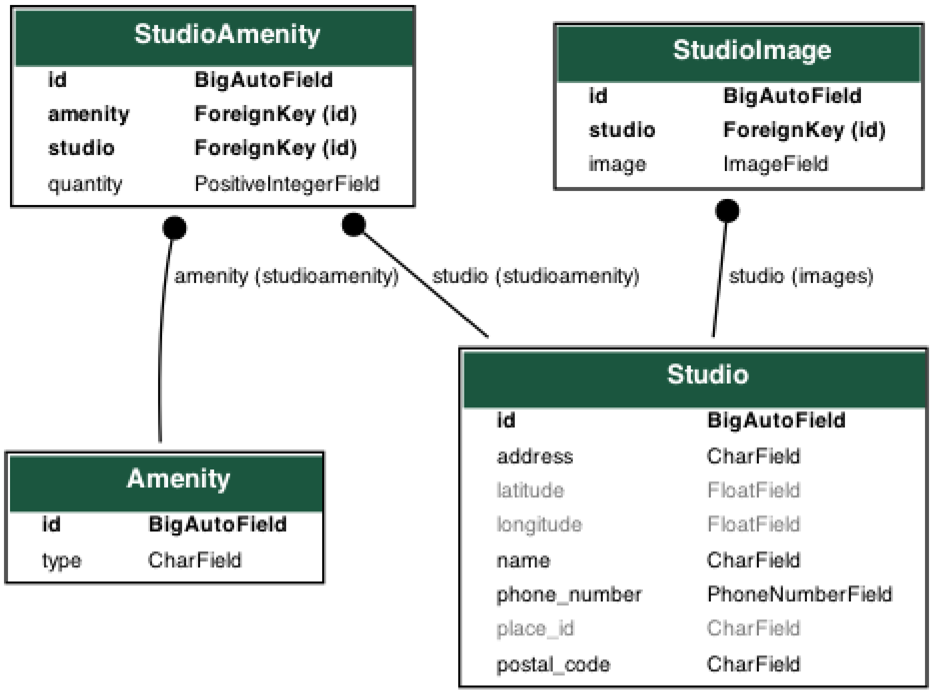
Toronto Fitness Club Backend Documentation

Models

Studios:

