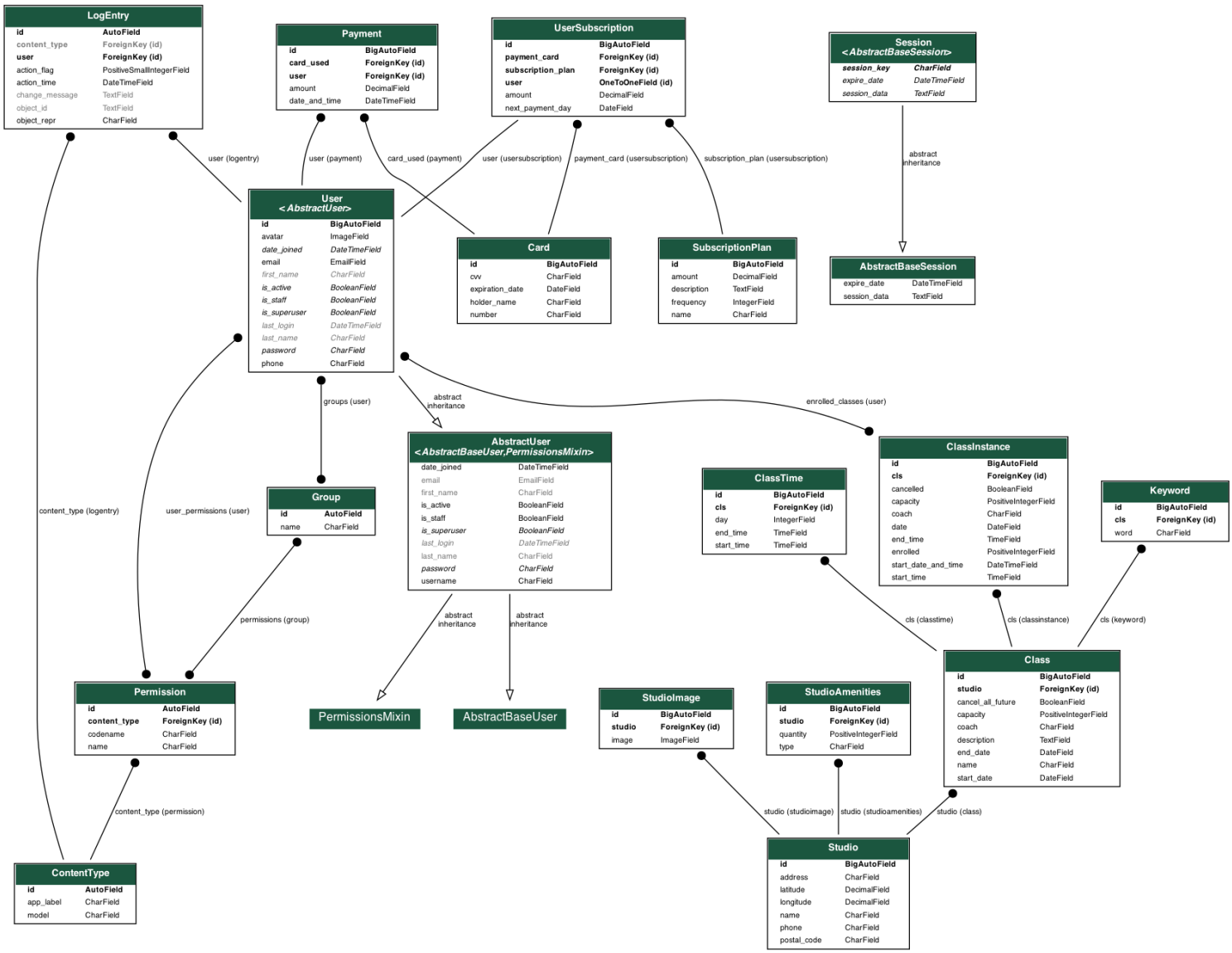
CSC309 Project Backend Documentation

Anis Singh

November 17, 2022

Model Description

Entity Relationship Diagram:



Brief Summary of Important Models:

To model users, I created a custom `User` model that conforms to the user stories regarding the user's profile and set it to the user model that Django will use. Both the email and phone number are required and must be unique, and the first and last name fields are optional.

