Toronto Fitness Club Django Backend Documentation

Group members: Nazanin, Parth and Tajwaar

0. Set up instructions

First run $startup.sh$ to setup the environment.

Then run $./manage.py\ createsuperuser$ to create an admin to access the admin panel

Then run $run.sh$ to start the server.

After creating user, adding card, and subscription plan, run $./tfc/manage.py\ payment$ command to make payments due for the day.

1. Models

1.1 $Card$

Name of Model	$Card$
Description	This represents the card that the user adds to their profile. Only 1 card per user allowed.
Required Fields	$number, cvv, expiration, postal\_code$, $used$

1.2 $MyUser$

Name of Model	$MyUser$
Description	Represents a user of the gym, which is a subclass of $AbstractUser$.
Required Fields	$username, password, password2, first\_name, last\_name, email, phone\_number, subscription, card, due\_date.$
Optional Fields	$avatar$
Notes	Subscription is the subscription plan of the user, card is the card the user puts for their account, and due_date is the date that the payment is due. These three are initiated as null when registering. They are set in other API endpoints($update\_card, subscribe, unsubscribe$)

1.3 $FuturePayment$

Name of Model	$FuturePayment$
Description	Represents a potential future payment that can be made.
Fields	$card, payment\_price, payment\_date$
Notes	These are based on the current card and subscription status of the user.

1.4 $PreviousPayment$

Name of Model	$PreviousPayment$
Description	Represents a previous payment made.
Fields	$card, payment\_price, payment\_date$
Notes	These are based on the former card and subscription status of the user.

1.5 $Subscription$

Name of Model	$Subscription$
Description	Represents a subscription plan of the gym.
Fields	$subscription\_interval$ ("$week$" or "$year$" or "$month$"), $price$
Notes	Created by admin in Django Admin panel

1.6 $Enrollment$

Name of Model	$Enrollment$
Description	Represents an enrollment of a user into a class on a specific date
Fields	$user$ ($foreign\ key$), $class\_instance$ ($foreign\ key$) , $date$
Notes	$user$ is a foreign key that represents the user, and $class\_instance$ is the foreign key that represents the class instance the user is enrolled in.

1.7 $Studio$

Name of Model	$Studio$
Description	Represents a studio in the company
Required Fields	$name, address, geo\_lat, geo\_long, postal\_code, phone\_number$

1.8 $StudioImages$

Name of Model	$StudioImages$
Description	Images of the given studio
Required Fields	$studio$ ($foreign\ key$), $image$
Notes	$studio$ is a foreign key to the studio object itself. $image$ is an $ImageField$ which stores the url to the image.

1.9 $StudioAmenities$

Name of Model	$StudioAmenities$
Description	The amenities of the given studio
Required Fields	$studio$ ($foreign\ key$), $type, quantity$
Notes	$studio$ is a foreign key to the studio object itself. $type$ is the type of amenity (pool, track etc). $quantity$ is the amount of that amenity.

1.10 $Classes$

Name of Model	$Classes$
Description	The class for a studio
Required Fields	$studio\ (foreign\ key), name, description, coach, capacity, day, start\_time, start\_date\ , end\_date, length$
Notes	$studio$ is a foreign key to the studio object itself. day is the day of the week the class takes place (Mon, Tues, etc). $start\_time$ is the time that the class starts (12pm, 1pm etc). $start\_date$ is the first date that this class can possibly start. end_date is the last possible date that this class can start.

1.11 $ClassKeywords$

Name of Model	$ClassKeywords$
Description	Class and its keyword
Required Fields	$class\_id\ (foreign\ key), keyword$
Notes	$class\_id$ is a foreign key to the $Classes$ object.

1.12 $ClassInstance$

Name of Model	$ClassInstance$
Description	An instance of $Classes$ which happens on a specific date
Required Fields	$class\_id\ (foreign\ key), date, num\_signed\_up, alerts$
Notes	$class\_id$ is a foreign key to the $Classes$ object. $date$ is the exact specific date that the class happens. $alerts$ is a string which holds information about if a class time or length has changed.

2. Admin-Only Abilities

2.1 Create/Edit/Delete a $Studio$

Only the admin can create and edit and delete a $studio$. To add studio amenities, the admin must add it to the $StudioAmenities$ model. To add studio images, the admin must add it to the $StudioImages$ model.

2.2 Create/Edit/Cancel a Class

Only the admin can create and edit and cancel a class. To do this, go to admin panel, and checkout the $classes$ section. To add, add in the $Classes$ model. $ClassInstances$ will be made for each time the class will occur. To edit, you can edit either the $Classes$ model or $ClassInstance$ model depending on if you want to edit all occurences or just 1. To cancel all occurences, delete it from $Classes$ model. To cancel a single occurrence, delete it from $ClassInstance$ model. You can also add/delete $Class\ keywords$.

2.3 Create/Edit/Delete a gym subscription plan

Only the admin can create and edit and delete a Subscription plan. This is done through the admin panel. Go to the $Subscriptions$ model, and add, edit or delete a plan.

