Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

base: master ... compare: new-features-branch

✓ **Able to merge.** These branches can be automatically merged.

update and refactorize the code #2

No description available

View pull request

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Use Closing keywords in the description to automatically close issues

Helpful resources

GitHub Community Guidelines

1 commit | 3 files changed | 1 contributor

Commits on Mar 9, 2024

update and refactorize the code

abbatistam committed 3 days ago

Showing 3 changed files with **85 additions** and **105 deletions**. Split | Unified

81 myproj/notifapi/consumers.py

```
...  ...  @@ -1,71 +1,52 @@
     1  + import json
1    2    from channels.generic.websocket import AsyncWebsocketConsumer
2    3    from django.apps import apps
3         - from asgiref.sync import sync_to_async, async_to_sync
4         - import json
5         - from channels.layers import get_channel_layer
6         - import asyncio
7         -
     4  + from channels.db import database_sync_to_async
8    5
9    6    class NotificationConsumer(AsyncWebsocketConsumer):
     7  +     """
     8  +     WebSocket consumer for handling notifications.
     9  +     """
    10  +
10   11        async def connect(self):
11        -         # Allow all connections
12   12            await self.channel_layer.group_add("public_room", self.channel_name)
13   13            await self.accept()
14   14            await self.update_notification_count()
15   15
16        -
17        -     async def update_notification_count(self, event=None):
18        -         print("update_notification_count method called with event:", event)
19        -         # you can't use a model directly from a consumer
    16  +     @database_sync_to_async
    17  +     def get_notifications(self):
20   18            NotifyModel = apps.get_model('notifapi', 'NotifyModel')
21        -         notifications = await sync_to_async(list)(NotifyModel.objects.all().values('is_read'))
22        -         messages = await sync_to_async(list)(NotifyModel.objects.all().values('message'))
23        -         # Get the notification count asynchronously using a custom utility method
    19  +         return list(NotifyModel.objects.all().values('is_read', 'message'))
```

```
                +
                +     async def update_notification_count(self):
                +         notifications = await self.get_notifications()
24              23            notification_count = len(notifications)
25              -
26              -
27              -         #print(f"Notification count is {notification_count}")
28              -         # Extracting is_read values from notifications
29              24            is_read_values = [notification['is_read'] for notification in
                notifications]
30              -         messages_values = [notification['message'] for notification in
                messages]
31              -         #print("Am I here?")
32              -         print(f"Messages values are: {messages_values}")
33              -         """ Sends a notification to the client by converting the event data
                (a dictionary) to a JSON string and sending it as text data through the
                WebSocket connection. """
                25        +         messages_values = [notification['message'] for notification in
                notifications]
                26        +
                27        +         await self.send_notification_update(notification_count,
                is_read_values, messages_values)
                28        +
                29        +     async def send_notification_update(self, count, is_read_values,
                messages_values):
34              30            await self.send(text_data=json.dumps({
35              31                "type": "notification.update",
36              -                 "count": notification_count,
                32        +         "count": count,
37              33                "is_read_values": is_read_values,
38              34                "messages_values": messages_values
39              35            }))
40              36
41              -
42              37        async def disconnect(self, close_code):
43              -         print("Consumer disconnected")
44              -         # Remove the channel from the public_room group when the WebSocket
                connection is closed
45              -         await self.channel_layer.group_discard(
46              -             "public_room",
47              -             self.channel_name
48              -         )
                38        +         await self.channel_layer.group_discard("public_room",
                self.channel_name)
49              39
50              40        async def receive(self, text_data):
51              -         # Handle incoming messages (if any)
52              -         data = json.loads(text_data)
53              -         await self.update_notification_count(data)
54              -
55              -
```

```
56  -  # this function is just for test, but it does not work
57  -  def send_notification_update_to_clients():
58  -      channel_layer = get_channel_layer()
59  -      notification_count = 10  # Example notification count
60  -      is_read_values = [False, True, False]  # Example list of read statuses
61  -      messages_values = ["New message 1", "New message 2", "New message 3"]
        # Example list of messages
62  -
63  -      async_to_sync(channel_layer.send)(
64  -          "public_room",
65  -          {
66  -              "type": "send.notification.update",
67  -              "count": notification_count,
68  -              "is_read_values": is_read_values,
69  -              "messages_values": messages_values
70  -          }
71  -      )
        ⊖
41  +          try:
42  +              data = json.loads(text_data)
```

```
+         except json.JSONDecodeError as e:
+             await self.send_error_message("Error decoding JSON: {}".format(e))
+         else:
+             await self.update_notification_count()
+
+     async def send_error_message(self, error_message):
+         await self.send(text_data=json.dumps({
+             "type": "error",
+             "message": error_message
+         }))
```

