CSC 309 Backend Project Documentation

Table of Contents

Descriptions

Accounts:

There are three models in this application.

- **CustomUser**: a custom user model that extends the AbstractUser class. Most attributes are inherited from the AbstractUser class except for “avatar”, “is_subscribed”, “phone_num”, and “pmt_option”.
  Model attributes:
  - username: a username field to take in the user's username.
  - password: the user’s password for this account.
  - email: an email registered for this account.
  - avatar: an image field for the user to upload profile pictures.
  - is_subscribed: a boolean indicates if the user is subscribed to our plan.
  - pmt_option: a choice field to indicate which plan the user is subscribed to, or if the plan is cancelled. The choices are “Monthly”, “Yearly”, and “Inactive / Cancelled”.
  - sub_edate: the end date of the user’s subscription service for the current payment period. This date will be renewed every time the user makes a payment on time.

- **Card**: a debit / credit card object that records the user’s payment method.
  Model attributes:
  - card_num: the card number of the card the user wants to register.
  - billing_addr: the billing address of the user’s card.
  - expires_at: the expiry date of this card.
  - cvv: the 3-digit card verification value of a card. Usually required for online payments.
  - holder: the holder of this card. It references a CustomUser object.

- **Payment**: a recurring payment of a user after it subscribes to one of our plans. Depending on the user's choice of the plan type (monthly or yearly), the recurring period varies, but the payment cycle is 1 year no matter what the plan type is unless the user chooses to renew.
  Model attributes:
    - amount: the amount needs to be paid for this payment.
    - pmt_method: the card number of the card that will be used to pay for this payment. A card registered at the beginning of the subscription will be set automatically for all subsequent payments unless modified.
    - pmt_date: the due date of this payment.
    - recur: the recurring period of this payment. It is either "monthly" or "yearly".
    - edate: the end date of the payment cycle.
    - pmt_status: the status of this payment to show whether it is paid already, pending to be paid, or cancelled due to unsubscription.

- **MembershipPlan**: a gym subscription plan.
  Model attributes:
    - price: the plan price.
    - type: the pricing structure of this subscription. Monthly or Yearly.

- **Endpoints & Methods**
    - Endpoint: api/token/
      Methods: POST
      Payloads: username, password

    - Endpoint: account/signup/
      Methods: POST
      Payloads: username, password, password2, email, phone_num, avatar, first_name, last_name, pmt_option, sub_edate

    - Endpoint: account/add_payment_method/

Methods: POST
Payloads: card_num, billing_addr, expires_at, cvv, holder

- Endpoint: account/<int:pk>/profile/edit/
  Methods: PATCH
  Payloads: username, password, email, phone_num, avatar, first_name, last_name, pmt_option
  Additional note: “pk” in the url refers to an user id. None of the payloads is required here.

- Endpoint: account/<int:pk>/profile/edit/
  Methods: PATCH
  Payloads: billing_addr, expires_at
  Additional note: “pk” in the url refers to a card id. None of the payloads is required here.

- Endpoint: account/payment_history/
  Methods: GET
  Payloads:

- Managing command “pay”: a managing command to manually make a payment for an existing user. It will allow payments when the due date is today.
  Usage: python3 manage.py pay <username>

Studios:

Model Attributes:

- id: Auto field to generate the added sequence of a studio
- name: Studio's Name
- address: Studio's Address, usually includes street name, city, and province.
- geolocation: Longitude followed by latitude.
- postalcode: Postal code of studio address
- phonenumber: The phone number of the studio
- amenitiestype: The type of amenities included in the studio.
- amenitiesquantity: The amount of each amenity in the studio.
- imagelist: A set of images, images link needed.

Example of Model (Assuming lon=1&lat=3):


```
{
    "id": 6,
    "name": "Victoria College fitness",
    "address": "Victoria College, Toronto, ON",
    "geolocation": "160,10",
    "postalcode": "M5S 1K6",
    "phonenumber": "1452369870",
    "amenitiestype": "Squash, Pool",
    "amenitiesquantity": "2, 4",
    "imagelist":"https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.vic.utoronto.
    ca%2Fassets%2FUploads%2FHomePage%2FTwoColumn%2Fold-vic-homepage.jpg&i
    mgrefurl=https%3A%2F%2Fwww.vic.utoronto.ca%2F&tbnid=eRrQxDMAG8SAGM&vet=
    12ahUKEwjL5bvEtrv7AhUgrXIEHZJjDJUQMygAegUIARDhAQ..i&docid=UpJ_Vlk5usjPN
    M&w=339&h=226&q=Victoria%20College&ved=2ahUKEwjL5bvEtrv7AhUgrXIEHZJjDJU
    QMygAegUIARDhAQ",
    "distance": 17276.85625152215
}
```

