Молодцы! Хороший старт. Есть над чем поработать.

Михаил Ивановревьюер

Ваш проект с комментариями

Проект "Yamdb" ПРОВЕРЕНА 20 ЯНВАРЯ

api_yamdb-master 8 4 27

api_yamdb 1

py __init__.py НЕТ КОММЕНТАРИЕВ

py asgi.py НЕТ КОММЕНТАРИЕВ

py settings.py 1

```
"""
```

Давайте удалим лишние комментарии по **всему** проекту - чтобы код был чище и читабильнее :)

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
2  Django settings for YaMDb project.
3
4  Generated by 'django-admin startproject' using Django 3.0.5.
5
6  For more information on this file, see
7  https://docs.djangoproject.com/en/3.0/topics/settings/
8
```

```
For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.0/ref/settings/
"""

import os
from datetime import timedelta

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'p&l%385148kslhtyn^##a1)ilz@4zqj=rq&agdol^##zgl9(vs'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = [
    '127.0.0.1',
    '.kromm.info',
    'localhost',
]

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'django_filters',
    'rest_framework.authtoken',

    'title_api',
    'users_api',

]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
```

```
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]


ROOT_URLCONF = 'api_yamdb.urls'

TEMPLATES_DIR = os.path.join(BASE_DIR, "templates")
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATES_DIR],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'api_yamdb.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-
validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
```

```
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },

    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'

# STATICFILES_DIRS = (os.path.join(BASE_DIR, 'static/'),)

STATIC_ROOT = os.path.join(BASE_DIR, 'static')

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticatedOrReadOnly',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
    # 'DEFAULT_AUTHENTICATION_CLASSES': [
    #     'rest_framework.authentication.BasicAuthentication',
    #     'rest_framework.authentication.SessionAuthentication',
    # ],
    'DEFAULT_PAGINATION_CLASS':
'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 100
}
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(days=3),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=30),
```

```
150 }
151
152 AUTH_USER_MODEL = 'users_api.YamdbUser'
153
154 EMAIL_BACKEND = "django.core.mail.backends.filebased.EmailBackend"
155 EMAIL_FILE_PATH = os.path.join(BASE_DIR, "sent_emails")
```

py	urls.py	НЕТ КОММЕНТАРИЕВ		
py	wsgi.py	НЕТ КОММЕНТАРИЕВ		
	data	НЕТ КОММЕНТАРИЕВ		
	static	НЕТ КОММЕНТАРИЕВ		
	templates	НЕТ КОММЕНТАРИЕВ		
	tests	НЕТ КОММЕНТАРИЕВ		
	title_api	6	2	21
	migrations	НЕТ КОММЕНТАРИЕВ		
py	__init__.py	НЕТ КОММЕНТАРИЕВ		
py	admin.py	1		

Было бы неплохо добавить модели в админку - очень удобно

МОЖНО ЛУЧШЕ Отметить как выполненный Михаил Иванов ревьюер

py apps.py НЕТ КОММЕНТАРИЕВ

py filters.py НЕТ КОММЕНТАРИЕВ

py models.py 3 1 5

```
1  from django.core.validators import MaxValueValidator, MinValueValidator
2  from django.db import models
3  from django.contrib.auth import get_user_model
4
5  User = get_user_model()
6
7
   class Category(models.Model):
```

Для моделей сразу лучше прописывать:
1) verbose_name для полей
2) в class Meta verbose_name и verbose_name_plural

МОЖНО ЛУЧШЕ Отметить как выполненный Михаил Ивановревьюер

```
       '''Категории (типы) произведений'''
```

https://www.python.org/dev/peps/pep-0257/ обратите внимание, какие кавычки используются для докстрингов
Нужно проверить и поправить **все** докстринги в проекте

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
10     name = models.CharField(max_length=200)
11     slug = models.CharField(max_length=200, unique=True, blank=True,
   null=True)
12
13     class Meta:
14         verbose_name = "Category"
           ordering = ['name']
```

Стоит придерживаться единообразия - использовать только одинарные или двойные кавычки в одном файле. Нужно проверить и поправить по **всему** проекту (кроме докстрингов, f-строк, в которых внутри есть двойные кавычки т.д)

