WAFFLE STUDIO

Model Relationships, Authentication, Serialization

Instructors:
김찬욱 @narayo9

2022.10.04 (화) 19:30

과제 1 리뷰

수업 피드백 반영

많이, 솔직하게, 작성해주세요!

Table of contents

Model Relationships

User authentication and permissions

Serializer customization

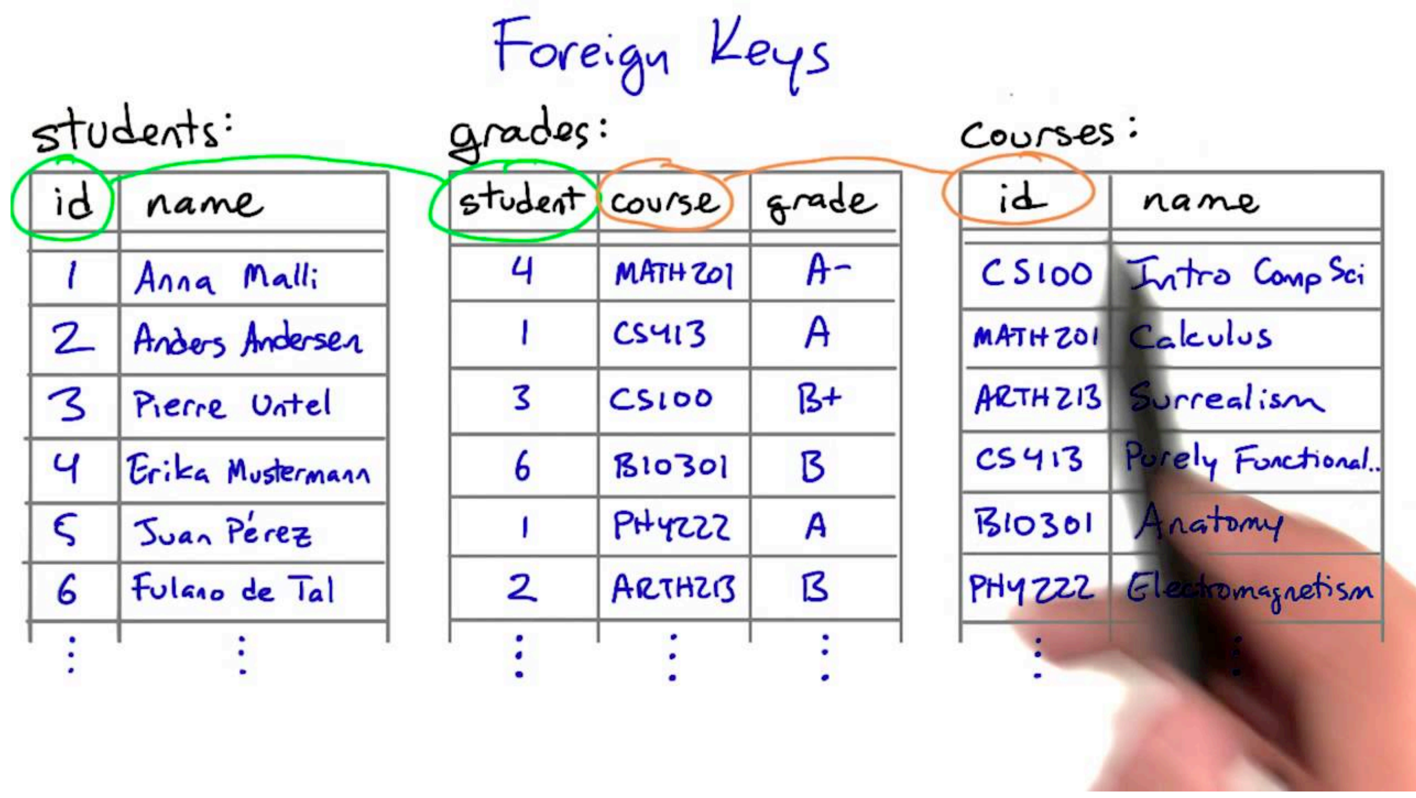
Project 1 설명

1. Model Relationships

Django models - relationships~

RDB

the power of relational databases lies in <u>relating</u> tables to each other.


Foreign Keys

students:

id	name
1	Anna Malli
2	Anders Andersen
3	Pierre Untel
4	Erika Mustermann
5	Juan Pérez
6	Fulano de Tal
⋮	⋮

grades:

student	course	grade
4	MATH201	A-
1	CS413	A
3	CS100	B+
6	BIO301	B
1	PHY222	A
2	ARTH213	B
⋮	⋮	⋮

courses:

id	name
CS100	Intro Comp Sci
MATH201	Calculus
ARTH213	Surrealism
CS413	Purely Functional..
BIO301	Anatomy
PHY222	Electromagnetism
⋮	⋮

Many-to-one relationships

To define a many-to-one relationship, use **django.db.models.ForeignKey**.

```python
from django.db import models

class Manufacturer(models.Model):
    # ...
    pass

class Car(models.Model):
    manufacturer = models.ForeignKey(Manufacturer, on_delete=models.CASCADE)
    # ...
```

on_delete?

링크

When an object referenced by a ForeignKey is deleted, Django will emulate the behavior of the SQL constraint specified by the on_delete argument.

Cascade: emulates the behavior of the SQL constraint ON DELETE CASCADE and also deletes the object containing the ForeignKey.

