Project Backend Documentation

Will Gotlib, Shalev Lifshitz

NOTE: The database is uploaded on Markus right now. This is bad practice, but given that it's not very big and it will be a big help making the Postman tests run easier, it seemed like an OK thing to do. If not, just ignore or delete it.

MODELS

ACCOUNTS

- TFCUser
  - Custom extension of the standard Auth user.
  - Contains phone number (not required), credit card number (not required, but you can't get a subscription without it), active subscription (starts off blank), and an avatar image field.
- Payment
  - When a user makes a transaction to buy a subscription, one of these is created to remember it.
  - Contains the associated user (as a foreign key), the type of subscription (also a foreign key), date time of the payment, amount of money paid, and the credit card number used.
  - All fields required except for subscription – if a subscription is deleted from the database, the payment will stick around and the subscription field will be set to null.
- Subscription
  - A type of gym subscription that a user can subscribe to.
  - Contains name of subscription plan, price, and how often that price is charged (annually, monthly, weekly, daily).
  - All fields required.

STUDIOS

This is significantly more complicated. We used a system based around indefinite repetition, which we were told was way beyond expectation. As such we have a lot of models here, since we couldn't just do explicit session objects.

- Studio
  - A studio in which classes take place.
  - Contains name, latitude and longitude of the studio, and optionally the address, postal code (these two are redundant to latitude and longitude from a backend perspective but are more user-friendly) and phone number.
- StudioImage
  - An image used by a studio.
  - Needs to be its own model just because there should be able to be multiple images to a studio; we use foreign key for this.
- Amenity
  - An amenity (like a weight machine, or a treadmill) that a gym studio has.
  - Contains associated studio, type of amenity, and quantity of how many that studio owns.

- Class
  - A class run inside a studio. For example, “HIIT Class”
  - Contains associated studio, name of class, description of class, the name of the coach who runs class, keywords relating to this class, and its capacity (a positive integer)
- ClassTime
  - A repeating instance of a class. For example, the HIIT Class may run every week on Wednesdays 5-7pm, but it might ALSO run on Sundays 2-4pm. They’re still the same class, hence this abstraction.
  - Contains the associated class, which day of the week and time of day this ClassTime takes place on/at, the duration of the ClassTime in hours (e.g. 5-7pm would be 2) and the date of the first session.
- RecurringEnroll
  - Indicates a user’s enrollment in a ClassTime on a recurring basis (they will go to every session of this ClassTime).
  - Contains associated ClassTime, associated user, and the starting and ending dates of their recurring enrolment. End-date is left blank by default to leave enrolment unbounded.
- RecurringEnrollSpecificSessionDrop
  - For when a user with a recurring enrolment in a ClassTime wants to drop a specific session.
  - Contains associated RecurringEnroll object, and for efficiency, the session number which the user wants to drop out of. For example, if the ClassTime’s start date is November 1st and we dropped out of session 2, that would drop us out of the session on November 8th.
- IndividualEnroll
  - Indicates that a user’s enrolment in a SPECIFIC session of a ClassTime, NOT on a recurring basis.
  - Contains associated ClassTime, user and a session number like in the previous model.
- SpecificSessionCancellation
  - Indicates that a specific session of a recurring ClassTime is being cancelled outright; nobody can come. This should only be created by an admin.
  - Contains associated ClassTime and session number.
  - Will automatically create a RecurringEnrollSpecificSessionDrop for everybody enrolled recurring-ly in this course.

API ENDPOINTS

- `/accounts/register/`
  - Method: POST
    - Send essential user fields: username, email, password, and password2 for confirmation.
- `/accounts/login/`
  - Method: POST
    - Send your username and password – both CharFields. You will be

