



Django Model und Form



Einleitung



pythonTM

django

Kapitel IV
Model und Form

Lernziele Kapitel IV:

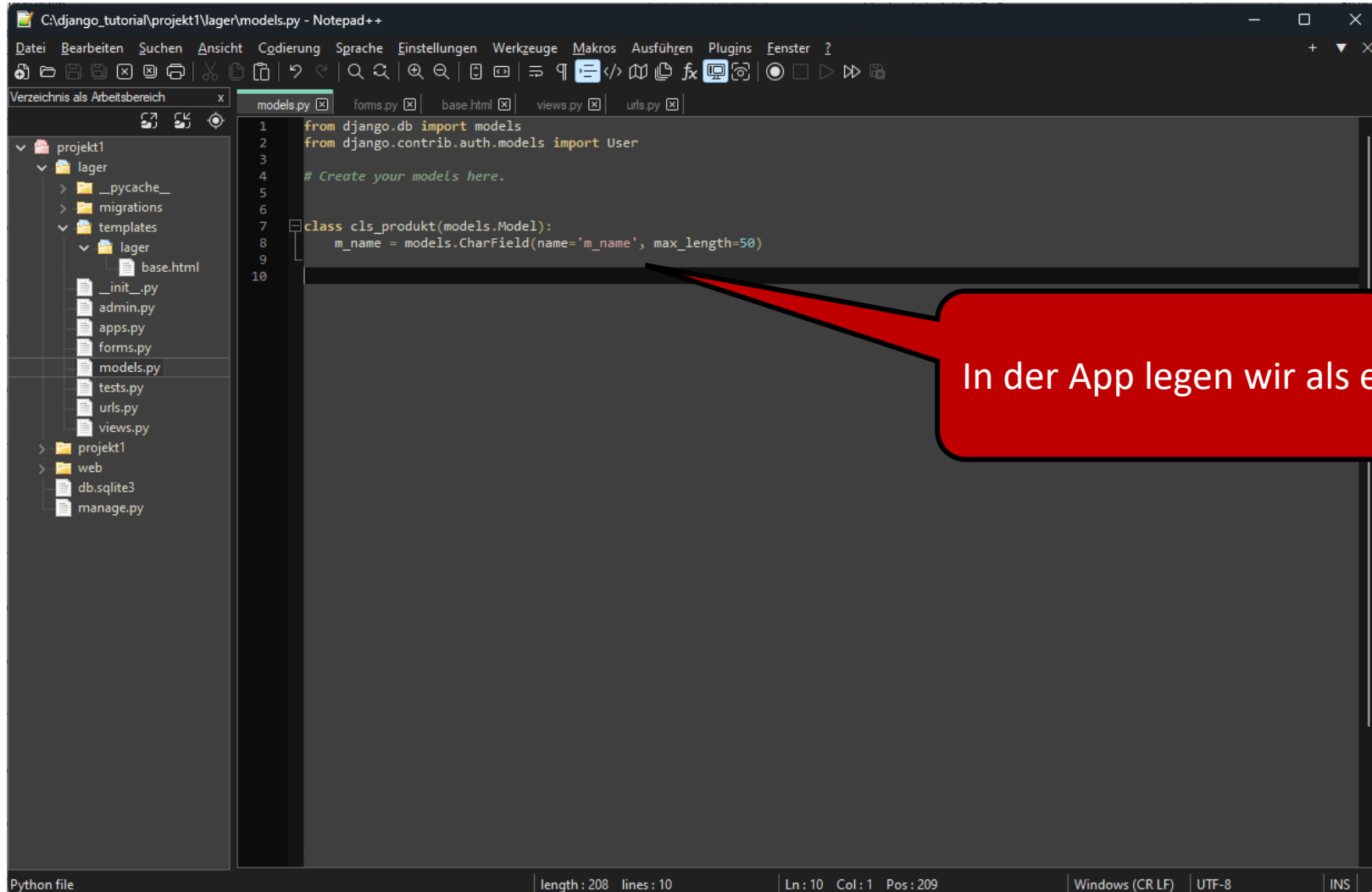
Ich kann ein Django Modell programmieren.

Ich kann eine Django Form programmieren.

Ich kann ein Django ForeignKey Field programmieren.

Ich kann Django Modelle Speichern und Laden.

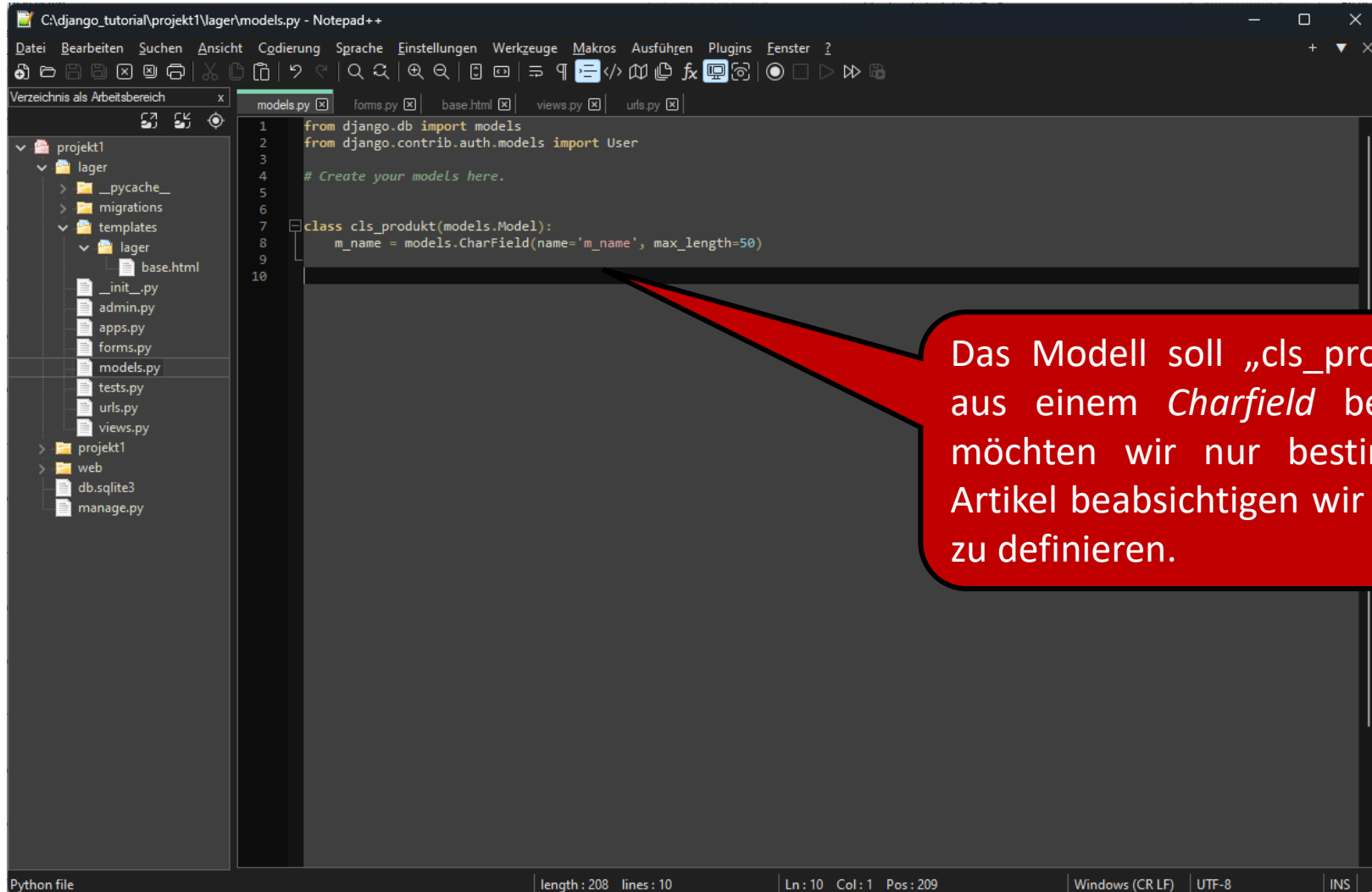
Modelle und Form



```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4 # Create your models here.
5
6
7 class cls_produkt(models.Model):
8     m_name = models.CharField(name='m_name', max_length=50)
9
10
```

In der App legen wir als erstes ein Modell an.

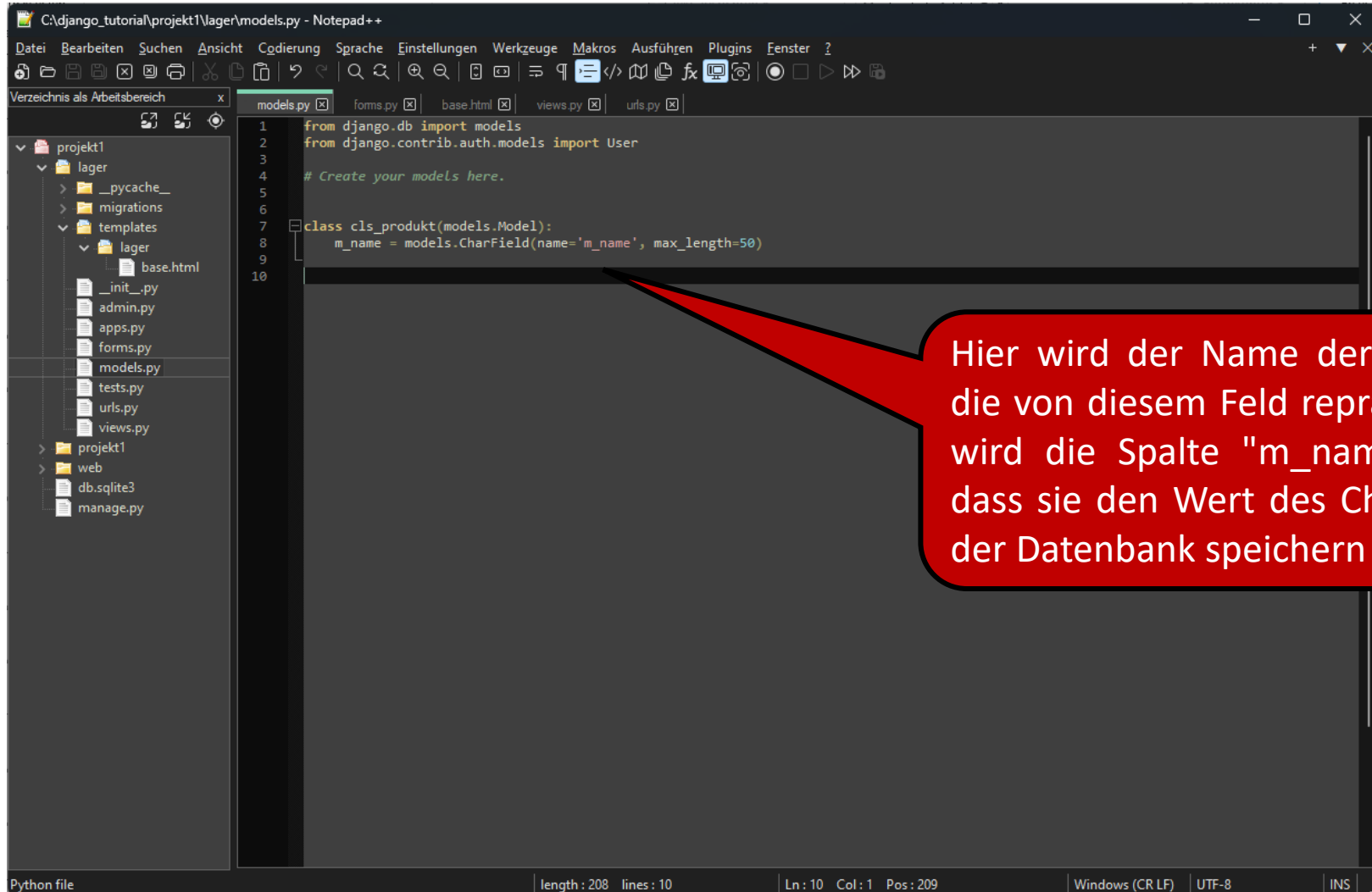
Modelle und Form



```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4 # Create your models here.
5
6
7 class cls_produkt(models.Model):
8     m_name = models.CharField(name='m_name', max_length=50)
9
10
```

Das Modell soll „cls_produkt“ genannt werden und aus einem *Charfield* bestehen. In unserem Lager möchten wir nur bestimmte Artikel lagern. Diese Artikel beabsichtigen wir grundlegend in dieser Klasse zu definieren.

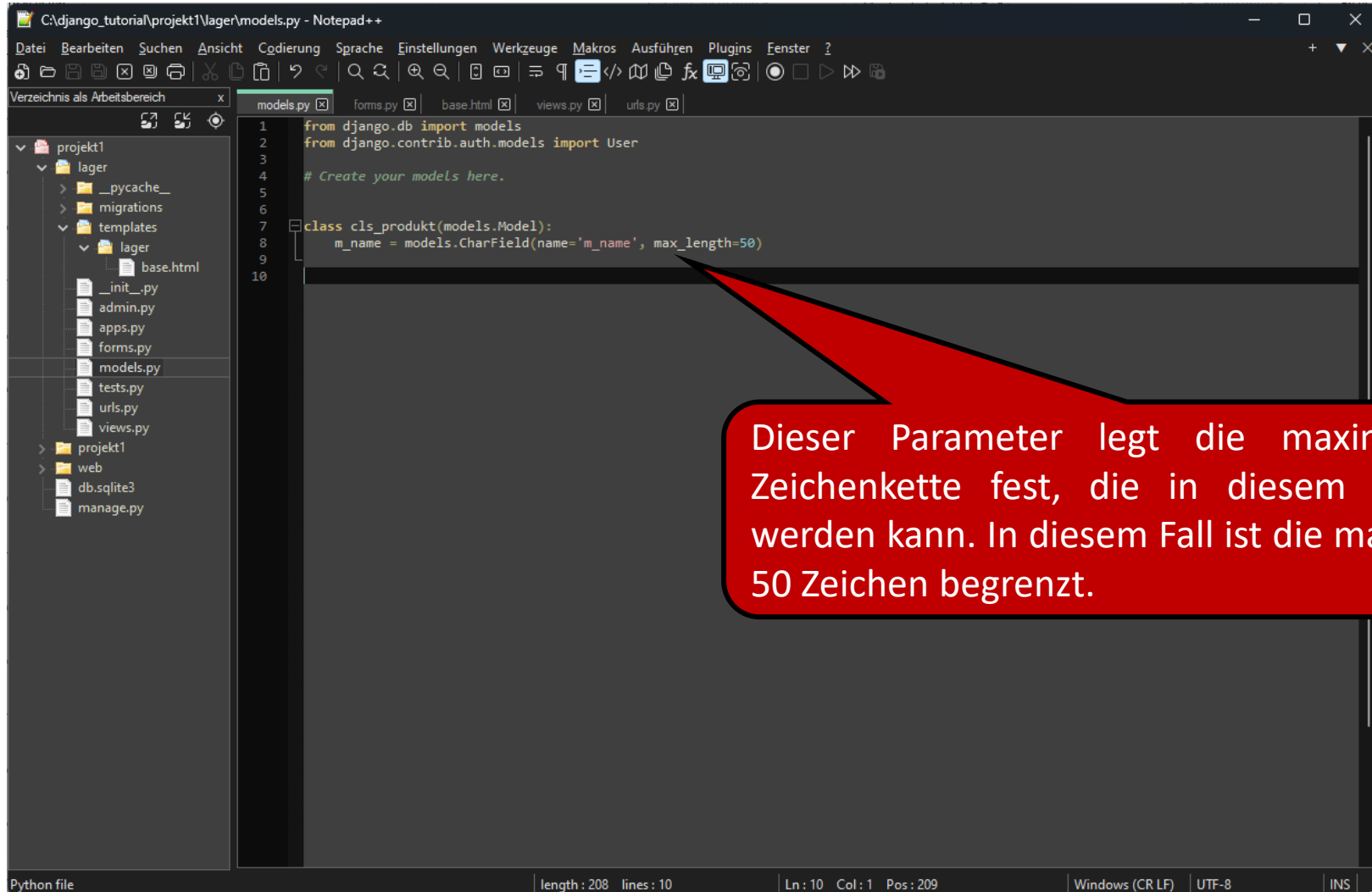
Modelle und Form



```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4 # Create your models here.
5
6
7 class cls_produkt(models.Model):
8     m_name = models.CharField(name='m_name', max_length=50)
9
10
```

Hier wird der Name der Datenbankspalte festgelegt, die von diesem Feld repräsentiert wird. In diesem Fall wird die Spalte "m_name" genannt, was bedeutet, dass sie den Wert des CharField für diesen Namen in der Datenbank speichern wird.

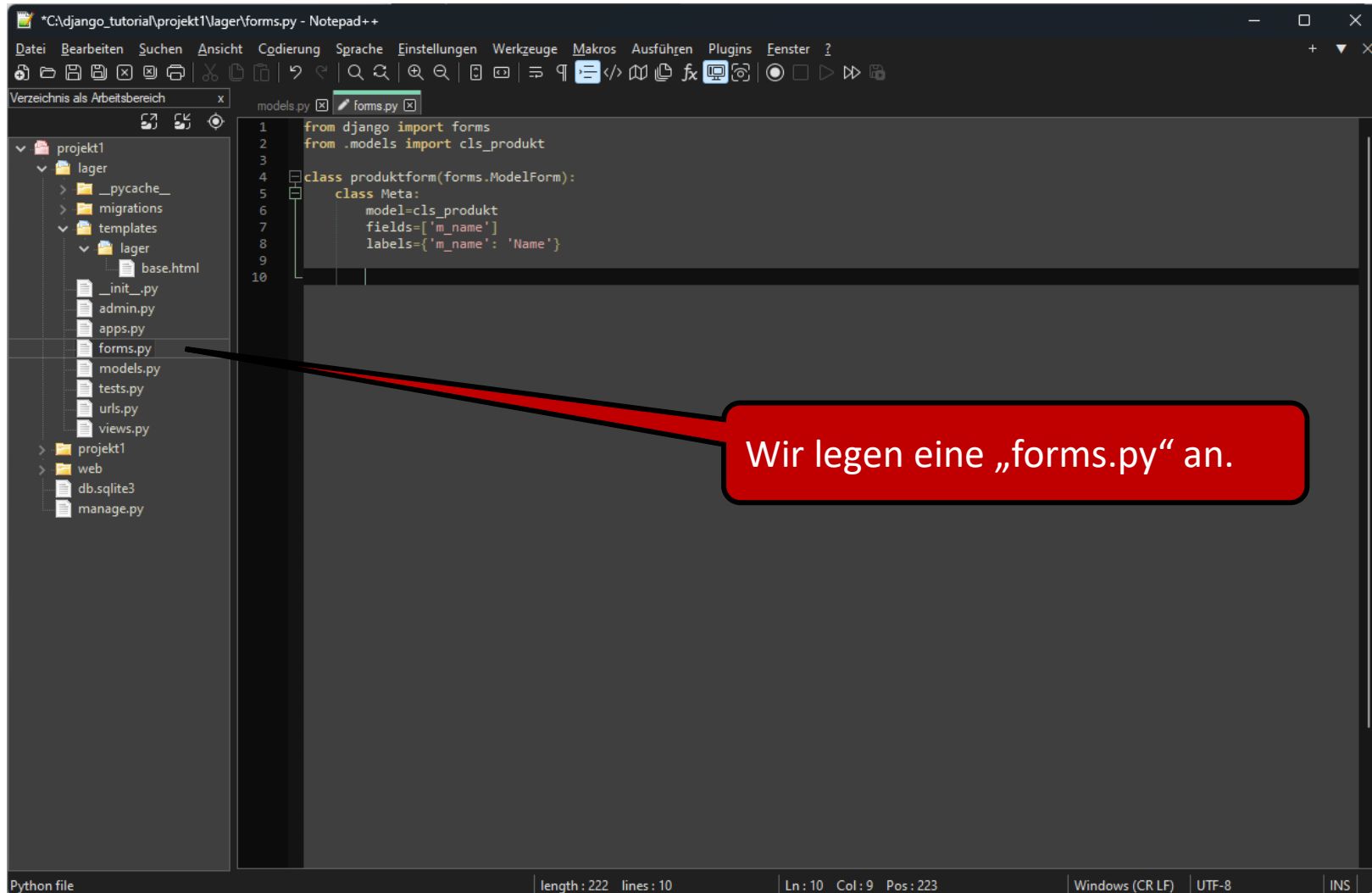
Modelle und Form



```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4 # Create your models here.
5
6
7 class cls_produkt(models.Model):
8     m_name = models.CharField(name='m_name', max_length=50)
9
10
```

Dieser Parameter legt die maximale Länge der Zeichenkette fest, die in diesem Feld gespeichert werden kann. In diesem Fall ist die maximale Länge auf 50 Zeichen begrenzt.