실습: Tesla

python manage.py startapp tesla

Modeling

Migrate and check ddl

Insert and remove!

실습: Tesla

```
root@efa523605d5e:/code# python manage.py shell
Python 3.8.14 (default, Sep 13 2022, 16:41:27)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from tesla.models import Manufacturer, Car
>>> tesla = Manufacturer.objects.create(title="tesla")
>>> cyber_truck = Car.objects.create(manufacturer=tesla,
title="cyber_truck")
>>> tesla.delete()
(2, {'tesla.Car': 1, 'tesla.Manufacturer': 1})
>>>
now exiting InteractiveConsole...
```

Many-to-many relationships

To define a many-to-many relationship, use **ManyToManyField**.

```
from django.db import models

class Tag(models.Model):
    # ...
    pass

class Car(models.Model):
    # ...
    tags = models.ManyToManyField(Tag)
```

실습: Tesla

Add Tesla tag to models.py

Add column and migrate

Check databases

How ManyToManyField works

```
class Car(models.Model):
    manufacturer = models.ForeignKey(Manufacturer, on_delete=models.CASCADE)
    title = models.CharField(max_length=100)


class Tag(models.Model):
    content = models.CharField(max_length=100)


class TagToCar(models.Model):
    tag = models.ForeignKey(Tag, on_delete=models.CASCADE)
    car = models.ForeignKey(Car, on_delete=models.CASCADE)
```

그냥 양쪽 ForeignKey 추가하는 것과 같음

ManyToManyField: through model

```
class Car(models.Model):
    manufacturer = models.ForeignKey(Manufacturer, on_delete=models.CASCADE)
    title = models.CharField(max_length=100)
    tags = models.ManyToManyField('Tag', through="TagToCar")


class Tag(models.Model):
    content = models.CharField(max_length=100)


class TagToCar(models.Model):
    tag = models.ForeignKey(Tag, on_delete=models.CASCADE)
    car = models.ForeignKey(Car, on_delete=models.CASCADE)
```

Tag에서 직접 접속 가능,
다양한 ManyToManyField methods 추가

ManyToManyField: through model

```
class Car(models.Model):
    manufacturer = models.ForeignKey(Manufacturer, on_delete=models.CASCADE)
    title = models.CharField(max_length=100)
    tags = models.ManyToManyField('Tag', through="TagToCar")


class Tag(models.Model):
    content = models.CharField(max_length=100)


class TagToCar(models.Model):
    tag = models.ForeignKey(Tag, on_delete=models.CASCADE)
    car = models.ForeignKey(Car, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Through model 활용 가능

OneToOne Relationships

```
class Car(models.Model):
    manufacturer = models.ForeignKey(Manufacturer, on_delete=models.CASCADE)
    title = models.CharField(max_length=100)
    tags = models.ManyToManyField('Tag', through="TagToCar")


class CarInfo(models.Model):
    car = models.OneToOneField(Car, on_delete=models.PROTECT)
    mass = models.IntegerField()
```

Model을 다른 테이블로 확장하는 역할이지만...
꼭 필요할지 고민해보는 것을 추천

읽을거리: 다루지 못한 Model 꿀정보

Model Methods: Django 내부에서 있을 것이라 가정하고 쓰는 메소드. 알아두면 좋음

Model Inheritance: SNS의 피드를 생각해보자. 같은 스키마일 수 있을까?

2. User authentication and permissions

Django: User authentication in django
DRF: Authentication / Permission

Authentication?

Authentication is the mechanism of associating an incoming request with a set of identifying credentials, such as the user the request came from, or the token that it was signed with.

The permission and throttling policies can then use those credentials to determine if the request should be permitted.

User authentication system

Django comes with a user authentication system.

Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
Groups: A generic way of applying labels and permissions to more than one user.
A configurable **password hashing system**

…

실습: Django user 살펴보기

django.contrib.auth.models.User
(Find symbol)

Db table 살펴보기 (auth_~)

실습: Django user 살펴보기

