Decyphr Documentation

Release 0.0.1

Aaron Sinnott

May 28, 2020

Table of Contents

- *Index*
- *Module Index*
- *Search Page*

The accounts app.

The accounts will handle a couple of different scenarios, which are:

- store basic user information, such as the users name, email, password, etc so that a user can be indentified and authorized within the application
- store any preferences that a user may have about how they interact with the application, such as the language that they would like to read the site in
- store information about how often the user is logging in. This will us to award a user for logging everyday, and it will also allow us to send reminders to users to remind them to come back to the site in the event that they haven't accessed the platform in some time

The authentication system is token based. When a user creates an account, a token will be generated for them. After both registration and login, the token will be returned to the requesting client to enable authentication to be performed on all API calls that require authentication.

In addition to this, we will need to be able to store additional information about the user and their login habits. In order to achieve this, we had to build a custom view that handles the login process, rather than using the view that the DRF provides out of the box.

The first version of this app should provide the following functionality:

- User registration - Validate incoming data - Create the user profile - Generate the user's token - Send a confirmation email - Return token to client
- User login - Validate incoming data - Authenticate user - Return token to client
- User logout - Log user out - Return message to the client
- User profiles - Users should be able to view their profiles, update any aspect of the profile and even delete the profile if they choose
- Password reset - Send an email to the user as confirmation - Validate the incoming data - Update the password

The accounts models.

This is where we'll store the information relevant to the users' accounts.

We want to able to store the user's information, such as their usual profile information, their preferences for the site, and also a login history to give users points for logging into the app everyday.

class accounts.models.**UserProfile** (**args*, ***kwargs*)

The user model that stores the user's overall profile.

This model extends Django's *AbstractUser* and inherits the majority of the necessary fields like *username*, *email* and *password*. As such, not all validation is done at the model level. Unique emails are enforced within the *serializers*.

first_language and *language_being_learned* are also required fields, but only enforced by the *serializer*. Users created by other means, like superusers, will be given a default value upon the creation of the account.

The *language_preference* field will allow users to choose what lanugage they would like to view the site in.

exception DoesNotExist

exception MultipleObjectsReturned

The accounts serializers

The Accounts app contains a number of different serializers that will all be used for various instances.

The *EmptySerializer* is provided as a dummy serializer required by Django Rest Framework's viewset classes.

Upon login, a user's incoming data will deserialized by the *UserLoginSerializer*. This will simply just contain the login credentials provided by the user. Once the user has successfully logged in and authenticated, the view will return the *AuthorisedUserSerializer*.

Upon registration, a users's incoming data will be deserialized by the *RegisterUserSerializer*. This will contain the information that the user provided when filling out the registration form. Once the user has successfully been created, the view will return the *AuthorisedUserSerializer*.

class accounts.serializers.**AuthorisedUserSerializer** (*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Authorised User Serializer

The serializer that will contain the general information about the user, after they have logged in

get_auth_token (*obj*)

Get auth token

Get the user's authentication token.

class accounts.serializers.**EmptySerializer** (*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

class accounts.serializers.**RegisterUserSerializer** (*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Register New User

The serializer that will be used to regsiter a new user

validate_email (*value*)

Validate email

Validate the user's email address to ensure that the email address doesn't already exist in the database. An error will be raise if the email exists within the database

class accounts.serializers.**UserLoginSerializer** (*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Login Serializer

The serializer that will be used for the user's incoming login credentials.

Parameters
- **username** (*str*) – The username provided by the user
- **password** (*str*) – The password provided by the user

class accounts.views.**AuthViewSet** (***kwargs*)

Authentication Viewset

The viewset that will be responsible for handle the user's authentication and authorization

get_serializer_class ( )

Get Serializer Class

This viewset has multiple actions and has potentially different serializers per action. This will handle the switching out of each of the serializer classes

login (*request*)

Login

Authenticate the user and log them in, and return the user and the user's token to the client. The request data will be serialized as per the requirements of the *UserLoginSerializer*.

Returns A JSON-ified UserProfile object which also includes the token for that user

Return type AuthorizedUserSerializer

serializer_class

alias of *accounts.serializers.EmptySerializer*

Accounts Utilities

The module contains a set of helper methods that will be used to create and authenticate users within the application. These functions will mostly exist for the purpose of wrapping existing functionality within Django

accounts.utils.**create_user_account** (*email*, *password*, *username=''*, *first_language=0*, *language_being_learned=0*, *language_preference=0*)

Create User Account

Create a new user account using the information provided. This function will just wrap Django's *.create_user* method in order to create a new user instance with the protected password.

Parameters
- **email** (*str*) – The email provided by the user
- **password** (*str*) – The password provided by the user
- **username** (*str*) – The username provided by the user
- **first_language** (*int*) – The ID of the language that the user chose as
- **first language** (*their*) –
- **language_being_learned** (*int*) – The ID of the language that the user
- **as the language that they wish to learn** (*chose*) –
- **language_preference** (*int*) – The ID of the language that the user
- **as the language that they wish to view the site in** (*chose*) –

Returns The newly created user instance

Return type *UserProfile*
Example
This function can be called by explicitly passing in the arguments:

```
user = create_user_account(email, password, username,
    first_language, language_being_learned, language_preference)
```

Or, by unpacking the *RegisterUserSerializer*:

```
user = create_user_account(**serializer.validated_data)
```

accounts.utils.**get_and_authenticate_user** (*username*, *password*)
Authenticate the user
Authenticate the user that's attempting to log in using their username and password.

Parameters
- **username** (*str*) – The username that the user provided
- **password** (*str*) – The password that the user provided

Returns An authenticated user instance is returned if the authentication process is successful

Return type *UserProfile*

Raises ValidationError – If the username and password cannot be matched
Examples
This function can be called by explicitly passing the username and password:

```
user = get_and_authenticate_user(username, password)
```

Or, by unpacking the *UserLoginSerializer*:

```
user = get_and_authenticate_user(**serializer.validated_data)
```

a

A

C

E

G

L

R

S

U

V