To model studios, I created a `Studio` model that contains a name, an address, a geolocation (stored as a latitude `DecimalField` and a longitude `DecimalField`), a postal code, and a phone number. The set of images are stored as instances of `StudioImage` models, each of which has a foreignkey to the `Studio` model that it is for. The amenities are stored as `StudioAmenities` models, each of which has a foreignkey to the `Studio` model it is for.

Classes were modelled on an individual and generic (abstract) basis. Abstract notions of a class (e.g., Core Training every Monday from 8pm-9pm) were modelled using a `Class` model. This model has a foreignkey to the `Studio` model representing the studio it takes place in, as well as various other attributes that a class should have (as shown in the ERD). The `cancel_all_future` attribute is a `BooleanField` that, upon being set to `true` in the admin panel, called a method that cancelled all future instances of this class. Individual class instances were modelled by `ClassInstance` models. These cannot be created by the admin and are auto-generated / auto-updated anytime the `Class` model specified in their foreignkey is created or undergoes a time change or cancellation. Note that neither `Class` nor `ClassInstance` models can be deleted by the admin.

Subscription plans were modelled by the `SubscriptionPlan` model. To store user subscriptions, I created a one-to-one relationship between the `UserSubscription` model and the `User` model. A `UserSubscription` model for a specific user only exists when a user has an active subscription. Payments are recorded in the database by the `Payment` model, which we generate an instance of every time a user makes a payment.

In the above ERD, the models that were generated by `simplejwt` and Django itself are `LogEntry`, `Permission`, `ContentType`, `Group`, `AbstractUser`, `PermissionsMixin`, `AbstractBaseUser`, and `Session`.

API Documentation

Below is documentation for the REST APIs that were implemented. Note that optional field lists contain the fields that do not need to be included in the payload. All endpoints that require authentication require an access token to be added to the headers with key `Authorization` and value `Bearer {your token here}`.

Register User

- **Endpoint:** `/accounts/register/`
- **Methods:** `POST`
- **Auth Required:** `false`
- **Payload:** `email`, `phone`, `password`, `avatar`, `first_name`, `last_name`
- **Optional Fields:** `avatar`, `first_name`, `last_name`
- **Description:** Creates a user from the given information in the payload and stores it in the database. If `avatar` is not included in the payload, a default profile picture is used.

Generate User Login Token

- **Endpoint:** `/accounts/api/token/`
- **Methods:** `POST`
- **Auth Required:** `false`
- **Payload:** `email`, `password`
- **Description:** Generates and returns a refresh token and an access token upon receiving valid login information. Its intended use is to use the access token to provide a way to authenticate users through token authentication.

Edit User Profile

- **Endpoint:** `/accounts/edit/`
- **Methods:** `PUT`, `PATCH`
- **Auth Required:** `true`
- **Payload:** `email`, `phone`, `password`, `avatar`, `first_name`, `last_name`
- **Optional Fields:** `avatar`, `first_name`, `last_name`
- **Description:** Updates the user's profile based on the data provided in the payload. Detects which user needs to be updated through the auth token provided in the header. To prevent an optional field from being updated, do not include it in the payload. To prevent a required field from being updated, a `PATCH` request must be sent, and the field must be excluded from the payload as well.

View User Information

- **Endpoint:** `/accounts/view/`
- **Methods:** `POST`
- **Auth Required:** `true`
- **Payload:** `email`
- **Description:** Returns information about the user given by `email`.

Get Studio Information

- **Endpoint:** `/studios/<studio_id>/info/`
- **Methods:** `GET`
- **Auth Required:** `false`
- **Description:** Returns the information about the studio with id `<studio_id>`. This includes its set of images and its amenities.

