Ergänzung zu Skript 16 & 17: Praktische API-Anfragen

Einleitung

- **Ziel**: Dieses Skript zeigt, wie man mit den in Skript 16 und 17 erstellten Django-REST-APIs von externen Clients aus interagiert.
- **Werkzeuge**: Wir verwenden zwei gängige Methoden:
    1. **Insomnia**: Ein API-Client-Programm, ideal zum Testen und Debuggen von API-Endpunkten. (Alternativen wie Postman funktionieren sehr ähnlich).
    2. **"Vanilla" JavaScript**: Direkte API-Anfragen aus einer einfachen HTML-Datei mit der `fetch`-API, so wie es eine moderne Frontend-Anwendung tun würde.

Kontext: Unser `Product`-API-Beispiel

Stellen wir uns vor, wir haben eine einfache Django-App `shop` mit folgendem Modell, Serializer und ViewSet:

`shop/models.py`:

```
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField(blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return self.name
```

`shop/serializers.py`:

```
from rest_framework import serializers
from .models import Product

class ProductSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = ['id', 'name', 'description', 'price']
```

`shop/views.py`:

```
from rest_framework import viewsets
from .models import Product
from .serializers import ProductSerializer

class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    # Berechtigungen werden wir in Teil 2 hinzufügen
```

Der Router macht diese API unter `/api/products/` verfügbar.

Teil 1: Anfragen an eine ungeschützte API (Bezug zu Skript 16)

Hier gehen wir davon aus, dass in `settings.py` noch keine globalen Berechtigungen gesetzt sind und jeder auf die API zugreifen kann.

Beispiel A: API-Test mit Insomnia

1. **GET-Anfrage (Alle Produkte abrufen)**:
    - Öffne Insomnia und erstelle eine neue HTTP-Anfrage (`New Request`).
    - Wähle die Methode `GET`.
    - Gib die URL ein: `http://127.0.0.1:8000/api/products/`
    - Klicke auf "Send".
    - **Ergebnis**: Im rechten Fenster siehst du den `200 OK` Status und eine JSON-Liste aller Produkte in deiner Datenbank.
2. **POST-Anfrage (Neues Produkt erstellen)**:
    - Erstelle eine neue Anfrage mit der Methode `POST` und derselben URL.
    - Wechsle zum "Body"-Tab und wähle als Format "JSON".
    - Gib das JSON für ein neues Produkt ein:

```
{
    "name": "High-End Tastatur",
    "description": "Mechanisch, mit RGB-Beleuchtung.",
    "price": "149.99"
}
```

    - Klicke auf "Send".
    - **Ergebnis**: Du erhältst einen `201 Created` Status und im Antwort-Body die Daten des neu erstellten Produkts, inklusive seiner neuen `id`.

Beispiel B: API-Anfrage mit "Vanilla" JavaScript (`fetch`)

Erstelle eine einfache `index.html`-Datei auf deinem Computer und öffne sie im Browser.

1. **GET-Anfrage (Alle Produkte abrufen & in der Konsole anzeigen)**:

```
<!DOCTYPE html>
<html lang="de">
<head><title>API Test</title></head>
<body>
    <h1>API-Test</h1>
    <p>Öffne die Browser-Konsole (F12), um die Ergebnisse zu sehen.</p>
    <script>
        // GET-Anfrage an die API
        fetch('http://127.0.0.1:8000/api/products/')
            .then(response => {
                if (!response.ok) {
                    throw new Error('Netzwerk-Antwort war nicht ok.');
                }
                return response.json(); // Wandelt die Antwort in JSON um
            })
            .then(data => {
                console.log('Produkte erfolgreich abgerufen:', data);
            })
            .catch(error => {
                console.error('Fehler bei der GET-Anfrage:', error);
            });
    </script>
</body>
</html>
```

