API Documentation

# User endpoints:

## GET /api/v1/users:

### Using no parameters:

Returns a list of all the users (deleted and active, all roles included) in the database in the following format:

A computer screen shot of white text

Description automatically generated

### Available parameters:

#### ?role=role&deleted=boolean

* **role** can take any of the following values: “SITTER”, “OWNER”, or “ADMIN”.
* **deleted** can either be set to ‘true’ or to ‘false’.
* These parameters are not mandatory. They also don’t have to (but can be) be used together.
* If role is specified, a list of users with that specific role will be returned.
* If deleted is specified and is set to true, only the deleted users will be returned and vice versa.
* Only the relevant relations are loaded. The three types of users share ‘role’ and ‘address’ so these will be loaded for all three. Owners and sitters share the relations ‘reviews\_received’ and ‘reviews\_given’, so these will be loaded for both user types. The relations ‘pets’ and ‘bookings’ will be loaded for owners, and ‘certifications’ and ‘sitting’ for sitters (along with all relations mentioned before).
* Example:

A computer screen shot of white text

Description automatically generated

#### ?start\_date=date&end\_date=date&country=country&city=city

* start\_date and end\_date must be valid dates in the format YYYY-MM-DD and the chosen interval must be at least one day long with start date on the day following the current date of the request.
* country can either be ‘MOROCCO’ or ‘GHANA’
* city must be a valid string with a city name.
* These parameters are not mandatory but MUST be used together. The purpose of these parameters is to get all available sitters in a given country/city in a given date interval.

## GET /api/v1/users/<user\_id>:

This route takes no parameters. Based on the role of the user being requested, the proper relationships will be loaded:

* Address and role for all users.
* Reviews\_received and reviews\_given for both owners and sitters.
* Pets and bookings for owners
* Certifications and sittings for sitters.

## POST /api/v1/users

This route requires the use of one of the following parameters:

#### ?login:

the body must be in the following format:

A screenshot of a computer code

Description automatically generated

This will return the user with the matching email and password.

#### ?register:

this route creates a new user and saves it to the database. The fields required to create a user are:

* fname: the user’s first name. Must be a valid string.
* lname: the user’s first name. Must be a valid string.
* email: must be a valid and unique email
* password: must be a valid string, at least 8 characters long, containing at least one special character, one lowercase letter, one upper case letter, and one digit. gender: must be either ‘f’ (for female) or ‘m’ for male.
* birthday: must be a valid date string. The user must be at least 18 years old.
* role: must be one of the following values: ‘owner’, ‘sitter’, ‘admin’.

Some optional fields include:

* phone: must be a string representing a valid phone number. This field must be unique and must be in the format +212 X XX XX XX XX (for phone numbers in MA region) or +233 XX XXX XXXX (for phone numbers in GH region)
* bank\_account\_number: a string representing a bank account number (at least 10 characters long).
* fee: a number representing the fee to pay a sitter for their services. Only users with the role sitter can set this field.
* Image\_path: the path to the image (stored on the server) of the user

## PUT /api/v1/users/<user\_id>:

The only fields that can be updated for a user are:

* "email", "phone", "bank\_account\_number", "password" and "fee".

## DELETE /api/v1/users/<user\_id>:

* A user is only deleted if they’ve never had a booking or given a review. The data of a user is set to null and the id and all important data (required to keep the database in balance) is kept. The ‘account\_stat’ field is set to ‘DELETED’ if the user can’t be permanently deleted.
* If these constraints aren’t broken, then the user will be removed from the database.

# Address endpoints:

A user is allowed to have only one address at a time. For that reason, all endpoints for ‘address’ are the same except for the request method.

## GET /api/v1/users/<user\_id>/address

Gets the address of a user in the following format:

A computer screen with white text

Description automatically generated

## POST /api/v1/users/<user\_id>/address:

Creates a new address instance in the database. A user can only have one address at a time, meaning that if a user already has an address, this request will fail.

Once the address is created, a json representation (the same as the one in the image above) is returned.

The only fields required to create an address are:

* Street: a string representing a street name.
* City: a string representing a city name
* Country: the value of country can either be ‘morocco’ or ‘ghana’. Any other value will result in an error.
* postal\_code: a string representing the zip code of the address.

The following fields can be set but are not required:

* building\_num: a number representing the number of the building the user resides in.
* apartment\_num: a number representing the apartment number of the address.
* Floor: the floor of the user’s residence.

## PUT /api/v1/users/<user\_id>/address:

Updates the user’s address. All fields mentioned above are updatable. After the instance is updated, a json is returned containing the updated address.

## DELETE /api/v1/users/<user\_id>/address:

Deletes the address associated with the user whose id matches ‘user\_id’.

# Certification endpoints:

Certifications are only available for users with the role ‘SITTER’. If the user id specified in any of the certification endpoints belongs to an owner or an admin, an error will be thrown

## GET /api/v1/users/<user\_id>/certifications:

Fetches the list of certifications that a user has. The JSON returned is formatted as follows:

A computer screen shot of a program code

Description automatically generated

## GET /api/v1/users/<user\_id>/certifications/<certification\_id>:

Returns the certification instance whose id matches <certification\_id>. The format is the same as the JSON above.