Modelle und Form



The screenshot shows a Notepad++ window with the title bar "C:\django_tutorial\projekt1\lager\forms.py - Notepad++". The menu bar includes "Datei", "Bearbeiten", "Suchen", "Ansicht", "Codierung", "Sprache", "Einstellungen", "Werkzeuge", "Makros", "Ausführen", "Plugins", and "Fenster". The toolbar contains various icons for file operations and editing. The left sidebar shows a file explorer with the following structure:

- projekt1
 - lager
 - __pycache__
 - migrations
 - templates
 - lager
 - base.html
 - __init__.py
 - admin.py
 - apps.py
 - forms.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - projekt1
 - web
 - db.sqlite3
 - manage.py

The main editor area shows the content of `forms.py`:

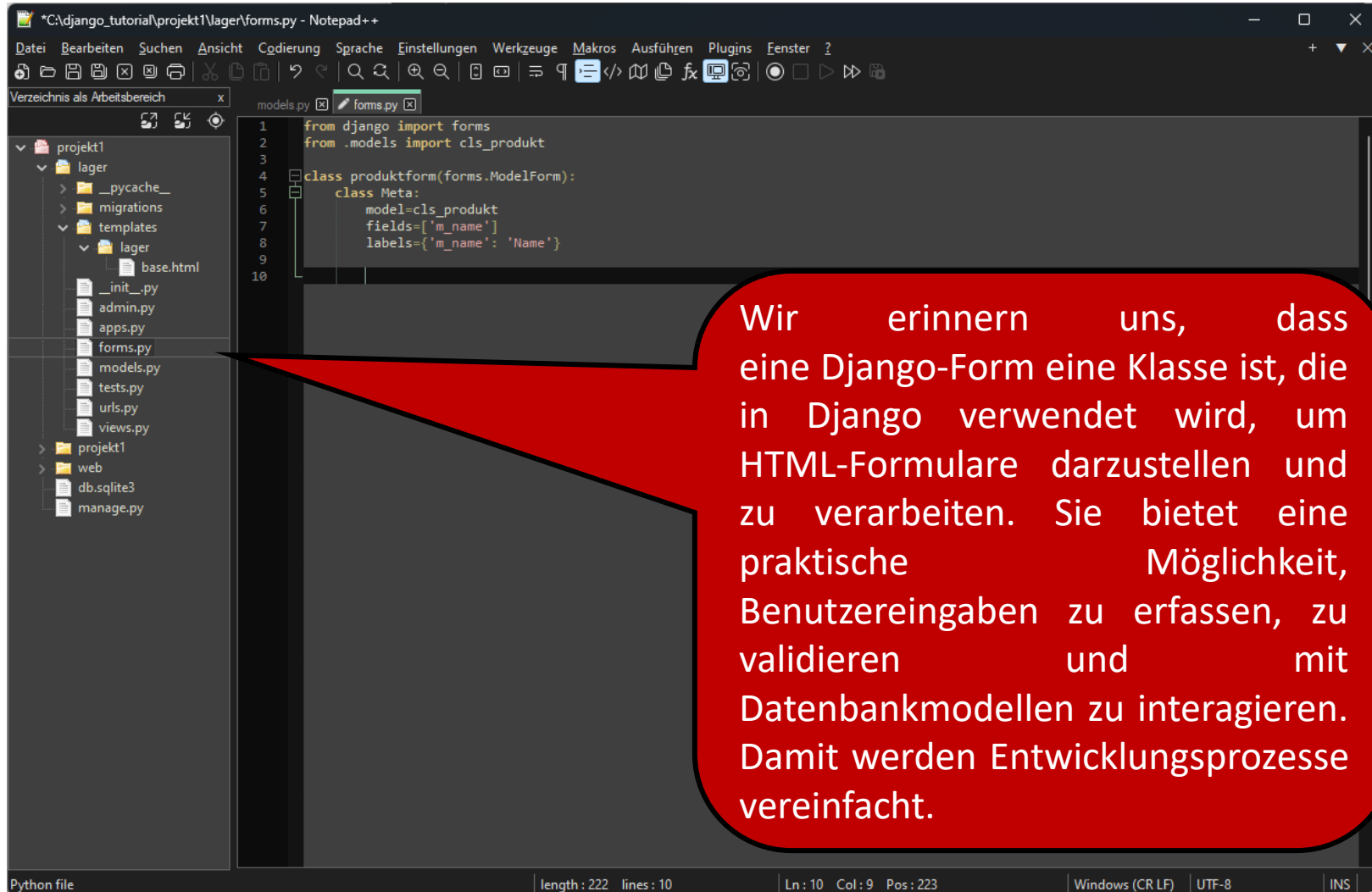
```
1 from django import forms
2 from .models import cls_produkt
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model=cls_produkt
7         fields=['m_name']
8         labels={'m_name': 'Name'}
9
10
```

A red callout box with a black border points to the `forms.py` file in the file explorer. The text inside the box is:

Wir legen eine „forms.py“ an.

The status bar at the bottom shows "Python file", "length : 222 lines : 10", "Ln : 10 Col : 9 Pos : 223", "Windows (CR LF)", "UTF-8", and "INS".

Modelle und Form



The screenshot shows a Notepad++ window with the file path `*C:\django_tutorial\projekt1\lager\forms.py`. The left sidebar displays a project file tree with folders like `lager`, `migrations`, `templates`, and `web`. The main editor area shows the following Python code:

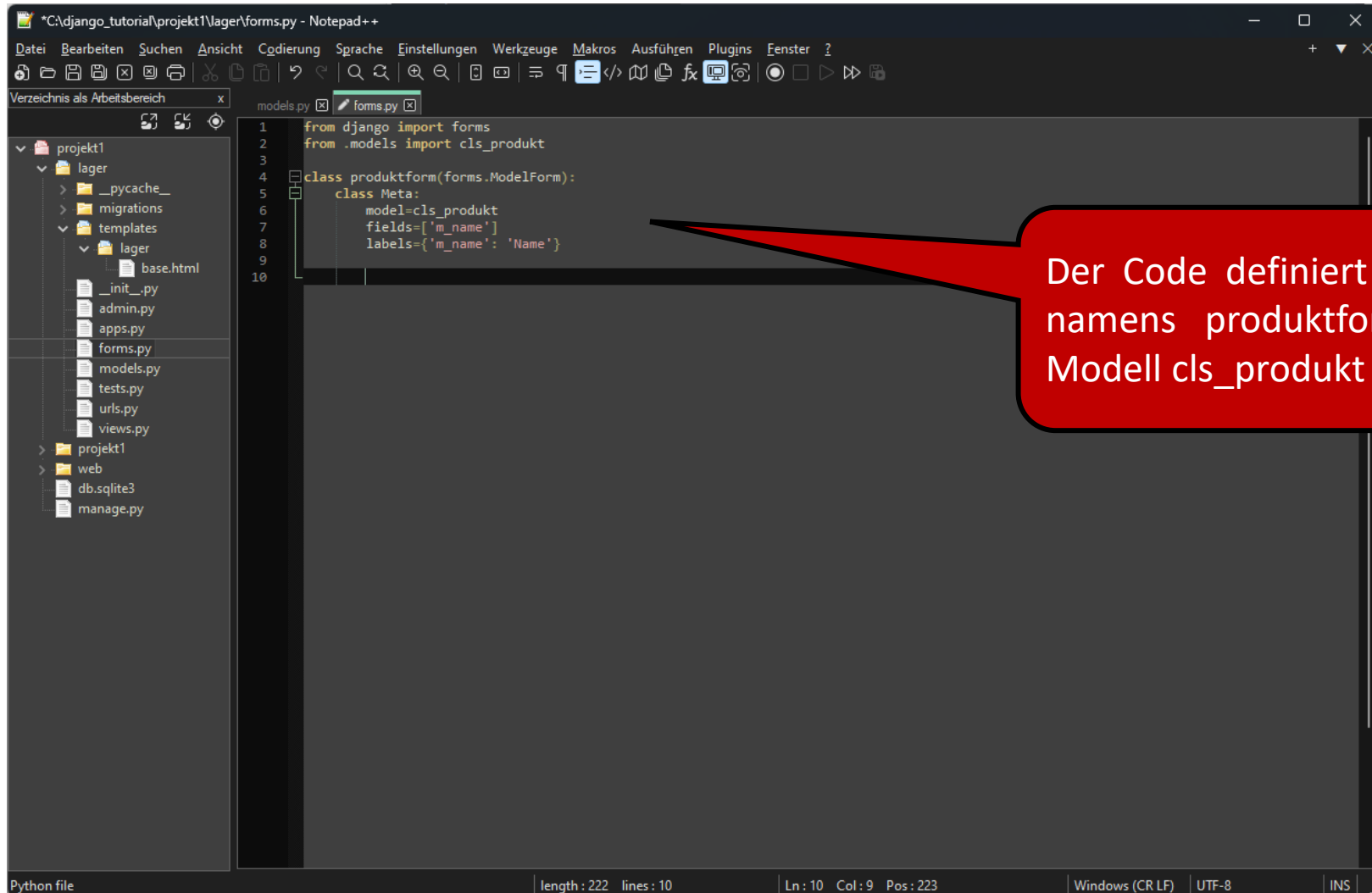
```
1 from django import forms
2 from .models import cls_produkt
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model=cls_produkt
7         fields=['m_name']
8         labels={'m_name': 'Name'}
9
10
```

A red speech bubble points to the `forms.py` file in the sidebar, containing the following text:

Wir erinnern uns, dass eine Django-Form eine Klasse ist, die in Django verwendet wird, um HTML-Formulare darzustellen und zu verarbeiten. Sie bietet eine praktische Möglichkeit, Benutzereingaben zu erfassen, zu validieren und mit Datenbankmodellen zu interagieren. Damit werden Entwicklungsprozesse vereinfacht.

At the bottom of the Notepad++ window, the status bar shows: `Python file`, `length : 222 lines : 10`, `Ln : 10 Col : 9 Pos : 223`, `Windows (CR LF)`, `UTF-8`, and `INS`.

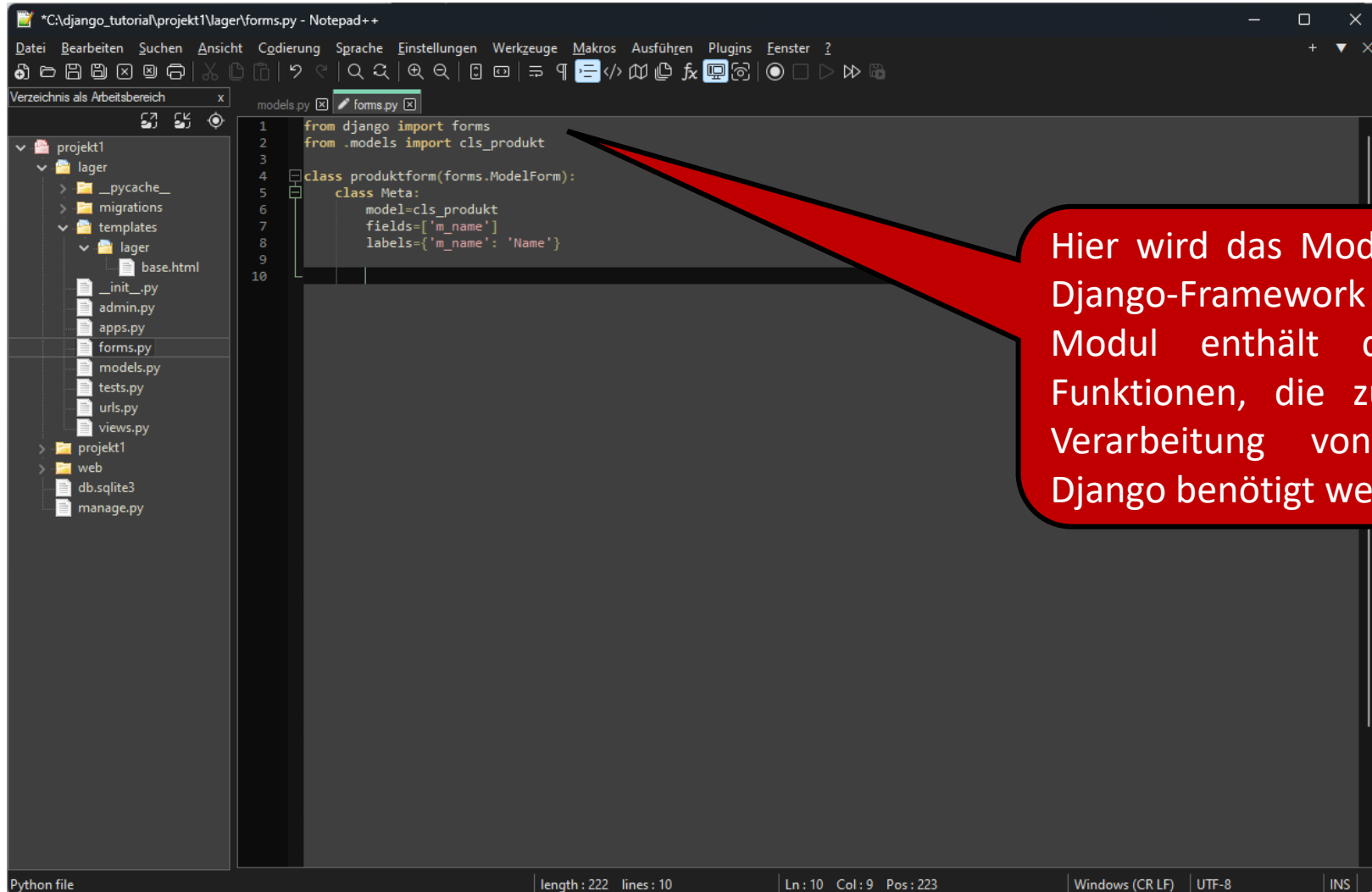
Modelle und Form



```
1 from django import forms
2 from .models import cls_produkt
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model=cls_produkt
7         fields=['m_name']
8         labels={'m_name': 'Name'}
9
10
```

Der Code definiert eine Django-Form namens produktform, die auf dem Modell cls_produkt basiert.

Modelle und Form



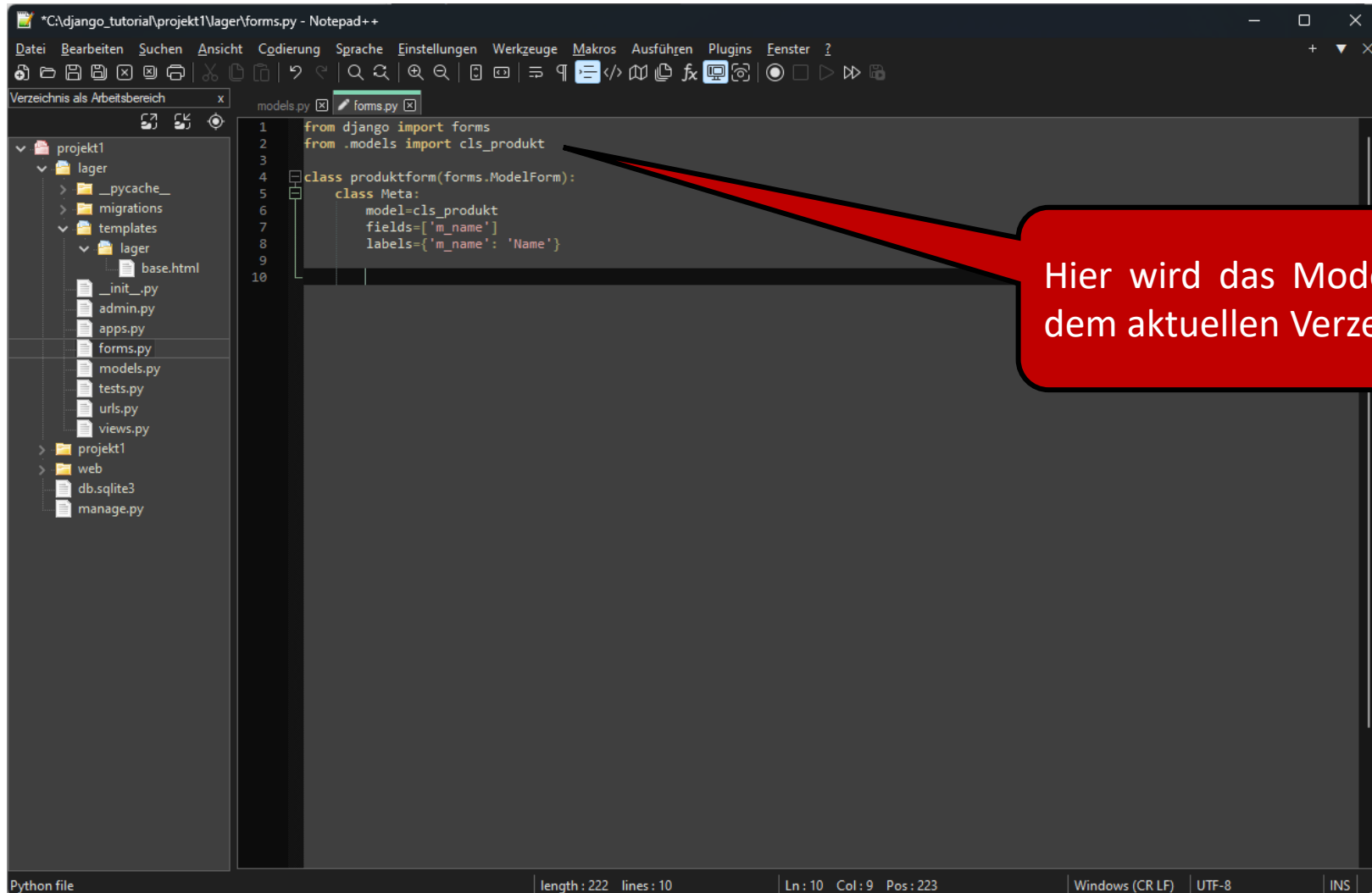
The screenshot shows a Notepad++ window with the file path `*C:\django_tutorial\projekt1\lager\forms.py`. The left sidebar displays a project tree with folders like `lager`, `templates`, and `project1`. The main editor area shows the following Python code:

```
1 from django import forms
2 from .models import cls_produkt
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model=cls_produkt
7         fields=['m_name']
8         labels={'m_name': 'Name'}
9
10
```

A red callout bubble points to the first line of code, `from django import forms`.

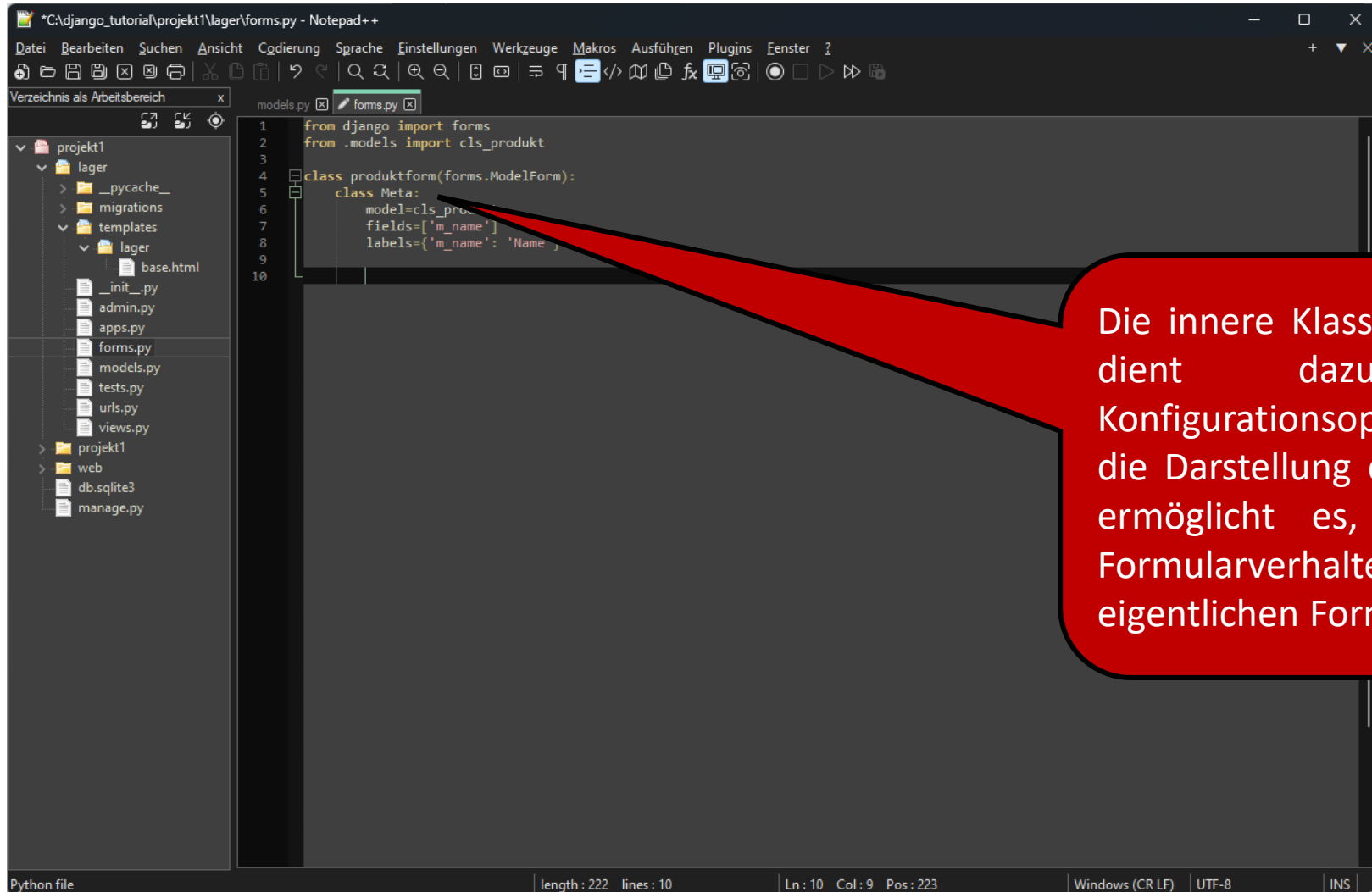
Hier wird das Modul forms aus dem Django-Framework importiert. Dieses Modul enthält die Klassen und Funktionen, die zur Erstellung und Verarbeitung von Formularen in Django benötigt werden.