```
root@efa523605d5e:/code# python manage.py shell
Python 3.8.14 (default, Sep 13 2022, 16:41:27)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from django.contrib.auth.models import User
>>> me = User.objects.create_user('chanuk', 'cksdnr987@gmail.com', 'password')
>>> # DB password 확인
>>>
>>> me.set_password('newPassword')
>>> me.save()
>>>
```

Authentication in web requests

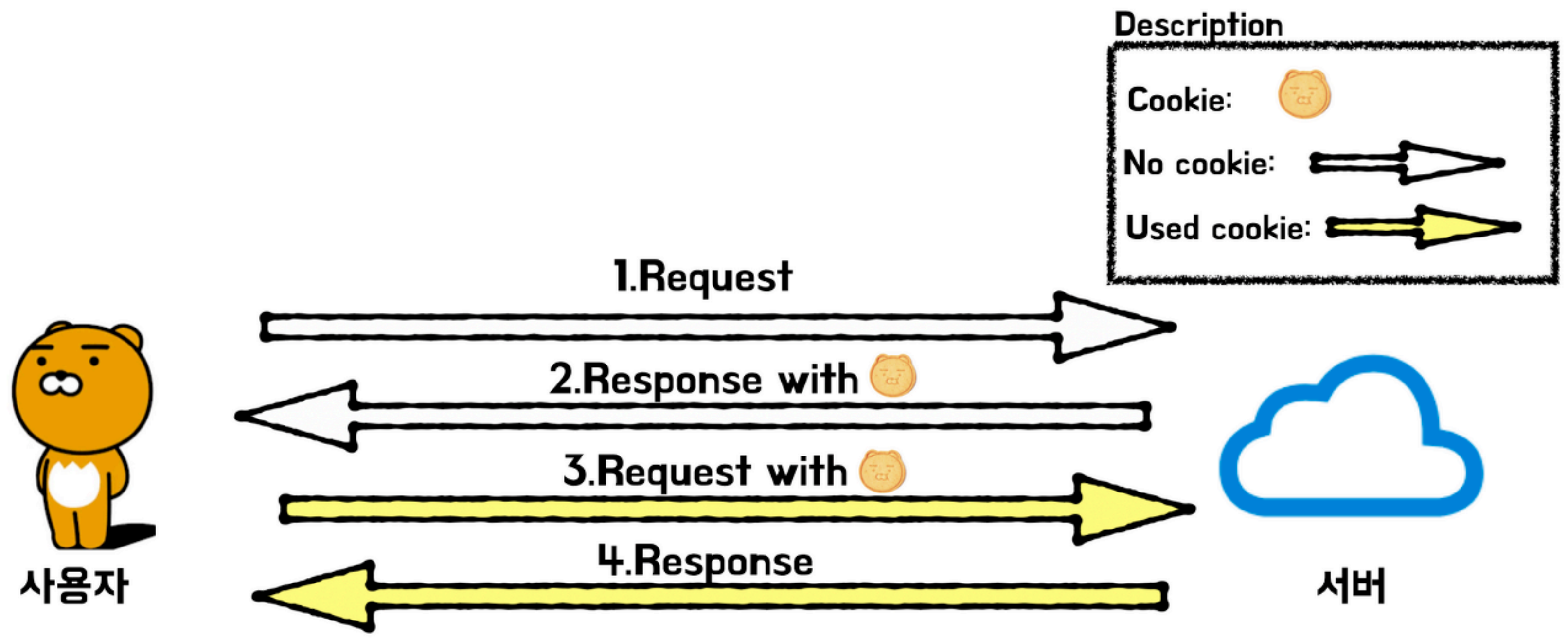
Django uses sessions and middleware to hook the authentication system into request objects.

Django 기본임

Try request.user!

읽을거리: What is session?

잘 설명된 한글문서



WAFFLE STUDIO

실습: check request.user

Completed 폴더에서 urls, serializers, views 복사하기

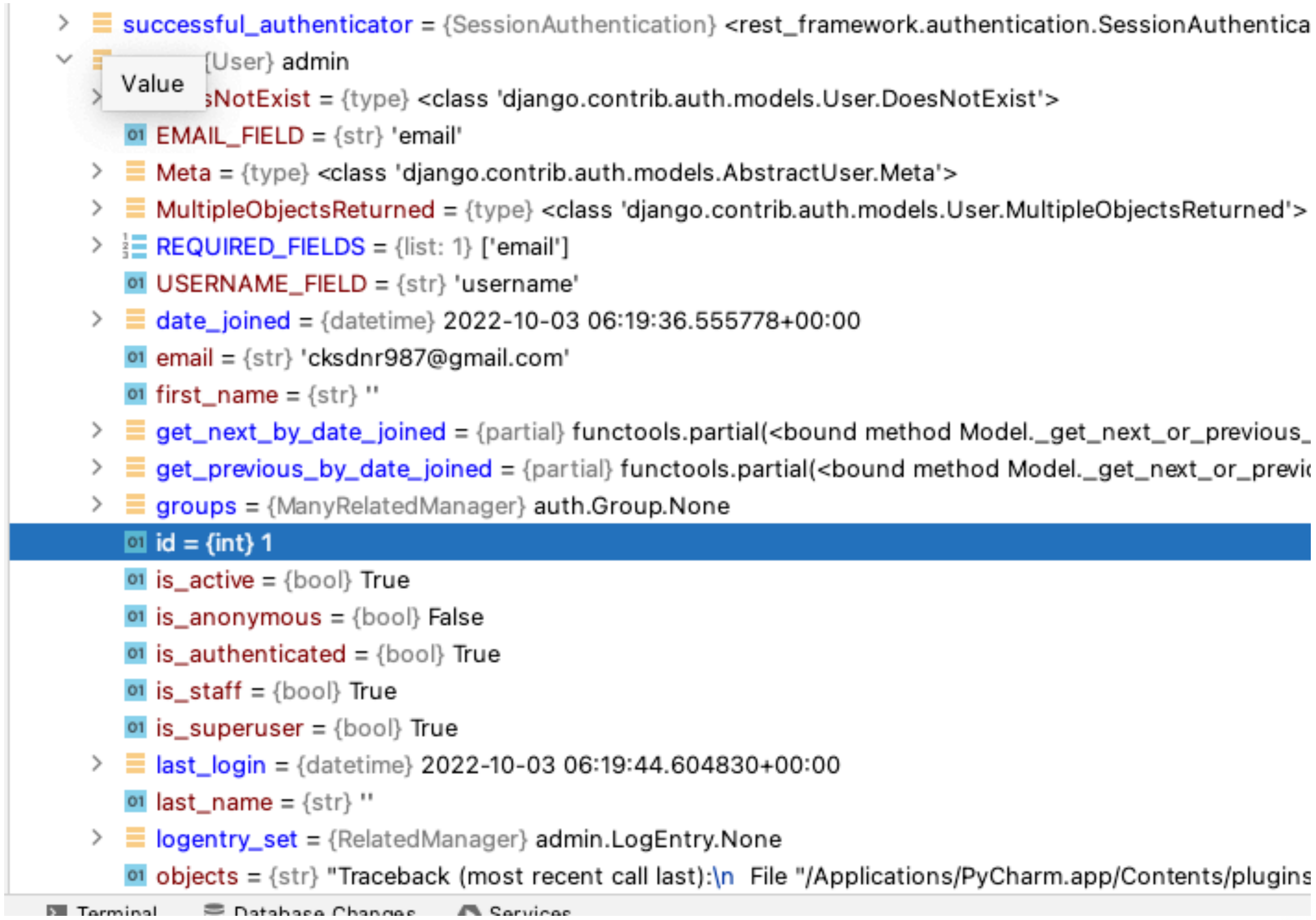
Stop 후 debug로 run

debug point on

Check request.user after login!

