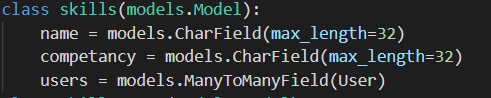
**Backend Design Approach-**

Backend is designed following *Django framework*. *Rest-framework* is used for creating all the APIs. *MySQL* is used for database. One API VIEW is designed for each component or section of a profile page. Class based API VIEWS are used. Each class is mapped to a particular URL. To know, how various API VIEWS are programmed, Go to *api/rest\_apis/ap*i and click on any view.

Below is the screenshot of Skills API. All other APIS are written in a similar manner. Detailed functioning of this API and its interaction with *serializers, models & end points* are explained below-

**1. Model-**

****

a. Each model is a subclass of *models.Model*.

b. *User* is a default Django model used for storing information about all the registered users of the app. *Username, password, email* etc are its default fields. To know how a new user can be created, go to *api/rest\_apis/api/authentication\_view.py*. this file contains three methods (one for registering a new user, another for login and last one is for logout). Each of the methods are mapped to a different URL. So in order to register a user, one can make a *post request* to the corresponding url with necessary credentials.

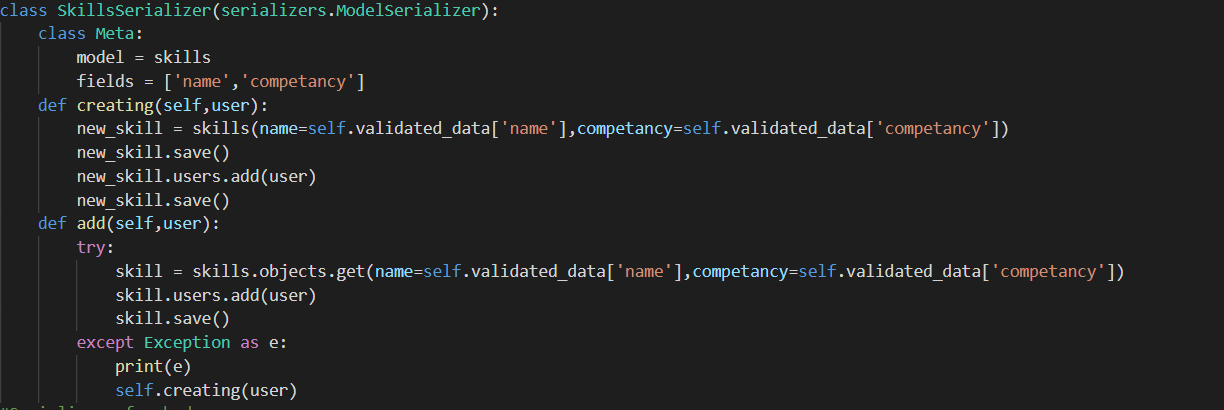
c. Any skill can be shared among multiple users. Also a user can have multiple skills with varying competencies, hence, there is a many to many relation between ‘skills’ and ‘user’ model.

d. In order to create a new model, Go to *models.py* file, write a new class, write required fields. Then to create a SQL table for the model in your database, use command line, Go to *API\_DEVELOPMENT\_USING\_DRF/api,* and execute following two commands sequentially-