Modelle und Form



```
1 from django import forms
2 from .models import cls_produkt
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model=cls_produkt
7         fields=[ 'm_name' ]
8         labels={ 'm_name': 'Name' }
9
10
```

Modelle und Form

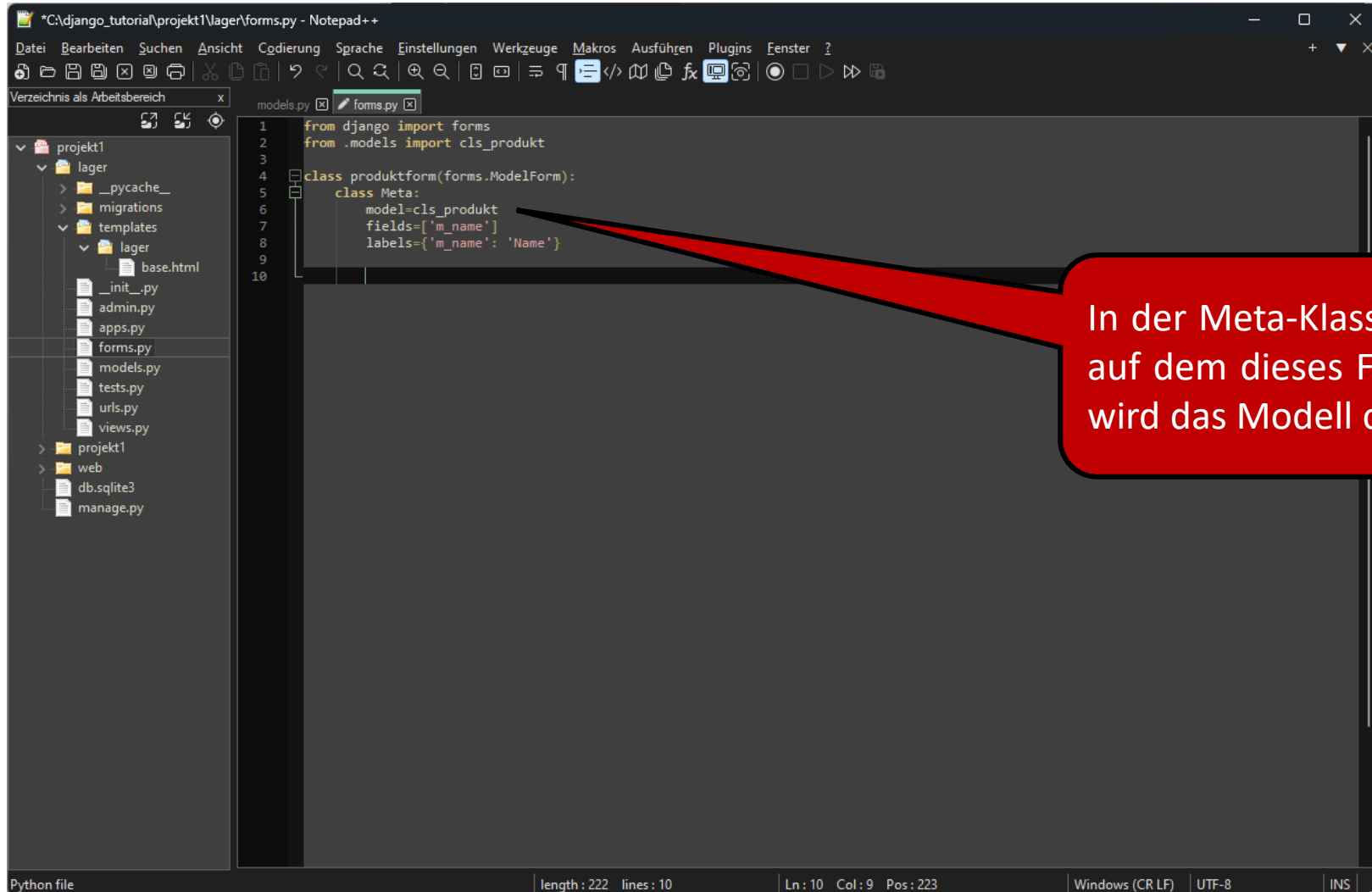


```
1 from django import forms
2 from .models import cls_produkt
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model = cls_produkt
7         fields = ['m_name']
8         labels = {'m_name': 'Name'}
9
10
```

The screenshot shows a Notepad++ window with the file path `*C:\django_tutorial\projekt1\lager\forms.py`. The left sidebar displays a project structure with folders like `lager`, `templates`, and `project1`. The main editor area shows the Python code for `produktform`, which inherits from `forms.ModelForm`. A red callout bubble points to the inner `Meta` class, highlighting its role in defining metadata and configuration options for the form.

Die innere Klasse Meta in einer Django-Form dient dazu, Metadaten oder Konfigurationsoptionen für das Verhalten und die Darstellung des Formulars anzugeben. Sie ermöglicht es, verschiedene Aspekte des Formularverhaltens anzupassen, ohne den eigentlichen Formularkörper zu verändern.

Modelle und Form

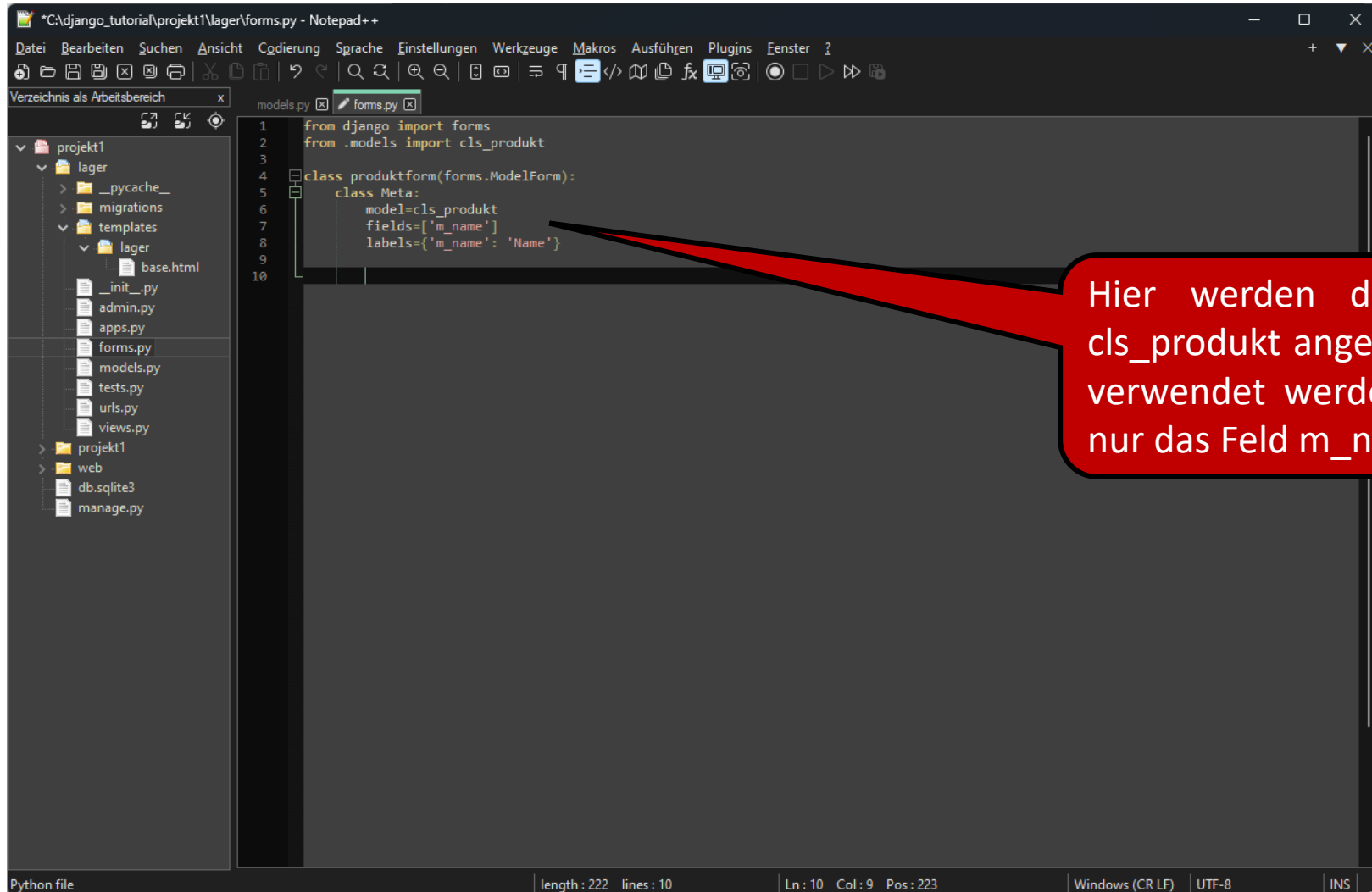


```
1 from django import forms
2 from .models import cls_produkt
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model=cls_produkt
7         fields=['m_name']
8         labels={'m_name': 'Name'}
9
10
```

The screenshot shows a Notepad++ window with the file path `*C:\django_tutorial\projekt1\lager\forms.py`. The left sidebar displays a project structure with folders like `lager`, `migrations`, `templates`, and `web`. The main editor area shows the Python code for `produktform`, which inherits from `forms.ModelForm`. A red callout bubble points to the `model=cls_produkt` line in the `Meta` class, explaining its purpose.

In der Meta-Klasse wird das Modell angegeben, auf dem dieses Formular basiert. In diesem Fall wird das Modell `cls_produkt` verwendet.

Modelle und Form



```
1 from django import forms
2 from .models import cls_produkt
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model=cls_produkt
7         fields=['m_name']
8         labels={'m_name': 'Name'}
9
10
```

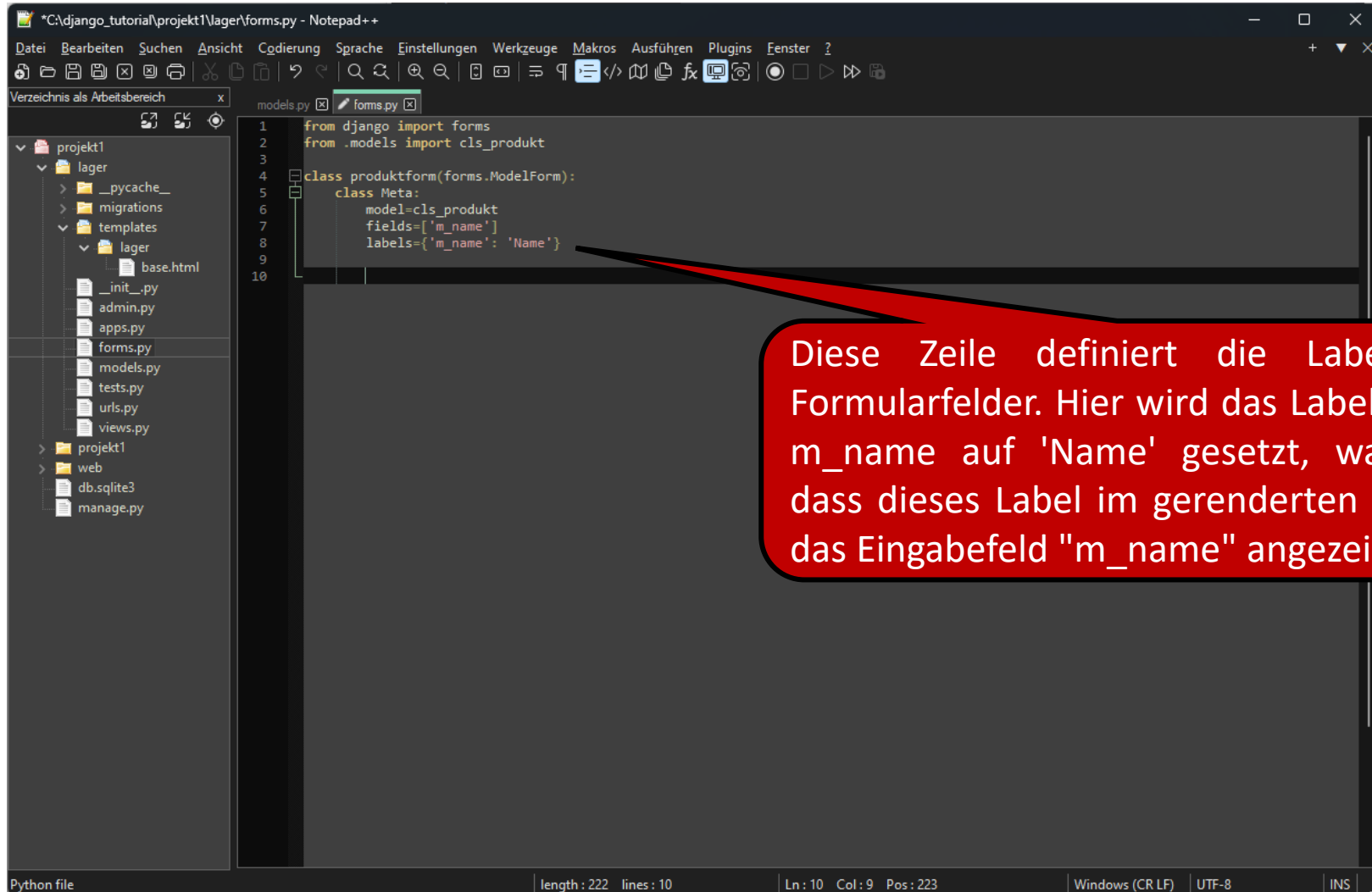
Verzeichnis als Arbeitsbereich

- projekt1
 - lager
 - __pycache__
 - migrations
 - templates
 - lager
 - base.html
 - __init__.py
 - admin.py
 - apps.py
 - forms.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - projekt1
 - web
 - db.sqlite3
 - manage.py

Python file | length : 222 lines : 10 | Ln : 10 Col : 9 Pos : 223 | Windows (CR LF) UTF-8 | INS

Hier werden die Felder aus dem Modell `cls_produkt` angegeben, die in diesem Formular verwendet werden sollen. In diesem Fall wird nur das Feld `m_name` ausgewählt.

Modelle und Form



```
1 from django import forms
2 from .models import cls_produkt
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model=cls_produkt
7         fields=['m_name']
8         labels={'m_name': 'Name'}
9
10
```

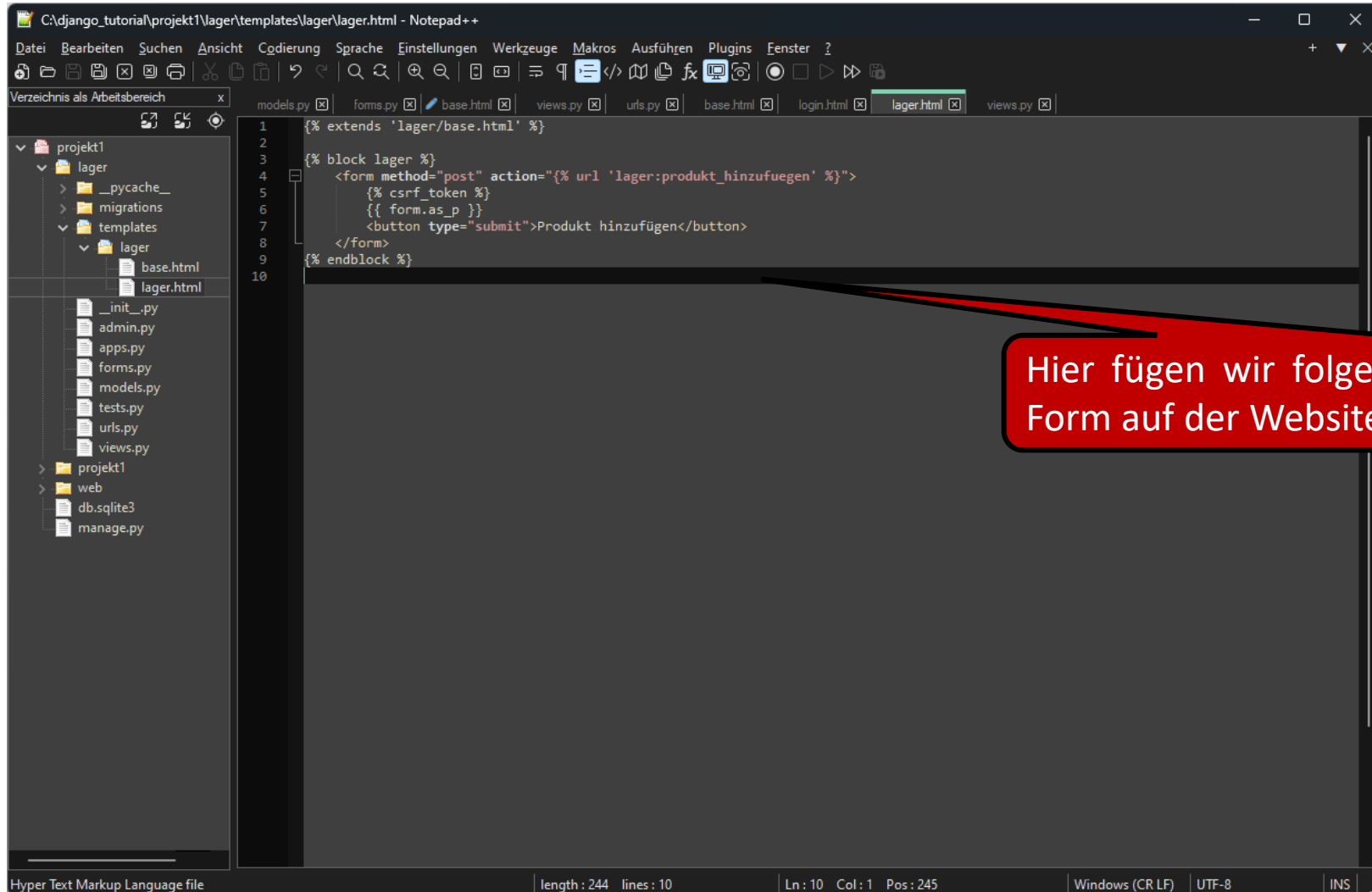
Diese Zeile definiert die Labels für die Formularfelder. Hier wird das Label für das Feld m_name auf 'Name' gesetzt, was bedeutet, dass dieses Label im gerenderten Formular für das Eingabefeld "m_name" angezeigt wird.

Modelle und Form

```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
5     {% csrf_token %}
6     {{ form.as_p }}
7     <button type="submit">Produkt hinzufügen</button>
8 </form>
9 {% endblock %}
10
```

Wir erstellen eine lager.html in den lager templates.

Modelle und Form



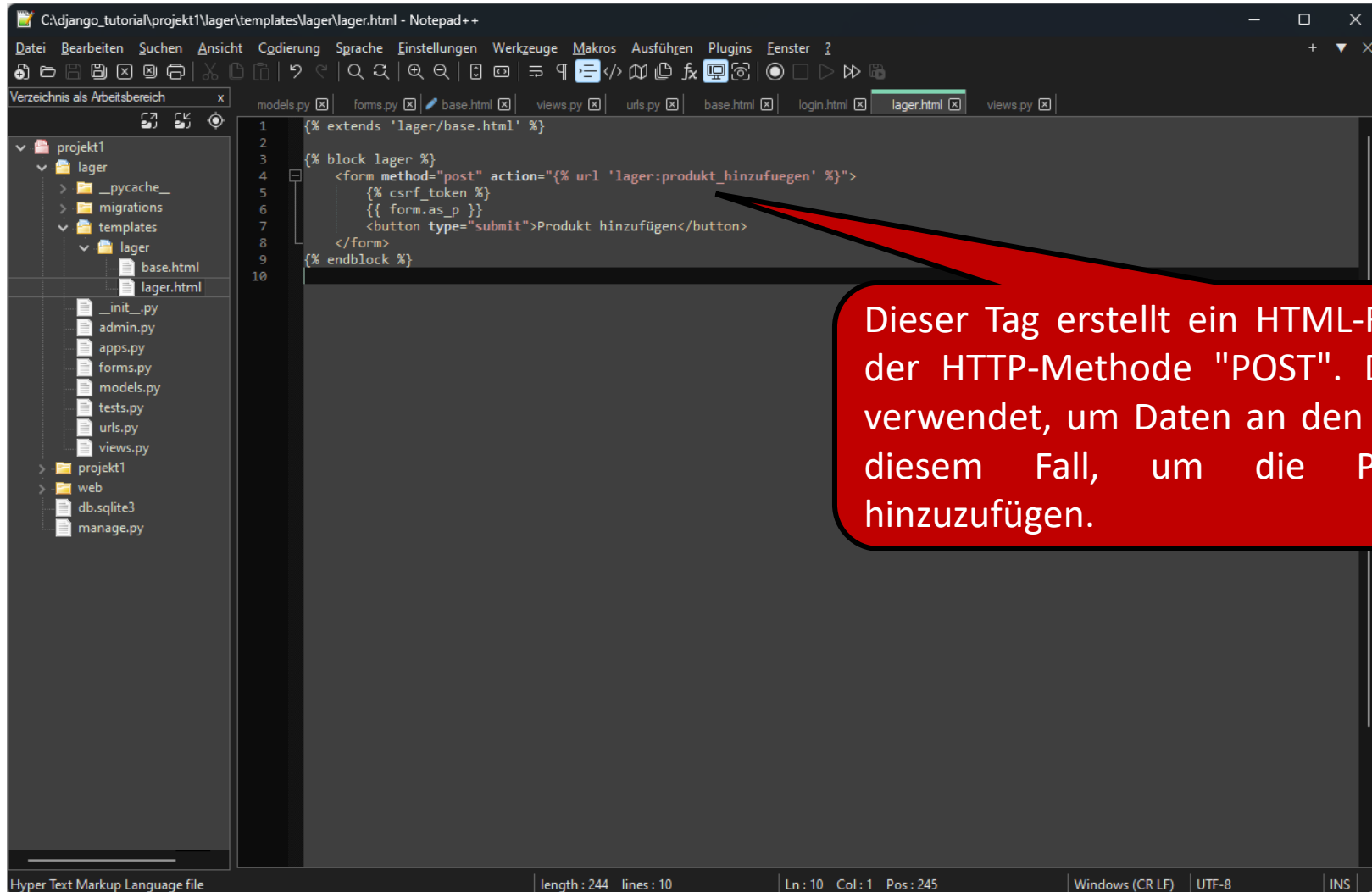
The screenshot shows a Notepad++ window with the file path `C:\django_tutorial\projekt1\lager\templates\lager\lager.html`. The left sidebar displays a project tree with folders like `lager`, `templates`, and `lager`. The main editor area shows the following HTML template code:

```
1 {% extends 'lager/base.html' %}
2
3
4 {% block lager %}
5     <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
6         {% csrf_token %}
7         {{ form.as_p }}
8         <button type="submit">Produkt hinzufügen</button>
9     </form>
10 {% endblock %}
```

The status bar at the bottom indicates the file is a Hyper Text Markup Language file, with a length of 244 bytes and 10 lines. The cursor is positioned at line 10, column 1, position 245.

Hier fügen wir folgende html-Zeilen ein, damit die Form auf der Website angezeigt wird.

