docs.pdf: You must submit a documentation that includes your design of models, as well as the full list of all API endpoints, a short **description**, their methods, and the payloads. You can use packages that automatically generate the API docs from your code. The TA will send requests based on the information you provide on that document. Therefore, it is important to have a usable and clear document.

Notifications:

List Notifications(also includes receiving notifs):

Endpoint: /notif/list/

Methods: GET

Description: After being authenticated, the user can ***view all the notifications*** they have through this endpoint. It should return JSONs of the different notifications the user has, mainly including the message and the ID of the notification. It also shows any received notifications when reservation status changes.

Read Notification:
Endpoint: /notif/<int: pk>/

Methods: GET

Description: This allows the user to manually ***read a notification*** and change the “is_read” flag to true. By passing <int: pk>, that marks the ID of the notification that the user wants to read.

Delete Notification:

Endpoint: /notif/delete/<int: pk>/

Methods: DELETE

Description: This allows the user to manually clear a notification and delete it from the dataset of notifications. By passing <int: pk>, that marks the ID of the notification that the user wants to delete. If we try to access the same notification ID again, it will not show.

Comments:

View Comments:

Endpoint: /comments/view/

Methods: GET

Description: After being authenticated, the user can ***view all the comments*** made so far through this endpoint. It returns the JSONs of the different comments and replies the user has made for both a guest and a property, mainly including the content, the ID of the comments and the boolean isComment which returns true if it's a comment and false otherwise. It also takes into account the pagination of the comments i.e we are only able to view 5 comments on one page at a time. It should return an HTTP 200 OK.

Add Comment for Guest/User:

Endpoint: /comments/add/

Methods: POST

Payloads: User, Name, Email, Content, isComment

Description: After being authenticated, you are allowed to choose the User/Guest that you want to write the comment for. User displays a dropdown list displaying all the users that have been created so far. Then you add your name, email and the comment you want to post. There is also an option to select whether you want to post a comment or a reply. It will return the JSON response of the comment added displaying the ID of the comment and all the payload fields except the User chosen. It should return HTTP 201 CREATED.

Add Comment for Property:

Endpoint: /comments/addproperty/

Methods: POST

Payloads: Property, Name, Email, Content, isComment

Description: After being authenticated, you are allowed to choose the Property that you want to write the comment for. Property displays a dropdown list displaying all the properties that have been created so far. Then you add your name, email and the comment you want to post. There is also an option to select whether you want to post a comment or a reply. It will return the JSON response of the comment added displaying the ID of the comment and all the payload fields except the property chosen. It should return HTTP 201 CREATED.

Add Reply to a Particular Comment:

Endpoint: /comments/reply/

Methods: POST

Payloads: Parent Comment, Content

Description: After being authenticated, choose a particular comment that you want to reply to from the dropdown of parent comment. (which gives you all the comments that have been created so far) Add your reply in the content field. It should return the JSON response displaying the id and the content. It should also return HTTP 201 CREATED.

Accounts:

Login:

Endpoint: /accounts/login/

Methods: POST

Payloads: username, password

Description: This allows the user to be authenticated and return to their account by entering their username and password. Then, they will be given a token to authenticate them. Note that to **logout**, we simply need to delete that token.

Signup:

Endpoint: /accounts/register/

Methods: POST

Payloads: username, password, email address, first name, last name

Description: This allows the user to register and make their account the first time (do not need authentication for this), i.e it adds Users. It should return the JSON response displaying all the payloads and the ID of the user created. It should also return HTTP 201 CREATED.

Update:

Endpoint: /accounts/update/<int:pk>/

Methods: POST

Payloads: username, password, email address, first name, last name

Description: After being Authenticated, this allows the user to update the information on their account that they set up when they registered. By passing <int: pk>, that marks the ID of the user that the user wants to update and only that user is updated with the particular fields. Also we cannot edit the username to something that already exists so error checking for that has also been performed.
It should return the JSON response displaying all the payloads and the ID of the user that was updated. It should also return HTTP 201 CREATED.

PROPERTY ENDPOINTS

ENDPOINT: /property/property/
METHOD: POST
PAYLOADS:

- Description
- Prop_type
- Address
- Rooms
- Baths
- Parking
- Max_guests
- rate

DESCRIPTION:
While authenticated, a host user can add a property to the database with the above payloads. If successful, it will return a 200 OK and a JSON response of the newly created property.

