بسم الله الرحمن الرحيم

واجب

اسم الطالب : ريان امين سيف عبد القوي
اسم المقرر:هندسة برمجيات
اسم الأستاذ : مالك المصنف
ايميل : riancomp.ye@gmail.com

بحث شامل عن آليات المصادقة في Django: مقارنة بين GWT، Sessions، وToken Authentication

جدول المحتويات

1. مقدمة عامة عن المصادقة فيDjango
2. Sessions Authentication
3. Token Authentication
4. JWT Authentication
5. مقارنة شاملة بين آليات المصادقة
6. أمثلة عملية وتطبيقات
7. توصيات واختيار الآلية المناسبة
8. الخاتمة

مقدمة عامة عن المصادقة في Django <a name="مقدمة-عامة"></a>

المصادقة (Authentication) هي عملية التحقق من هوية المستخدم، وهي عنصر أساسي في أي تطبيق ويب حديث. تقدم Django نظام مصادقة قويًا ومتكاملاً يسهل تخصيصه وفقًا لاحتياجات التطبيق.

يأتي Django مزودًا بنظام مصادقة مدمج يدعم:

- إدارة المستخدمين والتسجيل
- الجلسات(Sessions)
- صلاحيات المستخدمين(Permissions)
- مصادقة القوالب

تعتبر آلية المصادقة المناسبة أمرًا حاسمًا لأمان التطبيق وأدائه وتجربة المستخدم.

Sessions Authentication <a name="sessions-authentication"></a>

آلية العمل

تعتمد مصادقة الجلسات على تخزين حالة المستخدم على الخادم:

1. يقوم المستخدم بتسجيل الدخول بإرسال بيانات الاعتماد
2. يتحقق الخادم من صحة البيانات وينشئ جلسة
3. يتم تخزين معرف الجلسة في cookie على جهاز العميل
4. مع كل طلب لاحق، يرسل المتصفح cookie الجلسة تلقائيًا
5. يتحقق الخادم من صحة الجلسة ويحدد هوية المستخدم

المميزات

- **مدمجة في Django**: لا تحتاج إلى إعدادات إضافية
- **آمنة**: البيانات الحساسة مخزنة على الخادم فقط
- **دعم إدارة الجلسات**: يمكن إدارة الجلسات النشطة وإلغاؤها
- **مناسبة للتطبيقات التقليدية**: تعمل بشكل ممتاز مع التطبيقات التي تعتمد على القوالب

القيود

- **عدم قابلية التوسع**: تخزين الجلسات يستهلك ذاكرة الخادم
- **مشاكل مع APIs**: غير مناسبة للتطبيقات منفصلة الواجهة(Frontend)
- **مشاكل مع CORS**: صعوبة في التعامل مع نطاقات متعددة

مثال عملي

python

```python
# settings.py
INSTALLED_APPS = [
    'django.contrib.auth',
    'django.contrib.sessions',
    # ...
]

MIDDLEWARE = [
    'django.contrib.sessions.middleware.SessionMiddleware',
```

```
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    # ...
]

# views.py
from django.contrib.auth import authenticate, login
from django.http import JsonResponse

def login_view(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)
            return JsonResponse({'status': 'success'})
        else:
            return JsonResponse({'status': 'error', 'message': 'Invalid crede
ntials'})
```

Token Authentication <a name="token-authentication"></a>

آلية العمل

تعتمد مصادقة الرمز (Token) على tokens ثابتة:

1. يقوم المستخدم بتسجيل الدخول بإرسال بيانات الاعتماد
2. يتحقق الخادم من صحة البيانات وينشئ token فريد
3. يتم إرجاع token إلى العميل الذي يقوم بتخزينه
4. مع كل طلب لاحق، يرسل العميل token في رأس الطلب
5. يتحقق الخادم من صحة token ويحدد هوية المستخدم

المميزات

- **عديمة الحالة (Stateless)**: لا تخزن بيانات على الخادم
- **مناسبة لل APIs**: تعمل بشكل ممتاز مع التطبيقات منفصلة الواجهة
- **قابلة للتوسع**: لا تستهلك ذاكرة الخادم لتخزين الحالة

- **دعم متعدد المنصات** :مناسبة للجوال، سطح المكتب، والويب

القيود

- **عدم إمكانية الإلغاء** Tokens :صالحة حتى انتهاء صلاحيتها
- **مخاطر أمنية** :إذا سُرقtoken ، يمكن استخدامه حتى انتهاء صلاحيته
- **إدارة manual لل** :**tokens**تحتاج إلى نظام لإدارة Tokens المنتهية الصلاحية