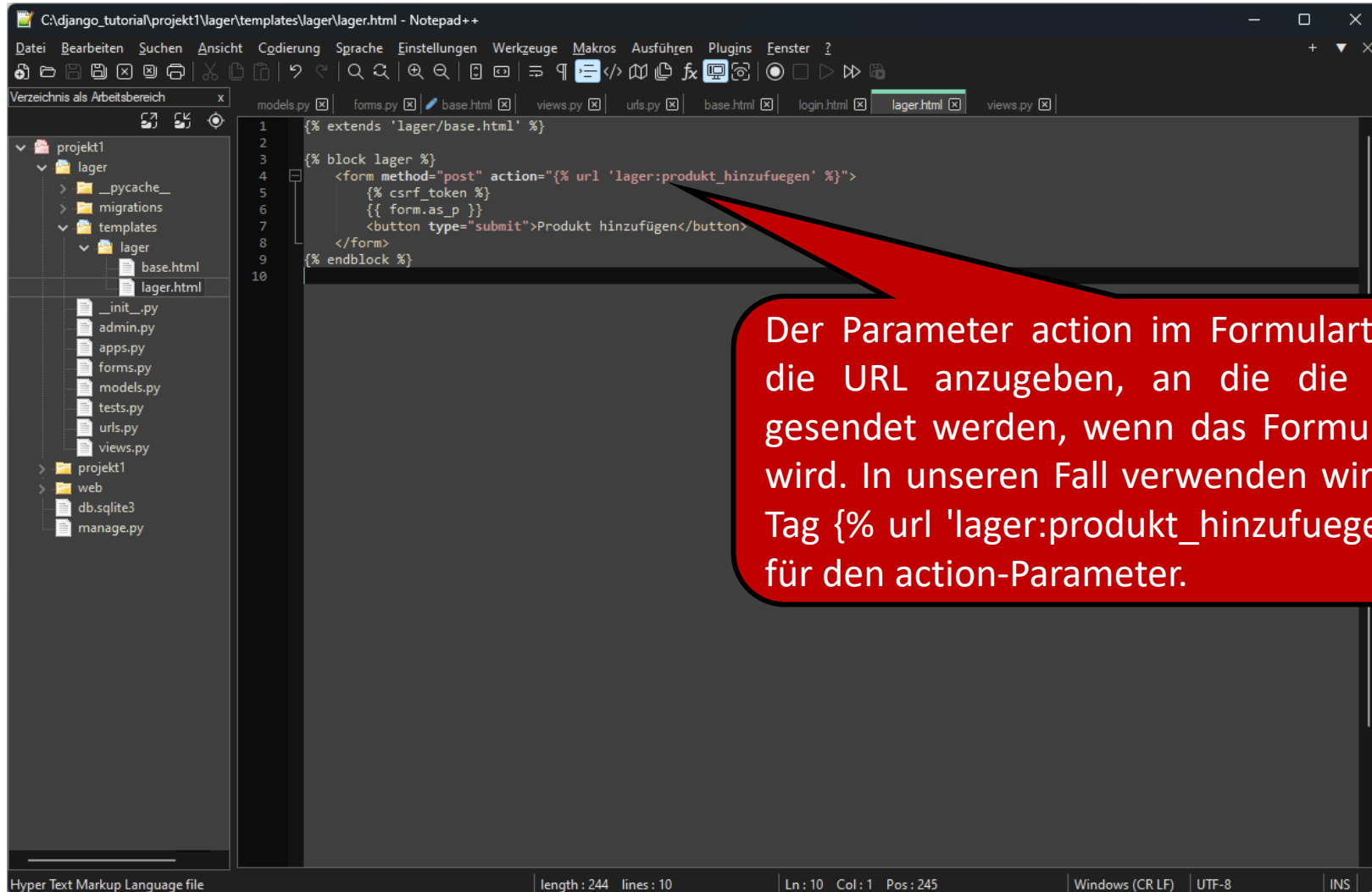
Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3
4 {% block lager %}
5     <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
6         {% csrf_token %}
7         {{ form.as_p }}
8         <button type="submit">Produkt hinzufügen</button>
9     </form>
10 {% endblock %}
```

Dieser Tag erstellt ein HTML-Formularelement mit der HTTP-Methode "POST". Diese Methode wird verwendet, um Daten an den Server zu senden, in diesem Fall, um die Produktinformationen hinzuzufügen.

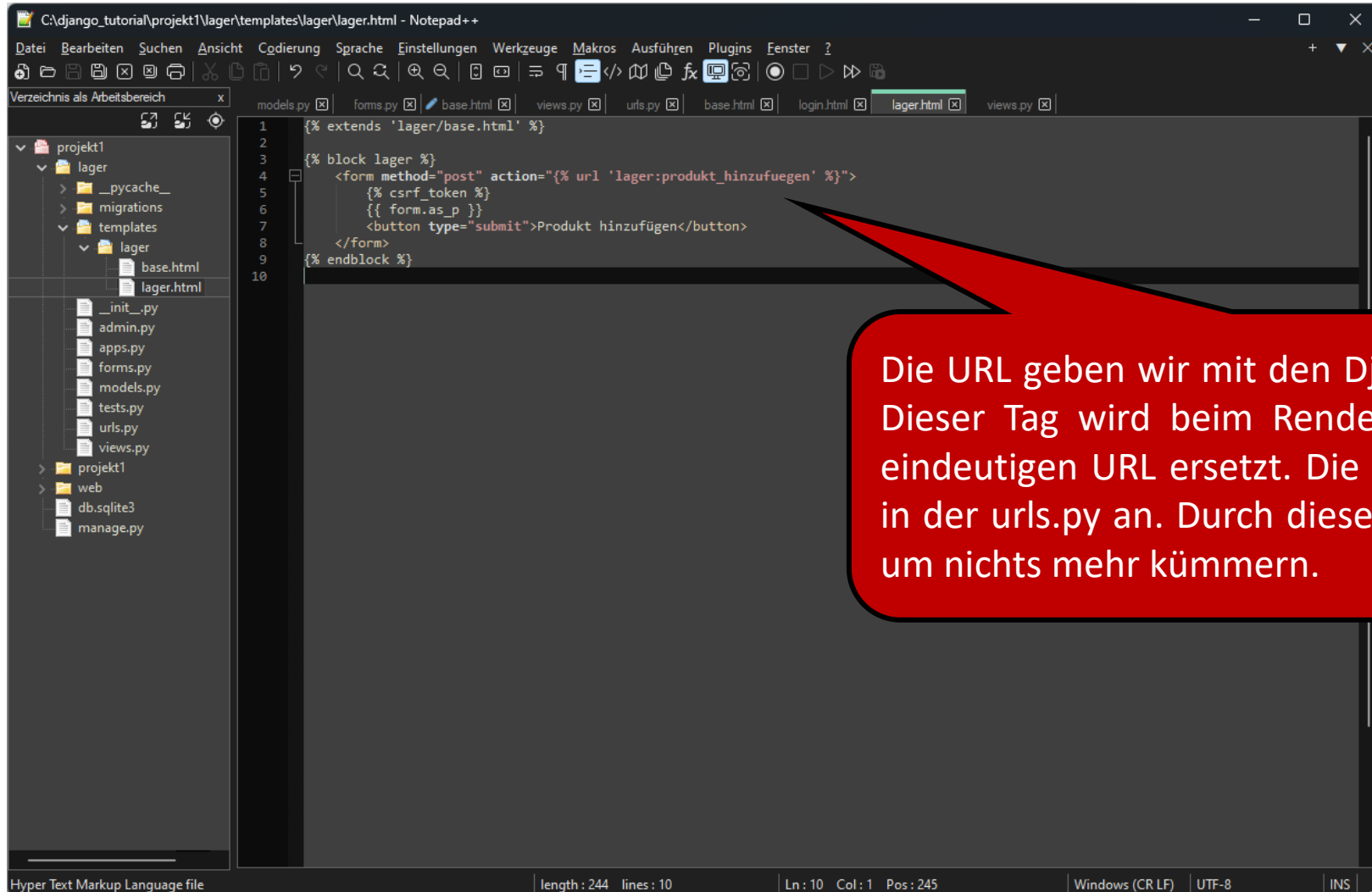
Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3
4 {% block lager %}
5     <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
6         {{ csrf_token }}
7         {{ form.as_p }}
8         <button type="submit">Produkt hinzufügen</button>
9     </form>
10 {% endblock %}
```

Der Parameter action im Formulartag dient dazu, die URL anzugeben, an die die Formulardaten gesendet werden, wenn das Formular abgesendet wird. In unseren Fall verwenden wir das Template-Tag `{% url 'lager:produkt_hinzufuegen' %}` als Wert für den action-Parameter.

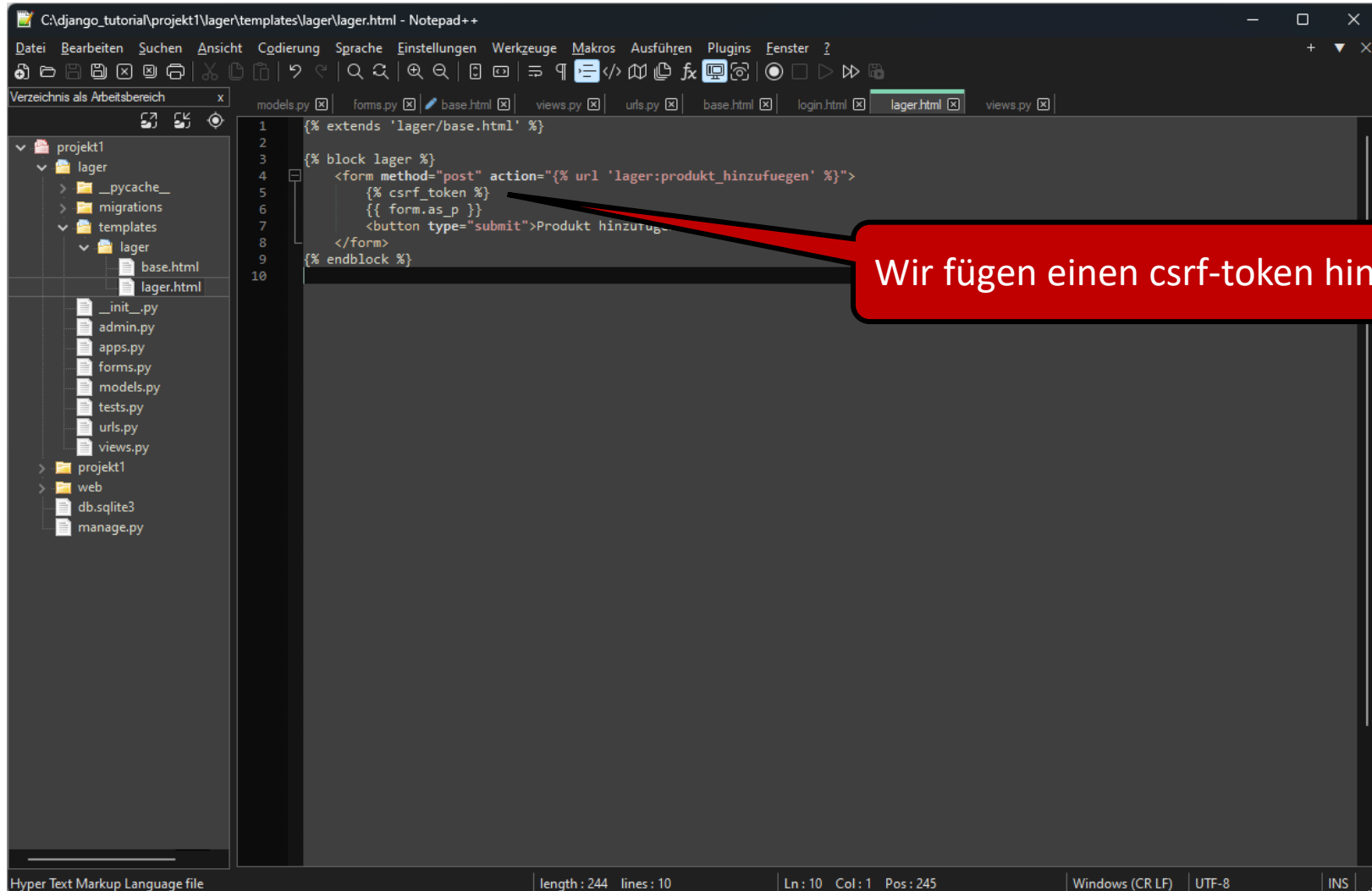
Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3
4 {% block lager %}
5     <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
6         {% csrf_token %}
7         {{ form.as_p }}
8         <button type="submit">Produkt hinzufügen</button>
9     </form>
10 {% endblock %}
```

Die URL geben wir mit den Django-Tag {% ... %} an. Dieser Tag wird beim Rendern der Seite mit der eindeutigen URL ersetzt. Die URL geben wir später in der urls.py an. Durch diesen Tag müssen wir uns um nichts mehr kümmern.

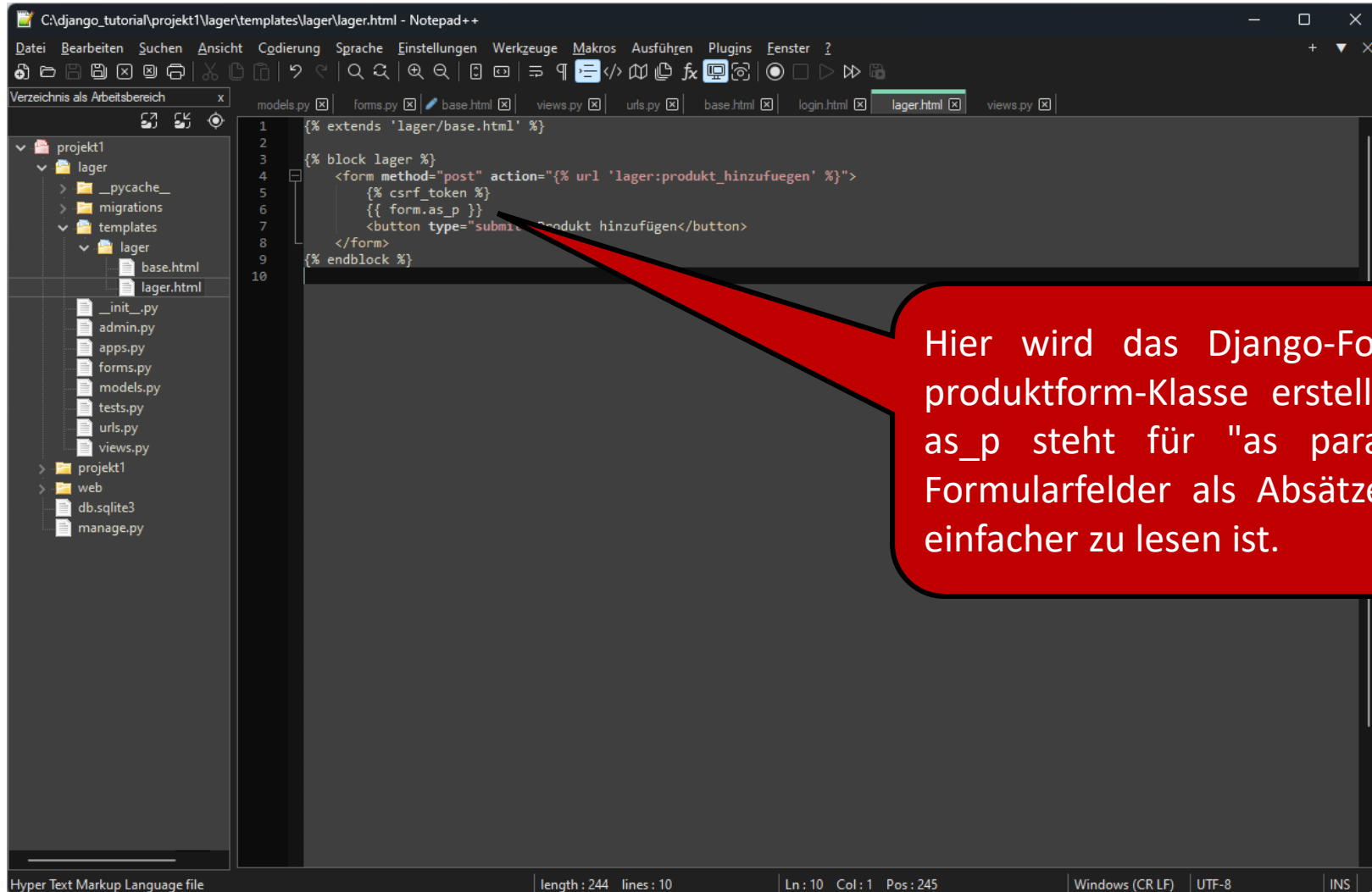
Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
5     {% csrf_token %}
6     {{ form.as_p }}
7     <button type="submit">Produkt hinzufuegen
8 </form>
9 {% endblock %}
10
```

Wir fügen einen csrf-token hinzu.

Modelle und Form

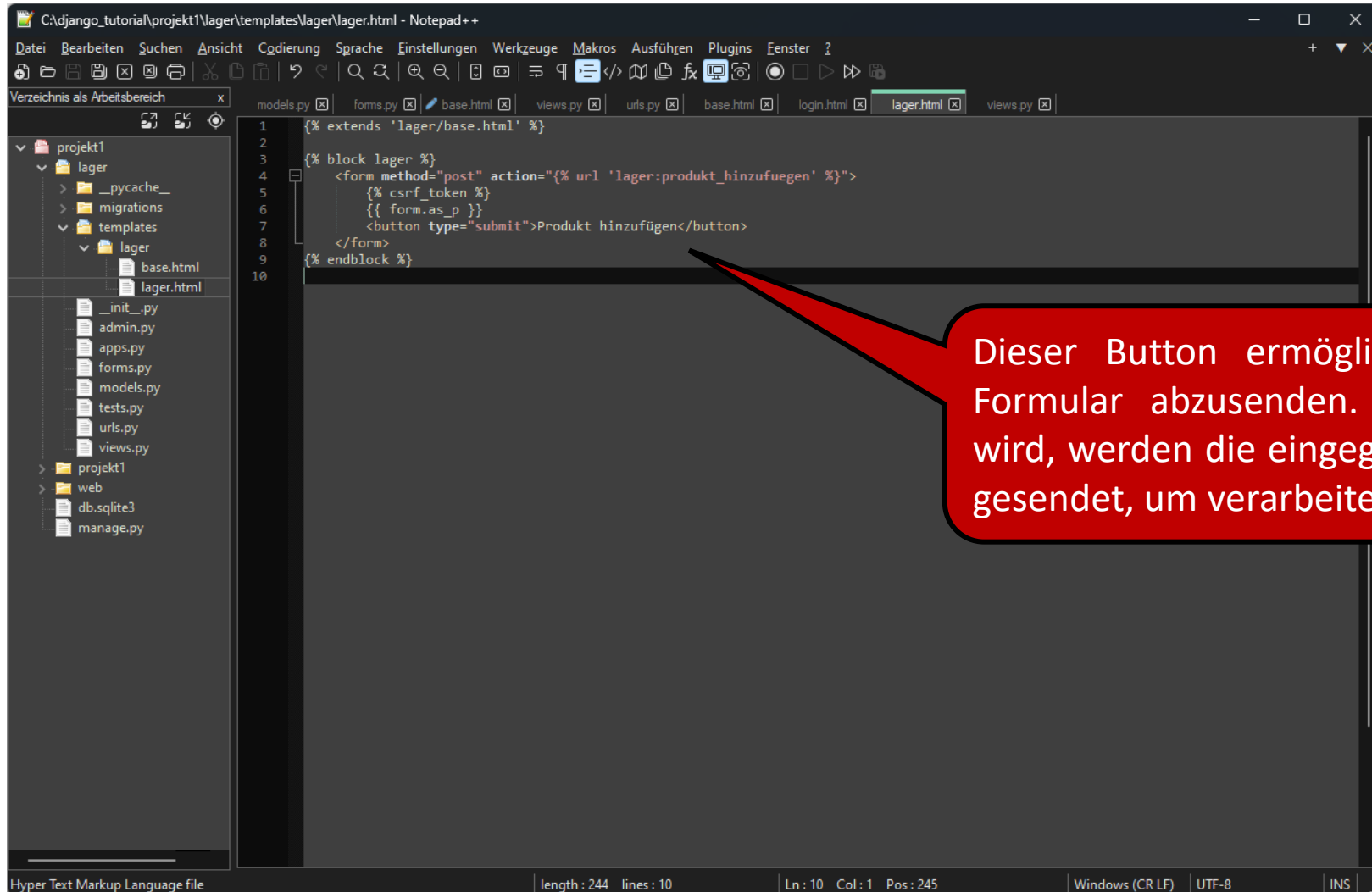


```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
5     {% csrf_token %}
6     {{ form.as_p }}
7     <button type="submit">Produkt hinzufügen</button>
8 </form>
9 {% endblock %}
10
```

Hier wird das Django-Formular, das wir in der produktform-Klasse erstellt haben, gerendert. Das as_p steht für "as paragraphs" und stellt die Formularfelder als Absätze dar, was üblicherweise einfacher zu lesen ist.

