**RepeatPro documentation (2)**

**Project Serializers**

**1. User Profile Serializer**

**======================================================================**

**Сериализатор для преобразования данных о профилях**

**======================================================================**

class UserProfileSerializer(serializers.ModelSerializer):

class Meta:

model = UserProfile

fields = ('id', 'email', 'firstname', 'lastname', 'password', 'avatar', 'age', 'gender', 'phone', 'rating')

extra\_kwargs = {'password': {'write\_only': True}}

def create(self, validated\_data):

print('212121')

user = UserProfile.objects.create\_user(\*\*validated\_data)

return user

class UserNotificationSerializer(serializers.ModelSerializer):

class Meta:

model = UserNotification

fields = ('id', 'user\_id', 'tutor\_id', 'status', 'text', 'price')

**======================================================================**

**`UserProfileSerializer` работает с моделью `UserProfile`, включая указанные в `fields` поля. Метод `create` используется для создания новых пользователей.**

**`UserNotificationSerializer` работает с моделью `UserNotification`, включая указанные в `fields` поля.**

**Project Views**

**1. User View**

**=============================================================================**

class UserView(APIView):

permission\_classes = (IsAuthenticated,)

parser\_classes = [JSONParser, MultiPartParser, FormParser]

def get(self, request):

print(request.user)

serializer = UserProfileSerializer(request.user, many=False)

return Response(serializer.data, status=status.HTTP\_200\_OK)

def put(self, request):

user = UserProfile.objects.get(email=request.user.email)

user.avatar = request.data['avatar']

user.save()

return Response({'message': 'Image updated'}, status=status.HTTP\_200\_OK)

**=============================================================================**

**Это API предоставляет доступ к списку активных репетиторов через GET запрос с параметрами: `salary`, `first\_name`, `last\_name`, `bio`, `avatar`, `experience`, `rating`, `course`. Фильтрация, поиск и сортировка поддерживаются через `django-filter`, используя `search\_fields` для поиска и `ordering\_fields` для сортировки. Пользовательские права управляют доступом: авторизованные пользователи могут взаимодействовать, остальные могут только читать. Для фильтрации доступны параметры: `salary`, `experience`, `degree`, `rating`, `course`.**

**2. Register User View**

**=============================================================================**

class RegisterUserView(APIView):

parser\_classes = [JSONParser, MultiPartParser, FormParser]

def post(self, request):

# if email is already in use

if UserProfile.objects.filter(email=request.data['email']).exists():

return Response({'error': 'Email already registered'}, status=status.HTTP\_400\_BAD\_REQUEST)

else:

serializer = UserProfileSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

**=============================================================================**

**Этот класс `RegisterUserView` представляет API-представление для регистрации пользователей. Он принимает данные в разных форматах с помощью различных парсеров. В методе `post`, производится проверка, зарегистрирован ли уже пользователь с указанным электронным адресом (`email`). Если такой адрес уже зарегистрирован, возвращается ошибка. В противном случае, данные проверяются с помощью `UserProfileSerializer`. Если данные валидны, новый пользователь сохраняется и возвращается успешный ответ с данными пользователя. Если данные невалидны, возвращаются ошибки валидации.**

**3. All Users View**

**=============================================================================**

class AllUsersView(APIView):

permission\_classes = (IsAuthenticated, )

def get(self, request):

if request.user.is\_staff:

users = UserProfile.objects.all()

serializer = UserProfileSerializer(users, many=True)

return Response(serializer.data, status=status.HTTP\_200\_OK)

else:

return Response(status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

**=============================================================================**

**Этот класс `AllUsersView` представляет API-представление для получения списка всех пользователей. Он требует аутентификации пользователя. В методе `get`, если пользователь является персоналом (администратором), то все пользователи извлекаются из базы данных с помощью `UserProfile.objects.all()`, затем сериализуются с использованием `UserProfileSerializer` и возвращаются в виде ответа. Если пользователь не является персоналом, то возвращается ответ с кодом `HTTP\_405\_METHOD\_NOT\_ALLOWED`, что означает, что метод не разрешен для данного пользователя.**

