Design of models

Accounts:
We are using JWT to manage authentication. Thus the login is handled automatically and the user is returned a token to access the server.

We are using the default User module to store the username, password, email,

- **Avatar**:
  Avatar essentially acts as the custom user model
  User: Foriegn Key to the Default User Model
  Avatar : Stores the Image of the User
- **Card**:
  User: Foriegn Key to the Default User Model
  Name: Name of the card
  Number: Card Number

Studios:

For the studios we have three models to incorporate the studio details, the details of the amenities, and for the set of images

- **Studios** :
  Fields: Name, Address, Longitude, Latitude, Postal Code, Phone Number, and User Distance.
- **Studio Images:**

  Each studio stores a set of images, we have included this in our design by creating another model which stores an image along with the studio as the Foreign Key.

  Fields:
  Studio - Foreign Key to Studios
  Image - Image corresponding to the studio

- **Studio Amenities:**
  Stores the studio amenities, which can be edited as the website admin
  Studio - Foreign Key to Studios
  Type - The type of amenity
  Quantity - The quantity of the amenity

Classes:
In order to implement the class methods, we are using three models, Class, ClassInstance, UserEnrolled. The classInstance model helps us implement recurring classes. The UsersEnrolled model helps us store the enrolled students in a class.

- **Class:**
    - studio = Foreign Key to Studios
    - name = Name of the Class
    - description = Class Description
    - coach = Name of the coach for the class
    - capacity = Maximum number of students allowed
    - start_time = Start Time of the Class

  end_time = End of the class
  period = Duration between two classes
  keywords = Keywords associated with the class

- **ClassInstance:**
  Fields:
  ClassObj: Foreign Key to Class
  Time: Time of the class
  CurrentCapacity: Stores the current capacity of the class

- **UsersEnrolled**:
  Fields:
  User - ForeignKey to the Default user model
  ClassInstance - ForeignKey to the User

Subscriptions:

1. **Subscription:** This enables creating a link between the Studio and the Subscription Plan model. A User creates a link ie: a subscription with the Studio through this model

    Studio: Foreign Key to the Studio to which the Subscription is made
    User: Foreign Key to the Default User Model
    Plan: Foreign Key to the Subscription Plan
    Start Date: Start Date of the Subscription
    End Date: End Date of the Subscription
    Payment Date: Date when the payment for the subscription was made
    Card Name: Card Used to make the subscription
    Card Number: Card Number of the card used to make the subscription

2. SubscriptionPlan: These are the plans available for selection for the users. Users select from these available subscription options.

name = Name of the Subscription
price = Price of the subscription
duration = Duration of the subscription
description = Description of the subscription plan
subscription_type = Type of the subscription

Important Note:
While using Postman, please set the access token as the environment variable “token”. This will enable authentication since we mentioned {{token}} as our Bearer Token for authentication

Snippet API

Authentication	none
Source Code	shell

Snippet API

```
# Install the command line client
$ pip install coreapi-cli
```

accounts

add_card > create

Interact

POST `/accounts/add_card/`

Add or update card details. Mandtory fields: number, name

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action accounts add_card create
```

edit_profile > create

Interact

POST `/accounts/edit_profile/`

Edit user profile. fields: username, password, email, first_name, last_name, avatar

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action accounts edit_profile create
```

login > create

Interact

POST `/accounts/login/`

Takes a set of user credentials and returns an access and refresh JSON web token pair to prove the authentication of those credentials.

Request Body

The request body should be a `"application/json"` encoded object, containing the following items.

Parameter	Description
`username` required	
`password` required	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action accounts login create -p username=... -p password=...
```

ping > list

⇄ Interact

GET `/accounts/ping/`

Test if the user is logged in

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action accounts ping list
```

refresh_token > create

⇄ Interact

POST `/accounts/refresh_token/`

Takes a refresh type JSON web token and returns an access type JSON web token if the refresh token is valid.

Request Body

The request body should be a `"application/json"` encoded object, containing the following items.