```
python manage.py createsuperuser
```

실습: check request.user



```
> successful_authenticator = {SessionAuthentication} <rest_framework.authentication.SessionAuthentica
v          {User} admin
  Value
  >      sNotExist = {type} <class 'django.contrib.auth.models.User.DoesNotExist'>
     EMAIL_FIELD = {str} 'email'
  >  Meta = {type} <class 'django.contrib.auth.models.AbstractUser.Meta'>
  >  MultipleObjectsReturned = {type} <class 'django.contrib.auth.models.User.MultipleObjectsReturned'>
  >  REQUIRED_FIELDS = {list: 1} ['email']
     USERNAME_FIELD = {str} 'username'
  >  date_joined = {datetime} 2022-10-03 06:19:36.555778+00:00
     email = {str} 'cksdnr987@gmail.com'
     first_name = {str} ''
  >  get_next_by_date_joined = {partial} functools.partial(<bound method Model._get_next_or_previous_
  >  get_previous_by_date_joined = {partial} functools.partial(<bound method Model._get_next_or_previc
  >  groups = {ManyRelatedManager} auth.Group.None
     id = {int} 1
     is_active = {bool} True
     is_anonymous = {bool} False
     is_authenticated = {bool} True
     is_staff = {bool} True
     is_superuser = {bool} True
  >  last_login = {datetime} 2022-10-03 06:19:44.604830+00:00
     last_name = {str} ''
  >  logentry_set = {RelatedManager} admin.LogEntry.None
     objects = {str} "Traceback (most recent call last):\n  File "/Applications/PyCharm.app/Contents/plugins
```

읽을거리: Permissions and Authorization

링크

Django 자체적으로 DB 테이블에 Permission을 추가/제거할 수 있도록 기능 제공

auth_user_user_permissions / auth_group_permissions 테이블 참고

다만, 우리는 DRF가 제공해주는 Permission 기능이 더 유용할 것

```
from myapp.models import BlogPost
from django.contrib.auth.models import Permission
from django.contrib.contenttypes.models import ContentType