Hyper Text Markup Language | length : 244 lines : 10 | Ln : 10 Col : 1 Pos : 245 | Windows (CR LF) UTF-8 | INS

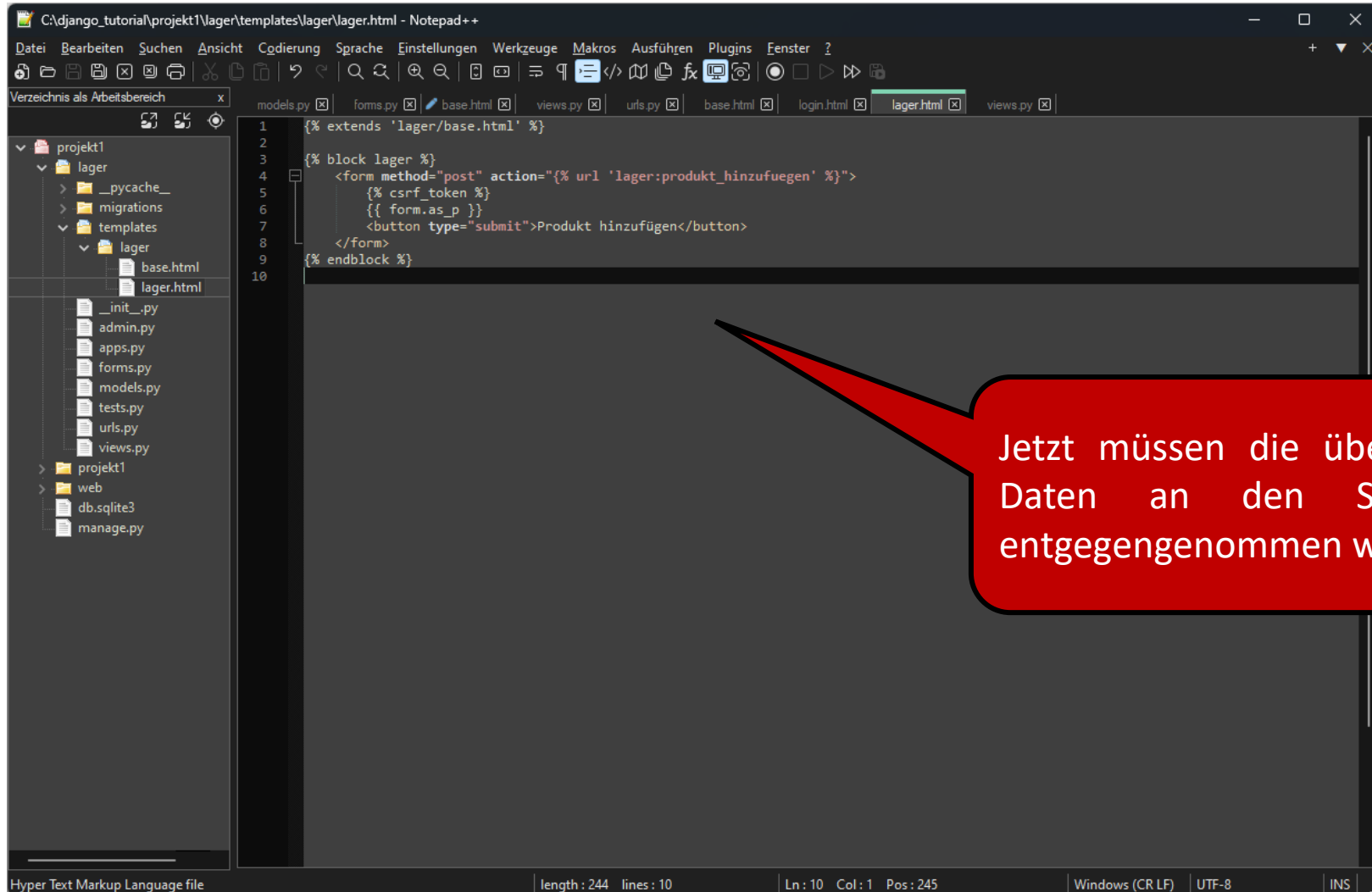
Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
5     {% csrf_token %}
6     {{ form.as_p }}
7     <button type="submit">Produkt hinzufügen</button>
8 </form>
9 {% endblock %}
10
```

Dieser Button ermöglicht es dem Benutzer, das Formular abzusenden. Wenn der Button geklickt wird, werden die eingegebenen Daten an den Server gesendet, um verarbeitet zu werden.

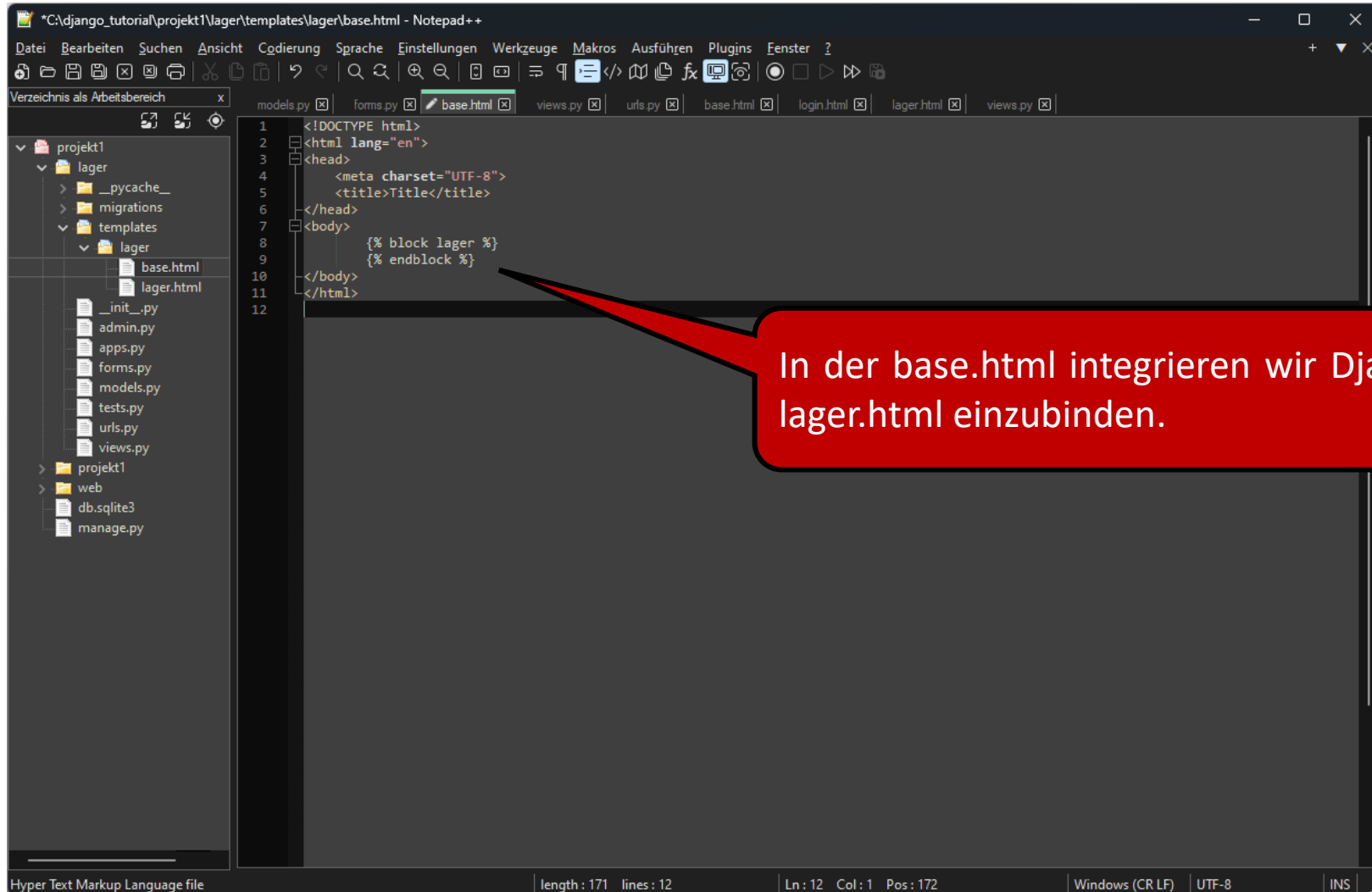
Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
5     {% csrf_token %}
6     {{ form.as_p }}
7     <button type="submit">Produkt hinzufügen</button>
8 </form>
9 {% endblock %}
10
```

Jetzt müssen die über diesen Button gesendeten Daten an den Server noch irgendwo in entgegengenommen werden...

Modelle und Form



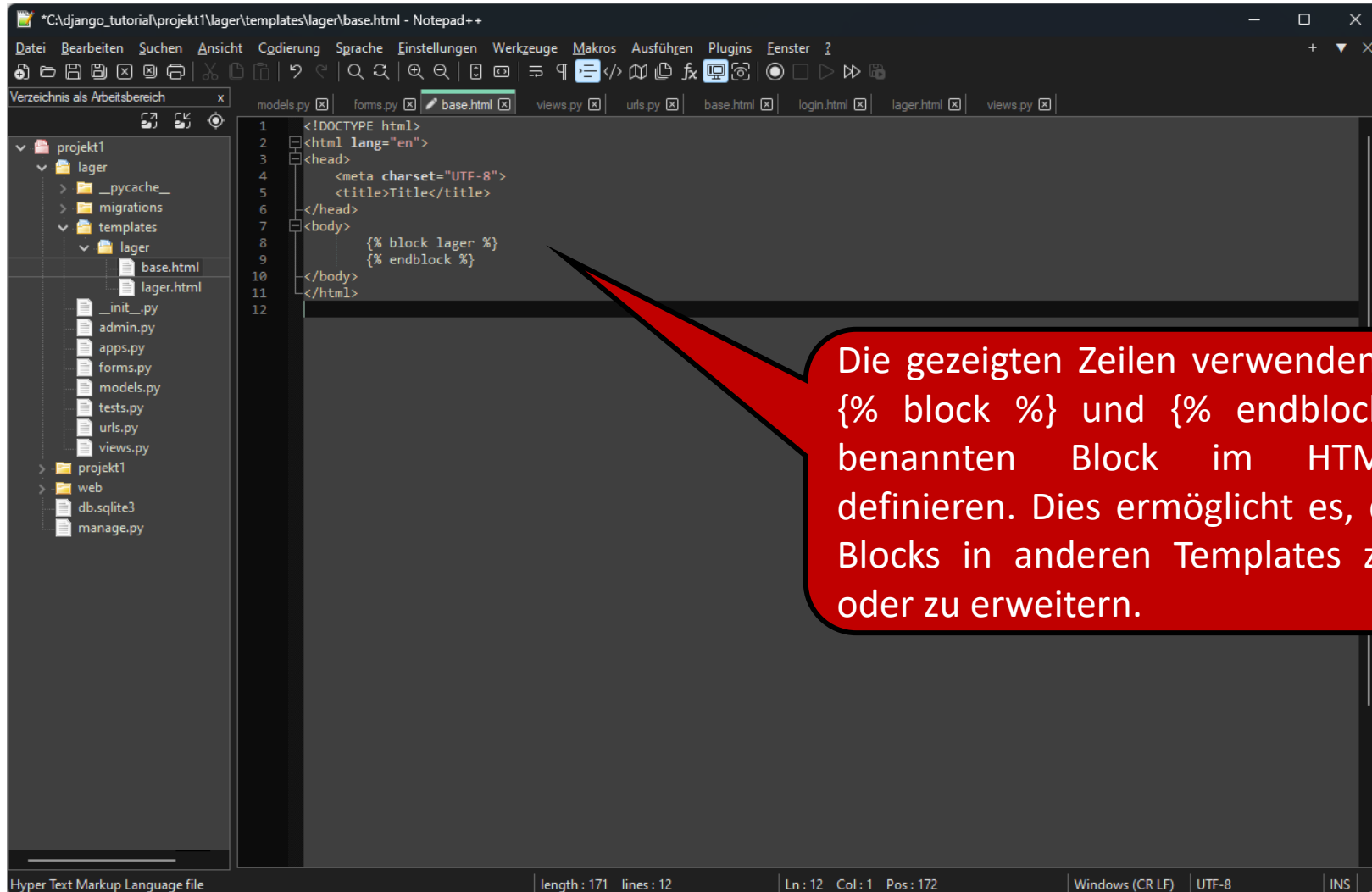
The screenshot shows a Notepad++ window with the file path `*C:\django_tutorial\projekt1\lager\templates\lager\base.html`. The left sidebar displays a file explorer for the `projekt1` directory, highlighting the `lager` subdirectory and its `templates` folder. The main editor area shows the content of `base.html`:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   {% block lager %}
9   {% endblock %}
10 </body>
11 </html>
12
```

A red callout bubble with a black border points to the `{% block lager %}` tag on line 8, containing the text: "In der base.html integrieren wir Django-Tags, um die lager.html einzubinden."

The status bar at the bottom indicates: "Hyper Text Markup Language | length : 171 lines : 12 | Ln : 12 Col : 1 Pos : 172 | Windows (CR LF) | UTF-8 | INS

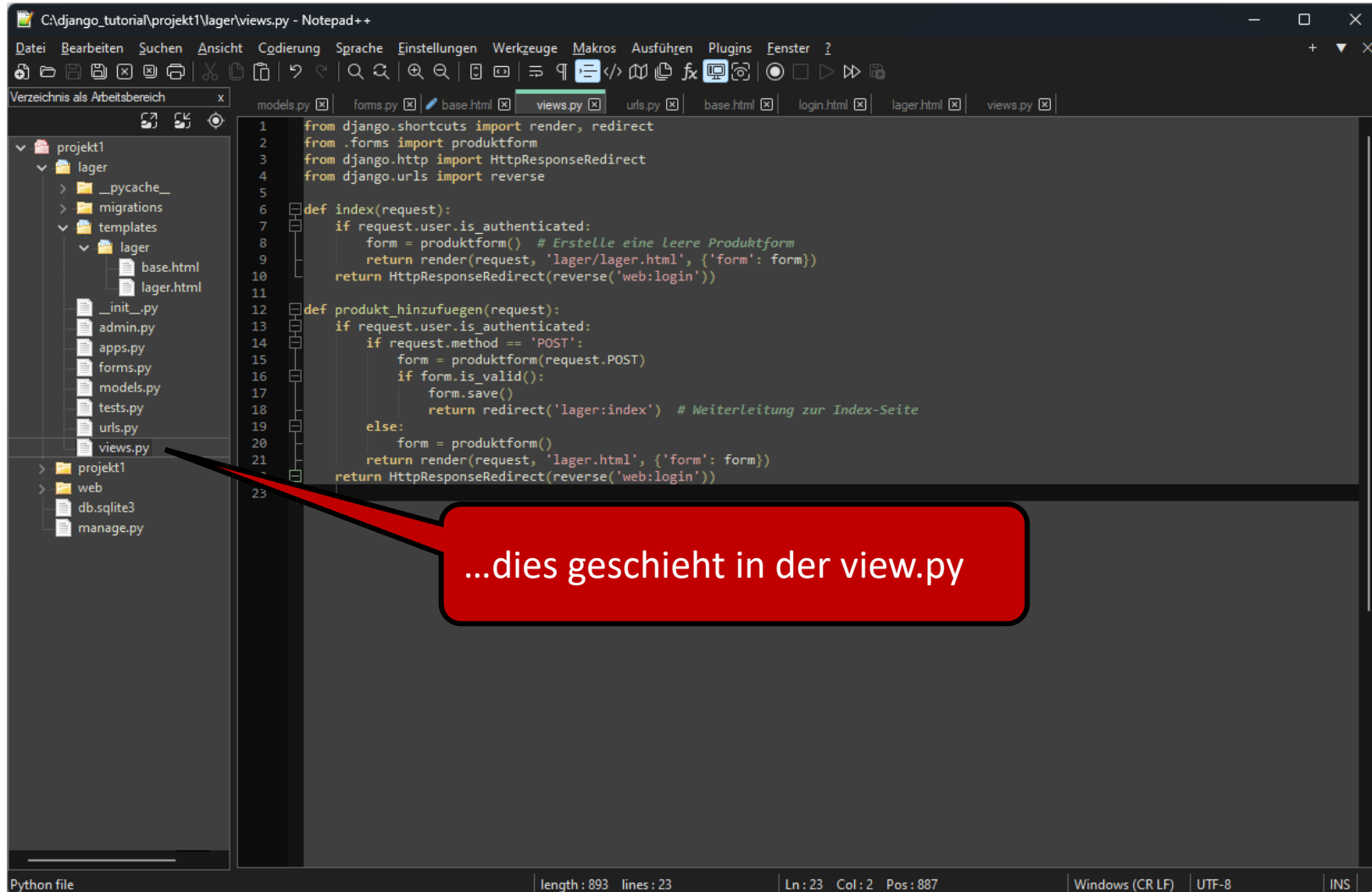
Modelle und Form



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   {% block lager %}
9   {% endblock %}
10 </body>
11 </html>
12
```

Die gezeigten Zeilen verwenden die Django-Tags `{% block %}` und `{% endblock %}`, um einen benannten Block im HTML-Template zu definieren. Dies ermöglicht es, den Inhalt dieses Blocks in anderen Templates zu überschreiben oder zu erweitern.

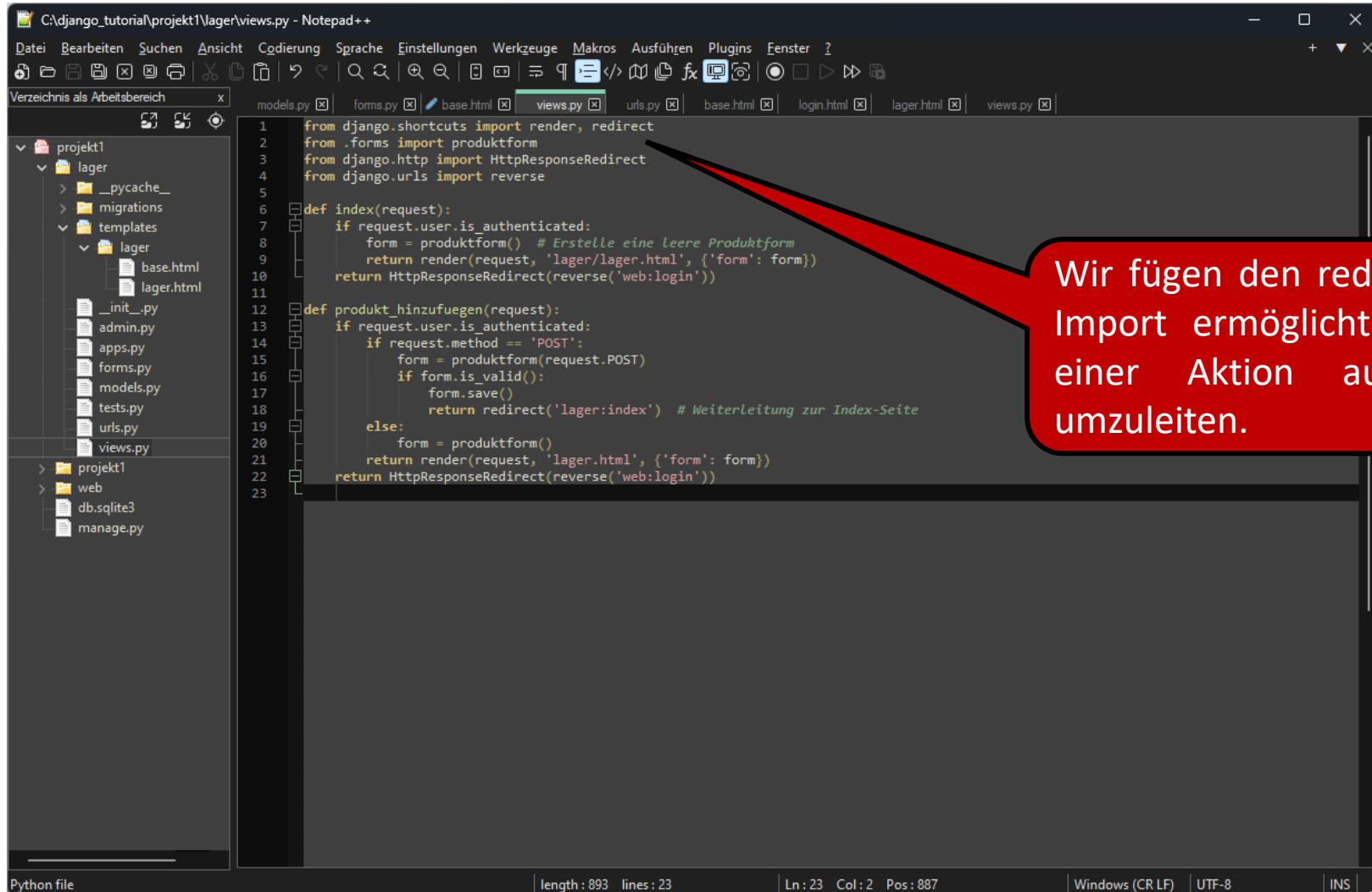
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19         else:
20             form = produktform()
21             return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

...dies geschieht in der view.py

Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19         else:
20             form = produktform()
21             return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Wir fügen den redirect import hinzu. Dieser Import ermöglicht es, den Benutzer nach einer Aktion auf eine andere Seite umzuleiten.

Modelle und Form

```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19             else:
20                 form = produktform()
21                 return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

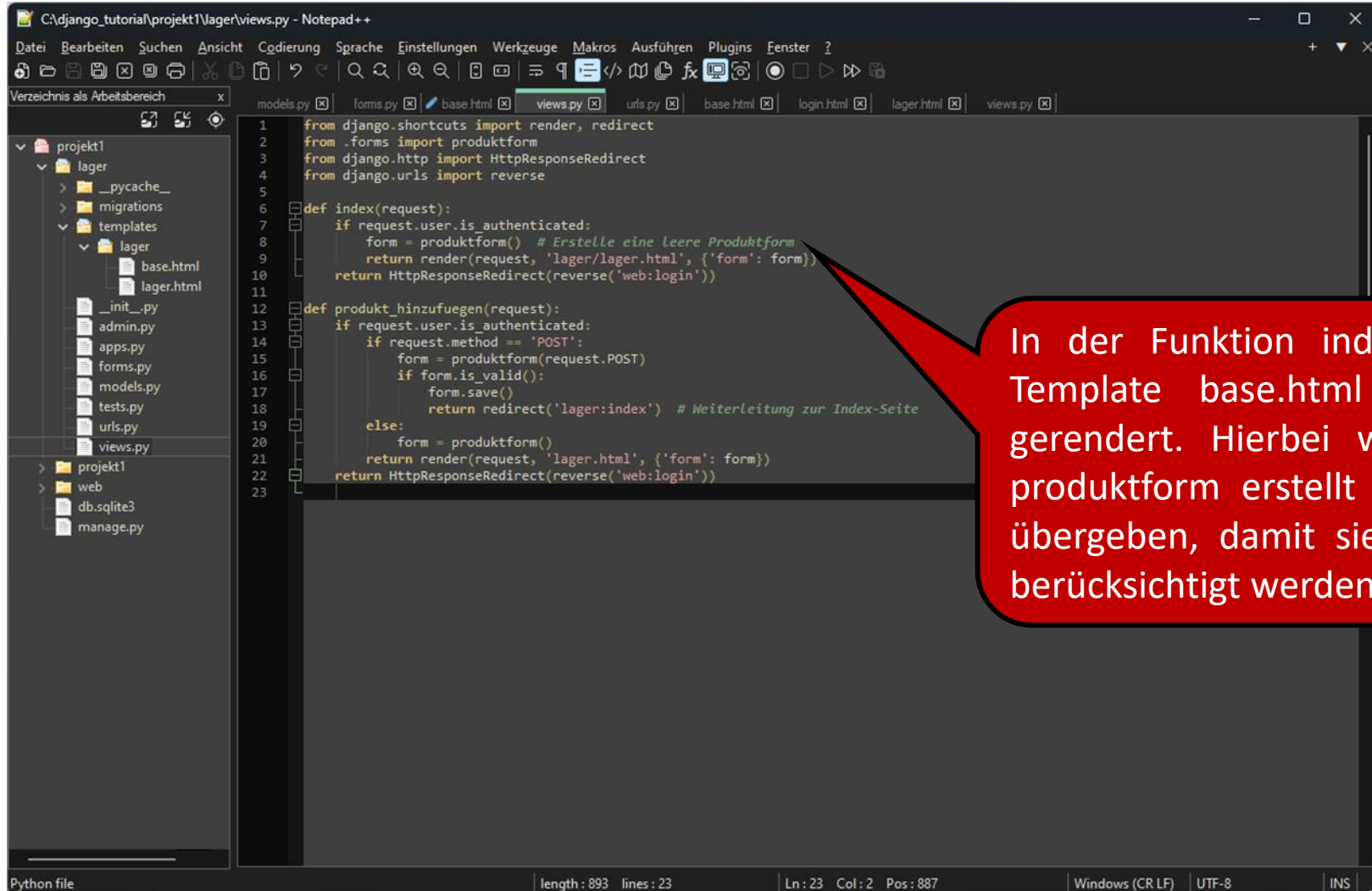
Hier wird das Formular produktform importiert, das wir zuvor definiert haben.

Modelle und Form

```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19         else:
20             form = produktform()
21             return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Zusätzlich fügen wir HttpResponseRedirect und reverse hinzu. Dies ermöglicht es uns, nicht eingeloggte Benutzer direkt auf die Login-Seite umzuleiten.