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
16          constraints = [
                models.UniqueConstraint(fields=['slug', 'name'],
```

Не совсем понятная эта связка, т.к. слаг у нас уже уникален.

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
18                                      name='unique_category')
19          ]
20
21      def __str__(self):
22          return self.name
23
24
25  class Genre(models.Model):
26      '''Жанры'''
27      name = models.CharField(max_length=80)
28      slug = models.SlugField(unique=True, null=True)
29
30      class Meta:
31          ordering = ['name']
32
33      def __str__(self):
34          return self.name
35
36
37  class Title(models.Model):
38      '''Заглавие'''
39      name = models.TextField(
40          max_length=100,
41      )
        year = models.IntegerField()
```

1) Год может быть больше текущего? Стоит написать свой валидатор https://stackoverflow.com/questions/41422565/django-year-validation-returns-ensure-this-value-is-less-than-or-equal-to-2016
2) Необязательно править - На часто используемые поля было бы неплохо повесить индексы https://ru.stackoverflow.com/questions/976248/django-db-index-in-field-%D1%87%D1%82%D0%BE-%D1%8D%D1%82%D0%BE-%D0%B8-%D0%B7%D0%B0%D1%87%D0%B5%D0%BC

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
43      description = models.TextField(
```

```
            max_length=500,
            null=True,
            blank=True
        )
        genre = models.ManyToManyField(
            Genre,
            related_name='titles'
        )
        category = models.ForeignKey(
            Category,
            on_delete=models.SET_NULL,
            related_name='titles',
            null=True,
            blank=True
        )

        class Meta:
            ordering = ['name', 'year']
            verbose_name = 'Произведение'
            verbose_name_plural = 'Произведения'

        def __str__(self):
            return self.name


    class Review(models.Model):
        author = models.ForeignKey(
            User,
            on_delete=models.CASCADE,
            related_name='reviews',
        )
        pub_date = models.DateTimeField('Дата публикации', auto_now_add=True)
        score = models.IntegerField(
            blank=True,
            validators=[MaxValueValidator(10), MinValueValidator(1)]
```

Вторым аргументом в MaxValueValidator/MinValueValidator можно прописать человекочитаемое описание ошибки

МОЖНО ЛУЧШЕ Отметить как выполненный Михаил Иванов ревьюер

```
        )
        text = models.TextField()
        title = models.ForeignKey(
            Title,
            on_delete=models.CASCADE,
            null=False,
            related_name='reviews',
```

```
86              default=''
87          )
88
89          class Meta:
                ordering = ['author']
```

Я думаю, тут лучше прописать сортировку по дате публикации. Так логичнее.

МОЖНО ЛУЧШЕ Отметить как выполненный Михаил Иванов ревьюер

```
91              constraints = [
                    models.UniqueConstraint(fields=['title', 'author'],
```

Интересное решение!

ОТЛИЧНО Отметить как выполненный Михаил Иванов ревьюер

```
93                                      name='unique_review_title_author')
94              ]
95
96
97      class Comment(models.Model):
            id = models.AutoField(primary_key=True)
```

А зачем? Модель автоматом создает id. Обычно это поле переопределяют для специфичного id в таблице

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Иванов ревьюер

```
99          author = models.ForeignKey(
100             User,
101             on_delete=models.CASCADE,
102             related_name='comments',
103         )
104
105         review = models.ForeignKey(
106             Review,
107             on_delete=models.SET_NULL,
108             blank=True,
109             null=True,
110             related_name='comments',
111             unique=False
112         )
113         text = models.TextField()
114         pub_date = models.DateTimeField('Дата публикации', auto_now_add=True)
```

```
115
116     class Meta:
117         ordering = ['author']
118
```

py permissions.py 1 1

```
1   from rest_framework import permissions
2   from users_api.models import YamdbUser
3
4
5   class AuthorPermissions(permissions.BasePermission):
6       def has_object_permission(self, request, view, obj):
7           if request.method not in permissions.SAFE_METHODS:
                return request.user.role == YamdbUser.Role.MODERATOR or
request.user == obj.author
```

Строка превышает 79 символов. Стоит оформить перенос по правилам pep8. Стоит проверить и поправить по **всему** проекту

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Иванов ревьюер

```
9           return True
10
11
12  class IsAdminPermissions(permissions.BasePermission):
13
14      def has_object_permission(self, request, view, obj):
            if request.method not in permissions.SAFE_METHODS:
```

Тело метода можно привести к одной строке с помощью оператора or

МОЖНО ЛУЧШЕ Отметить как выполненный Михаил Иванов ревьюер

