Accounts

Models:

For the user model, we decided to use Django’s built-in user model, and created a one-to-one relationship with the model ‘Profile’, to include things such as avatar, phone number, etc.

The Profile model is as follows:

Field	Type + Description
user	User (default from Django). This stores the User object this profile is associated with. There is a one-to-one relationship to ensure that each user has only one profile.
avatar	ImageField. This stores the image used as the profile’s avatar. It is an optional field.
phone_num	CharField. This stores the phone number associated with the profile. This is also an optional field.

Endpoints:

1. ‘accounts/register/’
    a. Method: POST
    b. Payload: username, password, password2 (repeat password), first_name (optional), last_name (optional), email (optional), phone_num (optional).
    c. Description: Given the details in the payload, it attempts to create a User and a Profile instance in the database. A User and Profile instance is created if no validation errors occur, i.e., if:
        i. The username is unique
        ii. The password is long and meets Django’s password constraints
        iii. The two password fields match
        iv. The email provided is of the right format
        v. The phone number (if provided) is of the right format
    d. Return: This returns the Profile object created as a result of registration
    e. Authentication Required: No
2. ‘accounts/login/’
    a. Method: POST
    b. Payload: username, password
    c. Description: Given the username and password, the endpoint authenticates the user and lets them log in, if the credentials are correct.
    d. Return: An AuthToken that is used to authenticate the user in later views.
    e. Authentication Required: No
3. ‘accounts/edit-profile/’
    a. Method: GET, PATCH
    b. Payload:
        i. For GET – nothing.

ii. For PATCH – any of the following fields: first_name, last_name, email, phone_num, avatar

c. Description: An authenticated user can send a GET request to this endpoint to retrieve their profile information. Data returned is first_name, last_name, email, phone_num, avatar, and username (this is read-only). The user can then send a PATCH request to update any of the fields listed in the payload.
d. Return: A paginated JSON content with first_name, last_name, email, phone_num, avatar, and username. If a PATCH request was sent, it displays the updated values. If a GET request was sent, it displays the value stored in the database.
e. Authentication Required: Yes

Studios

Models:

Studio: This stores information about each studio, not including its amenities and images.

Field	Type + Description
id	Automated ID field by Django
name	CharField, name of the studio
address	CharField, the address of the studio, stored in the following format = 'Street No. Street Name, City, Province'
longitude	DecimalField, the studio's longitude
latitude	DecimalField, the studio's latitude
postal_code	CharField, the studio's postal code with the format 'M7W 2W4'
phone_num	CharField, the studio's phone number
distance	FloatField (not required at the start), this field is overridden each time we display studios and holds the distance of a studio from an origin passed in.

Amenities: This has a one-to-one relationship with Studios and stores the amenities associated with a studio.

Field	Type + Description
id	Automated ID field by Django
name	CharField, the name of the amenity
quantity	PositiveIntField, the no. of this amenity available at the studio it is associated with
studio	Studio (foreign key), the studio this amenity is associated with

Images: This has a one-to-one relationship with Studios and stores the images associated with a studio.

Field	Type + Description
id	Automated ID field by Django
studio	Studio (foreign key), the studio this image is associated with
image	ImageField, the image stored

Endpoints:

1. ‘studios/all-studios/
    a. Method: GET
    b. Payload: none
    c. Query Params: origin. This is the location relative to which we want to display the studios from closest to farther. This is to be entered in the form of ‘Street No.%2C+ Street%2C+Name,%2C+City,%2C+Province’
    d. Description: This displays all the studios stored in the database ordered by those closest to the origin provided. If no origin is provided, or an invalid origin is provided, studios are displayed without being ordered by distance.
    e. Return: Paginated JSON with studios ordered by distance (if origin is provided). It returns 50 studios per page, with a link to the next and previous page.
    f. Authentication Required: No. We want all users regardless of having an account to be able to see the studios available to them
2. ‘studios/studio/<int:id>:
    a. Method: GET
    b. Payload: none
    c. Description: Displays information about the studio with id ‘id’ passed into the URL. This includes its name, address, phone number, and the images associated with it.
    d. Return: JSON with the studio details stated above
    e. Authentication Required: No. We want to allow all users to see the details of these studios even without having an account.

Subscriptions

Models:

Subscription: This model is used by admin to create/delete/update subscription models, and are the only subscription options provided to users

Field	Type + Description
id	Automated ID field by Django
price	DecimalField: Price of this subcription
duration	CharField: Duration of this subscription. There are 2 choices available: monthly (‘M’) and yearly (‘Y’)

SubscribedCustomer:

Field	Type + Description
id	Automated ID by Django
user	User (one-to-one), the user with which the subscription is affiliated
subscription	Subscription (one-to-one), stores the subscription chosen by the User
payment_method	CharField, stores the payment method for this subscription
card_num	CharField, stores the card number used to make payments for this subscription
active	BooleanField, denotes whether this user subscription is active or not. If active = False, then it is not.