2. **POST-Anfrage (Neues Produkt erstellen)**: Fügen wir einen Button und mehr JavaScript-Logik hinzu.

```
<button id="create-product-btn">Neues Produkt erstellen</button>

<script>
    // ... (der GET-Request von oben) ...
```

```
    document.getElementById('create-product-btn').addEventListener('click', () => {
        const newProductData = {
            name: "Webcam 4K",
            description": "Kristallklares Bild.",
            price: "89.50"
        };

        fetch('http://127.0.0.1:8000/api/products/', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                // CSRF-Token wird hier für einfache Beispiele oft weggelassen,
                // in einer echten SPA müsste man ihn handhaben!
            },
            body: JSON.stringify(newProductData) // Wandelt das JS-Objekt in einen JSON-
String um
        })
        .then(response => response.json())
        .then(data => {
            console.log('Produkt erfolgreich erstellt:', data);
        })
        .catch(error => {
            console.error('Fehler bei der POST-Anfrage:', error);
        });
    });
</script>
```

Teil 2: Anfragen an eine geschützte API (Bezug zu Skript 17)

Jetzt nehmen wir an, die API wurde wie in Skript 17 mit `TokenAuthentication` und `IsAuthenticatedOrReadOnly` abgesichert.

Beispiel A: API-Test mit Insomnia (mit Token)

1. **Schritt 1: Token holen**
    - Erstelle eine `POST`-Anfrage an `http://127.0.0.1:8000/api/token-auth/`.
    - Gib im JSON-Body den `username` und das `password` eines existierenden Benutzers an.
    - Klicke "Send". Du erhältst ein JSON-Objekt mit einem `token`-Feld. **Kopiere diesen Token-Wert.**
2. **Schritt 2: GET-Anfrage (ungeschützt)**
    - Eine `GET`-Anfrage an `/api/products/` funktioniert weiterhin ohne Token, da wir `IsAuthenticatedOrReadOnly` verwenden.
3. **Schritt 3: POST-Anfrage (geschützt)**
    - Erstelle die `POST`-Anfrage wie in Teil 1.
    - Wechsle zum **"Auth"**-Tab.
    - Wähle aus dem Dropdown-Menü **"Bearer Token"**.
    - Füge in das "TOKEN"-Feld den kopierten Token ein.
    - Klicke "Send".
    - **Ergebnis**: Die Anfrage ist jetzt erfolgreich (`201 Created`), weil Insomnia automatisch den `Authorization: Bearer <dein_token>`-Header mitsendet.

Beispiel B: API-Anfrage mit JavaScript (`fetch`) (mit Token)

1. **Token holen & speichern**: In einer echten Anwendung würdest du den Token nach dem Login einmalig abrufen und speichern (z.B. im `localStorage`). Für dieses Beispiel setzen wir ihn als feste Variable.
2. **POST-Anfrage (geschützt)**: Wir passen den `fetch`-Aufruf an und fügen den `Authorization`-Header hinzu.

```
```

```
<script>
    const authToken = 'HIER_DEINEN_KOPIERTEN_TOKEN_EINFÜGEN'; // Token aus Schritt 1

    document.getElementById('create-product-btn').addEventListener('click', () => {
        const newProductData = {
            name: "Monitor 27 Zoll",
            price: "299.00"
        };

        fetch('http://127.0.0.1:8000/api/products/', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                // Der entscheidende Header für die Authentifizierung
                'Authorization': `Token ${authToken}`
            },
            body: JSON.stringify(newProductData)
        })
        .then(response => {
            if (response.status === 401 || response.status === 403) {
                console.error('Authentifizierung fehlgeschlagen!');
            }
            return response.json();
        })
        .then(data => {
            console.log('Geschütztes Produkt erfolgreich erstellt:', data);
        })
        .catch(error => {
            console.error('Fehler bei der geschützten POST-Anfrage:', error);
        });
    });
</script>
```

Die Logik ist dieselbe, aber der `Authorization`-Header mit dem `Token` ist der Schlüssel zum Erfolg bei geschützten Endpunkten.