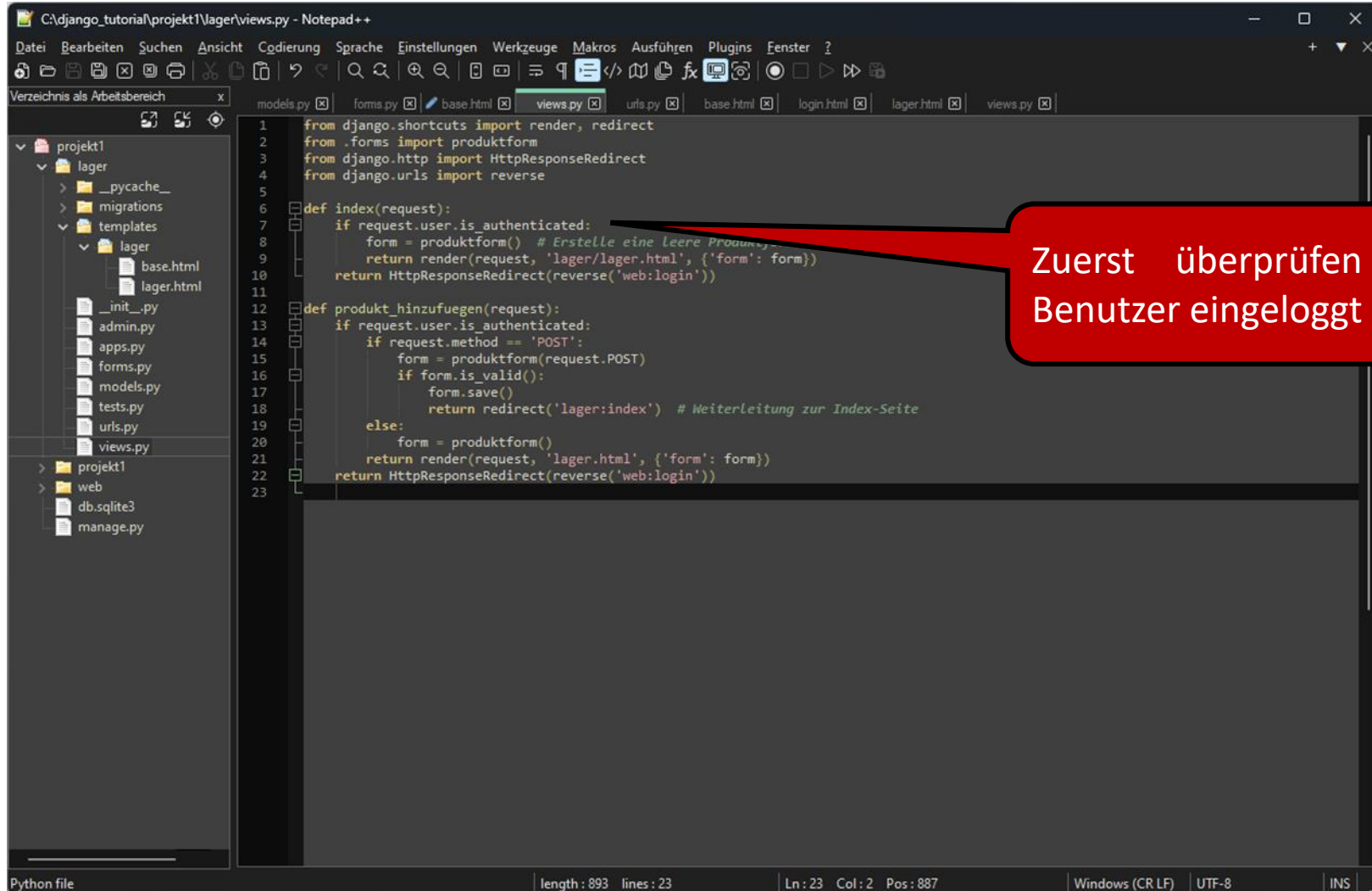
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10        return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19             else:
20                 form = produktform()
21                 return render(request, 'lager.html', {'form': form})
22     return HttpResponseRedirect(reverse('web:login'))
23
```

In der Funktion `index()` wird das HTML-Template `base.html` aus der Lager-App gerendert. Hierbei wird eine Instanz der `produktform` erstellt und an das Template übergeben, damit sie im Renderingprozess berücksichtigt werden kann.

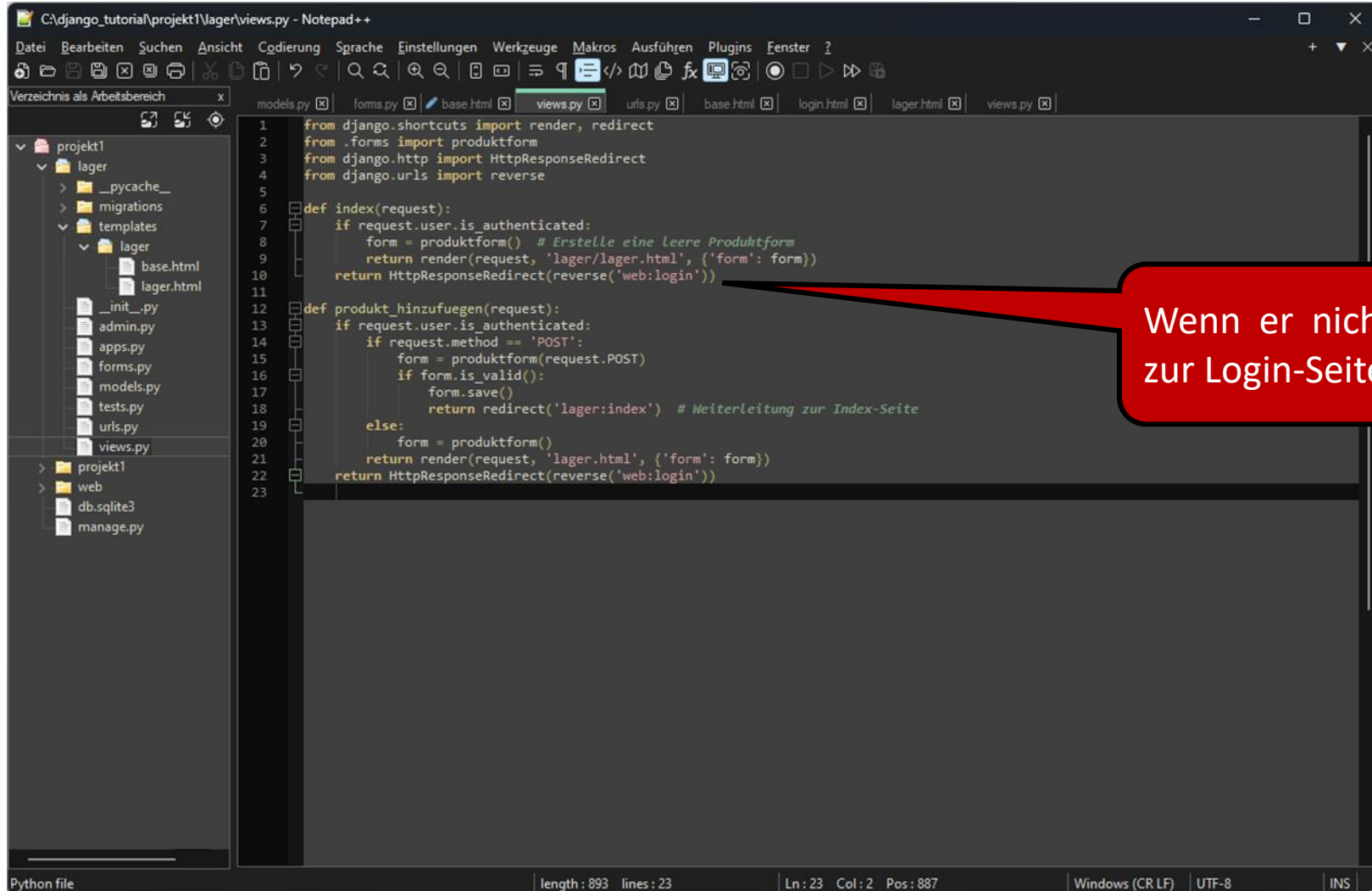
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19         else:
20             form = produktform()
21             return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Zuerst überprüfen wir, jedoch ob der Benutzer eingeloggt ist.

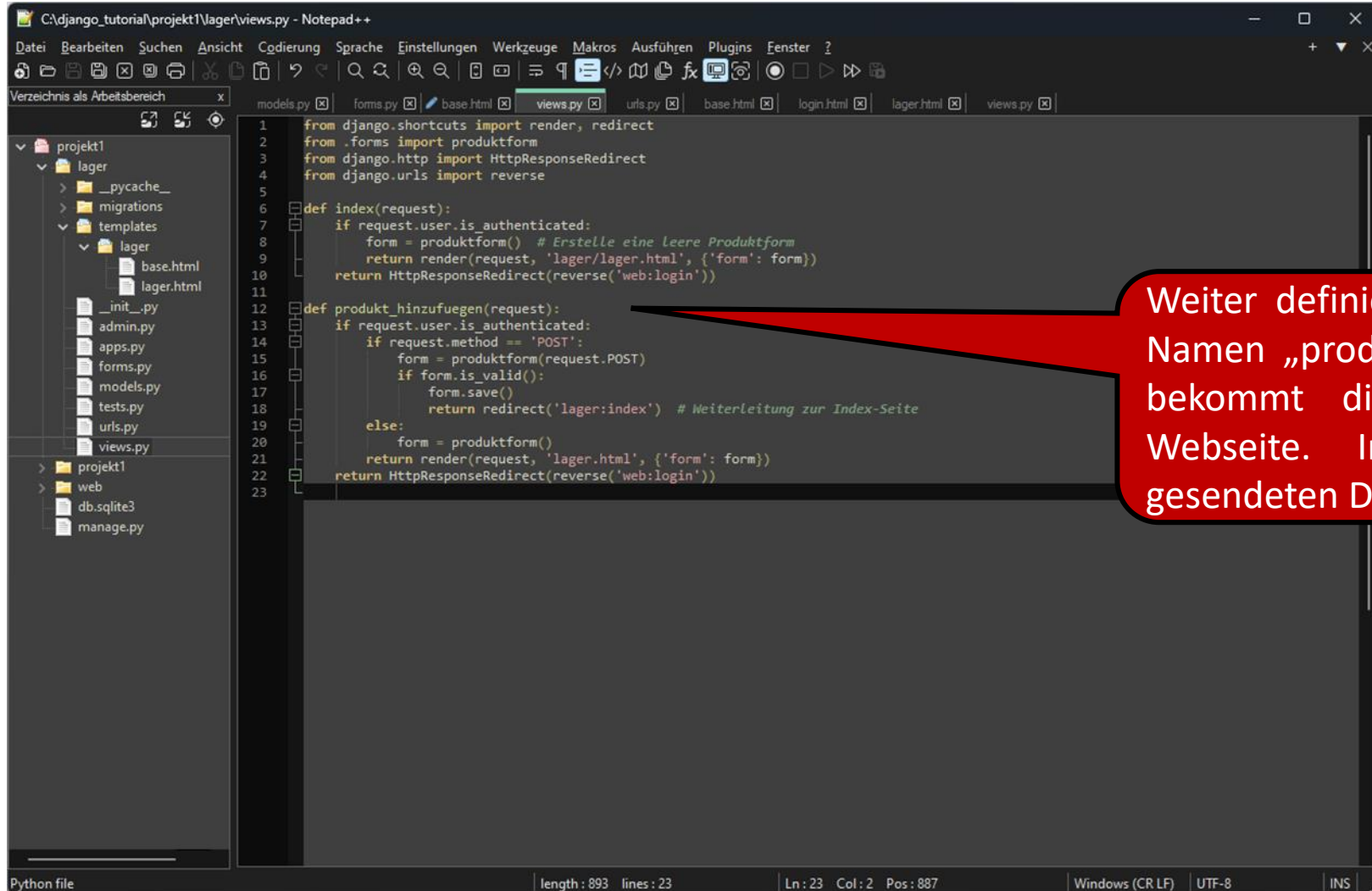
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19         else:
20             form = produktform()
21             return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Wenn er nicht eingeloggt ist, leiten wir ihn zur Login-Seite weiter.

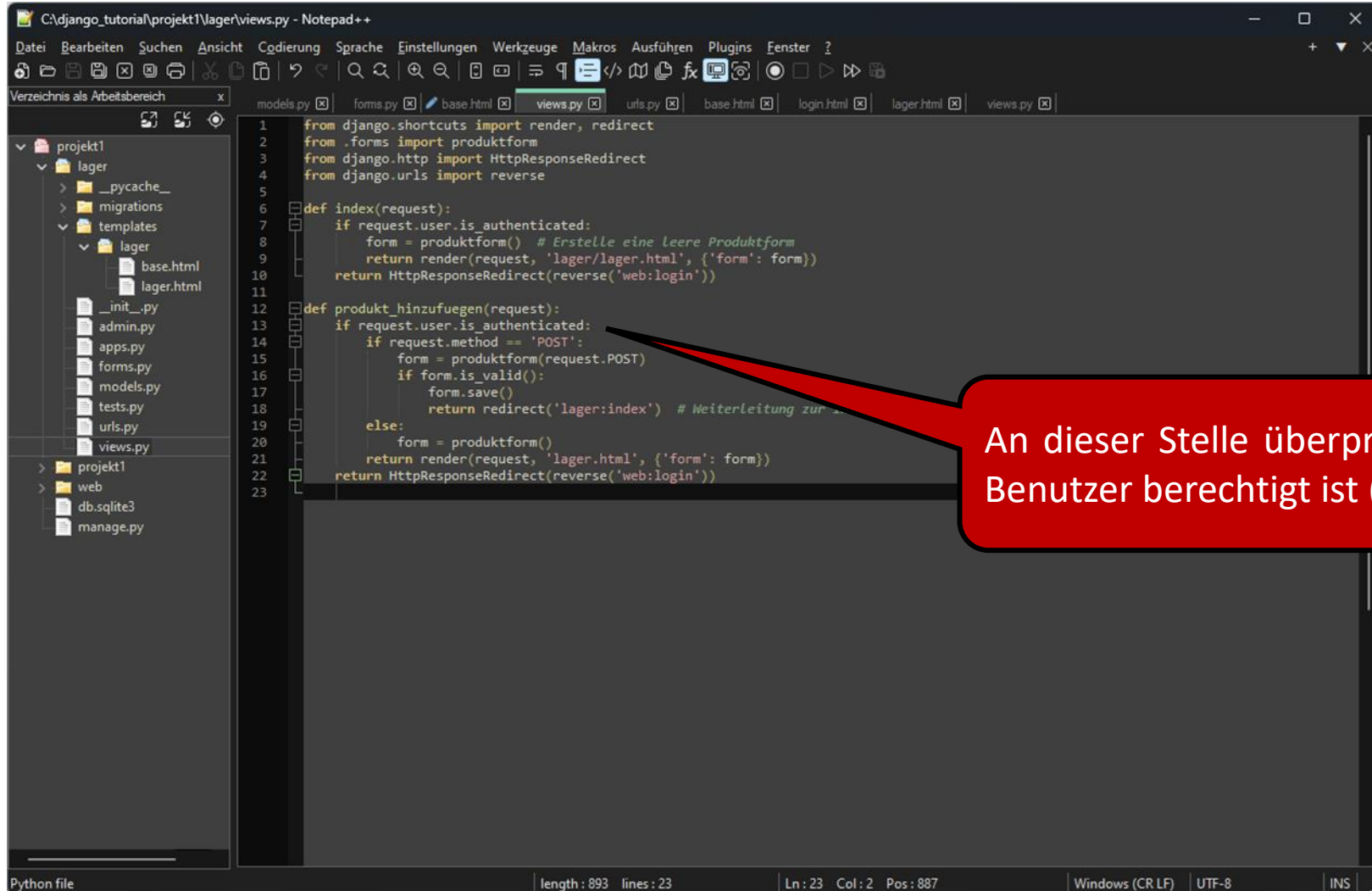
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19             else:
20                 form = produktform()
21                 return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Weiter definieren wir eine Funktion mit dem Namen „produkt_hinzufuegen“. Als Parameter bekommt die Funktion den request der Webseite. In diesen request sind alle gesendeten Daten der Website enthalten.

Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur ...
19         else:
20             form = produktform()
21             return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

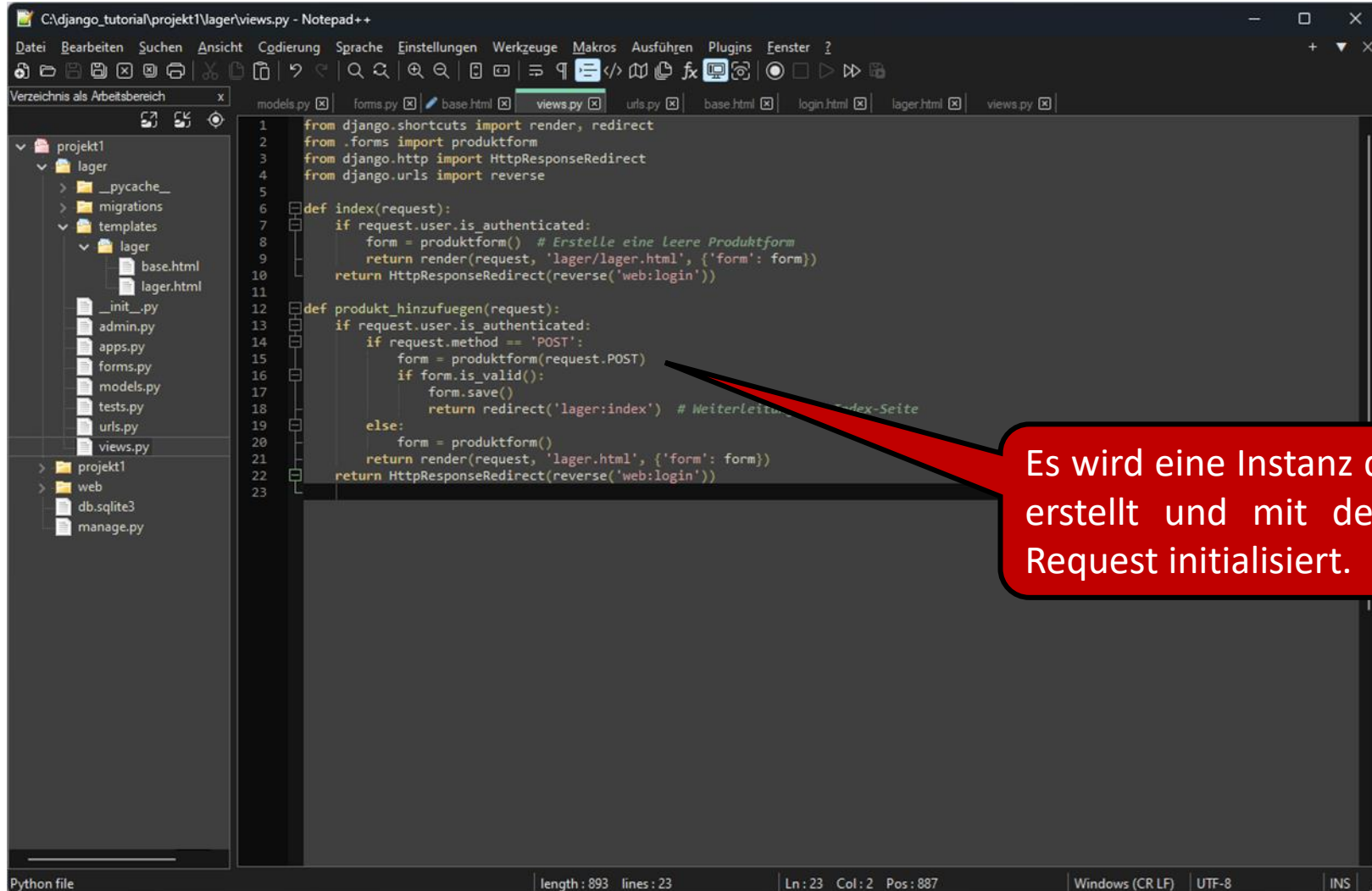
An dieser Stelle überprüfen wir erneut, ob der Benutzer berechtigt ist (eingeloggt ist).

Modelle und Form

```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19         else:
20             form = produktform()
21             return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Diese Bedingung überprüft, ob die Anfrage an die View ein POST-Request ist, was bedeutet, dass das Formular abgesendet wurde.

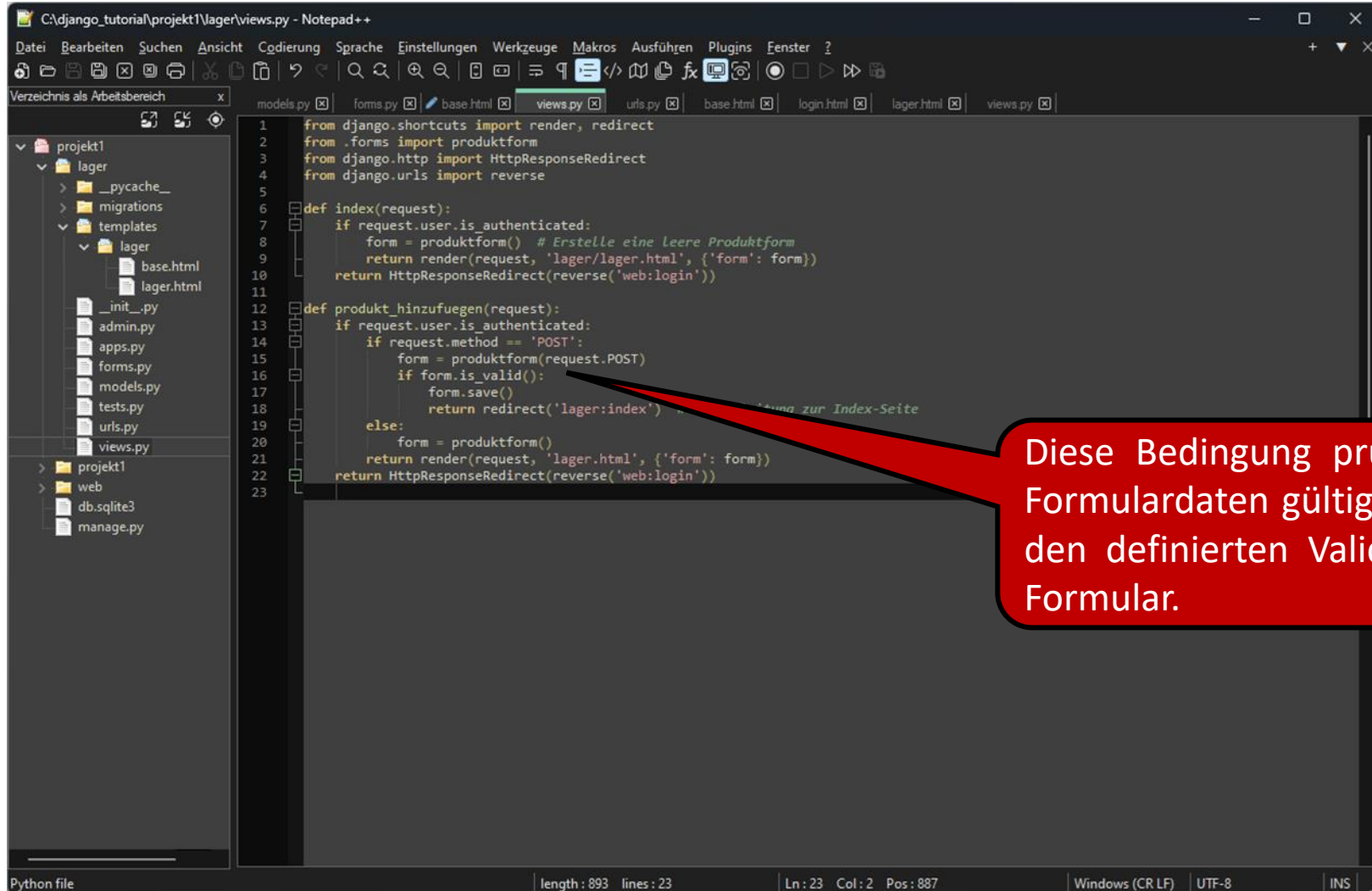
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung auf die Index-Seite
19             else:
20                 form = produktform()
21                 return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Es wird eine Instanz des produktform-Formulars erstellt und mit den Daten aus dem POST-Request initialisiert.