مثال عملي

python
```python
# settings.py
INSTALLED_APPS = [
    'rest_framework',
    'rest_framework.authtoken',
]

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

# models.py (إنشاء token تلقائي عند إنشاء مستخدم)
from django.db.models.signals import post_save
from django.dispatch import receiver
from rest_framework.authtoken.models import Token
from django.contrib.auth.models import User

@receiver(post_save, sender=User)
def create_auth_token(sender, instance=None, created=False, **kwargs):
    if created:
        Token.objects.create(user=instance)

# views.py
from rest_framework.authtoken.views import ObtainAuthToken
from rest_framework.authtoken.models import Token
from rest_framework.response import Response

class CustomObtainAuthToken(ObtainAuthToken):
    def post(self, request, *args, **kwargs):
```

```
        serializer = self.serializer_class(data=request.data,
                                           context={'request': request})
        serializer.is_valid(raise_exception=True)
        user = serializer.validated_data['user']
        token, created = Token.objects.get_or_create(user=user)
        return Response({
            'token': token.key,
            'user_id': user.pk,
            'email': user.email
        })
```

JWT Authentication <a name="jwt-authentication"></a>

آلية العمل

JSON Web Tokens (JWT)هي معيار مفتوح لإنشاء tokens تحتوي على بيانات:

1. يقوم المستخدم بتسجيل الدخول بإرسال بيانات الاعتماد
2. يتحقق الخادم من صحة البيانات وينشئJWT token
3. يتكون JWT من ثلاثة أجزاءHeader, Payload, Signature :
4. يتم إرجاع token إلى العميل الذي يقوم بتخزينه
5. مع كل طلب لاحق، يرسل العميل token في رأس الطلب
6. يتحقق الخادم من توقيع token ويستخرج بيانات المستخدم

المميزات

- **عديمة الحالة (Stateless)**: مثلToken Authentication
- **تحتوي على بيانات**: يمكن تخزين بيانات المستخدم في Token نفسها
- **قابلة للتوسع بشكل كبير**: مناسبة للتطبيقات الموزعة
- **دعم متعدد المنصات**: تعمل مع جميع أنواع التطبيقات

القيود

- **حجم أكبر** Tokens: أكبر من tokens العادية
- **تعقيد في التنفيذ**: تحتاج إلى فهم آلية عملJWT
- **عدم إمكانية الإلغاء**: مثل) Token Authentication ما لم تستخدم قائمة سوداء(

مثال عملي

```python
# settings.py
INSTALLED_APPS = [
    'rest_framework',
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ]
}

# urls.py
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)

urlpatterns = [
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair
'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refres
h'),
]

# views.py
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated

class ProtectedView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        content = {
            'message': 'Hello, authenticated user!',
            'user': str(request.user)
        }
        return Response(content)
```

مقارنة شاملة بين آليات المصادقة "><a name="مقارنة-شاملة</a>

جدول المقارنة

المعيار	Sessions Authentication	Token Authentication	JWT Authentication
الحالة	stateful	stateless	stateless
التخزين	على الخادم	قاعدة البيانات	في Token نفسها
حجم البيانات	صغير (معرف فقط)	متغير	كبير (يحتوي على بيانات)
الأمان	عالي	متوسط	عالي (مع التوقيع)
قابلية التوسع	محدودة	عالية	عالية جدًا
إمكانية الإلغاء	سهلة	صعبة	صعبة (ما لم تستخدم القائمة السوداء)
سهولة الاستخدام	سهلة	متوسطة	متوسطة إلى صعبة
مناسبة لـAPIs	محدودة	ممتازة	ممتازة
أداء الخادم	يستهلك ذاكرة	يعتمد على قاعدة البيانات	سريع (لا يحتاج لاستعلامات)

مقارنة الأداء

- **Sessions**: أسرع في القراءة ولكن يستهلك ذاكرة الخادم
- **Token**: يعتمد على أداء قاعدة البيانات في التحقق
- **JWT**: أسرع بشكل عام (لا استعلامات قاعدة بيانات) ولكن حجمه أكبر

مقارنة الأمان

- **Sessions**: آمنة مع HTTPS وضبط إعدادات Cookies بشكل صحيح
- **Token**: تحتاج إلى نقل آمن وتخزين آمن على العميل
- **JWT**: آمنة مع التوقيع القوي وضبط صلاحية مناسبة

