NOTE: After you have initialized the virtual machine, clone the directory, and go to P2 folder. Please "source startup.sh" to enter venv automatically, then ./run.sh
MODELS

RestifyUser { // Inherited from **AbstractUser**
host_or_not: if user is host → users become hosts once they create a property
avatar: user's avatar
phone_number: user's phone number
first_name: user's first name
last_name: user's last name
email: user's email address
}

UserHistory {
comment_for_this_user: ForeignKey(RestifyUser) → person for whom this review is
content: the comment content → the actual comment
}

Property {
property_owner = ForeignKey(RestifyUser) → user who made the property
address : address of the property
number_of_guest: number of guest allowed in the property
number_of_bed: number of beds allowed in property
number_of_rooms: number of rooms in property
baths: number of baths in the property
description: owner's description of the property
essentials: essentials of the property → multiselect field
features: features of the property → multiselect field
location: accessibility of other entertainment → multiselect field
safety feature: safety feature of the property → multiselect field
}

PropertyImage {
name: name of the image
property: ForeignKey(Property)
image: image file uploaded
}

RangePriceHostOffer {
property: ForeignKey(Property)
start_date: available start date of the property
end_date: available end date of the property
price_per_night: price of the property per night during this range of time
booked_for: if that property is booked for this range of time
}

Reservation {
property: Foreign key(Property) → property upon which this reservation is made
user: ForeignKey(RestifyUser) → person who made the reservation
status: a charfield with only 7 possible choices to represent the 7 states of a reservation
start_date: start date of the reservation
end_date: end date of the reservation

available_date: foreignKey(RangePriceHostOffer) → the available date object that cannot be attached to any other reservation, till this reservation frees
num_of_guests: number of guests coming to stay at this place
}

CommentBaseClass { // abstract base class
text_content: textfield containing the comments that someone inputs
reply: distinguishes reply from original comment and the author
}

PropertyComment { // inherits CommentBaseClass
author: ForeignKey(RestifyUser) → user who writes the comment
reservation: ForeignKey(Reservation) → property comment for a specific reservation
}

GuestComment { // inherits CommentBaseClass
author: ForeignKey(RestifyUser) → user who writes the comment
reservation: ForeignKey(Reservation) → property comment for a specific reservation
guest: ForeignKey(RestifyUser) → user who is the guest of this reservation
}

Notification {
user: ForeignKey(RestifyUser) → user who the notification belongs to
user_type: if this is a host notification or user notification since a user can be both user and host under the same profile
read: bool if this notification was read yet or unread
notification_message: the notification message that the user reads
reservation: ForeignKey(Reservation) → which reservation this specific notification is tied to

USER

Endpoint: /webpages/login/
Methods: POST
Fields/payload: username, password
Description: This logs the user in. It basically generates an authentication token that lasts for one day (previously 5 minutes but we changed it)

Endpoint: /webpages/signup/
Methods: POST
Fields/payload: username, email, first_name, last_name, phone_number, host_or_not, avatar, password
Description: Allows for the user to sign up.

Endpoint: /webpages/logout/
Methods: POST
Fields/payload:
Description: Blacklists the token and renders their current token not capable of staying logged in.

Endpoint: /webpages/profile/view/
Methods: GET
Fields/payload:
Description: This view shows all the details of the current logged in user's profile.

Endpoint: /webpages/profile/edit/
Methods: GET, PATCH
Fields/payload: username, email, first_name, last_name, phone_number, host_or_not, avatar, password
Description: Takes either of the fields and their changes values and updates the database with the new values. Also during a GET request, it works the same as profile/view.

PROPERTY

Endpoint: /webpages/property/add/
Methods: POST
Fields/payload: address, number_of_guest, number_of_bed, number_of_rooms, baths, description, essentials, features, location, safety_features
Description: This will add a property to the database, and the owner of the property will be set to the logged in user. Validation errors: This field is required (for all fields except for essentials, features, location, safety_features)

Endpoint: /webpages/properties/all/
Methods: GET
Fields/payload:
Description: This will list all existing properties in Restify that belong to the current logged in user.