ENDPOINT: /property/property/<int:prop_id>/
METHOD: DELETE
PAYLOADS:

- None

DESCRIPTION:
While authenticated, a host user can remove a property from the database with the above payloads. The url parameter prop_id is the id of the property that we want to delete. If successful, it will return a 200 OK and a JSON response of the newly created property.

ENDPOINT: /property/property/<int:prop_id>/
METHOD: PUT
PAYLOADS: (NOT ALL REQUIRED, ONLY WHAT IS BEING CHANGED)

- Description
- Prop_type
- Address
- Rooms
- Baths
- Parking
- Max_guests
- rate

DESCRIPTION:

While authenticated, a host user can modify a property in the database with the above payloads. If successful, it will return a 200 OK and a JSON response of the newly modified property.

ENDPOINT: /property/property/search/?[INSERT SEARCH FIELD]
METHOD: GET
PAYLOADS:

None

PAGINATION:

2 Properties displayed per page.

FILTERS:

- **Parking**
- **Baths**
- **Rooms**
- **owner**

ORDERING:

- **Baths**
- **Rooms**

FILTER AND ORDER:

To filter, add a ? mark to the end of the url followed by a filter parameter and the desired value for that parameter. Below, is an example:

/property/property/search/?bahts=2&rooms=1

To specify an order,

/property/property/search/?order=baths

DESCRIPTION:

While authenticated, a host user can search among all properties in the database with the above filters specified. The filtering is done through a django app called django-filter-backends and ordering is achieved by using django's filers.orderingfilter. The pages that resulted show only 2 properties per page, as per pagination requirements.

RESERVATION ENDPOINTS

ENDPOINT: /property/reservations/<int:prop_id>/
METHOD: POST
PAYLOADS:

- check_in
- check_out
- num_days
- numGuests

DESCRIPTION:
While authenticated, a user can create a reservation. They must specify a property they are interested in as prop_id in the url. When successful, it will create a reservation, create a notification for the host of that specific property, and return a 200.

Note, at creation, a reservations' status is set to PENDING

ENDPOINT: /property/reservations/cancel/<int:res_id>/
METHOD: GET
PAYLOADS:

- None

DESCRIPTION:
While authenticated, a user can request for a reservation to be cancelled. This is done through the above endpoint. The url parameter res_id is the reservation id passed on to the view to locate the desired reservation. When action is done successfully, the status of the reservation is set to CANCEL PENDING. A notification is also sent to the property owner notifying them of whether they would like to approve the cancellation.

ENDPOINT: /property/reservations/decision/<int:res_id>/<int:status>/
METHOD: GET
PAYLOADS:

- None

DESCRIPTION:
While authenticated, a host can decide to approve or deny a pending reservation. The payload red_id is the id of the reservation in question and status is defined as followed.

STATUS = 1 : This means that the host means to approve the reservation

STATUS = 2 : This means that the host means to deny the reservation.

Example: /property/reservations/decision/<int:res_id>/1
The above url would approve any given reservation.

When successful, the status of the reservation will change to either APPROVED or REJECTED. And a 200 is sent back to the user.

ENDPOINT: /property/reservations/cancel/pending/<int:res_id>/<int:status>/
METHOD: GET
PAYLOADS:

- None

DESCRIPTION:
While authenticated, a host can choose to either accept or deny a request an already created reservation for one of their properties. The res_id is the id of the said reservation and status is the parameter defined as the following:

STATUS = 1 : This means that the host means to approve the cancellation

STATUS = 2 : This means that the host means to deny the cancellation.

Example: /property/reservations/cancel/pending/<int:res_id>/2/
The above url would reject any given reservation cancellation.
When successful, the status of the reservation will change to either CANCEL APPROVED or CANCEL REJECTED. And a 200 is sent back to the user.

ENDPOINT: /property/reservations/delete/<int:res_id>/
METHOD: DELETE
PAYLOADS:

- check_in
- check_out
- num_days
- numGuests

DESCRIPTION:
While authenticated, a uhost can delete a reservation made on their property. The url parameter res_id is the id of the reservation. When successful, the reservation object will be deleted and 200 is sent back.

ENDPOINT: /property/reservations/search/?[insert filter paramer.
METHOD: POST
PAYLOADS:

- None

FILTERS:

- **Owner_id**
- **Customer_id**
- **status**

DESCRIPTION:

While authenticated, a user can see a list of reservations. They must specify one of the following.

?owner_id = id
To see all the reservations for their properties as a host.

OR

?customer_id = id
To see all the reservations they've made at other properties.

ADDITIONALLY,
They can apply a status filter to see the status of reservations of specific status.

The end point's results are paginated for 2 results per page.