Payments:

Field	Type + Description
id	Automated ID by Django
cust	SubscribedCustomer (foreign key), the customer who has to make this payment
due_date	DateField, date the payment is due
date_time	DateTimeField, date the payment was made (empty for pending payments)
amount	DecimalField, amount to be paid
payment_made	BooleanField, denotes whether this payment has been made (True) or is pending (False)

Endpoints:

1. ‘subscriptions/update-subscription/’
    a. Method: PATCH
    b. Payload: subs_id (subscription id)
    c. Description: It updates a user’s subscription to the subscription with the id provided, and updates future payments to reflect this change in the subscription.
    d. Return: JSON with the updated subscription information
    e. Authentication Required: Yes
2. ‘subscriptions/cancel-subscription/’
    a. Method: PATCH
    b. Payload: active
    c. Description: It updates a user’s ‘active’ status to ‘False’ to indicate that they no longer have a subscription.
    d. Return: JSON with active set to False
    e. Authentication Required: Yes
3. ‘subscriptions/subscribe/’
    a. Method: POST

   b. Payload:
   c. Description:
   d. Return:
   e. Authentication Required:
4. ‘subscriptions/update-card/’
   a. Method: PATCH
   b. Payload: username, subs_id, payment_method, card_num, active=True
   c. Description: allocates the subscription instance associated with subs_id to the user with the given username, along with the payment method, card number, and active status = True. This denotes that the user has a subscription, with the payment method and details specified above. It also creates all future payment instances that the user must make given the subscription chosen.
   d. Return: JSON with the payload details set as sent in the POST request
   e. Authentication Required: Yes
5. ‘subscriptions/payment-history/’
   a. Method: GET
   b. Payload: None
   c. Description: Returns all completed payments made by the logged-in user
   d. Return: JSON with past payment details like due_date, date_time, amount, card_num (card used to make the payment). If no past payments exist, return JSON with string indicating this.
   e. Authentication Required: Yes
6. ‘subscriptions/upcoming-payments/’
   a. Method: GET
   b. Payload: None
   c. Description: Returns all upcoming payments to be made by the logged-in user
   d. Return: JSON with upcoming payment details like due_date, amount, card_num (card that will be used to make the payment). If no upcoming payments exist, return JSON with string indicating this.
   e. Authentication Required: Yes

Class

Models:

For Class model, I tried to include all the required fields which are ‘name,’ ‘description,’ ‘coach,’ ‘keywords,’ ‘capacity,’ and ‘time.’ In addition, I added ‘studio,’ ‘enrolled,’ ‘group_id,’ and ‘end recursion’ to enhance the functionality. Since keywords in the gym are often shared and it might be useful to be able to find classes based on the keywords in the future, I have implemented a small model “KeyWord” to store keywords.

The Class model is as follows:

Field	Type + Description

name	CharField. This stores the name of the class. Mandatory field.
description	TextField. This stores the description of the class. I made it mandatory because a class without description seems unnatural.
studio	Foreign key to Studio model. Since many classes can belong in one studio, I believed many to one relationship (class to studio) would be reasonable choice.
keywords	Many to many field to KeyWord model. Since keywords are often shared for gym classes, I believed it would be useful to make them as a separate model. Multiple classes can have the samewords and multiple keywords can be in the same class, so I chose many to many relationships.
coach	Foreign Key to User model. After discussing with groupmates, we believed it is reasonable for a class to be run by a single coach. Since a coach can run multiple different classes, we believed foreign key would be the right choice. We are assuming that every coach will be registered in our database.
capacity	PositiveIntegerField with default value of 5. This indicates the available capacity in the class.
enrolled	ManyToManyField to User. Contains users enrolled to the class. Multiple classes can have multiple users and vice versa.
time	DateTimeField that indicates the start date and time of the class. Can be modified in admin panel.
end_recursion	DateField which indicates the end date of the class. Classes are usually held weekly between start time time and end_recursion date.
group_id	UUID that groups all the classes that were created together.

Endpoints:

1. ‘classes/studio/<int: studio_id>/’
   a. Method: GET
   b. Payload: None
   c. Description: Get information of future classes that belong to the studio which has studio_id. Sensitive information is not displayed (such as who is enrolled). Thus, this can be done by even for the people who are not logged in
   d. Return: Paginated REST response string with classes ordered by their starting time.