**4. Logout View**

**=============================================================================**

class LogoutView(APIView):

permission\_classes = (IsAuthenticated,)

def post(self, request):

try:

refresh\_token = request.data["refresh\_token"]

token = RefreshToken(refresh\_token)

token.blacklist()

return Response(status=status.HTTP\_205\_RESET\_CONTENT)

except Exception as e:

return Response(status=status.HTTP\_400\_BAD\_REQUEST)

**=============================================================================**

**Этот класс `LogoutView` представляет API-представление для выхода пользователя из системы (логаута). Он требует аутентификации пользователя. В методе `post`, извлекается `refresh\_token` из данных запроса, который используется для отзыва токена аутентификации пользователя. Токен помечается как недействительный с помощью метода `blacklist()` из библиотеки `RefreshToken` (предположительно, отвечающей за управление токенами обновления). Если операция проходит успешно, возвращается ответ со статусом `HTTP\_205\_RESET\_CONTENT` (сброс контента). Если возникает ошибка, возвращается ответ с кодом `HTTP\_400\_BAD\_REQUEST`, что означает, что запрос был неверным или содержал ошибку.**

**5. All Courses**

**=============================================================================**

class AllCourses(APIView):

permission\_classes = (IsAuthenticated,)

def get(self, request):

try:

data = requests.get('http://api')

return data.json()

except Exception as e:

return Response(status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

**=============================================================================**

**Этот класс `AllCourses` представляет API-представление для получения списка всех курсов. Он требует аутентификации пользователя. В методе `get`, попытка отправить запрос на внешний API с адресом `'http://api'` для получения данных о курсах. Если запрос успешен, возвращается JSON-подобный ответ с данными курсов. В случае возникновения ошибки (например, недоступность API), возвращается ответ с кодом `HTTP\_405\_METHOD\_NOT\_ALLOWED`, что означает, что метод не разрешен для данного пользователя.**

**6. All Tutors**

**=============================================================================**

class AllTutors(APIView):

permission\_classes = (IsAuthenticated,)

def get(self, request):

try:

data = request.get('http://127.0.0.1:8000/tutors/')

return data.json()

except Exception as e:

return Response(status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

**=============================================================================**

**Этот класс `AllTutors` представляет API-представление для получения списка всех репетиторов. Он требует аутентификации пользователя. В методе `get`, предположительно, есть ошибка в строке `data = request.get('http://127.0.0.1:8000/tutors/')`, она должна быть `data = requests.get('http://127.0.0.1:8000/tutors/')`. В данном случае, выполняется запрос к адресу `'http://127.0.0.1:8000/tutors/'` для получения данных о репетиторах. Если запрос успешен, возвращается JSON-подобный ответ с данными о репетиторах. В случае возникновения ошибки (например, недоступность API), возвращается ответ с кодом `HTTP\_405\_METHOD\_NOT\_ALLOWED`, что означает, что метод не разрешен для данного пользователя.**

**7. Tutor Request**

**=============================================================================**

class TutorRequest(APIView):

permission\_classes = (IsAuthenticated,)

def post(self, request):

try:

requests.post('http://127.0.0.1:8000/tutor\_requests/<int:request.data["tutor\_id"]>/', {'tutor\_id': request.data["tutor\_id"],

'client\_id': request.user.id,

'date\_time': request.data["date\_time"],

'text': request.data["text"],

'course\_id': request.data["course\_id"]

})

return Response(status=status.HTTP\_200\_OK)

except Exception as e:

return Response(status=status.HTTP\_400\_BAD\_REQUEST)

**=============================================================================**

**Этот класс `TutorRequest` представляет API-представление для отправки запроса репетитору. Он требует аутентификации пользователя. В методе `post`, выполняется запрос к адресу `'http://127.0.0.1:8000/tutor\_requests/<int:request.data["tutor\_id"]>/'`, предположительно, для отправки данных о запросе на репетитора. Данные, такие как `tutor\_id`, `client\_id`, `date\_time`, `text` и `course\_id`, передаются в теле POST-запроса. Если запрос успешен, возвращается ответ со статусом `HTTP\_200\_OK`. В случае возникновения ошибки (например, неверные данные или недоступность адреса), возвращается ответ с кодом `HTTP\_400\_BAD\_REQUEST`, что означает, что запрос был некорректным.**