1. *python manage.py migrate.*

2. *python manage.py makemigrations.*

**2. Serializers-**

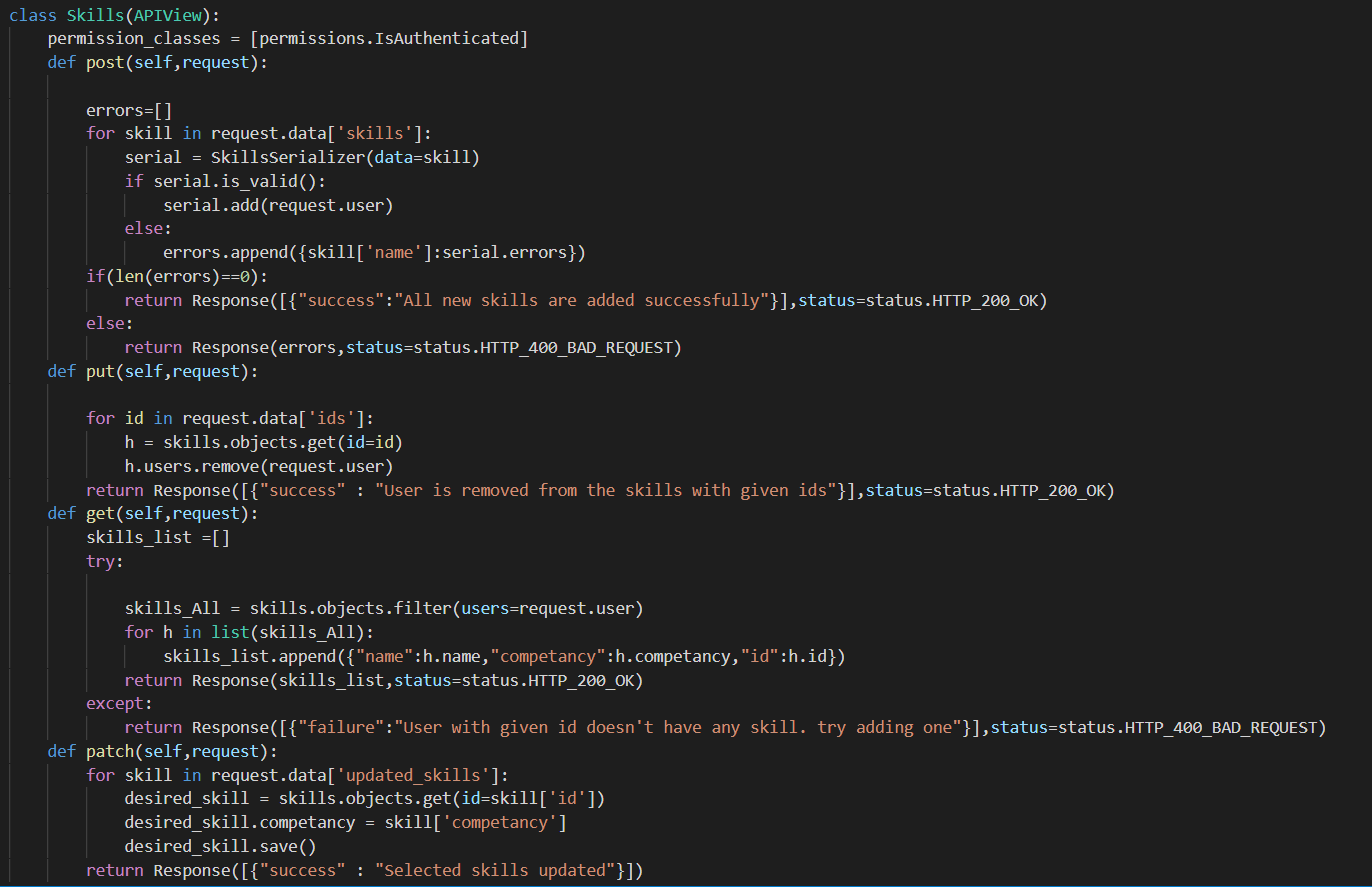
****

For each API, one serializer is defined in *serializers.py* file. A Serializer is used for serializing & deserializing the data. It is used in two cases-

1. Whenever a user make a POST request to create a new skill, it sends a json object with fields ‘name’ and ‘competancy’. Serializer will take ‘json object’ as an argument and creates a new instance of skill model. in this way, a new row is added into the ‘skill’ table.

2. When user make GET request to get all the skills for the user (currently logged in into the app). we first queries for all the entries in the ‘skills’ model for that user. Then in order to convert all the entries into ‘json object’, serializer is used. Newly created json object then sent to the user as a response to the GET request.

**3. API VIEW-**

****



1. in *urls.py* file, each API VIEW is mapped to at least one path. in the above image, ‘Skills’ api view is mapped to the path *‘http://localhost:8000/skills’* so whenever, user visits this path, one of the method of the class ‘Skills’ will be called.

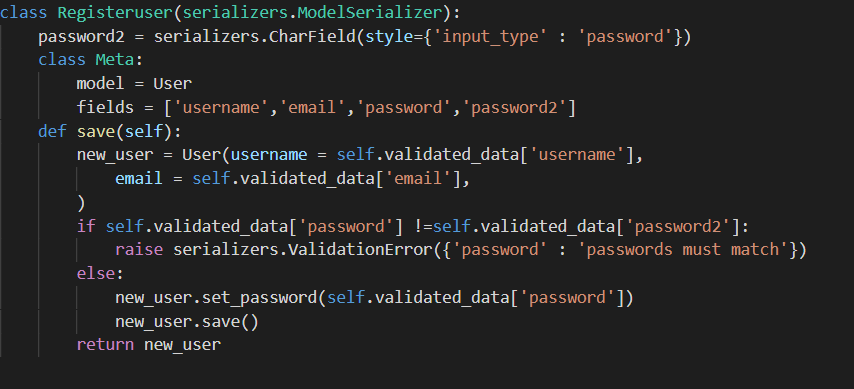
2. POST method will be called, when a user makes POST request to the path corresponding to ‘Skills’ API. ([*http://localhost:8000/skills*](http://localhost:8000/skills)*).*  Similarly, get, put, patch will be called.

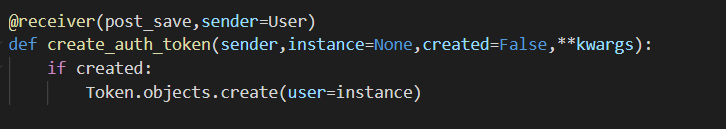
3. The ‘Skills’ api receives a list of json objects, with each object having two keys (name, competency). See the GET method- for each object, it is using serializer to create either a new instance of ‘Skill’ model or adding the user to an instance if the skill already exists (to avoid creation of duplicates).

4. All other methods can be understood in a similar way.

5. In the class, a list of *permission\_classes* is defined, permissions.IsAuthenticated means, the end point ([*http://localhost:8000/skills*](http://localhost:8000/skills)*) can’t be accessed without signing in. user can’t make any requests to that end point if it isn’t signed in.*

**4. Authentication-**

****



Previously, skill api is discussed, similarly, for registering a new user, we have an *api view, a model (default Django User model), and a serializer.*

In order to register a new user, A POST request should be made to (<http://localhost:8000/registeruser>), ‘save’ method of serializer uses ‘User’ model and creates a new instance, then sets the password for that user.

As soon as a new user is created, a signal is sent and is received by ‘*create\_auth\_token*’ method, it creates a unique ‘token’ for that user. Using this ‘token’, user can access all the API end points where authentication permissions are required for access.