    e. Authentication Required: No
2. ‘classes/add/<int: class_id>’
    a. Method: GET, POST
    b. Payload: None for both
    c. Description: I implemented two methods that do not require body because in the future, I thought it would be useful to do so. For example, if a user wants to simply find classes for whatever reason without enrolling, GET request which simply tells the user if the user can enroll or not would be sufficient. POST request, however, enrolls the user if the class has a spot and the user is subscribed.
    d. Return: REST response string tells if the user could or could not enroll. If successfully enrolled, the message will include which class the user enrolled
    e. Authentication Required: Yes (login and subscription)
3. ‘classes/drop/<int: class_id>’
    a. Method: GET, POST
    b. Payload: None for both
    c. Description: Works similarly to the above endpoint except that this drops the student from the class.
    d. Return: REST response string tells if the user could or could not enroll. If successfully dropped, the message will include which class the user dropped from.
    e. Authentication Required: Yes (login and subscription)
4. ‘classes/addall/<int: class_id>’
    a. Method: GET, POST
    b. Payload: None for both
    c. Description: Works similarly to ‘classes/add/<int: class_id>’ except that this time, it enrolls the student to the class with class_id and every subsequent classes that share the same ‘group_id.’ If used with GET method, it will only show basic information as if the user is already enrolled or they can enroll, etc. If used POST method, the user will be enrolled in every classes that the user is not already enrolled if the user is subscrib (and of course if the class has available capacity).).
    d. Return: REST response string tells if the user could or could not enroll. If successfully enrolled, the message will include how many classes the user could successfully enroll in.
    e. Authentication Required: Yes (login and subscription)
5. ‘classes /dropall/<int: class_id>’
    a. Method: GET, POST
    b. Payload: None for both
    c. Description: Similar to above except this drops the student from classes instead of enrolling them.
    d. Return: REST response string tells if the user could or could not drop. If successfully dropped, the message will include the number of classes the user could drop from.
    e. Authentication Required: Yes (login and subscription)
6. ‘classes /view/<int: user_id>’
    a. Method: GET

    b. Payload: None
    c. Description: Shows every class that has included user with ‘user_id’ in their enrolled field in chronological order.
    d. Return: This returns the list of classes the user has ever been enrolled in (and didn’t drop at the end) in chronological order. A user can only view his/her information.
    e. Authentication Required: Yes (login)

Search and Filter

Endpoints:

1. search_filter/studio-filter-name/studio__name=<str:studio_name>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all studios with the name ‘studio name’
    d. Return: JSON with the studio details
    e. Authentication Required: No
2. search_filter/studio-filter-coach/<str:coach_name>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all studios where coach ‘coach_name’ takes a class.
    d. Return: JSON with the studio details
    e. Authentication Required: No
3. search_filter/studio-filter-class/<str:class_name>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all studios with class ‘class_name’ offered
    d. Return: JSON with the studio details
    e. Authentication Required: No
4. search_filter/studio-filter-amenities/<str:amenity_name>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all studios with amenity with name ‘amenity_name’ available
    d. Return: JSON with the studio details
    e. Authentication Required: No
5. search_filter/class-filter-name/<str:class_name>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all classes with the name ‘class_name’
    d. Return: JSON with the class details
    e. Authentication Required: No
6. search_filter/class-filter-coach/<str:coach_name>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all classes coached by ‘coach_name’

    d. Return: JSON with the class details
    e. Authentication Required: No
7. search_filter/class-filter-day/<date>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all classes offered on day ‘date’
    d. Return: JSON with the class details
    e. Authentication Required: No
8. search_filter/class-filter-hour/<hour>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all classes offered at time ‘hour’
    d. Return: JSON with the class details
    e. Authentication Required: No
9. search_filter/class-filter-month/<month>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all classes offered in month ‘month’
    d. Return: JSON with the class details
    e. Authentication Required: No
10. search_filter/class-filter-year/<year>/
    a. Method: GET
    b. Payload: NONE
    c. Description: Shows all classes offered in year ‘year’
    d. Return: JSON with the class details
    e. Authentication Required: No

Admin Panel Feature for Class

What’s added:

When classes are first created (saved), subsequent classes with the same information except enrolled are created based on the ‘time’ (shows as ‘Start Time’) and ‘end_recursion’ (shows as ‘End Time’). Classes are created so that the ‘Start Time’ is incremented by 7 day each until reaching ‘End Time’ date.

This brings one disadvantage which is if an admin clicks the class that has already been created and wants to modify the information, they would have to be careful to check the ‘End Time’ to prevent creating more recursive (possibly duplicate) classes. I tried to tackle this problem in two ways. All the subsequently created classes have their ‘End Time’ set to their ‘Start Time’ so that they don’t have this issue (except for the very first class that has been created). And basic information such as start time can be modified in the list view of admin panel directly without going into the individual item.

In the list view of classes, an admin can choose to delete selected classes, selected classes and their subsequent classes, modify the date and time of the selected classes and finally modify just the time of the selected classes.

Because we are not allowed to implement any html, I couldn’t change the structure of the page explicitly. Instead, I added functionalities so that most of the important modifications can be done in bulk. If necessary fields are not provided, no change would occur.

For “Change start date and time of the selected classes,” datetime field needs to be filled to change the date and time of the selected classes.

For “Change start time of the selected classes,” time field needs to be filled to change the time of the selected classes.