Parameter	Description
`refresh` required	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action accounts refresh_token create -p refresh=...
```

signup > create

⇄ Interact

POST `/accounts/signup/`

Create user profile. Mandatory fields: username, password, email, first_name, last_name. Optional fields: avatar

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action accounts signup create
```

studios

classes > drop > read

⇄ Interact

GET `/studios/classes/drop/{id}/`

Drop a user from a class. Required fields: class id

Path Parameters

The following parameters should be included in the URL path.

Parameter	Description
id **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios classes drop read -p id=...
```

classes > drop > session > read

⇄ Interact

GET `/studios/classes/drop/session/{id}/`

Drop a user from a class session. Required fields: class instance id

Path Parameters

The following parameters should be included in the URL path.

Parameter	Description
id **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios classes drop session read -p id=...
```

classes > enroll > read

⇄ Interact

GET `/studios/classes/enroll/{id}/`

Enroll a user in a class. Required fields: class id

Path Parameters

The following parameters should be included in the URL path.

Parameter	Description
`id` **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios classes enroll read -p id=...
```

classes > enroll > session > read

⇄ Interact

GET `/studios/classes/enroll/session/{id}/`

Enroll a user in a class session. Required fields: class instance id

Path Parameters

The following parameters should be included in the URL path.

Parameter	Description
`id` **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios classes enroll session read -p id=...
```

classes > search > list

⇄ Interact

GET `/studios/classes/search/`

Search for classes using name of the studio, class name, coach name, date, or range. Required fields: studio id

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios classes search list
```

closest > read

⇄ Interact

GET `/studios/closest/{longitude}/{latitude}/`

Return studios in the order of closest to user

Path Parameters

The following parameters should be included in the URL path.

Parameter	Description
`longitude` **required**	
`latitude` **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios closest read -p longitude=... -p latitude=...
```

details > read

⇄ Interact

GET `/studios/details/{id}/`

Get the details of a studio. Required fields: studio id

Path Parameters

The following parameters should be included in the URL path.

Parameter	Description
`id` **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios details read -p id=...
```

schedule > read

⇄ Interact

GET `/studios/schedule/{id}/`

Get the classes schedule of a studio. Required studio id

Path Parameters

The following parameters should be included in the URL path.

Parameter	Description
`id` **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios schedule read -p id=...
```

search > list

⇄ Interact

GET `/studios/search/`

Search for studios by name, type, class, coach (provided as query parameters)

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios search list
```

user > schedule > list

⇄ Interact

GET `/studios/user/schedule/`

Return the schedule of a user.

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action studios user schedule list
```

subscriptions

read

GET `/subscriptions/{subscription_id}/`

Get subscription details. Mandatory fields: subscription_id

⇄ Interact

Path Parameters

The following parameters should be included in the URL path.

Parameter	Description
`subscription_id` **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action subscriptions read -p subscription_id=...
```

cancel > create

⇄ Interact

POST `/subscriptions/{subscription_id}/cancel/`

Enables a User to cancel subscription

Path Parameters

The following parameters should be included in the URL path.

Parameter	Description
`subscription_id` **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action subscriptions cancel create -p subscription_id=...
```

create > create

⇄ Interact

POST `/subscriptions/create/`

Enables Users to add a Subscription, ie: subscribe to one of the available subscriptions.

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action subscriptions create create
```

paymenthistory > list

⇄ Interact

GET `/subscriptions/paymenthistory/`

Gets the Payment history for Subscribed Subscriptions, if the User hasn't subscribed or has canceled a Subscription, they wouldn't be able to view the payment history.

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action subscriptions paymenthistory list
```

subscriptionplans > list

⇄ Interact

GET `/subscriptions/subscriptionplans/`

Gets the Subscription Plans available for subscription for the users

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action subscriptions subscriptionplans list
```

update > create

⇄ Interact

POST `/subscriptions/{subscription_id}/update/`

Enables editing the subscription, ie: enables Users to change Subscription Plans. For instance another plan is available ie: a yearly subscription, then the User can subscribe to that particular Plan.

Path Parameters

The following parameters should be included in the URL path.

Parameter	**Description**
`subscription_id` **required**	

```
# Load the schema document
$ coreapi get http://localhost:8000/docs/

# Interact with the API endpoint
$ coreapi action subscriptions update create -p subscription_id=...
```