Endpoint & Methods:

DistanceView endpoint: studios/distance

Get the user's current position as geolocation, and return the studio info list from the closest one. Studio info includes its name, address, geolocation, postal code, phone number, amenities type, amenities quantity, images, and distance between the user and studios. And five studios per page.

- Method: Get
- Payloads: Three payloads needed: lon, lat, page. And lon stands for longitude, range between -180 and 180; lat stands for latitude, range between -90 and 90; page stands for current page number, it always starts from 1.
- Api URL examples:
    - localhost:8000/studios/distance?lon=116.63914&lat=39.607054&page=1
    - localhost:8000/studios/distance?lon=-80&lat=0&page=2

StudioView endpoint: studios/query

Get user input as keywords, and return the studio info list with corresponding information. If there is more than one studio match, list the studios from the closest one. Studio info includes its name, address, geolocation, postal code, phone number, amenities type, amenities quantity, images, and distance between user and studios. And five studios per page.

- Method: Get
- Payloads: Four payloads needed: key, page, lon, lat. Key stands for keywords, and no space after comma; page stands for current page number, it always starts

from 1; lon stands for longitude, range between -180 and 180; lat stands for latitude, range between -90 and 90.

Addition information to key: we only do searching on name, address, postal code, phone number, amenities type, *class name*, and *coach name*.

Example of valid searching on each field (Regarding to example model above):

- Name(Studio Name): Victoria College fitness/Victoria College / fitness/ Victoria
- Address: Victoria College,Toronto,ON/Victoria College/Toronto/ON
- Postal code: M5S 1K6
- Phone number: 1452369870
- Amenity type: Squash,Pool/ Squash/ Pool/pool
- Class name (i.e. Swim Beginner): Swim Beginner/Swim/Beginner/swim
- Coach (i.e.Taylor Swift)：Taylor Swift

Api url examples (valid search based on example model):

- studios/query?key=Pool&page=1&lon=116.63914&lat=39.607054
- studios/query?key=Taylor Swift&page=1&lon=116.63914&lat=39.607054
- studios/query?key=fitness&page=1&lon=116.63914&lat=39.607054
- studios/query?key=1452369870&page=2&lon=12.63914&lat=39.607054

Class:

There are 3 models for Class.

1, Class

Model Attributes:

- Name: the name of the class. e.g.swimming
- Description: the description of the class

- Coach: the coach of the class
- Keywords: the keywords of the class. Use white space to separate different keywords. E.g. core fun
- Capacity: the capacity of the class
- Start_time: the start time of the first instance of this class. E.g. the swimming classes have instances with start time 2022-11-1 9:00, 2022-11-8 9:00,.... respectively. The start time of swimming is 2022-11-1 9:00.
- duration : the duration of the class. Each instance of this class has the same duration
- End_time: the end time of this class. Every instance of this class ends before end time.
- Studio: the studio that holds this class

2, ClassInstance: ClassInstances are automatically created when its corresponding Class is created in admin panel

Model Attributes:

- The_class: the corresponding class object of this class. Different class instance objects may share the same class objects.
- Class_name: this is the of name the_class by default
- Description: this is the of description the_class by default
- Coach: this is the of coach the_class by default
- Keywords: this is the of keywords the_class by default
- Space_avaibability: the space availability of this class instance, which equals the capacity of the_class when the class instance is created at the first place.
- Start_time : the start time of this class instance
- End_time: the end time of this class instance which equals the start_time of this class instance + the duration of the_class
- Is_cancelled: a boolean which indicates if the class instance is cancelled, and it is False by default.

3, UserEnrolledClass: a model that is used to record that each user has enrolled in what classes.

Model Attributes:

- user_id : the user id of a user who has enrolled in a specific class instance
- Class_instance: the class instance that the user has enrolled
- Class_instance_name
- Class_instance_start_time
- Class_instance_end_time

Endpoint: /classes/<str:studio>/get_classes/
Method: GET
Payloads:
Description: <str:studio> refers to a studio's name, we can get all class instances held in this studio.

