost:8000/comment/commentonreply/1/ Send ▼

Params ● Authorization ● Headers (10) ● Body ● Pre-request Script Tests Settings Cookies

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

| | | | |
|-------------------------------------|------------|---------------------|-------------------|
| <input type="checkbox"/> | property | 1 | |
| <input type="checkbox"/> | start_date | 2023-07-01 | |
| <input type="checkbox"/> | end_date | 2023-08-01 | |
| <input type="checkbox"/> | id | 2 | |
| <input checked="" type="checkbox"/> | content | no worries | |
| <input type="checkbox"/> | rating | 5 | |
| | Key | Text ▼ | Value Description |

Body Cookies (1) Headers (10) Test Results 🌐 Status: 201 Created Time: 15 ms Size: 382 B Save Response ▼

Pretty Raw Preview Visualize JSON ▼ 🔍

```

1  {
2    "id": 1,
3    "content": "no worries",
4    "created_at": "2023-03-14",
5    "reply": 1
6  }

```

Viewing the reply to the host's reply

The url for viewing a reply to a host's is

localhost:8000/comment/commentonprop/<porperty_id>/<comment_id>/<reply_id>/

The method supported is GET

The endpoint does not require the user to be authorized

The url will allow the user to see the user's reply to the host's reply with reply_id.

An example request may look like this:

GET http://localhost:8000/comment/commentonprop/1/1/1/ Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

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

| | | | |
|-------------------------------------|------------|------------|-------------|
| <input type="checkbox"/> | property | 1 | |
| <input type="checkbox"/> | start_date | 2023-07-01 | |
| <input type="checkbox"/> | end_date | 2023-08-01 | |
| <input checked="" type="checkbox"/> | id | Text 2 | |
| <input type="checkbox"/> | content | no worries | |
| <input type="checkbox"/> | rating | 5 | |
| | Key | Value | Description |

Body Cookies (1) Headers (10) Test Results Status: 200 OK Time: 6 ms Size: 384 B Save Response

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "id": 1,
4     "content": "no worries",
5     "created_at": "2023-03-14",
6     "reply": 1
7   }
8 ]

```

Create comment on a user

The url for signup is localhost:8000/comment/commentonuser/<user_id>/<reservation_id>/

The method supported is POST

The endpoint requires the user to be authorized

The url will allow the host of reservation with reservation_id to comment on the user of the reservation. The host can only comment on a user once per reservation.

The request body is expected to be in form-data format and expects all the following parameters:

Content: A textarea that contains the content of the reply

Rating: An integer in [1,2,3,4,5]

An example request may look like this:

POST http://localhost:8000/comment/commentonuser/2/1/ Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

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

| | | | | |
|-------------------------------------|-------------|------|-----------------|--|
| <input type="checkbox"/> | username | Text | another@one.com | |
| <input type="checkbox"/> | password | | 123 | |
| <input type="checkbox"/> | phone | | +14378783377 | |
| <input type="checkbox"/> | profile_pic | | flag.jpeg | |
| <input checked="" type="checkbox"/> | content | | good guest | |
| <input checked="" type="checkbox"/> | rating | | 4 | |
| <input type="checkbox"/> | id | | 7 | |

Body Cookies (1) Headers (10) Test Results Status: 201 Created Time: 16 ms Size: 408 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "content": "good guest",
4   "rating": 4,
5   "created_at": "2023-03-14",
6   "reservation": 1,
7   "user": 2
8 }
```

View comments on a user

The url for viewing a user's comments is localhost:8000/comment/commentonuser/<user_id>/

The method supported is GET

The endpoint requires the user to be authorized

The url will allow the hosts (at least own one property) to see the comments on a user with user_id

An example request may look like this:

GET

http://localhost:8000/comment/commentonuser/2/

Send

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettingsCookies

noneform-datax-www-form-urlencodedrawbinaryGraphQL

| | | | |
|--------------------------|-------------|-----------------|--|
| <input type="checkbox"/> | username | another@one.com | |
| <input type="checkbox"/> | password | 123 | |
| <input type="checkbox"/> | phone | +14378783377 | |
| <input type="checkbox"/> | profile_pic | flag.jpeg × | |
| <input type="checkbox"/> | content | good guest | |
| <input type="checkbox"/> | rating | 4 | |
| <input type="checkbox"/> | id | 7 | |

BodyCookies (1)Headers (10)Test Results

Status: 200 OKTime: 7 msSize: 410 BSave Response

PrettyRawPreviewVisualizeJSON

```
1  [
2    {
3      "id": 1,
4      "content": "good guest",
5      "rating": 4,
6      "created_at": "2023-03-14",
7      "reservation": 1,
8      "user": 2
9    }
10 ]
```

Reservations

Model

Reservation

The Reservation model has the following fields:

- property: a Property object
- user: a User object; the guest of the reservation
- host: a User object; the host of the reservation & the owner of the property
- start_date: a DateField item
- end_date: a DateField item
- state: the current state fo the reservation; can be one of the following, with default being "Pd":
 - "Pd" = Pending
 - "Dn" = Denied
 - "Ex" = Expired
 - "Ap" = Approved
 - "Pc" = Pending Cancellation'),
 - "Cc" = Canceled
 - "Tm" = Terminated
 - "Cm" = Completed
- previous_sate: the previous state whenever a state change happens

URLs

View All Reservations

The url for viewing all reservations that is related to the user is **localhost:8000/reservations/all/**

The method supported is **GET**

The endpoint requires the user to be authorized.

