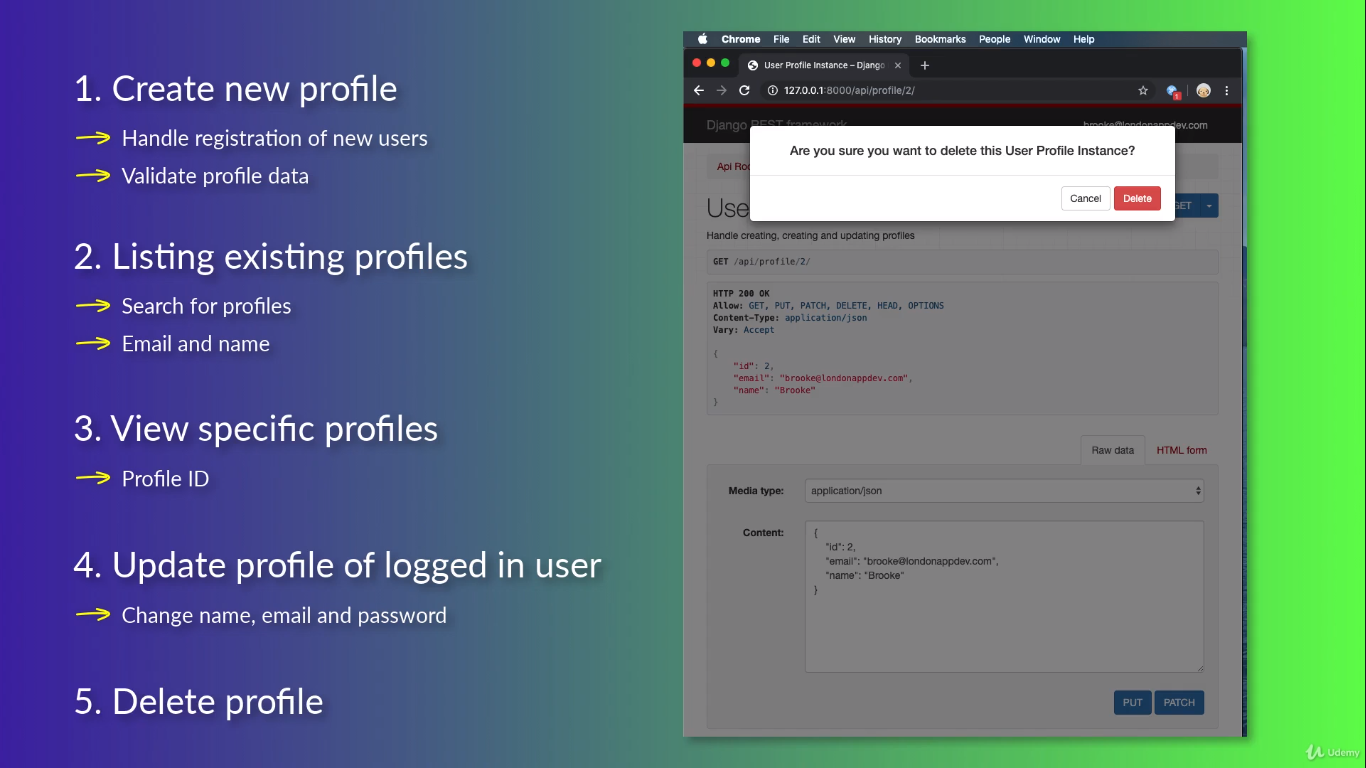
**Plan Our Profile API:**

Specifications:

Profile API will be able to handle the following:



1. Create new profile

-> Handles registration of new users in the system.

-> Validating the profile data to ensure that a user provided all the required fields.

2. Listing existing profiles

-> So users can find other users in the system (Search for profiles).

-> Ways to search for users by email and name.

3. View specific profiles

-> using profile ID.