Endpoint: /webpages/property/<int:pk>/detail/
Methods: GET
Fields/payload: pk
Description: This will retrieve the information of a single property with its id=pk

Endpoint: /webpages/property/<int:pk>/edit/
Methods: GET
Fields/payload: pk, address, number_of_guest, number_of_bed, number_of_rooms, baths, description, essentials, features, location, safety_features

Description: This will allow the logged in user to edit their property with id=pk. If that user is not the owner of the property, it will return 403 error

Endpoint: /webpages/property/<int:pk>/delete/
Methods: DELETE
Fields/payload: pk
Description: This will allow the logged in user to delete their property with id=pk. If that user is not the owner of the property, it will return 403 error

Endpoint: /webpages/property/search/
Methods: GET
Fields/payload: ?page=1, start_date, end_date, location, number_of_guest
Description: This will allow people to search for the properties based on the required fields: start_date, end_date, location, number_of_guest. For pagination support, the default is 10 results.

Endpoint: /webpages/property/filter/
Methods: GET
Fields/payload: ?page=1, price_per_night, number_of_rooms, number_of_bed, baths, essentials, features, safety_features, location
Description: It will take the body as the result from the search function. This will allow people to filter for the properties based on price_per_night, number_of_rooms, number_of_bed, baths, essentials, features, safety_features, location, which all can be left empty. For pagination support, the default is 10 results.

Endpoint: /webpages/property/order/
Methods: GET
Fields/payload: ?page=1, ordering=baths or ordering=-baths
Description: It will take the body as the result from the search function. It can order the resulting properties based on the number of baths. For pagination support, the default is 10 results.

Endpoint: /webpages/property/price_order/
Methods: GET
Fields/payload: ?page=1, ordering=price_per_night or ordering=-price_per_night
Description: It will take the body as the result from the search function. It can order the resulting properties based on the offered price_per_night. For pagination support, the default is 10 results.

Endpoint: /webpages/<int:pk>/create_timerange_price/
Methods: POST
Fields/payload: price_per_night, start_date, end_date, pk
Description: It will create a RangePriceHostOffer instance that specifies a single range of available dates and its price_per_night of property which property.id=pk. If the logged in user is not the owner of the property, return 403 errors.

Endpoint: /webpages/available_date/<int:pk>/edit/
Methods: GET
Fields/payload: pk, price_per_night, start_date, end_date, pk
Description: It will edit a RangePriceHostOffer instance which RangePriceHostOffer.id=pk. If the logged in user is not the owner of the property, return 403 errors.

Endpoint: /webpages/available_date/<int:pk>/delete/
Methods: DELETE
Fields/payload: pk
Description: It will delete a RangePriceHostOffer instance which RangePriceHostOffer.id=pk. If the

logged in user is not the owner of the property, return 403 errors.

Endpoint: /webpages/available_date/<int:pk>/detail/
Methods: GET
Fields/payload: pk
Description: It will return detailed information of a RangePriceHostOffer instance which RangePriceHostOffer.id=pk. If the logged in user is not the owner of the property, return 403 errors.

Endpoint: /webpages/available_date/<int:pk>/list/
Methods: GET
Fields/payload: pk
Description: It will return all RangePriceHostOffer instances which have the property's id=pk.

Endpoint: /webpages/picture/<int:pk>/add/
Methods: POST
Fields/payload: image, pk, name
Description: It will create a PropertyImage instance that specifies the name and image url with property.id=pk. If the logged in user is not the owner of the property, return 403 errors.

Endpoint: /webpages/picture/<int:pk>/delete/
Methods: DELETE
Fields/payload: pk
Description: It will delete a PropertyImage instance that has property.id=pk. If the logged in user is not the owner of the property, return 403 errors.

Endpoint: /webpages/picture/<int:pk>/detail/
Methods: GET
Fields/payload: pk
Description: It will return detailed information of a PropertyImage instance which PropertyImager.id=pk. If the logged in user is not the owner of the property, return 403 errors.

Endpoint: /webpages/picture/<int:pk>/list/
Methods: GET
Fields/payload:
Description: It will return all PropertyImageinstances which have property's id=pk.

RESERVATIONS

ALL OF THE RELEVANT APIS BELOW INCLUDE PAGINATION SUPPORT (NOT MENTIONED EXPLICITLY) There is pagination support for the endpoints below which can be modified by adding '?page_size=5&page=1' to the end of the endpoint based on what page size or page number you want (page_size=5, page=2 will return results 6-10 if it goes up to 10 items). The default page size is 5 if not inputted

Endpoint: webpages/<int:property_id>/reservations/add/
Methods: POST
Fields/payload: property_id, start_date, end_date, number_of_guest
Description: This view creates a reservation tied to the property specified. It also makes sure there is no reservation already booked for those dates on the property with property ID=property_id

Endpoint: /webpages/reservations/approved/
Methods: GET
Fields/payload: pagination support as above
Description: This view returns all the currently logged in user's approved reservations. That is all the currently logged in user's reservations that have been approved by the host that owns the property with which this reservation is connected. The current logged in user is the one that made the reservation.