77 myproj/notifapi/models.py

```
@@ -1,55 +1,64 @@
 from django.db import models
-from datetime import datetime
 from django.contrib.auth.models import User

 class NotifyManager(models.Manager):
-    def addNotification(self, typerel, person_from, person_to, redirect_to, msg):
-        # created specify if the object is created or not
-        (notification, created) = NotifyModel.objects.get_or_create(
-            sender_id=person_from,
-            receiver_id=person_to,
-            type=typerel,
-            redirect_url=redirect_to,
-            message=msg
-        )
-        if created is False: # object exists
-            if notification.is_read is False:
-                # if the notification already exists, update the date
-                notification.created_at = datetime.now()
-                notification.save()
-        return notification
+    def add_notification(self, type_relationship, person_from, person_to, redirect_to, message):
+        """
+        Add a new notification.
+
+        Args:
+            type_relationship (int): The type of relationship.
+            person_from (User): The sender of the notification.
+            person_to (User): The receiver of the notification.
+            redirect_to (str): The URL to redirect to.
+            message (str): The message of the notification.

+        Returns:
```

```
+            Notification: The created or existing notification object, or None if there was an error.
+        """
+        try:
+            notification, created = self.get_or_create(
+                sender=person_from,
+                receiver=person_to,
+                type=type_relationship,
+                redirect_url=redirect_to,
+                message=message
+            )
+            if not created and not notification.is_read:
+                notification.save()
+            return notification
+        except Exception as e:
+            # Log the error or handle it accordingly
+            print(f"Error adding notification: {e}")
+            return None

class NotifyModel(models.Model):
    class Meta:
        db_table = 'notifications'
-
-    BLOCK = 0 # will be used only in activity log
+
+    BLOCK = 0
    FAVORITE = 1
    TEASE = 2
    UPVOTE = 3
-    DOWNVOTE = 4 # will be used only in activity log
+    DOWNVOTE = 4
    VISIT = 5
    MESSAGE = 6

    TYPE_CHOICES = [
-        (BLOCK, 'blocked'),
-        (FAVORITE, 'favorites'),
-        (TEASE, 'teased'),
-        (UPVOTE, 'upvoted'),
-        (DOWNVOTE, 'downvoted'),
-        (VISIT, 'visited'),
-        (MESSAGE, 'message'),
+        (BLOCK, 'Blocked'),
+        (FAVORITE, 'Favorites'),
+        (TEASE, 'Teased'),
+        (UPVOTE, 'Upvoted'),
+        (DOWNVOTE, 'Downvoted'),
+        (VISIT, 'Visited'),
+        (MESSAGE, 'Message'),
    ]
```

```
45          -     """
46          -     Blank values for Django field types such as DateTimeField or ForeignKey
              will be stored as NULL in the DB. blank determines whether the field will
              be required in forms.
47          -     """
48          -     sender_id = models.ForeignKey(User, related_name='sender',
              on_delete=models.CASCADE, null=True)
49          -     receiver_id = models.ForeignKey(User, related_name='receiver',
              on_delete=models.CASCADE, null=True)
50          -     created_at = models.DateTimeField(auto_now=True)
51          -     type = models.PositiveSmallIntegerField()
52          -     is_read = models.BooleanField(null=False, default=False)
        57  +     sender = models.ForeignKey(User, related_name='sent_notifications',
              on_delete=models.CASCADE, null=True)
        58  +     receiver = models.ForeignKey(User,
              related_name='received_notifications', on_delete=models.CASCADE, null=True)
        59  +     created_at = models.DateTimeField(auto_now_add=True)
        60  +     type = models.PositiveSmallIntegerField(choices=TYPE_CHOICES)
        61  +     is_read = models.BooleanField(default=False)
53      62        redirect_url = models.CharField(max_length=120, null=True)
54      63        message = models.CharField(max_length=250, null=True)
55      64
```

32 myproj/notifapi/signals.py

```
7       7
8       8   @receiver(post_save, sender=Notification)
9       9   def notification_created(sender, instance, created, **kwargs):
10          -
11      10      if created:
```

```
12      -           print(f"A new notification was created {instance.message}")
    11  +           print(f"A new notification was created: {instance.message}")
13  12              channel_layer = get_channel_layer()
14      -           print(channel_layer)
15  13              try:
16      -               async_to_sync(channel_layer.group_send)(
17      -                   'public_room',
18      -                   {
19      -                       "type":"update_notification_count",
20      -                       "message":instance.message
21      -                   }
22      -               )
23      -               """
24  14                  async def send_notification():
25      -                   await channel_layer.send("public_room", {
26      -                       # OJO, update_notification_count nunca es llamada
27      -                       # ese es el verdadero problema. Solo es llamada después
28      -                       # de refrescar con F5
29      -                       "type": "update_notification_count",
30      -                       "message": instance.message
31      -                   }) """
    15  +                   await channel_layer.group_send(
    16  +                       'public_room',
    17  +                       {
    18  +                           "type": "update_notification_count",
    19  +                           "message": instance.message
    20  +                       }
    21  +                   )
32  22
33      -               # Call the async function to send the notification
34      -               #async_to_sync(send_notification)()
    23  +               # Llama a la función asíncrona para enviar la notificación
    24  +               asyncio.run(send_notification())
35  25
```

```
36  26          except Exception as e:
    27  +           # Maneja errores de manera más robusta
37  28              print(f"Error in group_send: {e}")
38      -
```