```
16              return request.user.is_superuser or request.user.role ==
YamdbUser.Role.ADMIN
17          return True
18
```

py serializers.py 1 5

```
1   from django.db.models import Avg
```

```
2  from rest_framework import serializers
3  from rest_framework.generics import get_object_or_404
4
5  from title_api.models import Review, Comment, Title, Category, Genre
6  from rest_framework.validators import UniqueValidator
7
8
9  class ReviewSerializer(serializers.ModelSerializer):
       title = serializers.ReadOnlyField(source='title.name')
```

Тут оптимальнее использовать *RelatedField https://www.django-rest-framework.org/api-guide/relations/#api-reference

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
       author = serializers.ReadOnlyField(source='author.username')
```

*RelatedField

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
12
       def validate(self, data):
```

Правильно - логика валидации должна быть в сериализаторе

ОТЛИЧНО Отметить как выполненный Михаил Ивановревьюер

```
14         request = self.context.get('request')
15         title_id = self.context.get('view').kwargs.get('title_id')
16         title = get_object_or_404(Title, pk=title_id)
17         if request.method != "PATCH" and
   Review.objects.filter(author=request.user, title=title).exists():
18             raise serializers.ValidationError("Validation error. Review
   object with current author and title already "
19                                               "exist!")
20         return data
21
22     class Meta:
23         fields = ('id', 'title', 'text', 'author', 'score', 'pub_date',)
24         model = Review
25
26
27 class CommentSerializer(serializers.ModelSerializer):
       author = serializers.ReadOnlyField(source='author.username')
```

*RelatedField

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
29
30      class Meta:
31          fields = ('id', 'text', 'author', 'pub_date')
32          model = Comment
33
34
35  class CategorySerializer(serializers.ModelSerializer):
36      class Meta:
37          model = Category
            fields = ('name', 'slug')
```

Для исключения одного поля лучше использовать exclude

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
39          lookup_field = 'slug'
40
41
42  class GenreSerializer(serializers.ModelSerializer):
43      class Meta:
44          model = Genre
            fields = ('name', 'slug')
```

exclude

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
46          lookup_field = 'slug'
47
48
49  class TitleViewSerializer(serializers.ModelSerializer):
50      genre = GenreSerializer(many=True, read_only=True)
51      category = CategorySerializer(read_only=True)
52      rating = serializers.FloatField()
53
54      class Meta:
55          fields = '__all__'
56          model = Title
57
58
59  class TitlePostSerializer(serializers.ModelSerializer):
```

```
class TitlePostSerializer(serializers.ModelSerializer):
    genre = serializers.SlugRelatedField(
        queryset=Genre.objects.all(),
        slug_field='slug',
        many=True
    )
    category = serializers.SlugRelatedField(
        queryset=Category.objects.all(),
        slug_field='slug',
    )

    class Meta:
        fields = '__all__'
        model = Title

```

py tests.py 1

```
from django.test import TestCase
```

Давайте удалим неиспользуемые файлы по **всему** проекту

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```


# Create your tests here.

```

py urls.py 1

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from title_api.views import ReviewViewSet, CommentViewSet, TitleViewSet,
CategoryViewSet, GenreViewSet

# Elena, create your urls here.
```

см. замечания про удалению лишних комментариев

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```


```

```python
# Lidia, create your urls here.

router = DefaultRouter()


router.register('titles', TitleViewSet, basename='title')

router.register('categories', CategoryViewSet, basename='category')

router.register('genres', GenreViewSet, basename='genre')

router.register(
    r'titles/(?P<title_id>\d+)/reviews',
    ReviewViewSet, basename='review'
)

router.register(
    r'titles/(?P<title_id>\d+)/reviews/(?P<review_id>\d+)/comments',
    CommentViewSet, basename='comment'
)
urlpatterns = [
    path('v1/', include(router.urls))
]

```

py	views.py	1		8
	users_api	2	2	5
	management	НЕТ КОММЕНТАРИЕВ		
	migrations	НЕТ КОММЕНТАРИЕВ		
py	__init__.py	НЕТ КОММЕНТАРИЕВ		
py	admin.py			1