Endpoint: /webpages/<int:reservation_id>/terminate_request/
Methods: PATCH
Fields/payload: reservation_id
Description: All this UpdateAPIView does is change the status of a reservation from approved to cancellation requested. To cancel a reservation that has already been approved, the user must request to cancel. This request may be approved or denied. The user, using this view, requests to cancel their reservation.

Endpoint: /webpages/reservations/requested/
Methods: GET
Fields/payload: pagination support as above
Description: This view returns all reservations for a logged in user such that the logged in user made the reservation and that the status of each reservation is "approval requested". Basically, all the reservations which the host has not approved yet.

Endpoint: /webpages/<int:reservation_id>/terminate/
Methods: PATCH
Fields/payload: reservation_id
Description: This UpdateAPIView changes the status of a reservation with reservation ID = reservation_id from approved requested to canceled. Note that since this is for a reservation that was not approved in the first place, the user does not need the approval of the host to cancel this reservation. Note that we are also freeing the RangePriceHostOffer object that is attached to this reservation and make sure the corresponding available date object is available for new reservations to book (changing the RangePriceHostOffer objects booked_for attribute back to false)

Endpoint: /webpages/reservations/cancellations/
Methods: GET
Fields/payload: pagination described as above
Description: This view returns all reservations for a logged in user such that the logged in user made the reservation and that the status of each reservation is "canceled"

Endpoint: /webpages/reservations/completed/
Methods: GET
Fields/payload: pagination support as described above
Description: This view returns all reservations for a logged in user such that the logged in user made the reservation and that the status of each reservation is "Completed"

Endpoint: /webpages/<int:reservation_id>/review_for_host/
Methods: POST
Fields/payload: reservation_id, text_content
Description: This endpoint will add property comments for a specific reservation_id after it has been either completed or terminated. Note that this review for hosts will be counted as a first comment on their property page. It will look for the reservation to ensure it exists, ensure that the current user that is logged in is either the user or the host of the reservation_id. There are max 3 comments allowed: first comment must be from the user, second reply must be from the host, third comment must be from the user. Cannot go in any other order and no additional comments allowed (from P1).

Endpoint: /webpages/reservations/terminated/
Methods: GET
Fields/payload: pagination described as above
Description: This view returns all reservations for
a logged in user such that the logged in user made the reservation and that the
status of each reservation is "Terminated". Reservations with this status are
basically the ones that are follow these routes:
1. Reservation Creation → approval required → approved → request to cancel by user → cancellation request approved by host → status = "Terminated"
2. Reservation Creation → Terminated by host at any point (Current status is irrelevant since Host can terminate with the reservation being in any state)

Endpoint: /webpages/listings/all/
Methods: GET
Fields/payload: pagination described as above
Description: This view returns all the listings that a Host has. Note that this is not reservation dependent. This view solely aims to return all active properties the current logged in user (the host in this case) has.

Endpoint: /webpages/listings/requested/
Methods: GET
Fields/payload: pagination described as above
Description: This returns the current logged in user's (a host in this case) reservations for which he is the property owner (property_owner of the Property that the relevant reservations are linked to, the reservations do not have a property_owner attribute).

Endpoint: /webpages/<int:reservation_id>/approve/
Methods: PATCH
Fields/payload: reservation_id
Description: This view changes the status of a reservation with status = approval requested to status = approved accessing the reservation using reservation_id . In this view the host is basically approving a 'request to reserve' another user has made on their property.

Endpoint: /webpages/<int:reservation_id>/deny/
Methods: PATCH
Fields/payload: reservation_id

Description: This view changes the status of a reservation with status = approval requested to status = denied accessing the reservation using reservation_id . In this view the host is basically denying a 'request to reserve' another user has made on their property. This also frees the RangePriceHostOffer object attached to the reservation and changes the booked_for attribute of that object to false allowing it to be open for other reservations.