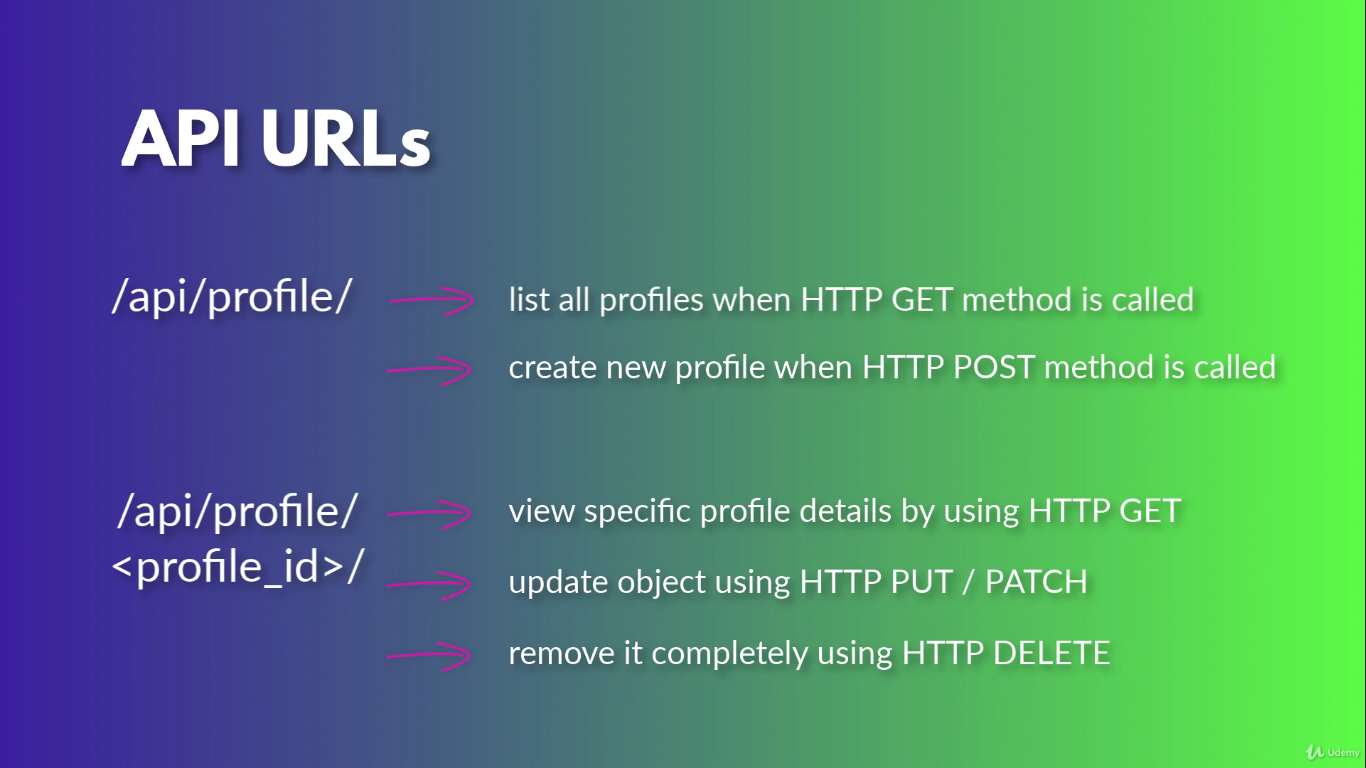
4. Update profile of logged in user

-> Users in the system will be able to change their name, email and password in their profiles.

5. Delete profile

-> Users will be able to delete their own profile.

What URLs and methods our API have?



**Create User Profile Serializer:**

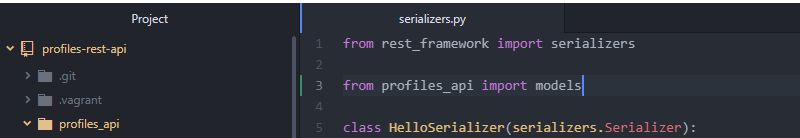
Creating serializer for user profile objects:

1. Go to serializers.py under profiles\_api in Atom editor

2. Add a new serializer class

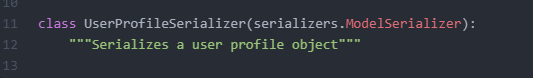
Model serializer – similar to regular serializer but it has some extra functionality which makes it easier to work with Django database models.