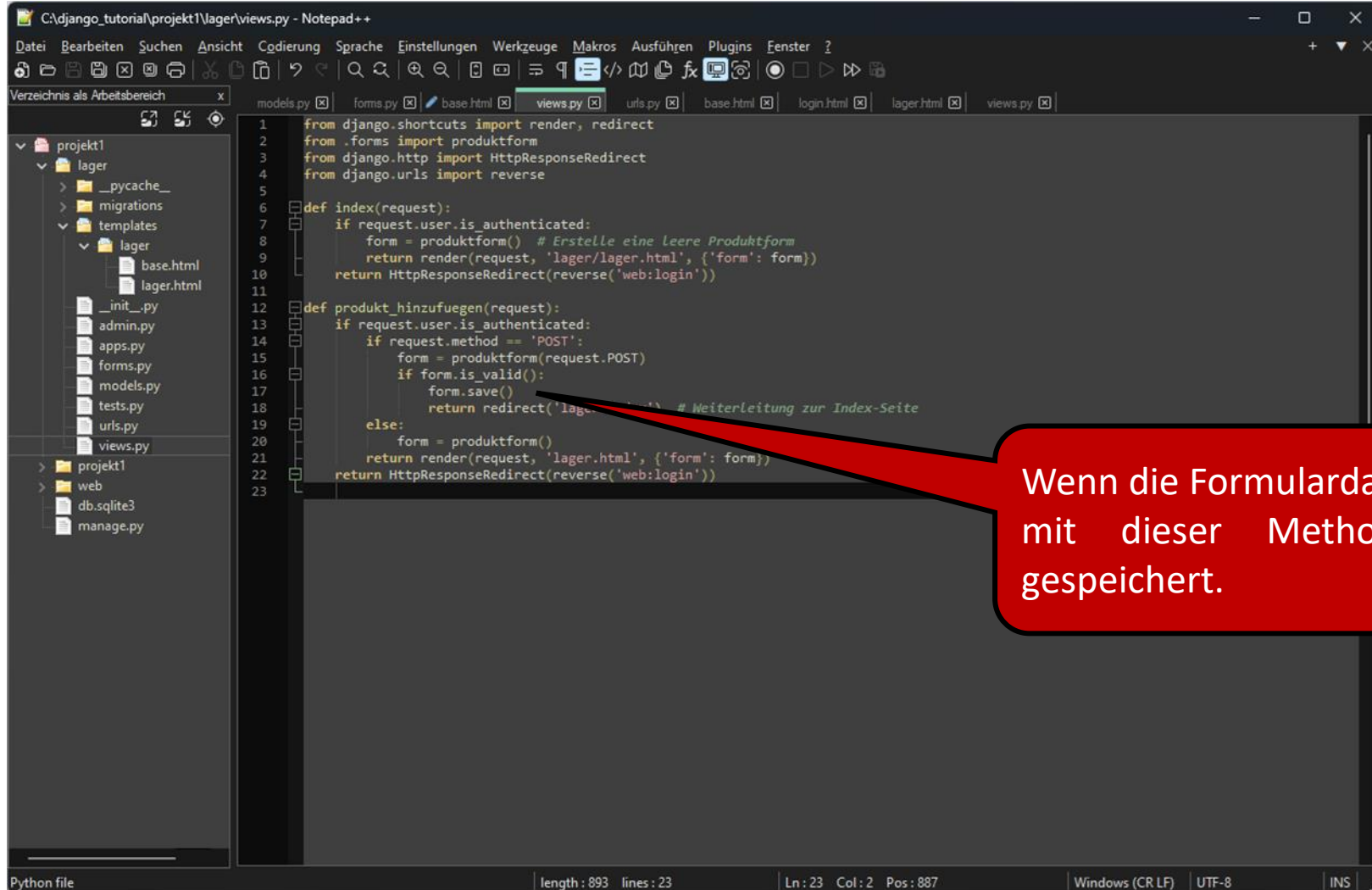
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Umleitung zur Index-Seite
19             else:
20                 form = produktform()
21                 return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Diese Bedingung prüft, ob die eingegebenen Formulardaten gültig sind, d.h. sie entsprechen den definierten Validierungsregeln in unseren Formular.

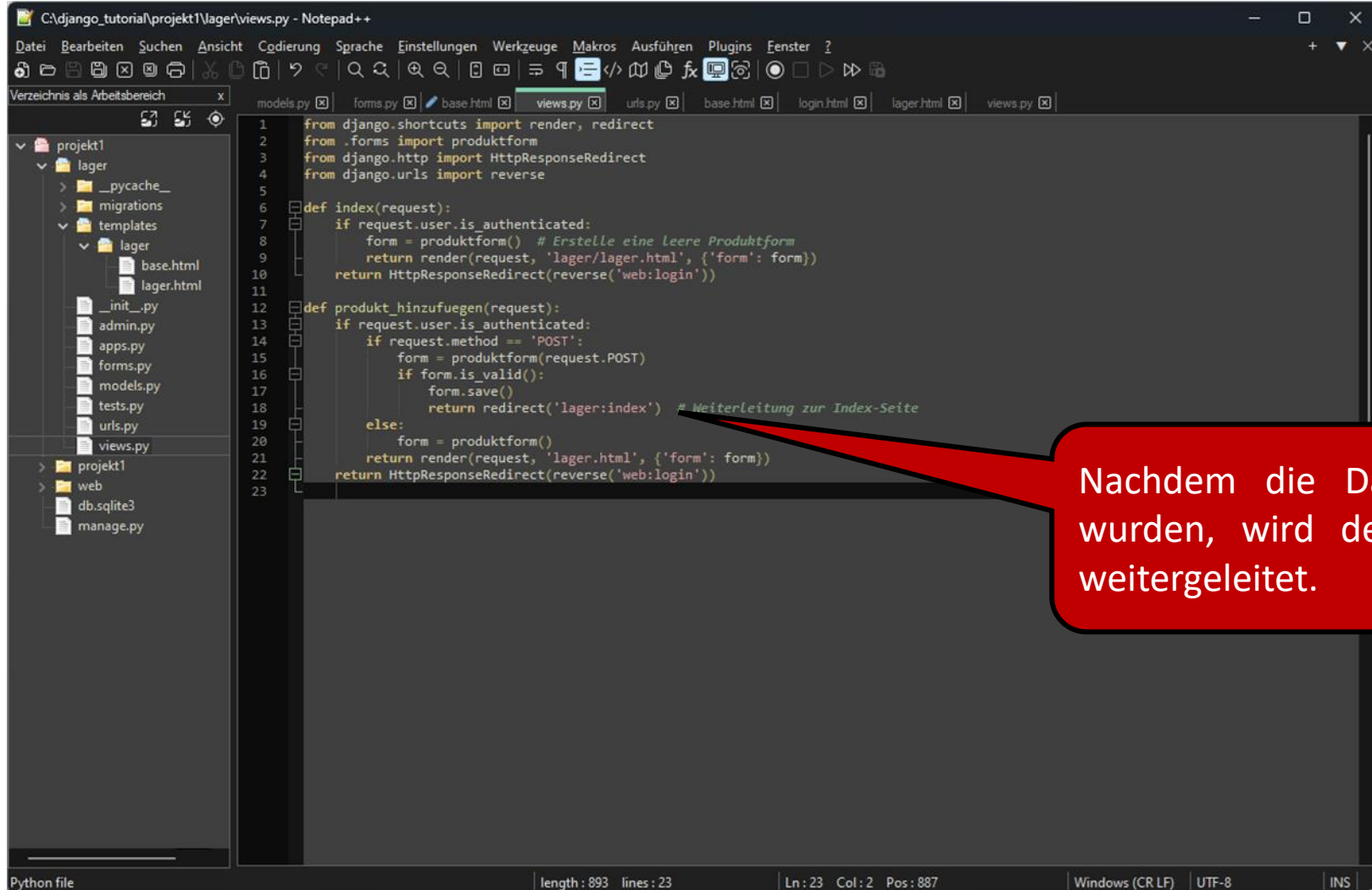
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return HttpResponseRedirect(reverse('web:login')) # Weiterleitung zur Index-Seite
19             else:
20                 form = produktform()
21                 return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Wenn die Formulardaten gültig sind, werden sie mit dieser Methode in die Datenbank gespeichert.

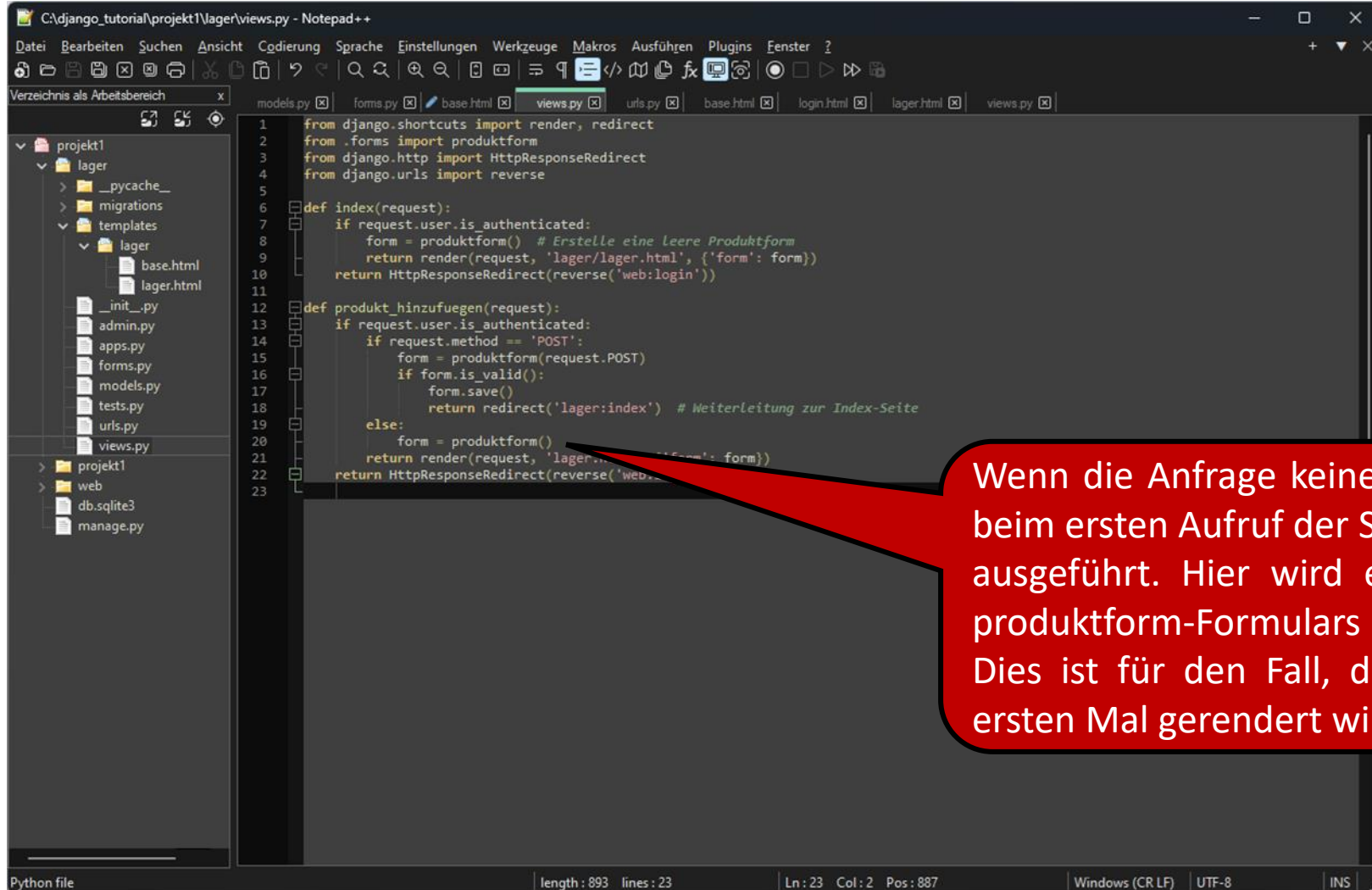
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19             else:
20                 form = produktform()
21                 return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Nachdem die Daten erfolgreich gespeichert wurden, wird der Benutzer zur Index Seite weitergeleitet.

Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19         else:
20             form = produktform()
21             return render(request, 'lager/lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

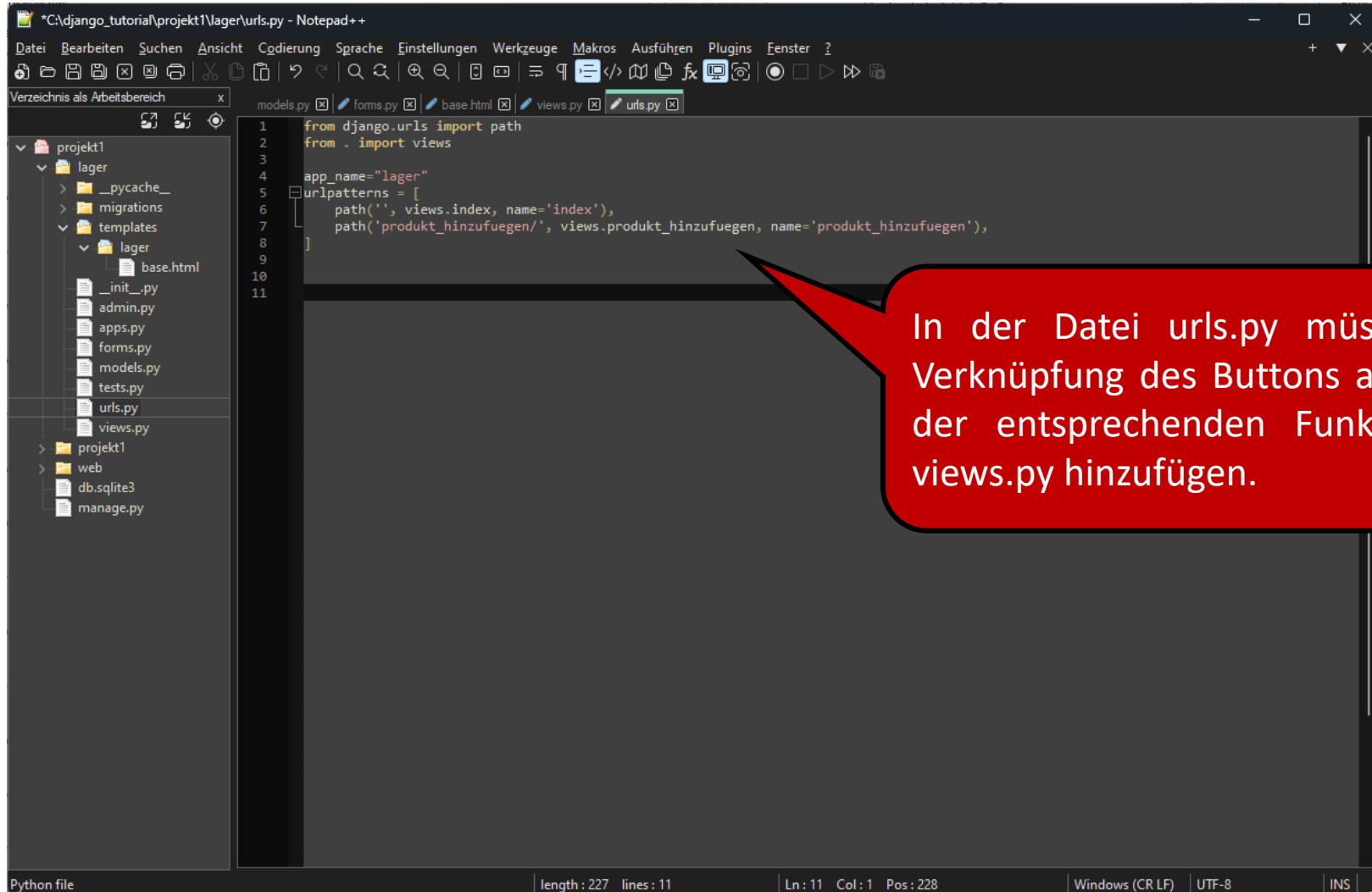
Wenn die Anfrage keine POST-Anfrage ist (z.B. beim ersten Aufruf der Seite), wird dieser Block ausgeführt. Hier wird eine neue Instanz des produktform-Formulars erstellt, ohne Daten. Dies ist für den Fall, dass das Formular zum ersten Mal gerendert wird.

Modelle und Form

```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5
6 def index(request):
7     if request.user.is_authenticated:
8         form = produktform() # Erstelle eine Leere Produktform
9         return render(request, 'lager/lager.html', {'form': form})
10    return HttpResponseRedirect(reverse('web:login'))
11
12 def produkt_hinzufuegen(request):
13     if request.user.is_authenticated:
14         if request.method == 'POST':
15             form = produktform(request.POST)
16             if form.is_valid():
17                 form.save()
18                 return redirect('lager:index') # Weiterleitung zur Index-Seite
19             else:
20                 form = produktform()
21                 return render(request, 'lager.html', {'form': form})
22    return HttpResponseRedirect(reverse('web:login'))
23
```

Hier wird das HTML-Template base.html gerendert und dabei das produktform-Formular als Kontext übergeben. Dadurch wird das Formular im Template angezeigt und der Benutzer kann Daten eingeben.

Modelle und Form



```
1 from django.urls import path
2 from . import views
3
4 app_name="lager"
5 urlpatterns = [
6     path('', views.index, name='index'),
7     path('produkt_hinzufuegen/', views.produkt_hinzufuegen, name='produkt_hinzufuegen'),
8 ]
9
10
11
```

In der Datei urls.py müssen wir noch die Verknüpfung des Buttons auf der Website mit der entsprechenden Funktion in der Datei views.py hinzufügen.

Modelle und Form

```
Eingabeaufforderung

(tutorial-env) C:\django_tutorial\projekt1>python manage.py makemigrations
No changes detected

(tutorial-env) C:\django_tutorial\projekt1>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, lager, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying lager.0001_initial... OK
  Applying sessions.0001_initial... OK

(tutorial-env) C:\django_tutorial\projekt1>
```

Wir erstellen die Datenbank mit den Befehlen makemigrations und migrate.

Modelle und Form



A screenshot of a web browser interface. The address bar shows the URL `127.0.0.1:8000/lager/`. Below the address bar, there is a header with a logo and the text "Meine Bibliothek |...". Below the header, there is a form with a label "Name:" followed by a text input field containing the word "Buch". Below the input field, there is a button labeled "Produkt hinzufügen".

Wir rufen über diesen Link die base.html der lager App auf. (Wir haben leider noch keinen Hyperlink eingerichtet... 😞)

Modelle und Form



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/lager/". Below the address bar, there is a navigation bar with a logo and the text "Meine Bibliothek |...". The main content area contains a form with a label "Name:" followed by a text input field containing the word "Buch". Below the input field is a button labeled "Produkt hinzufügen".

Hier wird die Django-Form gerendert, die wir geschrieben haben.

Modelle und Form

← → ↻ 127.0.0.1:8000/lager/

Meine Bibliothek [...]

Name:

Produkt hinzufügen

Senden wir den Eintrag ab, wird dieser in der Datenbank gespeichert.

DB Browser for SQLite - C:\django_tutorial\projekt1\ldb.sqlite3

Datei Bearbeiten Ansicht Werkzeuge Hilfe

Neue Datenbank Datenbank öffnen Änderungen schreiben Änderung...

Datenbankstruktur Daten durchsuchen Pragma bearbeiten SQL ausführen

Tabelle: lager_db_produkt

id	m_name
1	Buch

Modus: Text

1

Art der Daten in dieser Zelle: Text / Numerisch
1 Zeichen

Übernehmen

Entfernt

Identität Select an identity to connect

DBHub.io Lokal Current Database

Name Letzte Änderung

SQL-Log Diagramm DB-Schema Entfernt

UTF-8

Modelle und Form

← → ↻ 127.0.0.1:8000/lager/

Meine Bibliothek [...]

Name:

Produkt hinzufügen

Datenbankstruktur

Tabelle: lager_cls_produkt

id	m_name
1	Buch

Jetzt möchten wir sicherstellen, dass die Produkte auf der Seite angezeigt werden.