Endpoint: /webpages/<int:user_id>/history/
Methods: GET
Fields/payload: user_id, pagination described as above
Description: This ties in with the above. The host can make the decision to approve or deny the user that has made a reservation on their property by checking the user's history out. Therefore, this view accesses the user's UserHistory objects (each of which are just comments the same host or another host made after this user completed a stay at their property) and returns all of them helping the host make an informed decision on whether to let someone stay or not. These are confidential and are only shown to the host when he is deciding to approve or deny a reservation request.

Endpoint: /webpages/listings/approved/
Methods: GET
Fields/payload: pagination described as above
Description: This is a bit different from reservations/approved endpoint. Here we are returning all those reservations for which the current logged in user is a host for, that is all those reservations with properties attached to them for which the current logged in user is a property_owner

Endpoint: /webpages/<int:reservation_id>/termination_by_host/
Methods: PATCH
Fields/payload: reservation_id
Description: Here we are giving the host the option to terminate a reservation no matter what state it is in. This view changes the status of a reservation with pk = reservation_id to "Terminated". This also frees the RangePriceHostOffer object attached to the reservation and changes the booked_for attribute of that object to false allowing it to be open for other reservations.

Endpoint: /webpages/listings/cancellations/
Methods: GET
Fields/payload: pagination described as above
Description: This returns all those reservations which have a status = "Cancelled". Note that this view returns only those reservations with properties for which the current logged in user is a property_owner.

Endpoint: /webpages/<int:reservation_id>/approve_cancellation/
Methods: PATCH
Fields/payload: reservation_id
Description: When a user requests to cancel an approved reservation, they must be approved by the host of the property upon which we have this reservation. Here we are enabling the host to approve the cancellation request which changes the status of the reservation from "Cancellation Request" to "Canceled". This also frees the RangePriceHostOffer object attached to the

reservation and changes the booked_for attribute of that object to false allowing it to be open for other reservations.

Endpoint: /webpages/<int:reservation_id>/deny_cancellation/
Methods: PATCH
Fields/payload: reservation_id
Description: When a user requests to cancel an approved reservation, they must be approved by the host of the property upon which we have this reservation. Here we are enabling the host to deny the cancellation request which changes the status of the reservation from "Cancellation Request" back to "Approved". This also frees the RangePriceHostOffer object attached to the reservation and changes the booked_for attribute of that object to false allowing it to be open for other reservations.

Endpoint: /webpages/listings/completed/
Methods: GET
Fields/payload:
Description: Here we are returning all those reservations that have the status = "Completed". Note that the current logged in user is the property_owner of the properties on which there are completed reservations. This view is for a host to see all his completed reservations wherein he is the property_owner of the corresponding reservations attached.

Endpoint: /webpages/<int:reservation_id>/review_for_guest/
Methods: PATCH
Fields/payload: reservation_id
Description: This view is the backbone of the history button for when a host is deciding to approve or deny a reservation. After the completion of a reservation, each host has the option to leave a review for the guest, which then is added as a UserHistory object to the list of UserHistory objects attached to any given User (In this case the UserHistory object will be attached to the User that made the reservation).

Endpoint: /webpages/listings/terminated/
Methods: GET
Fields/payload:
Description: Here we are returning all those reservations that have the status = "Terminated". Note that the current logged in user is the property_owner of the properties on which there are terminated
reservations. This view is for a host to see all his terminated reservations
wherein he is the property_owner of the corresponding properties attached.

COMMENTS

Endpoint: /webpages/reservations/<int:reservation_id>/property-comments/add/
Methods: POST
Fields/payload: reservation_id, text_content
Description: This endpoint will add property comments for a specific reservation_id after it has been either completed or terminated. It will look for the reservation to ensure it exists, ensure that the current user that is logged in is either the user or the host of the reservation_id. There

are max 3 comments allowed: first comment must be from the user, second reply must be from the host, third comment must be from the user. Cannot go in any other order and no additional comments allowed (from P1).

Endpoint: /webpages/reservations/<int:reservation_id>/property-comments/view/
Methods: GET
Fields/payload: reservation_id
Description: This gets all property comments tied to a specific reservation_id if it exists. Maximum 3 comments will be returned since the rules of user original comment, host reply, user reply is followed.

Endpoint: /webpages/reservations/property/<int:property_id>/property-comments/view/
Methods: GET
Fields/payload: property_id, ?page_size=5&page=1
Description: This will get all the property comments for a specific property, if it exists, so all the comments that have been added from all completed/terminated reservations on this property will be listed here. There is pagination support for this endpoint which can be modified by adding '?page_size=5&page=1' to the end of the endpoint based on what page size or page number you want (page_size=5, page=2 will return results 6-10 if it goes up to 10 items). The default page size is 5 if not inputted.