content_type = ContentType.objects.get_for_model(BlogPost)
permission = Permission.objects.create(
    codename='can_publish',
    name='Can Publish Posts',
    content_type=content_type,
)
```

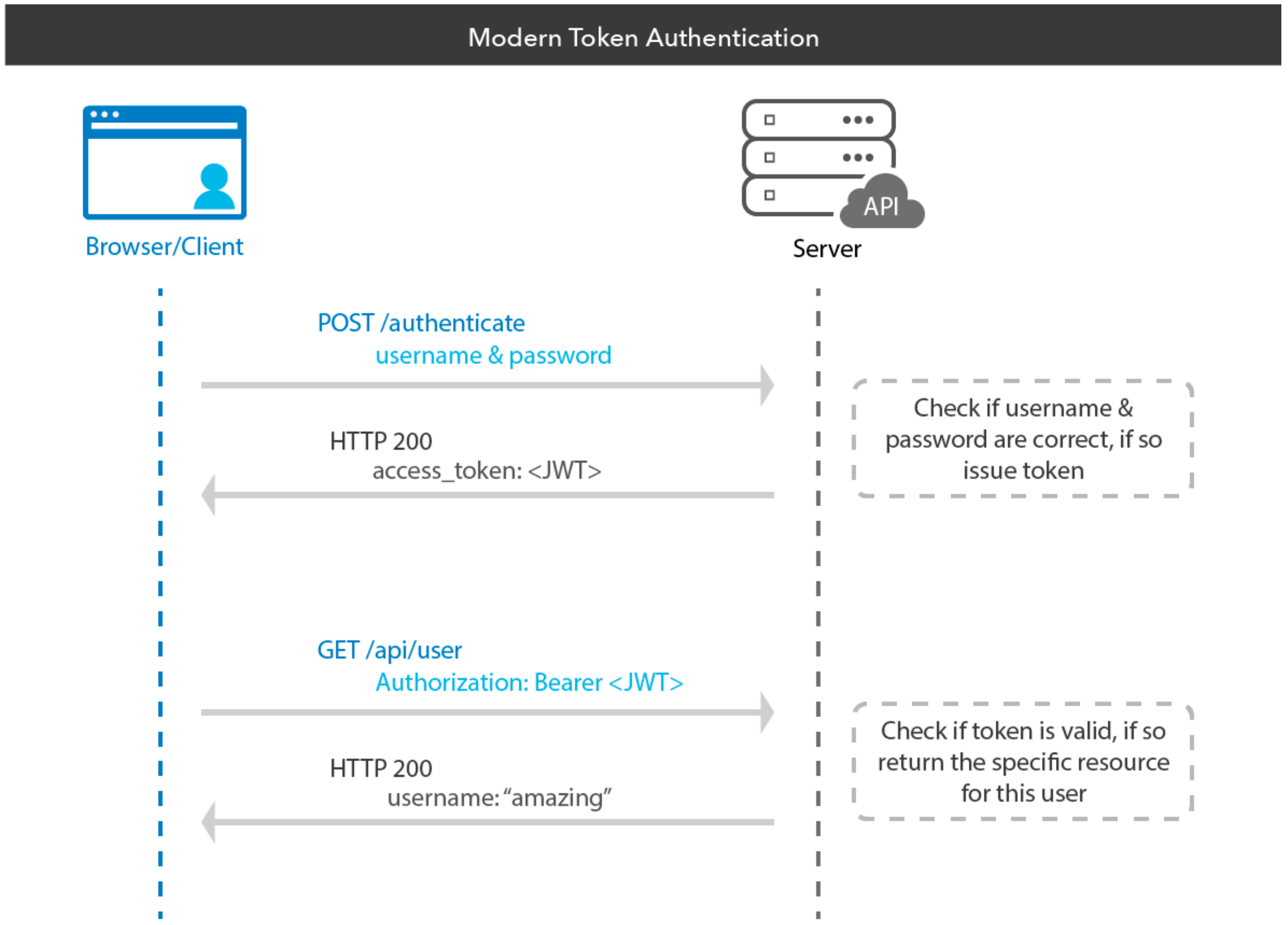
DRF Authentication methods

BasicAuthentication: generally only appropriate for testing.

SessionAuthentication: Django default, but only usable on web browser (not mobile app)

We will learn **TokenAuthentication**

Token Authentication


Modern Token Authentication
Browser/Client
API
Server
POST /authenticate
username & password
HTTP 200
access_token: <JWT>
Check if username & password are correct, if so issue token
GET /api/user
Authorization: Bearer <JWT>
HTTP 200
username: "amazing"
Check if token is valid, if so return the specific resource for this user

실습: Token Authentication 적용

링크

token.key까지~

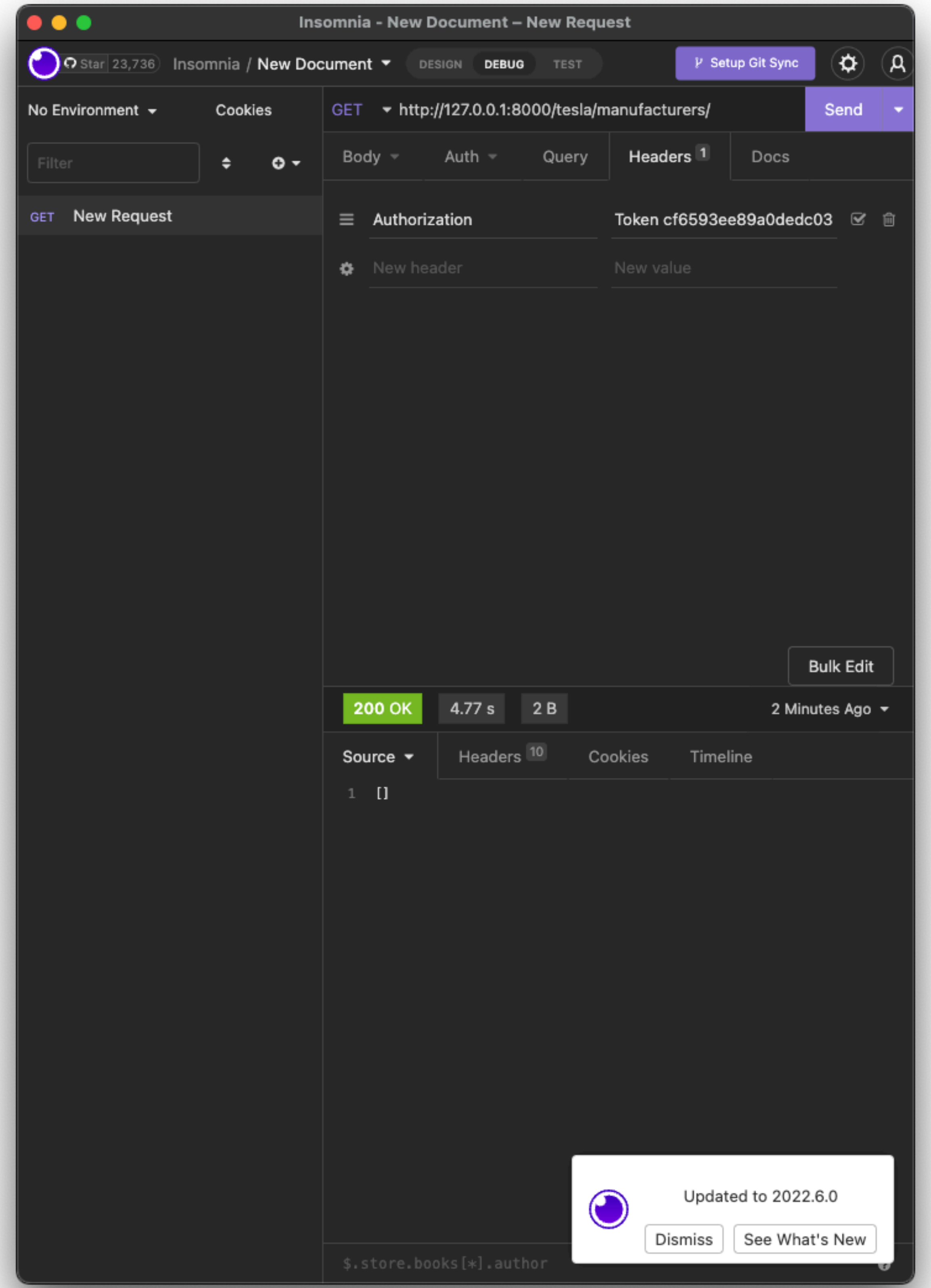
insomnia로 Token Authentication test

settings에 rest_framework.authentication.TokenAuthentication 추가

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.TokenAuthentication',
    ]
}
```