Endpoint: /classes/enrol_class/
Method: POST
Payloads: class_id
Description: the user that has logged in can enrol a class instance with class_id.
(a user can get the id of class instances by the endpoint /classes/<str:studio>/get_classes/)

Endpoint: /classes/my_class_history/
Method: GET
Payloads:
Description: get the class history of the user that has logged in.

Endpoint: /classes/my_class_schedule/
Method: GET
Payloads:
Description: get the class schedule of the user that has logged in.

Endpoint: /classes/drop_class/
Method: POST
Payloads: class_id
Description: the user that has logged in can drop a class instance with class_id.
(a user can get the ids of the class instances that he or she has enrolled by the end point /classes/my_class_history/)

Endpoint: /classes/search_or_filter_1/<str:studio>/<str:class>/
Method: GET
Payloads:
Description: <str:studio> refers to a studio name. <str:class> refers to a class name like swimming. A user can search/filter the class instances with class name <str:class> in studio with studio name <str:studio>.

Endpoint: /classes/search_or_filter_2/<str:studio>/<str:coach>/
Method: GET
Payloads:
Description: <str:studio> refers to a studio name. <str:coach> refers to a coach name. A user can search/filter the class instances with coach name <str:coach> in studio with studio name <str:studio>.

Endpoint: /classes/search_or_filter_3/<str:studio>/<int:year>/<int:month>/<int:day>/
Method: GET
Payloads:
Description: <str:studio> refers to a studio name. A user can search/filter the class instances on the date <int:year> - <int:month> - <int:day> in studio with studio name <str:studio>.

Endpoint:
/classes/search_or_filter_4/<str:studio>/<int:hour1>/<int:minute1>/<int:hour2>/<int:minute2>/
Method: GET
Payloads:
Description: <str:studio> refers to a studio name. A user can search/filter the class instances which are held from <int:hour1>: <int:minute1> to <int:hour2>: <int:minute2> in studio with studio name <str:studio>.

Test Guide

- **IMPORTANT**: Please create at least a yearly MembershipPlan object through the website admin before creating any other instances because our design is based on the assumption that a gym subscription plan already exists for users to choose from. One plan for one plan type only.
- **NOTE**: A card and picking a plan are required for a user to be properly subscribed to a gym plan.
- Users are only able to update the billing address and the expiry date of an existing card. If users want to update the card number and the CVV code, they would need to register a new card.
- A managing command named “pay” was included to allow manual payments. The argument of this command is an existing username.
  Usage:
  python3 manage.py pay <username>
  e.g. python3 manage,py pay user1

Procedures:

1. Create a user. Username, password, password2, and email are required.
2. Log in.
3. Add a card. All fields are required except for “holder”, it is the logged-in user by default. After this step, the user should be subscribed, and a set of payments were generated with the first payment being already paid automatically. This can be viewed in payment history.
4. Now you can update any fields in the edit profile view. Values are pre-filled in postman, but you are free to change the values. All fields are optional. If you change “pmt_option”, the payments already created will be changed to reflect this change. For example, if the plan was changed from monthly to yearly, the number of payments would be less, and the amount would increase. The payment date of each payment would also be updated to match the new billing cycle. Although the first paid payment would not be changed because it has already been processed.
5. To update the card info, as explained above, you can only update “billing_addr” and “expires_at”. These fields are also optional.
6. Now, you can view the payment history.
7. need to create a studio named studio1
    a. Studio Model

- name: Characters, maximum length 50
- address: Enter street name, city, and province, and separate each field by comma
- geolocation: Longitude followed by latitude, longitude range between -180 and 180, and latitude range between -90 and 90, separated by a comma
- postalcode: Entire postal code, space needed.
- phonenumber: 10-digit integer

- amenitiestype: Amenities' name included in the studio, separated by commas
- amenitiesquantity: integers, corresponding to each amenity, separated by commas
- Imagelist: link of images, separated by a comma

Note: For the two endpoints of the studio, many example URLs have been created in the postman collection, and you can use the information to build the studio without changing the params. This is just a suggestion, you can create the studio according to your needs.

8. create a class named swimming whose coach is james and start time is 2022-12-10 09:00:00 and duration is 01:00:00 and end time is 2023-10-11 22:00:00.

   These are necessary since some inputs are taken from url