## POST /api/v1/users/<user\_id>/certifications:

Creates a certification instance for the user whose id matches <user\_id>. A certification has 3 fields and all three are required to have a value:

Title: the title of the certification.

Issue\_date: the date when the certification was issued. This must be a valid date string in the format YYYY-MM-DD.

Organization: the organization that issued the certification to the sitter.

If the request succeeds, the response received will have a 201 CREATED status code with a JSON representation of the certification instance saved to the database (see example above).

## PUT /api/v1/users/<user\_id>/certifications/<certification\_id>:

All three certification fields are updatable. On successful update, the response will have a 200 OK status code with a JSON representation of the updated certification instance (see example above).

## DELETE /api/v1/users/<user\_id>/certifications/<certification\_id>:

Deletes a certification instance belonging to the user whose id matches <user\_id>.

On successful deletion, the response will contain a 204 NO CONTENT status code. The body will be empty.

# Pet endpoints:

## GET /api/v1/users/<user\_id>/pets

Gets a list of all the pets associated with a user. This route only works if the user id is of a user with the role ‘owner’. The json returned takes the following format:

A computer screen shot of white text

Description automatically generated

## GET /api/v1/users/<user\_id>/pets/<pet\_id>

Gets the pet with ‘pet\_id’ that is associated with the user with ‘user\_id’. The json returned is in the same format as the example above.

## POST /api/v1/users/<user\_id>/pets

Creates a new pet instance. In order to create a pet instance, the following fields are required:

* Name: the name of the pet. A constraint to keep in mind when choosing a pet name is that a user cannot have two pets with the same name.
* Size: the pet size MUST be one of the following values:
* S for small.
* M for medium.
* L for large.
* Description: a string (at least 30 characters long) that describes the pet.

Birthdate: a valid date string representing the date of birth of the pet.

Gender: the gender of the pet. Must be either f (for female) or m (for male).

Breed: the breed of the pet. It must be one of the breeds available in the database (see breed endpoints).

The following fields can (but aren’t required to) be set for a pet:

* Temperament: the default value of this field is ‘friendly’, but it can also take the value ‘aggressive’. This field isn’t required since it has a default value.
* Image\_path: the path to the image (stored on the server) of the pet.

## PUT /api/v1/users/<user\_id>/pets/<pet\_id>:

This endpoint allows you to update a pet that’s associated with a given user. Not all fields can be updated. The only updatable fields are:

* Name, size, description, and temperament.

Make sure to follow the constraints for the values of size and temperament.

## DELETE /api/v1/users/<user\_id>/pets/<pet\_id>:

Deletes a pet instance. If the pet has had bookings in the past, the pet will not be removed. Instead, the status field will be set to DELETED and the image\_path set to null.

If the pet has had no bookings, it will permanently be removed from the database.

# Role endpoints:

Roles cannot be created, deleted, or updated through the API. The only available request method is ‘GET’:

## GET /api/v1/roles:

Returns the list of roles available and for each role, a l of users having this role is included. The format looks like this:

A computer screen shot of white text

Description automatically generated

## GET /api/v1/roles/<role\_id>:

Fetches a single role instance whose id matches <role\_id>. The JSON returned is in the same format as the one above.

# Species endpoints:

Species cannot be created, deleted, or updated through the API. The only available request method is ‘GET’:

## GET /api/v1/species:

Returns a list of all the species in the database. A list of each species’ breeds is also included along with the number of pets in the database of that specific breed. The JSON returned looks like this:

A screenshot of a computer program

Description automatically generated

## GET /api/v1/species/<species\_id>:

Gets a specific species. The json returned is the same as above.

# Breed endpoints:

## GET /api/v1/species/<species\_id>/breeds:

Gets all the breeds of a specific species:

A screenshot of a computer program

Description automatically generated

## GET /api/v1/species/<species\_id>/breeds/<breed\_id>:

Gets a specific breed.

## POST /api/v1/species/<species\_id>/breeds:

Creates a new breed instance whose species is the one with <species\_id>. The only required field to fill is:

* Name: A string representing the name of the breed. Note that this field MUST be UNIQUE. In other words, if you try to create two instances of breed with the same name, an error will be thrown.

If the breed creation is successful, the response status code will be 201 CREATED and the json returned will take the following form:

A computer screen shot of a code

Description automatically generated

## PUT /api/v1/species/<species\_id>/breeds/<breed\_id>:

Edits a specific breed instance. Only the ‘name’ field of the breed can be edited. If a breed has at least one pet associated with it, it cannot be edited. If the PUT request is successful, the response will have a status code 200 OK and the JSON returned will be in the same format as the POST request.

## DELETE /api/v1/species/<species\_id>/breeds/<breed\_id>:

A breed can only be deleted if no pets are associated with it.

# Booking endpoints:

Only users with either the role “SITTER” or “OWNER” can create bookings. Admins do not have bookings. Any request to any booking endpoint that has an admin id (either in the uri or the request body) will result in an error.