실습: Token Authentication 적용

```
Python 3.8.9 (default, May 17 2022,
12:55:41)
[Clang 13.1.6 (clang-1316.0.21.2.5)] on
darwin
Type "help", "copyright", "credits" or
"license" for more information.
(InteractiveConsole)
>>> from django.contrib.auth.models import
User
>>> from rest_framework.authtoken.models
import Token
>>> token =
Token.objects.create(user=User.objects.get(i
d=1))
>>> token.key
'cf6593ee89a0dedc03e7ce3f8c27de39c60449d8'
>>>
```


Insomnia - New Document – New Request

GET http://127.0.0.1:8000/tesla/manufacturers/

Authorization: Token cf6593ee89a0dedc03

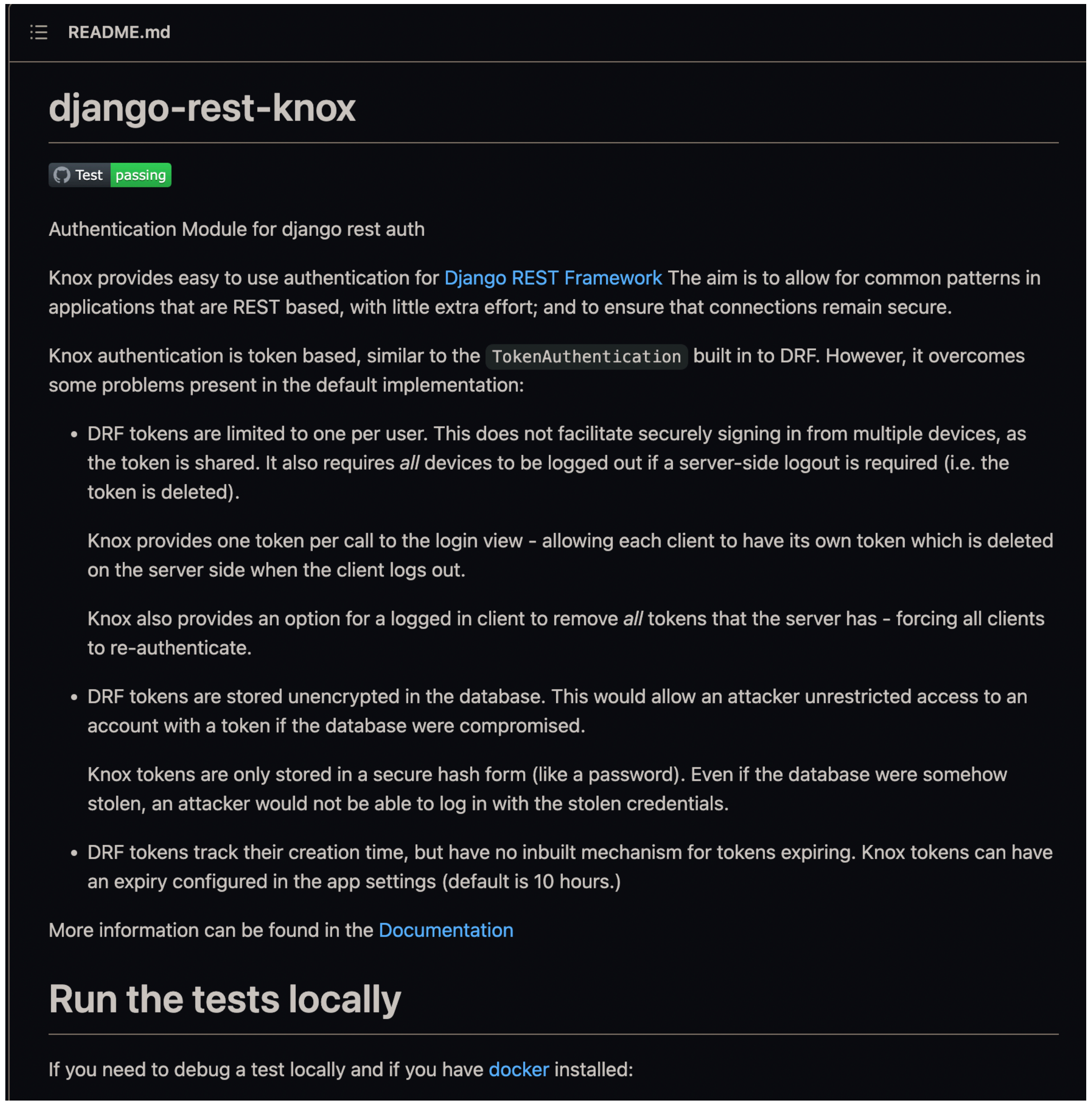
200 OK 4.77 s 2 B 2 Minutes Ago

```
[]
```

Updated to 2022.6.0

Token Authentication... more

Production 환경에서 token expire, allow more than one token per user 등을 구현하기 위해서는 django-rest-knox를 사용하라고 합니다.