3. Endpoints

3.1 Register user

URL	http://127.0.0.1:8000/accounts/register/
Method	POST
Payloads	$username$, $password$, $password2$, $first\_name$, $last\_name$, $email$, $phone\_number$ (which all are required and none blank) $avatar$ (which is optional)
Description	Register a user when user provides the payloads
Payload Requirements	$password$ has to be at least 8 characters long and contain at least one small letter, one capital letter, one number, and one special character[!@#$%^&*]. $password$ and $password2$ should match. $phone\_number$ should be 11 characters long. The response is the information of the user
Django Class	$class\ MyUserCreateView(CreateAPIView)$

3.2 Generate token for user

URL	http://127.0.0.1:8000/api/token/
Method	POST
Payloads	$username, password$
Description	Generate token for latter requests (bearer token) , then set the environment to the exported environment (project). After the token is generated, copy and paste the “access” token in “authToken” variable of project environment. The latter requests that require auth token, will use this token without needing to copy and paste everytime.

3.3 User Login

URL	http://127.0.0.1:8000/accounts/login/
Method	GET
Headers	Authorization: Bearer (token)
Description	Login a user with the generated token and show a response with the information of the user.
Django Class	$class\ MyUserLoginView(RetrieveAPIView)$, with get_object method to get user

3.4 User Logout

URL	http://127.0.0.1:8000/accounts/logout/
Method	GET
Headers	Authorization: Bearer (token)
Description	Simply logout user and show a response of logged out successfully.
Django Class	$MyUserLogoutView(APIView)$ with get method to logout

3.5 Edit user details

URL	http://127.0.0.1:8000/accounts/edit/<str:username>/
Method	PUT/PATCH
Payloads	$Username, password, password2, first\_name, last\_name, email, phone\_number$ $avatar$
Headers	Authorization: Bearer (token)
Description	Provide the username the user wants to edit in queries and the payloads for updating. If the username doesn't exist, 404 not found response is returned. If the username exists but doesn't match the provided token, 403 forbidden with message "You do not have permission to perform this action" is returned. If the username exists and matches the token, information is updated with status 200 OK.
Payload Requirements	In PUT all of the payloads except for $avatar$ have to be provided but in PATCH, partial updates are made
Django Class	$MyUserEditView(UpdateAPIView)$, with get_object to get the user

3.6 Subscription Options

URL	http://127.0.0.1:8000/subscriptions/all/
Method	GET
Description	Return a response of all available subscriptions with their details(id, interval (week or year or month) and price)
Django Class	$SubscriptionsView(ListAPIView)$, with get_query_set to get a list of all available subscriptions

3.7 Subscribe

URL	http://127.0.0.1:8000/subscriptions/subscribe/
Method	GET
Query Param	$subscription\_id$
Headers	Authorization: Bearer (token)
Description	Subscribe or update subscription of the user to subscription with id subsciption_id and return a response of the information of the user. Inside the information of user, subscription and due_date, are updated.
Payload Requirements	$subscription\_id$ is the id of the subscription user wants to subscribe to
Django Class	$AddSubscription(RetrieveAPIView)$, with get_object to get the user with $username = user\_name$ and update subscription

3.8 Unsubscribe

URL	http://127.0.0.1:8000/subscriptions/unsubscribe/
Method	GET
Headers	Authorization: Bearer (token)
Description	Unsubscribe user and return a response of the information of the user. Inside the information of user, subscription and due_date, will be set to None.
Django Class	$DeleteSubscription(RetrieveAPIView)$, with get_object to get the user with $username = user\_name$ and delete subscription

3.9 Update card

URL	http://127.0.0.1:8000/accounts/update_card/
Method	POST
Payloads	$number, cvv, expiration, postal\_code$ (which are all required and none blank)
Headers	Authorization: Bearer (token)
Description	Add or update the provided card of the user and return a response with the cards's information. When you login the user with token, updated card can be seen. If the user wants to update the same card(provides same card number but different queries of cvv, postal_code, or expiration), then this card will be updated in database. But, if the user wants to update a card to another new card, the old card will be marked as unused, and the new card will be added. A card can be used by one user at the time. If a user wants to update to a card that's already being used by another user, a 403 Forbidden with message "Card already in use by another user", is shown.
Payload Requirements	Card number has to be 16 characters long and all digits. cvv has to be 3 or 4 characters long and all digits. expiration has to have the format YYYY-MM. Postal code has to have the format LDLDLD or LDL DLD (with space in between)
Django Class	$UpdateCard(RetrieveAPIView, CreateAPIView)$, with post method to create the card in database and assign to user

3.10 View Future Payments

URL	http://127.0.0.1:8000/accounts/payment/future/
Method	GET
Headers	Authorization: Bearer (token)
Description	Show the future payments of user with the current subscription plan and card. If the subscription is per month or week, the next 12 payments will be shown. If it's per year, the next 3 payments are displayed. If the user updates subscription or card, future payments will also be updated.
Django Class	$PaymentFutureView(ListAPIView)$, with get_query_set to create and get the list of all future payments and get method to delete all the created payments(since they haven't been made yet)

3.11 View Payment History

URL	http://127.0.0.1:8000/accounts/payment/history/
Method	GET
Headers	Authorization: Bearer (token)
Description	Show the previous payments of user with the former subscription plan and card. The first payment which is made right after subscription is shown in history.
Django Class	$PaymentHistoryView(ListAPIView),$ with get_query_set to get the list of previous payments made by a specific user with former card and subscription. $./tfc/manage.py\ payment$ is a command that has to be manually run everyday to make payments that are due that day. So this command is run first and then the user is able to see the payment history. **Note**: that in a real-world application, we would be adding a crontab task which automates running the command every day 12:00am sharp.