Endpoint: /webpages/reservations/<int:reservation_id>/guest-comments/add/
Methods: POST
Fields/payload: text_content, reservation_id
Description: This creates a guest comment on a guest only after a specific reservation_id has been completed. Following the same rules as property-comments, there are a maximum of 3 comments and the endpoint can only be created by the logged in user who is either a host or a user in that reservation. This time, the host adds the first comment about the guest, the user will reply, then the host will reply. Cannot go in any other order and no additional comments allowed (from P1).

Endpoint: /webpages/reservations/<int:reservation_id>/guest-comments/view/
Methods: GET
Fields/payload: reservation_id
Description: This gets all guest comments tied to a specific reservation_id if it exists. Maximum 3 comments will be returned since the rules of host original comment, user reply, host reply is followed.

Endpoint: /webpages/reservations/guest/<int:guest_id>/guest-comments/view/
Methods: GET
Fields/payload: guest_id, ?page_size=5&page=1
Description: This will get all the guest comments for a specific guest, if they exist, so all the comments that have been added from all completed reservations on this guest will be listed here. There is pagination support for this endpoint which can be modified by adding '?page_size=5&page=1' to the end of the endpoint based on what page size or page number you want (page_size=5, page=2 will return results 6-10 if it goes up to 10 items). The default page size is 5 if not inputted.

NOTIFICATIONS

Endpoint: /webpages/notifications/<int:reservation_id>/<int:user_id>/create/
Methods: POST
Fields/payload: reservation_id, user_id
Description: This endpoint is meant to be accessed after some external event occurs i.e., status change. When this change occurs, the endpoint will use the reservation_id to get the reservation

status as well as the host and guest of the reservation. Given the user_id, the endpoint will create the correct notification for the user based on the status of the reservation. Note that not all statuses are notification worthy. Each notification will have a read = False Bool field, signifying if they read the notification or not, the notification message based on the reservation's status, and the user_type, which is whether this notification is for a host or a guest.

Endpoint: /webpages/notifications/list/
Methods: GET
Fields/payload: ?page_size=5&page=1
Description: This will take the requesting user's user_id to find all their notifications. This means that the user must be logged in. Then, all notifications that belong to this user will be retrieved regardless of the read status. There is pagination support for this endpoint which can be modified by adding '?page_size=5&page=1' to the end of the endpoint based on what page size or page number you want (page_size=5, page=2 will return results 6-10 if it goes up to 10 items). The default page size is 5 if not inputted.

Endpoint: /webpages/notifications/<int:notification_id>/view/
Methods: PUT
Fields/payload: notification_id
Description: This endpoint will view a specific notification but not clear it. Given the notification_id, the endpoint will first check to make sure that this is the user's own notification since I cannot view someone else's notifications. Then, it will update read = True for that specific notification.

Endpoint: /webpages/notifications/<int:notification_id>/clear/
Methods: DELETE
Fields/payload: notification_id
Description: This endpoint will essentially delete, or clear, a user's notification given the notification_id. First, it will check that the notification_id exists, and it belongs to the user since I cannot clear someone else's notifications. Then, it will clear this notification from the user's notifications.

Endpoint: /webpages/notifications/new/view/
Methods: GET
Fields/payload: ?page_size=5&page=1
Description: This endpoint will list all the user's notifications that are unread or read = False. There is pagination support for this endpoint which can be modified by adding '?page_size=5&page=1' to the end of the endpoint based on what page size or page number you want (page_size=5, page=2 will return results 6-10 if it goes up to 10 items). The default page size is 5 if not inputted.

Endpoint: /webpages/notifications/<str:position>/view/
Methods: GET
Fields/payload: ?page_size=5&page=1, position
Description: This endpoint will list all the user's notifications that are for the given position. This can essentially separate the notifications by 'host' notifications and 'guest' notifications. The string position must be entered as either 'host' or 'guest', or else it is an invalid position string. There is pagination support for this endpoint which can be modified by adding '?page_size=5&page=1' to the end of the endpoint based on what page size or page number you want (page_size=5, page=2 will return results 6-10 if it goes up to 10 items). The default page size is 5 if not inputted.