README.md

django-rest-knox

Authentication Module for django rest auth

Knox provides easy to use authentication for Django REST Framework The aim is to allow for common patterns in applications that are REST based, with little extra effort; and to ensure that connections remain secure.

Knox authentication is token based, similar to the `TokenAuthentication` built in to DRF. However, it overcomes some problems present in the default implementation:

- DRF tokens are limited to one per user. This does not facilitate securely signing in from multiple devices, as the token is shared. It also requires *all* devices to be logged out if a server-side logout is required (i.e. the token is deleted).

  Knox provides one token per call to the login view - allowing each client to have its own token which is deleted on the server side when the client logs out.

  Knox also provides an option for a logged in client to remove *all* tokens that the server has - forcing all clients to re-authenticate.

- DRF tokens are stored unencrypted in the database. This would allow an attacker unrestricted access to an account with a token if the database were compromised.

  Knox tokens are only stored in a secure hash form (like a password). Even if the database were somehow stolen, an attacker would not be able to log in with the stolen credentials.

- DRF tokens track their creation time, but have no inbuilt mechanism for tokens expiring. Knox tokens can have an expiry configured in the app settings (default is 10 hours.)

More information can be found in the Documentation

Run the tests locally

If you need to debug a test locally and if you have docker installed:

DRF Permission classes

```python
class ManufacturerPermissionListCreateView(generics.ListCreateAPIView):
    permission_classes = [IsAdminUser]
    queryset = Manufacturer.objects.all()
    serializer_class = ManufacturerSerializer

    def get(self, request, *args, **kwargs):
        print(request.user)
        return super().get(request, *args, **kwargs)
```

해당 view에 class based permission 적용

실습: DRF Permission classes

```python
class ManufacturerPermissionListCreateView(generics.ListCreateAPIView):
    permission_classes = [IsAdminUser]
    queryset = Manufacturer.objects.all()
    serializer_class = ManufacturerSerializer

    def get(self, request, *args, **kwargs):
        print(request.user)
        return super().get(request, *args, **kwargs)
```

AllowAny, IsAuthenticated, IsAdminUser 적용해보기
Response code?

401 / 403, Exceptions

Before running the main body of the view each permission in the list is checked. If any permission check fails, an **exceptions.PermissionDenied** or **exceptions.NotAuthenticated** exception will be raised, and the main body of the view will not run.

Custom permissions

Manufacturer를 만든 사람만 해당 요청에 성공하게 하고 싶다!

```python
from rest_framework import permissions

class CustomerAccessPermission(permissions.BasePermission):
    message = 'Adding customers not allowed.'

    def has_permission(self, request, view):
        ...
```