This url returns all reservations that the user booked (as guest) and/or booked by others on one of the user's property (as host), filtered by state if specified, and can be ordered by start date or end date (not specifically required in P2).

The query parameters of the request are as follows:

- **user_type:** can be either 'guest' or 'host', or be left empty; returns reserervations that the user is either the guest or the host of. By default the value is not specified, and all reservations satisfying either one of the above criteria will be returned.
- **state:** accepts a value from ["Pd","Dn","Ex","Ap","Pc","Cc","Tm","Cm"]; returns reservations in that state. By default the value is not specified, and reservations of all states are returned.

- **order_by:** can be either 'start_date', 'end_date', or be left empty. orders the results by the specified input. By default, this value is left empty and results are returned in order of their reservation IDs.
- paginator options:
 - **per_page:** specifies the number of reservations shown per page (default=1).
 - **page:** page number

On Postman, a sample request would look like this:



The request will return a paginated data containing reservations that satisfy the constraints and status code of 200 if successful. Otherwise it will return an error message outlining where the error occurred.

Create Reservations

The url for creating a reservation **localhost:8000/reservations/create/**

The method supported is **POST**

The endpoint requires the user to be authorized.

This url creates a reservation where the user in the request body is assigned as the guest and the property's owner is assigned as the host. If successfully created, the reservation will have a state of "Pd" (Pending).

The required fields of the request are as follows:

- **property:** ID of the property
- **start_date:** start date of the reservation in yyyy-mm-dd format
- **end_date:** end date of the reservation in yyyy-mm-dd format

On Postman, a sample request would look like this:

POST http://localhost:8000/reservations/create/

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

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

| KEY | VALUE |
|--|------------|
| <input checked="" type="checkbox"/> property | 4 |
| <input checked="" type="checkbox"/> start_date | 2023-01-01 |
| <input checked="" type="checkbox"/> end_date | 2023-12-01 |
| Key | Value |

Note: several restraints are placed on the fields:

- start_date must be smaller than end_date (it is assumed that each reservation must be at least 2 dates)
- the start_date and end_date must exactly match one of the availability period that the property has.
- property ID must be a number, and there must exist a property that has that ID (otherwise. a 404 error will be returned)
- a user cannot create reservations on a property that they're the owner of.

If successful, the request will create a Reservation object with status code of 200. Otherwise it will return an error message outlining where the error occurred.

Request Cancel

The url for requesting to cancel an approved reservation is

localhost:8000/reservations/cancel/

The method supported is **PUT**

The endpoint requires the user to be authorized.

This url takes the ID of the reservation that the user wish to be canceled, and after verifying that they're the guest of the reservation and the reservation is in the "approved" state, change the reservation state to "pending cancellation".

The required fields of the request are as follows:

- **id:** ID of the reservation

On Postman, a sample request would look like this:

http://localhost:8000/reservations/cancel/

PUT http://localhost:8000/reservations/cancel/

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

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

| KEY | VALUE |
|--|-------|
| <input checked="" type="checkbox"/> id | 4 |

Note: several restraints are placed on the fields:

- reservation ID must be a number, and there must exist a reservation that has that ID (otherwise. a 404 error will be returned)
- the user of the request must be the guest of the reservation, and the reservation must be in “approved (Ap)” state.

If successful, the request will change the state of the reservation in question to “Pc” (pending cancellation) with status code of 200. Otherwise it will return an error message outlining where the error occurred.

Approve/Deny Pending

The url for requesting to cancel an approved reservation is

localhost:8000/reservations/approve/pending/

The method supported is **PUT**

The endpoint requires the user to be authorized.

This url takes the ID of a pending reservation, and after verifying it’s in “pending” (Pd) state and that the user is the host of the reservation, approve or deny it.

The required fields of the request are as follows:

- **property:** ID of the property
- **action:** approve or deny

On Postman, a sample request would look like this:

The screenshot shows a Postman interface for a PUT request to the URL `http://localhost:8000/reservations/approve/pending/`. The 'Body' tab is selected, and the 'form-data' radio button is chosen. The request body contains two fields: 'id' with the value '5' and 'action' with the value 'deny'. Below the form data, there is a table with columns 'KEY' and 'VALUE'.

| KEY | VALUE |
|--|-------|
| <input checked="" type="checkbox"/> id | 5 |
| <input checked="" type="checkbox"/> action | deny |
| Key | Value |

Note: several restraints are placed on the fields:

- reservation ID must be a number, and there must exist a reservation in “Pc” state that has that ID (otherwise. a 404 error will be returned)
- user must be the host of the reservation.
- the property must have the availability specified in the reservation at the time of approval

If successful, the request will return a status code of 200. If action is “approve”, the reservation state will become “approved” (Ap) and the availability time range for the property of that reservation is removed; if action is “deny”, the reservation state will become “denied” (Dn).