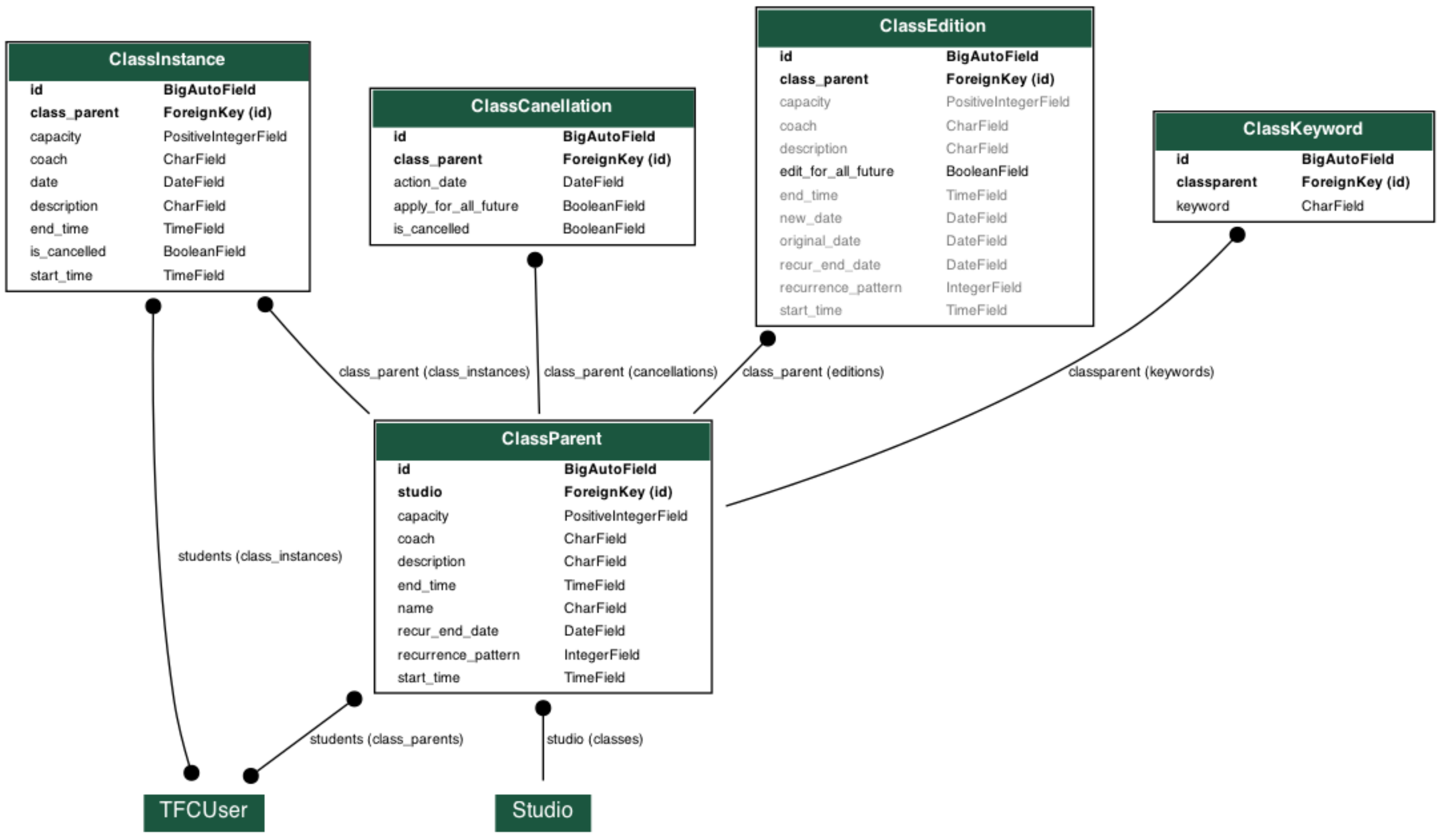


Studios’ latitude, longitude, and place id are automatically generated after the admin creates the studios with good addresses. And the location information will be used when calculating geological proximity from a specific location and getting direction to the studios.

Apart from some basic features in the studio table as shown in above figure, each studio has multiple images, which is represented by a one to many relation.

Besides, the relationship between studios and amenities is many to many, which is represented through an intermediate table “StudioAmenity”, and one extra column “quantity” is specified in the table to accommodate the fact that studios can have different numbers of amenities.

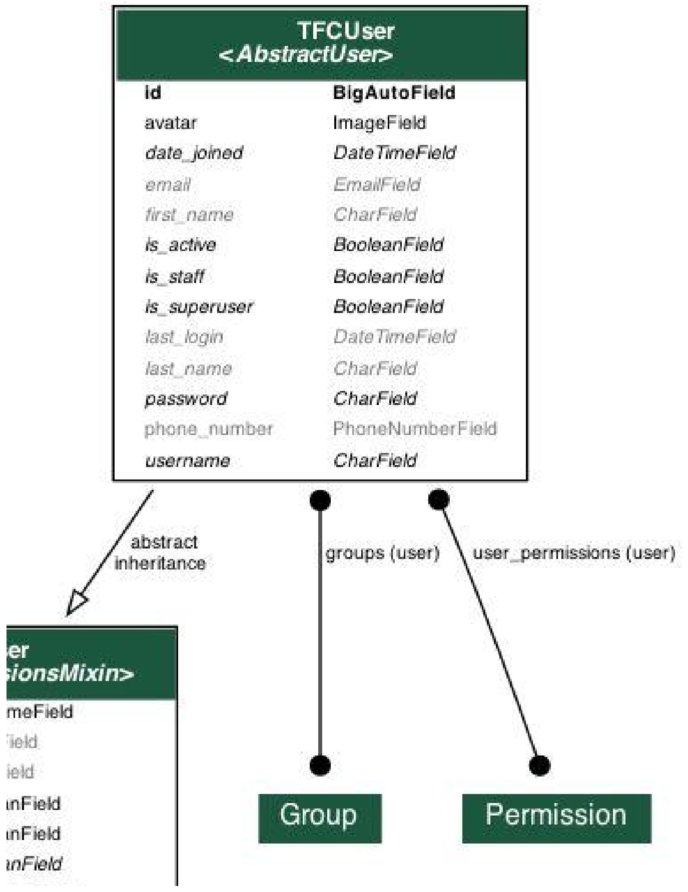
Classes:



The recurrence of the classes is mainly achieved by the relation between ClassInstance and ClassParent. When the admin creates a class parent under a studio, or when the admin edits the classes’ recursion end date, a bunch of class instances will be created automatically.