أمثلة عملية وتطبيقات "><a name="أمثلة-عملية</a>

تطبيق عملي شامل

python
```python
# settings.py - إعدادات متعددة للمصادقة
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.TokenAuthentication',
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ]
}

# authentication.py - مصادقة مخصصة متعددة
from rest_framework.authentication import SessionAuthentication, TokenAuthent
ication
from rest_framework_simplejwt.authentication import JWTAuthentication
from rest_framework.exceptions import AuthenticationFailed

class MultiAuthentication:
    def __init__(self):
        self.authenticators = [
            SessionAuthentication(),
            TokenAuthentication(),
            JWTAuthentication()
        ]

    def authenticate(self, request):
        for authenticator in self.authenticators:
            try:
                user_auth_tuple = authenticator.authenticate(request)
                if user_auth_tuple is not None:
                    return user_auth_tuple
            except AuthenticationFailed:
                continue
        return None

# views.py - واجهات متعددة لأنواع المصادقة
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
```

```python
from .authentication import MultiAuthentication

class MultiAuthView(APIView):
    authentication_classes = [MultiAuthentication]
    permission_classes = [IsAuthenticated]

    def get(self, request):
        return Response({
            'message': 'Authenticated successfully with multiple methods',
            'user': request.user.username,
            'auth_type': self.get_auth_type(request)
        })

    def get_auth_type(self, request):
        if hasattr(request, 'auth'):
            if hasattr(request.auth, 'token'):
                return 'Token Authentication'
            elif isinstance(request.auth, dict):
                return 'JWT Authentication'
        return 'Session Authentication'
```

إعدادات JWT متقدمة

python
```python
# settings.py
from datetime import timedelta

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': False,
    'BLACKLIST_AFTER_ROTATION': True,

    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'VERIFYING_KEY': None,
    'AUDIENCE': None,
    'ISSUER': None,

    'AUTH_HEADER_TYPES': ('Bearer',),
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',
```

```
    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),
    'TOKEN_TYPE_CLAIM': 'token_type',

    'JTI_CLAIM': 'jti',

    'SLIDING_TOKEN_REFRESH_EXP_CLAIM': 'refresh_exp',
    'SLIDING_TOKEN_LIFETIME': timedelta(minutes=5),
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),
}
```

توصيات واختيار الآلية المناسبة "<a name=توصيات-واختيار-الآلية-المناسبة></a>"

سيناريوهات الاستخدام الموصى بها

1. Sessions Authentication

- **التطبيقات التقليدية** التي تعتمد على قوالبDjango
- **التطبيصات التي تتطلب إدارة جلسات** متقدمة
- **عند الحاجة لإلغاء الجلسات** فورًا
- **التطبيقات التي تخدم محتوى ديناميكي** مباشرة من الخادم

2. Token Authentication

- **واجهات برمجة التطبيقات (APIs)** البسيطة
- **التطبيقات التي تحتاج مصادقة دائمة** بدون إعادة تسجيل دخول متكررة
- **عند عدم الحاجة لإلغاء Tokens** بشكل فوري
- **التطبيقات ذات عدد مستخدمين محدود**

3. JWT Authentication

- **التطبيقات الموزعة** والأنظمة الميكروسيرفيس
- **التطبيقات التي تحتاج أداء عالي** مع عدد كبير من المستخدمين
- **عند الحاجة لنقل بيانات المستخدم** في Token نفسها
- **التطبيقات التي تعمل عبر نطاقات متعددة**

خوارزمية الاختيار

```text
if (application_type == "traditional_web_app"):
    use SessionsAuthentication
elif (api_required == True && scalability_required == True):
    use JWTAuthentication
elif (api_required == True && simplicity_required == True):
    use TokenAuthentication
elif (microservices == True || distributed_system == True):
    use JWTAuthentication
else:
    use SessionsAuthentication // الخيار الافتراضي الآمن
```

أفضل الممارسات

1. **استخدم HTTPS دائمًا** لأي نوع من المصادقة
2. **اضبط صلاحية Tokens بشكل مناسب** لتحقيق التوازن بين الأمان والتجربة
3. **نفذ تجديد Tokens** لتحسين الأمان
4. **استخدم القوائم السوداء** لـ JWT إذا كنت تحتاج إمكانية الإلغاء
5. **حافظ على تحديث المكتبات** وإعدادات الأمان

الخاتمة "><a name="الخاتمة</a>

تقدم Django مجموعة قوية من آليات المصادقة التي تناسب مختلف أنواع التطبيقات ومتطلباتها. يعتمد اختيار الآلية المناسبة على عدة عوامل including نوع التطبيق، متطلبات الأداء، considerationsالأمان، وهيكل النظام.

- **Sessions Authentication**تظل الخيار الأمثل للتطبيقات التقليدية
- **Token Authentication**مناسبة للـ APIs البسيطة والمحدودة
- **JWT Authentication**هي الخيار الأفضل للتطبيقات الموزعة والكبيرة

يجب أن يكون اختيار آلية المصادقة جزءًا من التصميم architecture المبكر للتطبيق، مع مراعاة إمكانية التوسع future والأمان طويل المدى.