Project Backend API Documentation

1. **Design of models:**
   - **Accounts**

     - **User**: The default Django User.
       - first_name: A char field, stores the user's first name.
       - last_name: A char field, stores the user's last name.
       - email: A char field, stores the user's email.
       - phone number: A char field, stores the user's phone number.

     - **CardInfo**: The model that represents a card in the payment system.
       - card_num: A char field, stores the number of the card.
       - expiry_date: A date field, stores the expiry date of the card.

     - **Profile**: A decorator model for each of the users.
       - user: A one-to-one field to the django user.
       - avatar: An image field, stores the avatar of the user.
       - cardInfo: A foreign-key field, stores the user's payment card.

     - **Subscription**: A model that represents a subscription plan of a studio.
       - studio: A foreign-key field, stores the studio that the subscription is for.
       - subscription_type: A char field of choice 'Monthly' or 'Yearly', represents how long each billing cycle would be.
       - rate: An Integer Field, which represents the price of subscription for each billing cycle.

     - **UserSubscription**: A model that represents the subscription of a user to a studio's subscription plan.
       - subscription: A foreign-key field, stores the subscription plan that the user is subscribing to.
       - user: A foreign-key field, stores the user that is making this subscription.
       - start_time: A Datefield, stores the date of start of this subcription.
       - end_time: A Datefield, stores the date of end of this subscription.
       - auto_renew: A boolean field, stores whether this subscription will be auto-renewed when it ends.

- **Payment**: A model that represents a payment made by a user.
    - amount: An integer field, stores the amount of money of this payment.
    - CardInfo: A foreign-key field, stores the cardInfo that the payment is using.
    - time: A datetime field, stores the time of this payment.
    - user: A foreign-key field, stores the user that is making this payment.

- **Studios**

2. **API endpoints:**
    - **Accounts:**
        1. **/Accounts/register/**
           Authentication: Not required
           Web Admin: Not required

           post:Register an account, taking username, password1, password2, email, first_name, last_name, avatar(optional).

        2. **/Accounts/updateprofile/**
           Authentication: Required
           Web Admin: Not required

           post: Update the user profile, takes user_id, username(optional), password(optional), email(optional), first_name(optional), last_name(optional), and avatar(optional).

        3. **/Accounts/getprofile/**
           Authentication: Required
           Web Admin: Not required

           get: Get a json response of the current information of the user profile.

        4. **/Accounts/addcard/**
           Authentication: Required

Web Admin: Not required

post: Create a CardInfo Instance, taking card_num and expiry_date.

5. /Accounts/updatepayment/
   Authentication: Required
   Web Admin: Not required

   post: Take the user_id and card_num, update the user's payment method to this card.

6. /Accounts/createpayment/
   Authentication: Required
   Web Admin: Not required

   post: Create a new payment and save it to the database.
   Takes amount, card_num, time(optional, default is the current time) and
   user_id(optional, default is the user of the request).

7. /Accounts/manageusersubscription/
   Authentication: Required
   Web Admin: Not required

   post: Create a user subscription plan for the current user. Take a start_time, end_time, plan_id, and auto_renew of 'True' or 'False'.

   get: Get all the subscription plan of the current user.

8. /Accounts/editusersubscription/
   Authentication: Required
   Web Admin: Not required

post: Take a user_sub_id(UserSubscription id), start_time, end_time, auto_renew and update the subscription accordingly.

9. /Accounts/managesubscription/
   Authentication: Required
   Web Admin: Required

   post: Take a studio_id, type of 'Monthly' or 'Yearly', rate. Create the corresponding subscription plan.

   get: Take a studio_id, return a json response containing the subscription plans of it.

10. /Accounts/editsubscription/
    Authentication: Required
    Web Admin: Required

    post: Update the subscription of the given subscription_id.
    Take a sub_id, type of 'Monthly' or 'Yearly'(optional), rate(optional).

11. /Accounts/deletesubscription/
    Authentication: Required
    Web Admin: Required

    post: Take a sub_id(Subscription id), delete it from the system.

12. /Accounts/paymenthistory/
    Authentication: Required
    Web Admin: Not required

    get: Return the json response containing all the payment history of current user.

13. /Accounts/api/token/
    Authentication: Not Required

Web Admin: Not required

post: Take username and password, retrieve the authentication access token of this user, which can be used as bearer token in requests that requires authentications.