The admin’s operations (edit or cancel either one class or all future instances) are through ClassEdition and ClassCancellation. These models are displayed as inline models in the ClassParent admin panel, and are read-only when creating a new class parent. When editing the classes, the admin can change date for one instance, or change all classes' recurrence pattern, change classes' recursion end date, or just edit general information such as capacity, coach for classes.

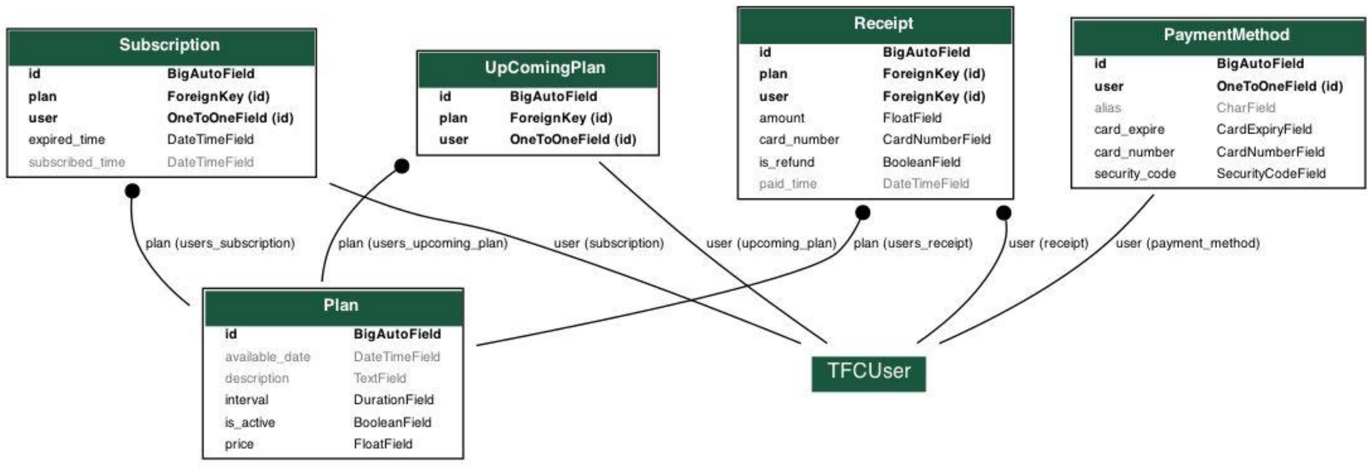
The user enrollment part is achieved by both ClassInstance and ClassParent, each through an intermediate table automatically generated by Django many to many fields. When users want to enroll for all future classes, not only the ClassInstance is used, the ClassParent should also be used, since we need keep track of previous users who wish to enroll all future classes and make sure they are automatically enrolled to new classes when the admin extend the recursion end date and generate a bunch of new class instances.



TFCUser:

TFCUser is extended from Django's AbstractUser. It has two additional fields: avatar and phone_number.

Subscription-related Models:



Each user has only one subscription, upcoming plan, payment method and multiple receipts.

There is a background thread running in the server periodically to check whether the user's subscription has expired. When the thread finds one, it will automatically subscribe the user to his/her upcoming plan if there is a payment method and upcoming plan set in advance.

When a user cancels the subscription before it expires, there will be a refund calculated by the passed time of this subscription period.

The way users update the subscription is by canceling one first and subscribing to a new one (which could be integrated by front end, as backend we think it's better to keep lower coupling).

Whenever there is a purchase or refund, there will be a receipt which is auto-generated and unchangeable. By looking up the receipts of a user and his/her upcoming plan, we can view the history and future of the user's subscriptions(can’t be done by a single restful url, but could be integrated by front end).

A plan is undeletable because it’s played as foreign key for receipts. However, the Administrator could set the is_active field of a plan to false to prohibit users from subscribing.