WA LE
STUDIO

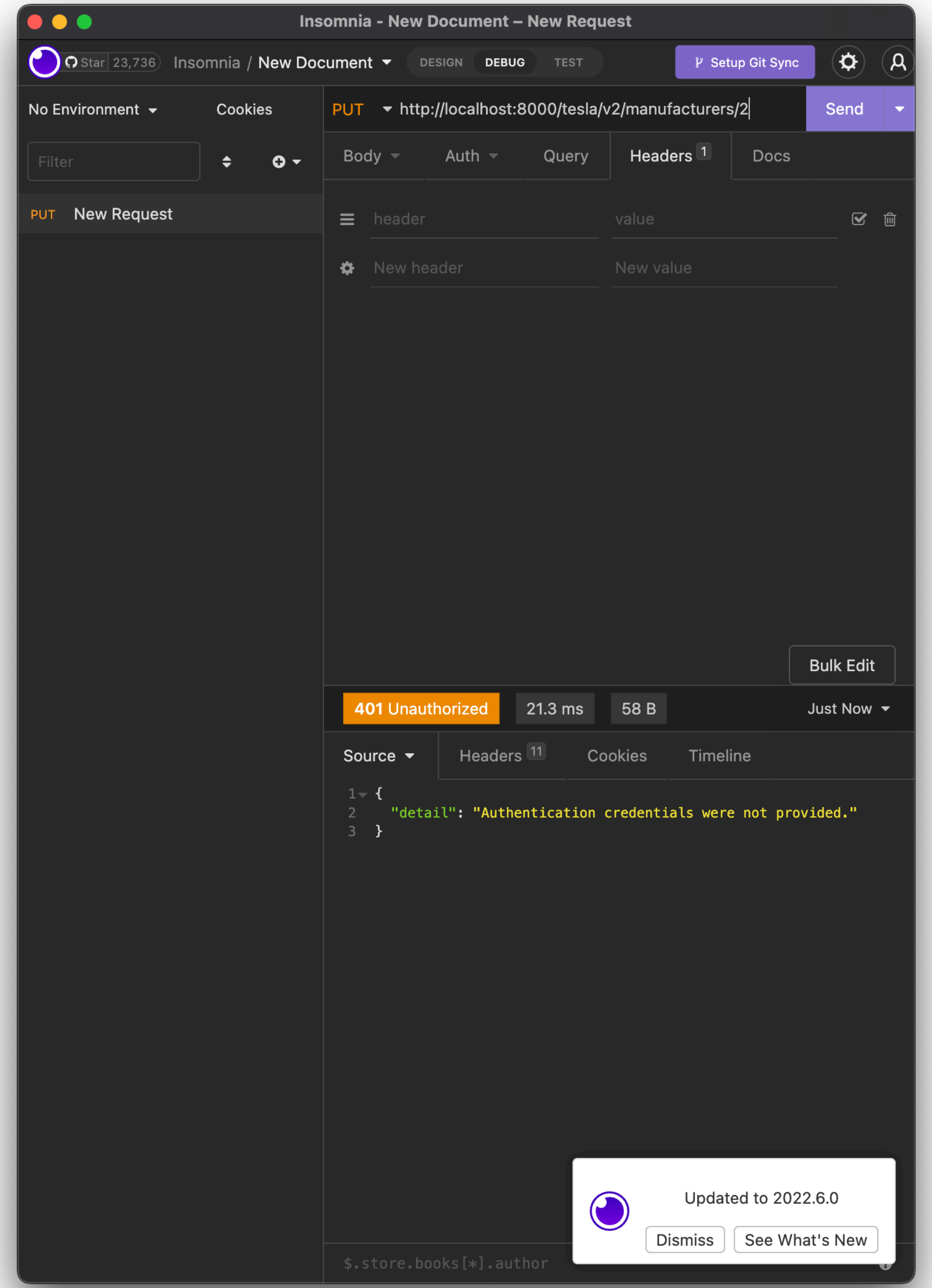
실습: Custom permissions

manufacturer에 created_by 추가

Manufacturer의 save시 request.user 포함되도록 (to_internal_value)

Custom Permission 추가

ListCreateView의 not SAFE_METHOD로 추가



PUT http://localhost:8000/tesla/v2/manufacturers/2

401 Unauthorized 21.3 ms 58 B

```
{
  "detail": "Authentication credentials were not provided."
}
```

읽을거리: Custom User Model

[링크]

Custom User Model을 결국 만드는 게 깔끔할 거에요.

3. Custom serializers

DRF: Serializers...
그러나 cmd+click이 더 나을수도?

Base Serializers overriding

.to_representation() - Override this to support serialization, for read operations.

.to_internal_value() - Override this to support deserialization, for write operations.

.create() and .update() - Override either or both of these to support saving instances.

Case 1: JSON에 속해있지 않는 정보를 저장

```
class ManufacturerSerializer(serializers.ModelSerializer):
    def to_internal_value(self, data):
        internal_value = super().to_internal_value(data)
        return {**internal_value, 'created_by':
self.context['request'].user}

    class Meta:
        model = Manufacturer
        fields = ['title', 'id']
```

created_by에 request.user 저장?

def get_serializer_context():

def to_internal_value() override!

Case 2: 저장시와 표현시의 데이터가 다름

```
class ManufacturerSerializer(serializers.ModelSerializer):
    id = serializers.PrimaryKeyRelatedField(read_only=True)

    def to_internal_value(self, data):
        internal_value = super().to_internal_value(data)
        return {**internal_value, 'created_by':
self.context['request'].user}

    class Meta:
        model = Manufacturer
        fields = ['title', 'id']
```

Case 3: List와 Detail시의 Serializer가 다름

```python
class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'email']


class ManufacturerListSerializer(serializers.ModelSerializer):
    id = serializers.PrimaryKeyRelatedField(read_only=True)

    def to_internal_value(self, data):
        internal_value = super().to_internal_value(data)
        return {**internal_value, 'created_by': self.context['request'].user}

    class Meta:
        model = Manufacturer
        fields = ['title', 'id', 'created_by']


class ManufacturerDetailSerializer(serializers.ModelSerializer):
    id = serializers.PrimaryKeyRelatedField(read_only=True)
    created_by = UserSerializer(read_only=True)

    def to_internal_value(self, data):
        internal_value = super().to_internal_value(data)
        return {**internal_value, 'created_by': self.context['request'].user}

    class Meta:
        model = Manufacturer
        fields = ['title', 'id', 'created_by']
```

Case 3: List와 Detail시의 Serializer가 다름

```
class ManufacturerRetrieveUpdateDestroyView(generics.RetrieveUpdateDestroyAPIView):
    permission_classes = [IsManufacturerCreator]
    queryset = Manufacturer.objects.all()

    def get_serializer_class(self):
        if self.request.method == 'GET':
            return ManufacturerDetailSerializer
        return ManufacturerListSerializer
```

Django의 난제

Q. 깔끔하게 override할 수 있는 method를 어떻게 알 수 있나요?

A. 내부 구현을 잘 살펴보고, 바뀌어야 할 부분을 찾아보세요.
의도를 파악하고 제일 작은 부분을 바꾸려고 노력해보세요.

원래 어떤건지? 가 궁금할 때는 super() 정의한다음 cmd +클릭해보세요.

```
class ManufacturerListSerializer(serializers.ModelSerializer):
    id = serializers.PrimaryKeyRelatedField(read_only=True)

    def to_internal_value(self, data):
        internal_value = super().to_internal_value(data)
        return {**internal_value, 'created_by': self.context['request'].user}

    class Meta:
        model = Manufacturer
        fields = ['title', 'id', 'created_by']
```

4. Project 1: The large blog service

seminar-2022-django-assignment2

Q&A