Get Closest Studios

- **Endpoint:** `/studios/closest/`
- **Methods:** `GET`
- **Auth Required:** `false`
- **Query Parameters:** `lat`, `long`, `page`
- **Description:** Returns a paginated list of studios that are ordered by closest geographical distance from the latitude and longitude specified in `lat` and `long`, respectively. Set `page = <num>` to view page `<num>`. If `page` is unspecified, returns the first page of data. If `lat` and `long` are invalid decimal numbers, returns an empty list of studios. Multiple query parameters of the same type can be specified, but only the last one will be processed.

Enroll in One Class

- **Endpoint:** `/studios/classes/<class_id>/enroll/one/`
- **Methods:** `POST`
- **Auth Required:** `true`
- **Payload:** `email`, `date`
- **Description:** Enrolls the user given by `email` in class `<class_id>` on date `date`, subject to the constraints outlined in the project handout. The format of `date` is `YYYY-MM-DD`.

Enroll in All Classes

- **Endpoint:** `/studios/classes/<class_id>/enroll/`
- **Methods:** `POST`
- **Auth Required:** `true`
- **Payload:** `email`
- **Description:** Enrolls the user given by `email` in all classes given by `<class_id>`, subject to the constraints outlined in the project handout.

Drop One Class

- **Endpoint:** `/studios/classes/<class_id>/drop/one/`
- **Methods:** `POST`
- **Auth Required:** `true`
- **Payload:** `email`, `date`

- **Description:** Drops the user given by `email` from class `<class_id>` on date `date`, subject to the constraints outlined in the project handout. The format of `date` is `YYYY-MM-DD`. Note that dropping from a future class that is cancelled is allowed.

Drop All Classes

- **Endpoint:** `/studios/classes/<class_id>/drop/`
- **Methods:** `POST`
- **Auth Required:** `true`
- **Payload:** `email`
- **Description:** Drops the user given by `email` from all classes given by `<class_id>`, subject to the constraints outlined in the project handout. Note that dropping from future classes that are cancelled is allowed.

Get Studio Class Schedule

- **Endpoint:** `/studios/<studio_id>/schedule/`
- **Methods:** `GET`
- **Auth Required:** `false`
- **Query Parameters:** `page`
- **Description:** Returns a paginated list of future and active class instances that take place in studio `<studio_id>`. Set `page = <num>` to view page `<num>`. If `page` is unspecified, returns the first page of data. Multiple query parameters of the same type can be specified, but only the last one will be processed.

Get User Class Schedule

- **Endpoint:** `/studios/classes/user/schedule/<user_id>/`
- **Methods:** `GET`
- **Auth Required:** `true`
- **Query Parameters:** `page`
- **Description:** Returns a paginated list of future class instances that the user is enrolled in, including cancelled class instances. The list is ordered chronologically. Set `page = <num>` to view page `<num>`. If `page` is unspecified, returns the first page of data. Multiple query parameters of the same type can be specified, but only the last one will be processed.

Get User Class History

- **Endpoint:** `/studios/classes/user/history/<user_id>/`
- **Methods:** `GET`
- **Auth Required:** `true`
- **Query Parameters:** `page`
- **Description:** Returns a paginated list of past class instances that the user was enrolled in, including class instances that were cancelled. The list is ordered chronologically. Set `page = <num>` to view page `<num>`. If `page` is unspecified, returns the first page of data. Multiple query parameters of the same type can be specified, but only the last one will be processed.

Search/Filter Studios

- **Endpoint:** `/studios/search/`
- **Methods:** `GET`
- **Auth Required:** `false`
- **Query Parameters:** `name`, `amenity`, `class-name`, `coach`, `page`
- **Description:** Filter through all studios and return a paginated list of those matching all the arguments given in the query parameters. Multiple query parameters of the same type can be specified, and this results in an or condition filter. For example, if the query string is `name=Studio&name=LA&coach=Ron`, we return a list of all studios whose name is "Studio" or "LA" and that have classes coached by "Ron". The performed search is **not** case-sensitive. Set `page = <num>` to view page `<num>`. If `page` is unspecified, returns the first page of data.