        logged in if they are authenticated.
- `/accounts/logout/`
  - Method: GET
    - Logs the user out if they're logged in.
- `/accounts/<int:user_id>/profile/`
  - Method: PATCH
    - Send in the form-data payload the user information you want to change. Validation applies.
  - Method: GET
    - Returns the user's information
- `/accounts/payments/history`
  - Method: GET
    - Returns a history of the currently logged-in user's payments.
- `/accounts/payments/future`
  - Method: GET
    - Shows the speculative payments that the currently logged-in user will have to make in the future, based on their current subscription.
- `/accounts/subscriptions/all/`
  - Method: GET
    - Returns a list of all the available subscriptions in the system.
- `/accounts/subscriptions/<int:sub_id>/subscribe/`
  - Method: POST
    - Given the subscription ID in the URL, the currently-authenticated user will be "charged" (a payment will be created tied to them) and their active subscription will be changed to the right one.
    - Note that if the user doesn't have a credit card (add it with <user_id>/profile) this won't be allowed as they can't pay.
    - This is a POST but no body data is required. Just doesn't seem like it should be a GET.
    - Will "change" the subscription if the user already has one.
- `/accounts/subscriptions/unsubscribe/`
  - Method: GET
    - Unsubscribe for whatever subscription the current user is subscribed to right now.
- `/studios/list/?lat=X&lng=Y`
  - Method: GET
    - Lists out studios, ordered by distance to the user's location which they supply as query parameters as above.
    - Required params are lat and lng. They both have to be between -90 and 90.
    - You can also so filtering here. If you attach `&name=...`, `&coach=...`, `&amenities=am1,am2,...`, or `&class_names=c1,c2,...` to the URL it will filter the studios in those respective ways.
- `/studios/<int:studio_id>/`
  - Method: GET

        - Returns the specified studio's basic information.
- `/studios/<int:studio_id>/schedule/`
    - Method: GET
        - Returns a paginated list of a studio's classes and all their ClassTimes. This is kind of difficult to order because of the hierarchical structure here.
- `/studios/class_time/<int:class_time_id>/enroll/`
    - Method: POST
        - Attempts to enroll the user in this ClassTime, as specified by the parameter in the URL.
        - The user can attach a session_num (integer) in the POST body. This will indicate that they want to enrol in an individual session. Leaving this field blank indicates that the user wants to enrol recurring.
- `/studios/class_time/<int:class_time_id>/drop/`
    - Method: POST
        - Attempts to drop the user from their enrollment in this ClassTime (if they have one), as specified by the parameter in the URL.
        - The user can attach a session_num (integer) in the POST body. This will indicate that they want to drop an individual session from their recurring enrollment. Leaving this field blank indicates that the user wants to drop all sessions from now on.
- `/studios/class_time/num_enrolled/<int:class_time_id>/<int:session_num>/`
    - Method: GET
        - Tells us how many people are in session <session_num> of class_time with id <class_time_id>.
- `/studios/schedule/`
    - Method: GET
        - Returns a list containing the currently-logged-in user's schedule of future classes. The hierarchical nature of this also makes it difficult to sort chronologically or paginate it effectively. It might be a little confusing to look through – it returns a list of classes the user has a relationship to; inside the appropriate ClassTime(s), a user's enrolment status (recurring or individual) shows up.

NOTE ON OUR DESIGN CHOICES

Attaching this extra section because apparently we did a lot more than we were expected to, and were told that we would get a bonus. The main thing is the indefinite class repetition structure, and there are a lot of specifics that splinter off of that which made our implementation way harder than it might have been otherwise. But Kianoosh has been positive about it. There are definitely limitations to the way our program works, but there are also possibilities opened by our use of patterns rather than explicit session objects which would be basically impossible if we had taken that path.

On lack of pagination in some places:
1) For studio schedule, we treat 'ordering classes by their start time' as a frontend issue. This is because we do not actually construct sessions in the backend, but we send the general pattern + the exceptions to the front end. So we can't 'order by session start time' in the backend. (We don't have session objects!)

2) Similarly for User History, we cannot order chronologically in the backend. We send the user's general patterns + exceptions and the frontend will construct the history.

In both these cases we had to remove pagination. Because we are doing infinite recurring sessions rather than explicit sessions for every classtime per week, the frontend needs to have everything loaded in order to create the sessions and order them by time. On the other hand, because of this format, there isn't a lot of data being sent to the frontend (not sending an element per session, just patterns and exceptions).

Some more loose notes:

- We consider a session starting as having past, for simplicity. So you cannot enrol or drop a session which just started but hasn't ended.
- Capacity for a classtime can be infinite (if capacity property is null)
- We can have recurring or individual enrolments, but class_times are recurring.
- "Re-enrolling" in a dropped individual instance when you have a recurring enrollment deletes the relevant RecurringIndividualSpecificSessionDrop object.
- Because of our pattern system, we can't change the class info of a specific session
- Class schedule for studio is constructed at frontend, which isn't the best. It's a tradeoff.
- Un-cancelling a cancelled session will retain all the people who were signed up for it.