3.12 Enroll user in class

URL	http://127.0.0.1:8000/classes/enroll/
Method	POST
Payloads	$class\_instance\_id$ (required) $date$ (optional, if not included, user will be enrolled in all future instances of the class)
Headers	Authorization: Bearer (token)
Description	Enrolls the user in a singular or recurring class (of type ClassInstance) depending on date value in payload, and returns all classes that the user has enrolled in (in chronological order) or valid status messages depending on if the user has a valid subscription or not, or whether the user is already enrolled in the desired class.
Payload Requirements	Date must be a valid string in the format YYYY-MM-DD if included
Django Class	$enroll\_in\_one\_class$: Enrolls the user in one specific class and creates the appropriate object in the $Enrollment$ model $enroll\_in\_all\_future\_classes$: Enrolls the user in all occurrences of the class and creates the appropriate objects in the Enrollment model

3.13 Drop user from class

URL	http://127.0.0.1:8000/classes/drop/
Method	GET
Payloads	$class\_instance\_id$ (required) $date$ (optional, if not included, user will be enrolled in all future instances of the class)
Headers	Authorization: Bearer (token)
Description	Drops the user from a singular or recurring class (of type ClassInstance) depending on date value in payload, and returns all classes that the user has dropped (in chronological order) or valid status messages depending on whether the user is already enrolled in the desired class.

Payload Requirements	Date must be a valid string in the format YYYY-MM-DD if included
Django Class	*drop_one_class*: Drops the user from one specific class and deletes the appropriate object in the *Enrollment* model, updating the ClassInstance.num_signed_up as well *drop_all_future_classes*: Drops the user from all future occurrences of the class and deletes the appropriate objects in the Enrollment model, updating the *ClassInstance.num*_signed_up as well

3.14 View User Schedule

URL	http://127.0.0.1:8000/classes/schedule/
Method	GET
Headers	Authorization: Bearer (token)
Description	Shows the list of all future classes that the user is enrolled in, along with the name of the class and any potential alerts relating to the class (e.g. if the class has been canceled, etc.)

3.15 View User History

URL	http://127.0.0.1:8000/classes/history/
Method	GET
Headers	Authorization: Bearer (token)
Description	Shows the list of all past classes that the user is enrolled in, along with the name of the class and any potential alerts relating to the class (e.g. if the class has been canceled, etc.)

3.16 Search Classes

URL	http://127.0.0.1:8000/classes/search/
Method	GET
Payloads	*studio_id* (required, the id/pk of the studio) *class_names* (not required, all potential class names the user wants to filter on) *coach* (not required, all potential coaches the user wants to filter on) *date* (not required, all potential dates the user wants to filter on) *time_range* (not required, all potential time ranges the user wants to filter on)
Headers	Authorization: Bearer (token)
Description	Searches for classes and filters them based on the values provided. Returns a list of all matched classes, or valid messages depending on if any classes were found.
Payload Requirements	studio_id must be a valid studio id/pk in the Studio model, else a 400 BAD REQUEST response is returned. class_names, coach, and time_range must be lists separated by a comma and whitespace (", "). date must be a string in the format YYYY-MM-DD. time_range must be a list of strings in the form HH:MM:SS.

3.17 View classes of a studio

URL	http://127.0.0.1:8000/classes/studio/<int:studio_id>/details/
Method	GET
Payloads	studio_id *(required, the id/pk of the studio; provided in URL)*
Headers	Authorization: Bearer (token)
Description	Searches for classes occurring in the studio with corresponding provided $studio\_id$. Returns a list of all matched classes, or valid messages depending on if no classes were found.
Payload Requirements	$studio\_id$ must be a valid studio id/pk in the Studio model, else a 400 BAD REQUEST response is returned.

3.18 Studio Details

URL	http://127.0.0.1:8000/studios/<int:studio_id>/details/
Method	GET
Description	Get the details of studio
Django Class	$StudioDetails$

3.19 Studio Search

URL	http://127.0.0.1:8000/studios/search/
Method	GET
Params	$location$ (required) this is used to return studios nearest by location $studioname$ (optional) this is used to filter studios with this name $amenities$ (optional) this is used to filter studios with these amenities. Separate amenities by comma $classes$ (optional) this is used to filter studios which have these classes. Separate class names by comma $coach$ (optional) this is to filter studios which have classes by this coach
Headers	Authorization: Bearer (token)
Description	Given the params, search for studios and return list of them
Django Class	$StudioSearch$