Otherwise it will return an error message outlining where the error occurred.

Approve/Deny Cancel

The url for requesting to cancel an approved reservation is

localhost:8000/reservations/approve/cancel/

The method supported is **PUT**

The endpoint requires the user to be authorized.

This url takes the ID of a reservation, and after verifying it's in "pending cancellation" (Pc) state and that the user is the host of the reservation, approve or deny it.

The required fields of the request are as follows:

- **property:** ID of the property
- **action:** approve or deny

On Postman, a sample request would look like this:

The screenshot shows a Postman interface for a PUT request to the URL `http://localhost:8000/reservations/approve/cancel/`. The 'Body' tab is selected, and the format is set to 'form-data'. The body contains two key-value pairs: 'id' with value '1' and 'action' with value 'deny'. Both fields have checkboxes checked in the left margin.

| KEY | VALUE |
|--------|-------|
| id | 1 |
| action | deny |

Note: several restraints are placed on the fields:

- reservation ID must be a number, and there must exist a reservation in "Pc" state that has that ID (otherwise, a 404 error will be returned)
- user must be the host of the reservation.

If successful, the request will return a status code of 200. If action is "approve", the reservation state will become "canceled" (Cc); if action is "deny", the reservation state will go back to "approved" (Ap).

Otherwise it will return an error message outlining where the error occurred.

Terminate

The url for requesting to cancel an approved reservation is

localhost:8000/reservations/terminate/

The method supported is **PUT**

The endpoint requires the user to be authorized.

This url takes the ID of the reservation that the user wish to be terminate, and after verifying that they're the host of the reservation and the reservation is in the "approved" state, change the reservation state to "terminated" (Tm).

The required fields of the request are as follows:

- **id**: ID of the reservation

On Postman, a sample request would look like this:

The screenshot shows a Postman interface for a PUT request to the URL `http://localhost:8000/reservations/terminate/`. The 'Body' tab is selected, and the 'form-data' radio button is chosen. A single key-value pair is shown in the body: the key is 'id' (checked with a checkbox) and the value is '4'.

| KEY | VALUE |
|--|-------|
| <input checked="" type="checkbox"/> id | 4 |

Note: several restraints are placed on the fields:

- id must be a number, and there must exist a reservation that has that ID (otherwise a 404 error will be returned)
- the user of the request must be the host of the reservation, and the reservation must be in “approved (Ap)” state.

If successful, the request will change the state of the reservation in question to “Tm” (terminated) with status code of 200. Otherwise it will return an error message outlining where the error occurred.

Notifications

Model

ReservationNotification

The ReservationNotification has the following fields:

- user: a User model object; the recipient of the notification.
- reservation: a Reservation model object; the reservation that was created/updated.
- content: a character string of maximum 120 characters; the content of the notification
- state: can either be “R” (read) or “U” (unread). Default is “U”

ReservationNotification objects are created automatically whenever a reservation:

- is created: host gets a notification that a reservation has been created on the property.
- has a state change from “Pd” to “Ap”: user (guest) gets a notification that their pending reservation has been approved.
- has a state change to “Pc”: host gets a notification that the guest of their reservation requested cancellation.
- has a state change to “Cc”: user (guest) gets a notification that their reservation has been cancelled. (pending cancellation request is approved)

URLs

ViewAllNotifications

The url for viewing all notifications that belong to a user is **localhost:8000/notifications/all/**

The method supported is **GET**

The endpoint requires the user to be authorized.

The query parameters of the request are as follows:

- **state:** U (unread) or R (read)
- paginator options:
 - **per_page:** specifies the number of reservations shown per page (default=1).
 - **page:** page number

On Postman, a sample request would look like this:

GET

▼

http://localhost:8000/notifications/all/?per_page=10&page=1&state=U

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Query Params

| | KEY | VALUE |
|-------------------------------------|----------|-------|
| <input checked="" type="checkbox"/> | per_page | 10 |
| <input checked="" type="checkbox"/> | page | 1 |
| <input checked="" type="checkbox"/> | state | U |
| | ... | ... |

Note: A notification becomes read when it is returned in the output. For example, if there are 20 unread notifications and we set per_page=5 and page=1, the first 5 notifications will become read.

On success, all notifications satisfying the conditions will be returned, with status 200.

Clear Notifications

The url for viewing all notifications that belong to a user is **localhost:8000/notifications/clear/**

The method supported is **DELETE**

The endpoint requires the user to be authorized.

No query parameter is needed. This request deletes all notifications that are in “read” state and returns the number of messages deleted.