```python
from django import forms
from django.contrib import admin

```

```
4  from users_api.models import YamdbUser
5
6
   class YamdbUserForm(forms.ModelForm):
```

Лучше не смешивать логику разных сущностей в одном файле - форму можно вынести в forms.py

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Иванов ревьюер

```
8      bio = forms.CharField(widget=forms.Textarea)
9
10     class Meta:
11         model = YamdbUser
12         fields = '__all__'
13
14
15 @admin.register(YamdbUser)
16 class YamdbUserAdmin(admin.ModelAdmin):
17     form = YamdbUserForm
18     list_display = ('username', 'pk', 'email', 'role',)
19
```

py apps.py НЕТ КОММЕНТАРИЕВ

py models.py 2 2 2

```
1  from uuid import uuid1
2
3  from django.contrib.auth.models import AbstractUser, BaseUserManager
4  from django.contrib.auth.tokens import default_token_generator
5  from django.core.mail import EmailMessage
6  from django.db import models
7  from django.db.models import signals
8  from django.dispatch import receiver
9
10
11 class YamdbUserManager(BaseUserManager):
12     def create_superuser(self, username=None, password=None, email=None):
13         user_obj = self.create_user(
14             username=str(uuid1()),
15             email=email,
16             password=password,
17             is_staff=True,
```

```
            is_admin=True,
            is_active=True,
        )

        return user_obj

    def create_user(self, email,
                    username,
                    password=None,
                    is_active=True,
                    is_staff=False,
                    is_admin=False,
                    ):
        if not email:
            raise ValueError("User must have an email address")
        user_obj = self.model(
            email=email
        )
        user_obj.set_password(password)
        user_obj.username = username
        user_obj.email = email
        user_obj.is_staff = is_staff
        user_obj.is_superuser = is_admin
        user_obj.is_active = is_active
        user_obj.save(using=self._db)
        return user_obj


class YamdbUser(AbstractUser):
```

Еще круто было бы добавить проперти(@property) для вычисления ролей. Тогда в коде можно будет просто вызывать их как методы, не сравнивая с константами и вся логика проверки будет сосредоточена в классе юзера.

МОЖНО ЛУЧШЕ Отметить как выполненный Михаил Иванов ревьюер

```
    class Role(models.TextChoices):
```

Отличное решение!

ОТЛИЧНО Отметить как выполненный Михаил Иванов ревьюер

```
        USER = 'user'
        MODERATOR = 'moderator'
        ADMIN = 'admin'

    bio = models.CharField(max_length=255, blank=True)
    role = models.CharField(max_length=10, default=Role.USER
```

```
    role = models.CharField(max_length=10, default=Role.USER,
choices=Role.choices)
```

Лучше сразу заложить кол-во символов с запасом - вдруг появится роль длиннее 10 символов?

МОЖНО ЛУЧШЕ Отметить как выполненный Михаил Ивановревьюер

```
53      email = models.EmailField(unique=True, db_index=True, blank=False,
null=False)
        username = models.CharField(unique=False, max_length=31)
```

Почему переопределено стандартное поле из AbstractUser?

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
55      USERNAME_FIELD = 'email'
56      REQUIRED_FIELDS = []
57
58      objects = YamdbUserManager()
59
60      class Meta:
61          ordering = ('-id',)
62
63
64  @receiver(signals.post_save, sender=YamdbUser)
65  def send_code(sender, instance, created, **kwargs):
        code = default_token_generator.make_token(instance)
```

Хорошее решение. Если можно генерировать из коробки, почему бы и нет?

ОТЛИЧНО Отметить как выполненный Михаил Ивановревьюер

```
67      if created:
68          email = EmailMessage(
69          'Confirmation code',
70          f'Your confirmation code: {code}',
            'from@example.com',
```

Емейл админа стоит вынести в settings.py, как и все настройки проекта

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Ивановревьюер

```
72          [f'{instance.email}', ],
73          )
```

```
        email.send()
```