3. Perform imports

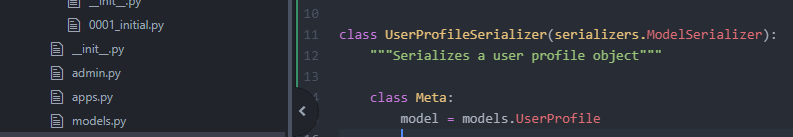


Allow us to access our user profile model that we created previously (profiles\_api -> models.py)

4. Create a serializer class



- define a meta class



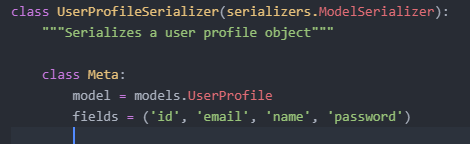
To work with ModelSerializer we use meta class to configure the serializer to point to a specific model in our project.

Variable

Sets our serializer to point to our user profile model.

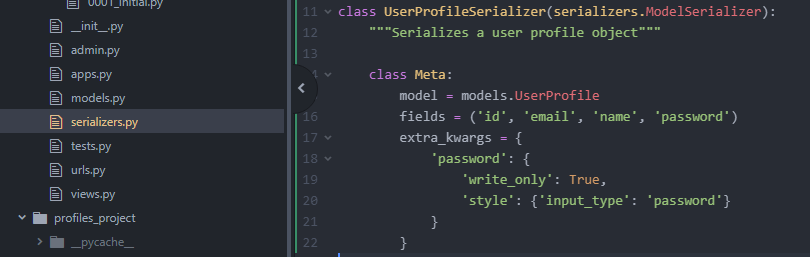
4.1. Specify list of fields in our model that we want to manage through our serializer.

Define the list of fields that we want to make accessible in our API or want to use to create new models with our serializer.



Tuple of fields that we want to make accessible In our model.

4.2. Make the password field as write only



Variable

Keys of the dictionary are the fields to which we want to add custom configuration.

‘write\_only’: True - can only use it to create new objects or update objects. (Can’t use it to retrieve objects)

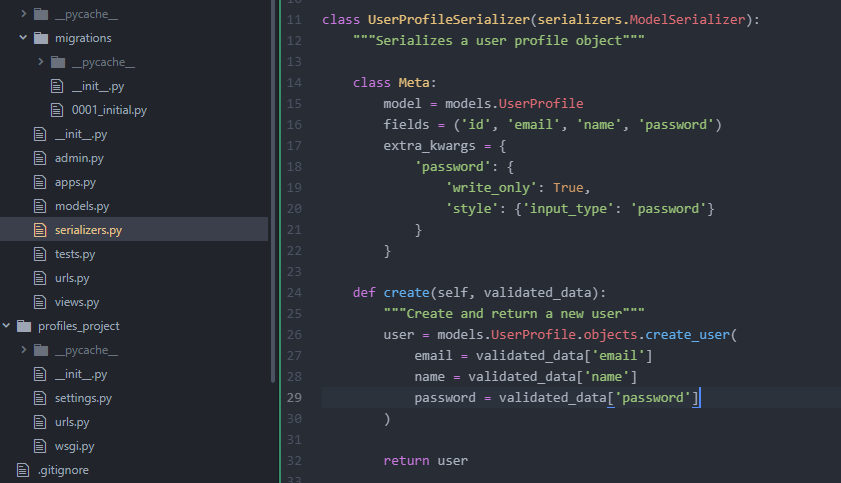
{When we request a GET, we will not see the password field included in the response}

Add a custom style. Set the field type to a password field (which means we will not be able to see the input as we are typing, we’ll see dots or stars)

5. Override the create function

By default the model serializer allows us to create simple objects in the database so it uses the default create function of the object manager to create the object.

We will override this functionality for this serializer so that it uses create user function instead of the create function. (The reason being that the password should get created as a hash and not as clear text password)



Whenever we create a new object with our UserProfileSerializer class, it will validate the object or validate the fields provided to the serializer, then it will call the create function passing in the validated data.

Create and return a new user from our UserProfile model Manager.

It will override the create function and call create\_user function (defined in models.py -> class UserProfileManager -> def create\_user)

Pass appropriate fields from the validated data

Return the new user