## GET /api/v1/users/<user\_id>/bookings:

Gets the list of bookings belonging to a user. The user id MUST belong either to a sitter or an owner. If the user is an admin, an error will be thrown.

## GET /api/v1/users/<user\_id>/bookings/<booking\_id>:

Gets a specific booking.

## POST /api/v1/users/<user\_id>/bookings:

Creates a new booking instance. The only users that can create bookings are those with the role “OWNER”.

The body of the booking request MUST be in the following format:

A screenshot of a computer program

Description automatically generated

* Sitter: the id of the sitter to be booked. This user MUST:
* Have the role “SITTER” (returns a 403 FORBIDDEN if not)
* Have a valid address. (returns a 400 BAD REQUEST if not)
* Must be in the same city and country as the owner. (returns a 400 BAD REQUEST if not)
* Must have a bank account number. (returns a 400 BAD REQUEST if not)
* Must have a fee specified. (returns a 400 BAD REQUEST if not)
* Must be available on the date interval specified (returns 409 CONFLICT if not)
* Pets: a list of pet ids that will be involved in the booking. These pets must belong to the user requesting the bookings.
* If the owner has no pets, a 409 CONFLICT response will be sent.
* If the one or more pet ids are invalid/don’t belong to the owner, a 404 NOT FOUND response will be semt.
* If one or more pets already have bookings on the specified date interval, a 409 CONFLICT response will be sent.
* Start\_date and end\_date: both must be valid date strings in the format YYYY-MM-DD. The start\_date field MUST be at least one day in the future of the day the request is sent. End\_date MUST be at least one day after the start\_date. In other words, the date interval cannot be less than 24 hours and the request MUST be made at least 24 hours before the start\_date.

If the POST request is successful, the response will have a status code of 201 CREATED with a body containing the owner and sitter, their addresses, the pets, and other booking info.

## PUT /api/v1/users/<user\_id>/bookings/<booking\_id>:

Unlike the POST request, PUT can be done either by sitters or owners. However, some constraints must be considered:

* This endpoint does not require a body. In fact, even if a body is sent, it will be ignored.
* The only field that can be edited is the “status” field. This field can only take the following values:
* ACCEPTED or REJECTED: Initially, when an owner creates a booking request, the status is automatically set to “PENDING”. The next step requires the sitter in question either to accept or reject the booking request.
* CANCELLED: Only a booking with the status “ACTIVE” or “ACCEPTED” can be cancelled.
* COMPLETED: A booking can only be marked as completed if the previous status was “ACTIVE”, the payment is not null, and the end\_date has passed.
* An owner can only set the status field to “CANCELLED”. Any other value will result in an error.
* A sitter can only accept or reject a pending booking.
* A sitter cannot cancel a booking that isn’t active or accepted.
* To edit the status of a booking, the uri must contain one of the following parameters:
* ?ACCEPTED, ?REJECTED, ?CANCELLED or ?COMPLETED

## DELETE /api/v1/users/<user\_id>/bookings/<booking\_id>:

The only bookings that can be deleted are those whose status has one of the following values:

* REJECTED
* CANCELLED
* PENDING

# Payment endpoints:

## GET /api/v1/users/<user\_id>/bookings/<booking\_id>/payment:

A booking can only have one payment. Therefore, there is no endpoint that uses the payment id.

## POST /api/v1/users/<user\_id>/bookings/<booking\_id>/payment:

The user id MUST belong to the owner who initially requested this booking. The sitter CANNOT create a payment instance. The post request doesn’t require a body (if present, the body will be ignored). A POST request to this endpoint automatically creates a payment instance and sets the booking’s status to “ACTIVE”. The response has a status code 201 CREATED and the body takes the following format

A computer code with green numbers

Description automatically generated

A payment cannot be edited once it has been created. It can’t be deleted either. The only way to delete a payment is by sending a PUT request to this endpoint /api/v1/users/<user\_id>/bookings/<booking\_id> and including the parameter ?CANCELLED. If the user cancelling the booking is the owner themselves, the payment will only be deleted if this request is at least 24h before the start\_date. If the sitter is the one cancelling the booking, the payment will be deleted regardless of any date constraints. The payment deletion is only included to simulate a refund.

# Review endpoints:

## GET /api/v1/users/<user\_id>/bookings/<booking\_id>/reviews:

Returns a list of reviews associated with a specific booking.

## GET /api/v1/users/<user\_id>/bookings/<booking\_id>/reviews/<review\_id>:

Returns a specific review associated with a specific booking.

## POST /api/v1/users/<user\_id>/bookings/<booking\_id>/reviews:

Creates a review instance for a specific booking. The fields to create a review are:

* Review: this field is mandatory. It’s supposed to be a string that’s at least 30 characters long.
* Rating: this field is not required. It is an integer with a default value of 1. It can be specified but its value can only range from 1 to 5.

The user giving the review is the one whose id is specified in the URI. Therefore, the reviewed user is automatically considered the other user that’s in the booking. A user can only give one single review. If a user attempts to give more than one review, an error will occur.

Reviews can not be deleted nor edited.