Search/Filter Through a Studio's Class Schedule

- **Endpoint:** `/studios/<studio_id>/classes/search/`
- **Methods:** `GET`
- **Auth Required:** `false`
- **Query Parameters:** `class-name`, `coach`, `date`, `start-time`, `end-time`, `page`
- **Description:** Filter through all class instances of studio `<studio_id>` and return a paginated list those matching all the arguments given in the query parameters. Multiple query parameters of the same type can be specified, and this results in an or condition filter. For example, if the query string is `coach=Ron&coach=John&class-name=Core%20Training`, we return a list of all class instances in `<studio_id>` whose coach is "Ron" or "John" and whose name is "Core". The performed search is **not** case-sensitive. The format of `date` is `YYYY-MM-DD` and the format of both `start-time` and `end-time` is `hh-mm-ss`. Set `page = <num>` to view page `<num>`. If `page` is unspecified, returns the first page of data.

Subscribe User

- **Endpoint:** `/subscriptions/subscribe/<plan_id>/`
- **Methods:** `POST`
- **Auth Required:** `true`
- **Payload:** `email`, `card_number`, `cardholder_name`, `expiration_date`, `cvv`
- **Description:** Subscribes the user given by `email` to the subscription plan given by `plan_id`. Does **not** update existing subscriptions. Updating subscriptions for users with active subscriptions (subscriptions in which the last billing period is not over[1]) should be done through the subscription update API instead. `expiration-date` is of the form `MM-YYYY`.

Update Payment Card

- **Endpoint:** `/subscriptions/update/card/`
- **Methods:** `POST`
- **Auth Required:** `true`

[1] Notice this includes cancelled subscriptions in which we are still within the user's billing period.

- **Payload:** email, card_number, cardholder_name, expiration_date, cvv
- **Description:** Updates the payment card of the user given by email if the user already has an active subscription. expiration-date is of the form MM-YYYY.

Get User Payment History

- **Endpoint:** /subscriptions/user/<user_id>/payments/history/
- **Methods:** GET
- **Auth Required:** true
- **Query Parameters:** page
- **Description:** Returns a paginated list of the payment history of the user given by user_id. Set page = <num> to view page <num>. If page is unspecified, returns the first page of data. Multiple query parameters of the same type can be specified, but only the last one will be processed.

Get User Future Payments

- **Endpoint:** /subscriptions/user/<user_id>/payments/future/
- **Methods:** GET
- **Auth Required:** true
- **Description:** Returns information about a user's next scheduled payment. The user is given by user_id. If the user does not have a subscription or their subscription has been cancelled, returns no information.

Update User Subscription Plan

- **Endpoint:** /subscriptions/update/plan/<plan_id>/
- **Methods:** POST
- **Auth Required:** true
- **Payload:** email
- **Description:** Updates a user's subscription plan to the subscription plan specified by <plan_id¿ if the user already has a subscription. The user is given by user_id.

Cancel User Subscription

- **Endpoint:** /subscriptions/cancel/
- **Methods:** POST
- **Auth Required:** true
- **Payload:** email
- **Description:** Cancels the user's subscription plan if the user already has a non-cancelled subscription. The user is given by user_id.

Get All Subscription Plans

- **Endpoint:** /subscriptions/plans/all/
- **Methods:** GET
- **Auth Required:** false

- **Query Parameters:** `page`
- **Description:** Returns a paginated list of all subscription plans that are offered. Set `page = <num>` to view page `<num>`. If `page` is unspecified, returns the first page of data. Multiple query parameters of the same type can be specified, but only the last one will be processed.