py permissions.py НЕТ КОММЕНТАРИЕВ

py serializers.py НЕТ КОММЕНТАРИЕВ

py tests.py НЕТ КОММЕНТАРИЕВ

py urls.py НЕТ КОММЕНТАРИЕВ

py views.py 2

```python
from uuid import uuid1

from django.contrib.auth.hashers import make_password
from django.contrib.auth.tokens import default_token_generator
from django.shortcuts import get_object_or_404
from rest_framework import generics
from rest_framework import mixins
from rest_framework import status
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated, AllowAny
from rest_framework.response import Response
from rest_framework_simplejwt.tokens import AccessToken

from users_api.models import YamdbUser
from users_api.permissions import IsYamdbAdmin
from users_api.serializers import UserSerializer, MeSerializer, \
    EmailRegistrationSerializer, UserVerificationSerializer


class CreateUser(generics.CreateAPIView):
    """
    Create user with POST request with email parameter.
    Wait for email confirmation code.
    """
    permission_classes = (AllowAny, )
    serializer_class = EmailRegistrationSerializer

    def perform_create(self, serializer):
```

```
29          serializer.save(
30              is_active=False,
31              password=make_password(None),
32              username=str(uuid1()),
33          )
34
35
36  class ConfirmUser(generics.UpdateAPIView):
37      """
38      Activate your user with POST request included email
39      and confirmation_code params
40      """
41      serializer_class = UserVerificationSerializer
42      permission_classes = (AllowAny, )
43      http_method_names = ['post', ]
44
45      def get_object(self):
            return YamdbUser.objects.get(email=self.request.data.get('email'))
```

1) get() - это потенциально необработанное исключение, если такого объекта не найдется или найдется несколько. Для таких случаев из коробки доступен шорткат https://docs.djangoproject.com/en/3.0/topics/http/shortcuts/#get-object-or-404 который или вернет объект или обработает исключение и ответит 404 статусом
2) А как же валидация сериализатора? В этом же его суть :)

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Иванов ревьюер

```
47
48      def post(self, request, *args, **kwargs):
49
50          token = request.data.get('confirmation_code')
51          user = self.get_object()
52          check = default_token_generator.check_token(user, token)
53          if not check:
54              return Response(
55                  {'Error': 'Confirmation code for this email is wrong'},
56                  status=status.HTTP_401_UNAUTHORIZED,
57                  content_type='application/json',
58              )
59
60          request.data._mutable = True
61          request.data['is_active'] = True
62          request.data._mutable = False
63
64          serializer = self.get_serializer(user, data=request.data,
    partial=True)
65          serializer.is_valid(raise_exception=True)
66          self.perform_update(serializer)
```

```
67
68          token = AccessToken.for_user(user)
69          return Response(
70              data=token.payload,
71              status=status.HTTP_202_ACCEPTED,
72              content_type='application/json',
73          )
74
75
76  class UsersViewSet(viewsets.ViewSetMixin,
77                     mixins.RetrieveModelMixin,
78                     mixins.CreateModelMixin,
79                     mixins.ListModelMixin,
80                     mixins.UpdateModelMixin,
81                     mixins.DestroyModelMixin,
82                     generics.GenericAPIView,):
83      """
84      Users List (for admin only), Send PATCH request to /api/v1/users/me/
85      for editing your user information.
86      """
87      queryset = YamdbUser.objects.all()
88      serializer_class = UserSerializer
89      lookup_field = 'username'
90      permission_classes = (IsYamdbAdmin, )
91
92
    class UserSelf(
```

Получение профиля должно быть в UserViewSet отдельным экшеном https://www.django-rest-framework.org/api-guide/viewsets/#marking-extra-actions-for-routing

НАДО ИСПРАВИТЬ Отметить как выполненный Михаил Иванов ревьюер

```
94      mixins.RetrieveModelMixin,
95      mixins.UpdateModelMixin,
96      generics.GenericAPIView,
97  ):
98      serializer_class = MeSerializer
99      permission_classes = (IsAuthenticated,)
100     http_method_names = ['get', 'patch']
101     queryset = YamdbUser.objects.all()
102
103     def get(self, request, *args, **kwargs):
104         return self.retrieve(request, *args, **kwargs)
105
106     def put(self, request, *args, **kwargs):
107         return self.update(request, *args, **kwargs)
```

```
108
109     def patch(self, request, *args, **kwargs):
110         return self.update(request, *args, **kwargs)
111
112     def get_object(self):
113         return get_object_or_404(YamdbUser,
    username=self.request.user.username)
114
```

md	README.md	НЕТ КОММЕНТАРИЕВ
py	manage.py	НЕТ КОММЕНТАРИЕВ
txt	old_requirements.txt	НЕТ КОММЕНТАРИЕВ
txt	requirements.txt	НЕТ КОММЕНТАРИЕВ