- **Studios:**

3. /Studios/<id>/
    a. Post:
        i. Payload: name, address, latitude, postal_code, phone_number
        ii. Update the studio with id with the playloads
    b. Get: get the detail of studio with the given in
4. <studio_id>/images/
    a. Get: get the images of the studio with given id
    b. Post: Delete the image with the given id (note the value of studio_id is not important here)
        i. Payload: img_id
    c. Put: Add the payload to the set of images of the studio with the given id
        i. Payload: image (a image file)
5. <studio_id>/amenities/
    a. Get: get the amenity of the studio with given id
    b. Post: update the amenity with type type, create or delete if needed
        i. Payload: type, amount
6. <studio_id>/classes
    a. Get: the classes in a studio
    b. Post: create a class
        i. Payload: coach (the user id of the coach), name, description, start_time, end_time, start_date, end_date, recurse_time (days between each class), max_size, key_word
7. classes/deleat_class/
    a. Post: delete a class on a date or all instance of the class
        i. class_id, date (in form of YYYY-MM-DD, or empty for deleting all class in the future)
    b. Get: get all class the current user enrolled
8. classes/<class_id>/<d>/
    a. Put: enroll the user in the class with class_id on date d(format YYYY-MM-DD, all for all future classes)
    b. Delete: delete the user in the class with class_id on date d(format YYYY-MM-DD, all for all future classes)
9. /
    a. Get: get 5 studios that is index-th to index+5-th closest to the given location
        i. Payload: latitude, longitude, index

   b. Post: create a new Studio
      i. Payload: name, address, latitude, longitude, postal_coad,phone_number

10. Search/Filter
   - The search and Filters are grouped into two categories, one for studios and one for classes
   - For all search/filter, only get, no post
   - Each type of the search(e.g. By name, by coach….) will have a different url
   - Search/filter through the listed studios
      - GetClass_name(View): user given the name that they want to search by entering for key=name, the function will filter through all the classes with the given name with end dates later than the current date. (In other words, it will not display classes that is no longer available)
      - GetClass_time(View): user given a date that wants to search, the function will call for a helper function check_has_class to loop through all classes. The helper function will return true if the class will occur on that day. In this case, we will include it in our Json response
      - GetClass_range(View): The user will be asked to type two dates, the first date must be before the second date. Very similar to GetClass_time, but instead, we are listing all the classes that occur during the range of two dates.
      - GetClass_coach(View): The user will be asked to input the coach, the function will return all classes with that coach
      - GetStudio_name(View): user given the name that they want to search by entering for key=name, the function will filter through all the studios with the given name with end dates later than the current date. (In other words, it will not display classes that is no longer available)
      - GetStudio_amentities(View): User type a type of amenities, the function returns all studios with that amenetites. The function basically filter all amenities objects with the type, and return it’s foreign key Studio Object
      - GetStudio_coach(View): The user will be asked to input the coach, the function will return all studios with at least one class taught by that coach
      - GetStudio_class(View): The user will type in a class that they want to search, the function will return all studios that contains that class including thus studios that only offered that class in the past

PostMan User Case:

For Studios:

- `path('<id>/',Stodio_Detail_View.as_view()),`
- `path('<studio_id>/images/',Image_view.as_view()),`
- `path('<studio_id>/amenities/',Amenities_view.as_view()),`
- `path('<studio_id>/classes/',stodio_class_view.as_view()),`
- `path('classes/deleat_class/',class_edit.as_view()),`
- `path('classes/<class_id>/<d>/',User_Enroll_class_view.as_view()),`
- `path('/',Stodio_View.as_view()),`
- `path('search/class/byname',GetClass_name.as_view()),`
- `path('search/class/bydate',GetClass_time.as_view()),`
- `path('search/class/bydaterange',GetClass_range.as_view()),`
- `path('search/class/bycoach',GetClass_coach.as_view()),`
- `path('search/studio/byclass',GetStudio_class.as_view()),`
- `path('search/studio/byamentities',GetStudio_amentities.as_view()),`
- `path('search/studio/bycoach',GetStudio_coach.as_view()),`
- `path('search/studio/name',GetStudio_name.as_view()),`

For Accounts:

- `path('register/', RegisterView.as_view()),`
- `path('updateprofile/', UpdateProfile.as_view()),`
- `path('getprofile/', GetProfile.as_view()),`
- `path('addcard/', AddCard.as_view()),`
- `path('updatepayment/', UpdatePayment.as_view()),`
- `path('createpayment/', CreatePayment.as_view()),`
- `path('manageusersubscription/', ManageUserSubscription.as_view()),`
- `path('editusersubscription/', EditUserSubscription.as_view()),`
- `path('managesubscription/', ManageSubscription.as_view()),`
- `path('editsubscription/', EditSubscription.as_view()),`
- `path('deletesubscription/', DeleteSubscription.as_view()),`

```
-     path('paymenthistory/', PaymentHistory.as_view()),
-     path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair')
-
```