Art der Daten in dieser Zelle: Text / Numerisch
1 Zeichen

Entfernt

Identität: Select an identity to connect

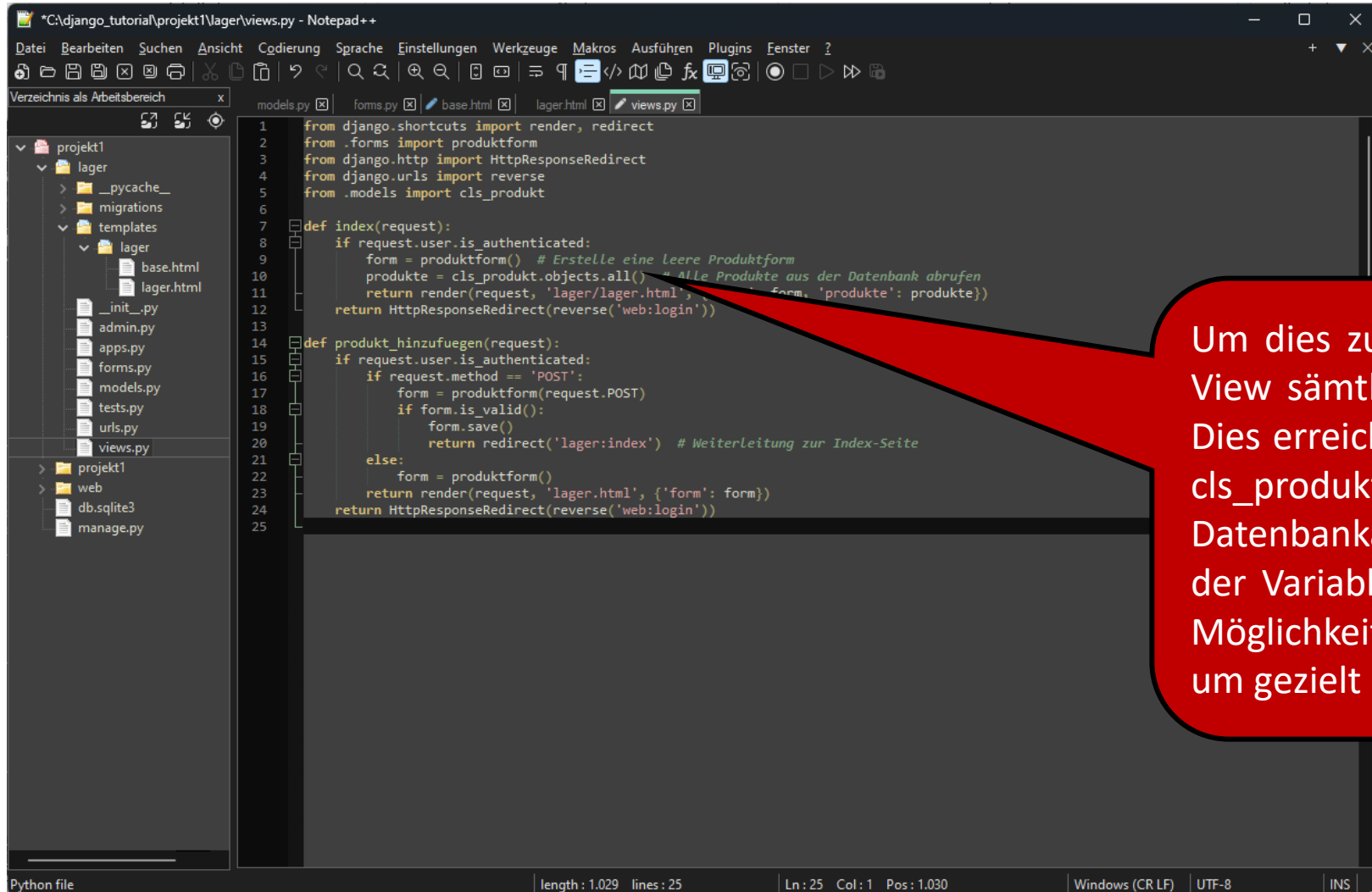
DBHub.io Lokal Current Database

Name Letzte Änderung

Springe zu: 1

SQL-Log Diagramm DB-Schema Entfernt

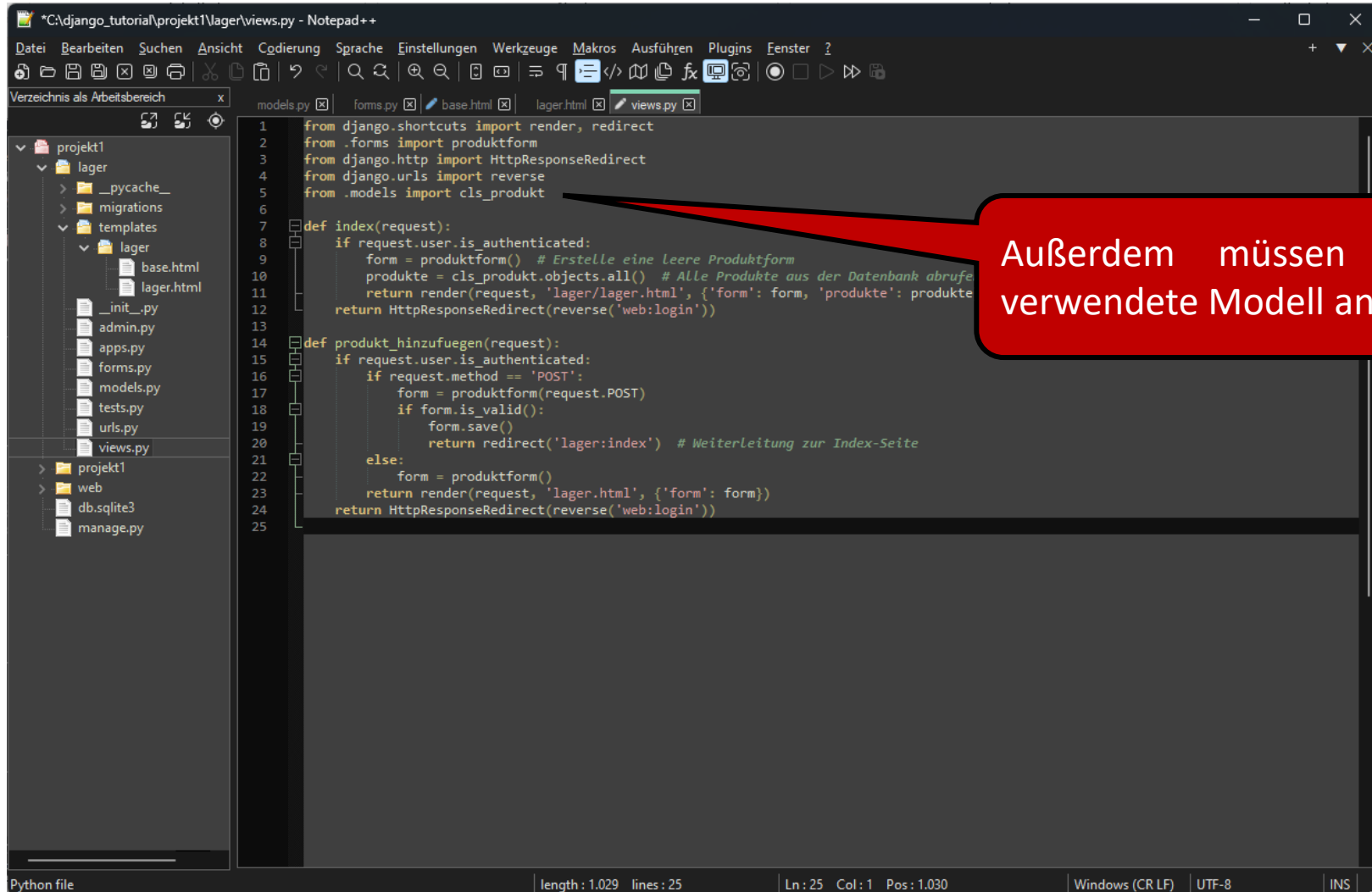
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5 from .models import cls_produkt
6
7 def index(request):
8     if request.user.is_authenticated:
9         form = produktform() # Erstelle eine leere Produktform
10        produkte = cls_produkt.objects.all() # Alle Produkte aus der Datenbank abrufen
11        return render(request, 'lager/lager.html', {'form': form, 'produkte': produkte})
12    return HttpResponseRedirect(reverse('web:login'))
13
14 def produkt_hinzufuegen(request):
15     if request.user.is_authenticated:
16         if request.method == 'POST':
17             form = produktform(request.POST)
18             if form.is_valid():
19                 form.save()
20                 return redirect('lager:index') # Weiterleitung zur Index-Seite
21         else:
22             form = produktform()
23             return render(request, 'lager.html', {'form': form})
24     return HttpResponseRedirect(reverse('web:login'))
25
```

Um dies zu erreichen, extrahieren wir in der View sämtliche Produkte aus der Datenbank. Dies erreichen wir durch die Verwendung von `cls_produkte.objects.all()`. Diese Datenbankabfrage speichert alle Einträge in der Variable "produkte". Es besteht auch die Möglichkeit, Filterfunktionen anzuwenden, um gezielt nach bestimmten Daten zu suchen.

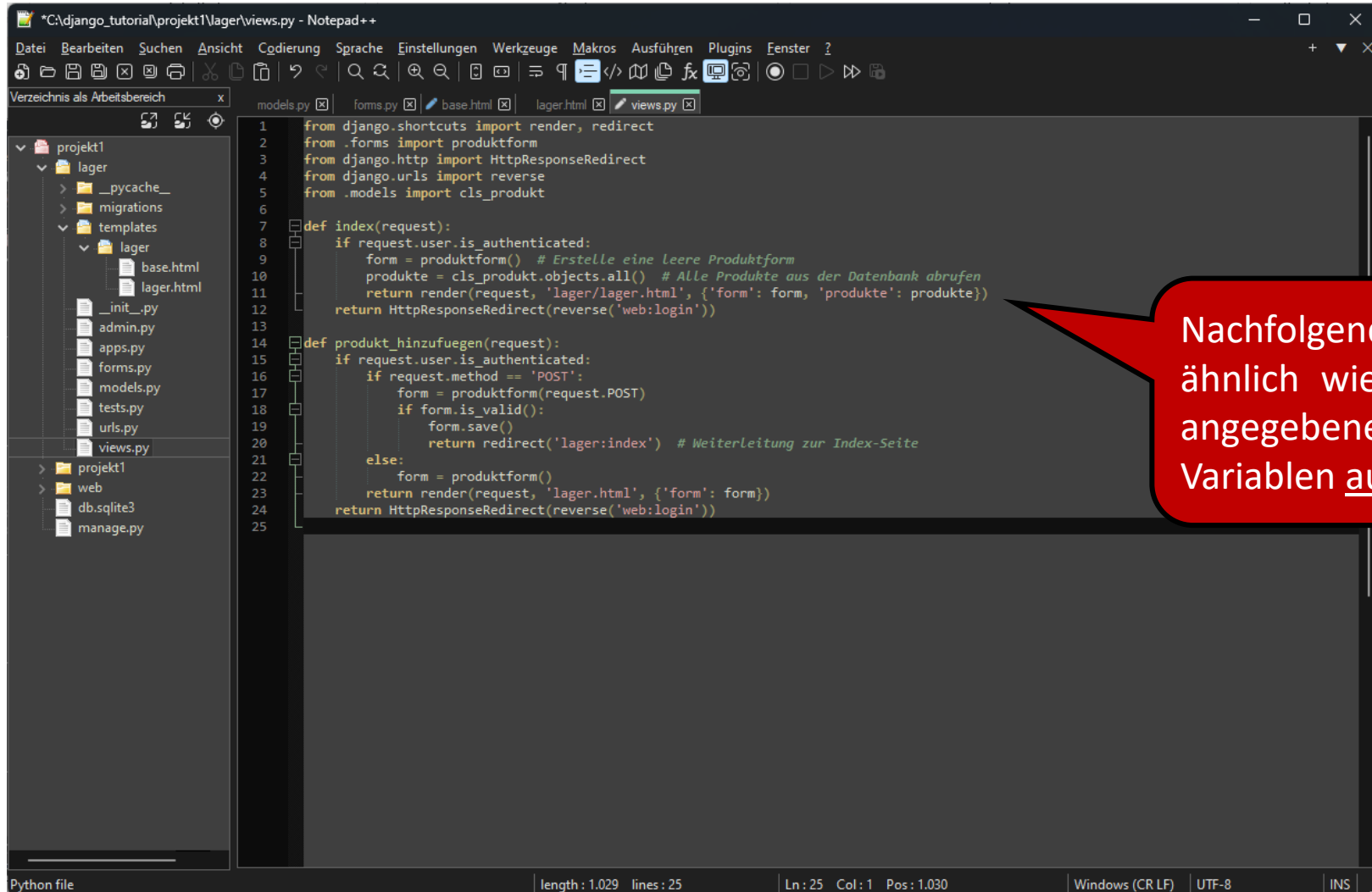
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5 from .models import cls_produkt
6
7 def index(request):
8     if request.user.is_authenticated:
9         form = produktform() # Erstelle eine leere Produktform
10        produkte = cls_produkt.objects.all() # Alle Produkte aus der Datenbank abrufe
11        return render(request, 'lager/lager.html', {'form': form, 'produkte': produkte})
12        return HttpResponseRedirect(reverse('web:login'))
13
14 def produkt_hinzufuegen(request):
15     if request.user.is_authenticated:
16         if request.method == 'POST':
17             form = produktform(request.POST)
18             if form.is_valid():
19                 form.save()
20                 return redirect('lager:index') # Weiterleitung zur Index-Seite
21         else:
22             form = produktform()
23             return render(request, 'lager.html', {'form': form})
24     return HttpResponseRedirect(reverse('web:login'))
25
```

Außerdem müssen wir noch das verwendete Modell angeben.

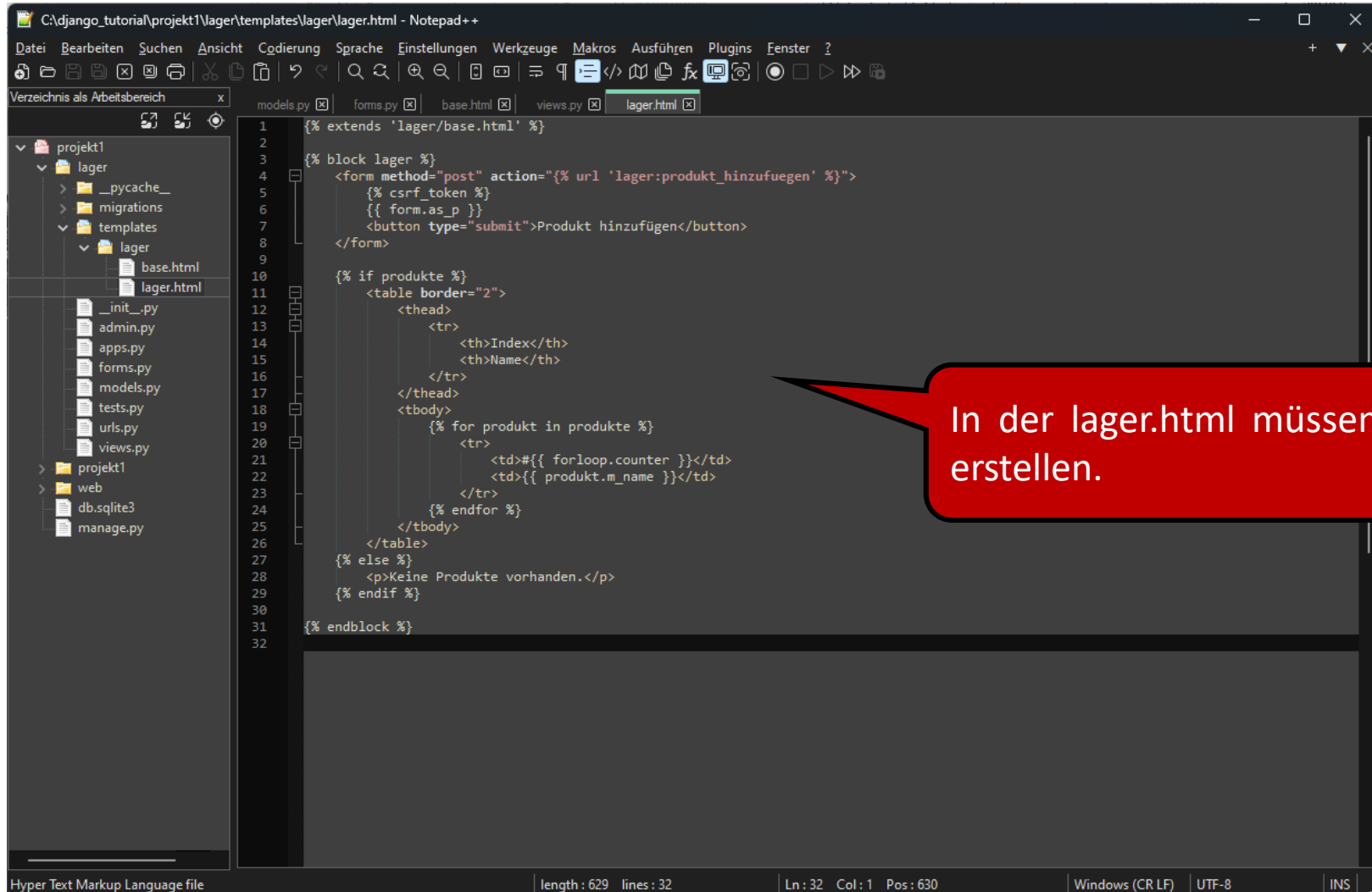
Modelle und Form



```
1 from django.shortcuts import render, redirect
2 from .forms import produktform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5 from .models import cls_produkt
6
7 def index(request):
8     if request.user.is_authenticated:
9         form = produktform() # Erstelle eine leere Produktform
10        produkte = cls_produkt.objects.all() # Alle Produkte aus der Datenbank abrufen
11        return render(request, 'lager/lager.html', {'form': form, 'produkte': produkte})
12        return HttpResponseRedirect(reverse('web:login'))
13
14 def produkt_hinzufuegen(request):
15     if request.user.is_authenticated:
16         if request.method == 'POST':
17             form = produktform(request.POST)
18             if form.is_valid():
19                 form.save()
20                 return redirect('lager:index') # Weiterleitung zur Index-Seite
21         else:
22             form = produktform()
23             return render(request, 'lager.html', {'form': form})
24     return HttpResponseRedirect(reverse('web:login'))
25
```

Nachfolgend übergeben wir die Variable, ähnlich wie die Form, an die Webseite. Der angegebene String stellt dabei den Namen der Variablen auf der Webseite dar.

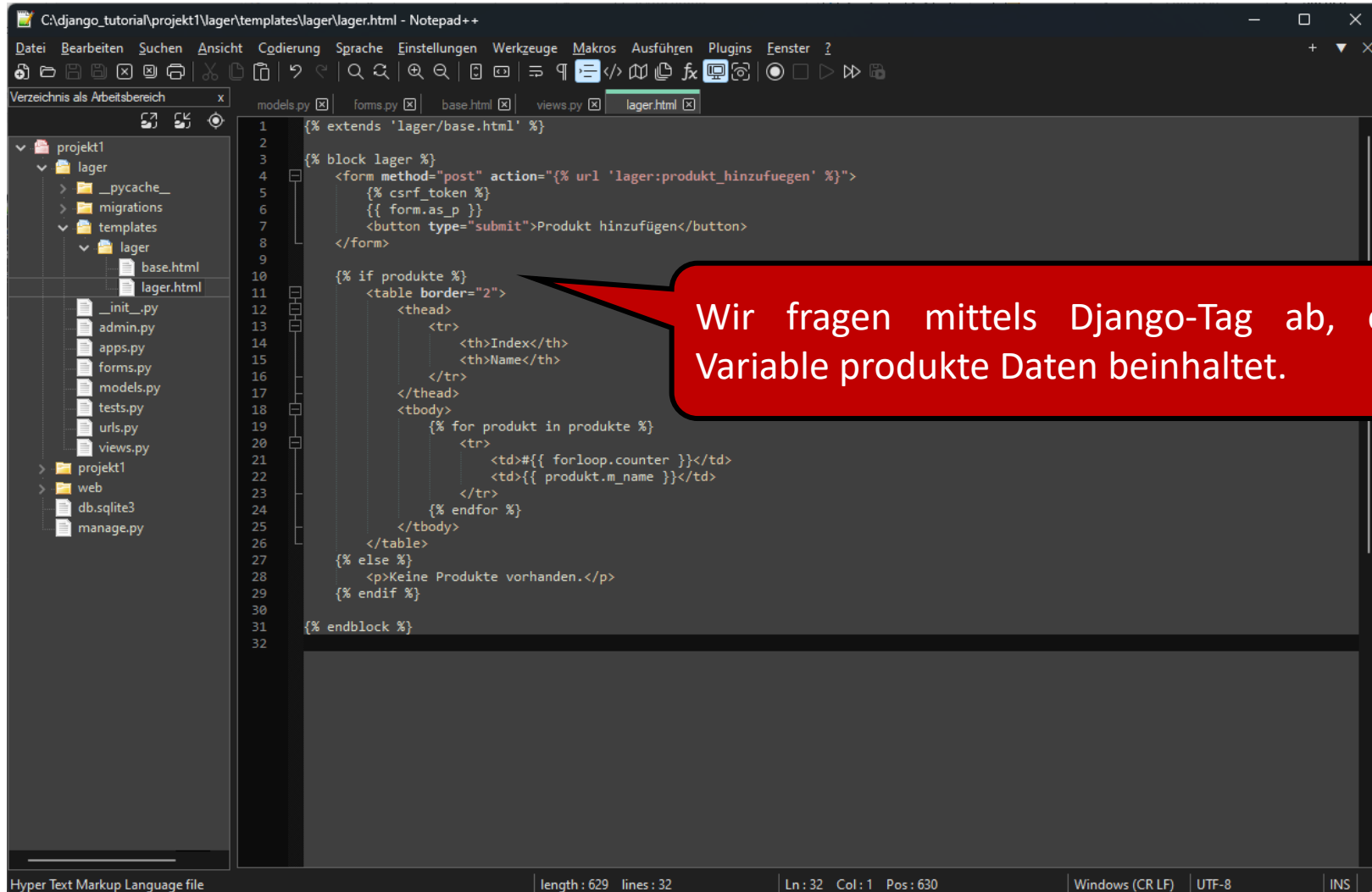
Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
5     {% csrf_token %}
6     {{ form.as_p }}
7     <button type="submit">Produkt hinzufügen</button>
8 </form>
9
10 {% if produkte %}
11 <table border="2">
12     <thead>
13         <tr>
14             <th>Index</th>
15             <th>Name</th>
16         </tr>
17     </thead>
18     <tbody>
19         {% for produkt in produkte %}
20             <tr>
21                 <td>#{{ forloop.counter }}</td>
22                 <td>{{ produkt.m_name }}</td>
23             </tr>
24         {% endfor %}
25     </tbody>
26 </table>
27 {% else %}
28 <p>Keine Produkte vorhanden.</p>
29 {% endif %}
30
31 {% endblock %}
32
```

In der lager.html müssen wir die Tabelle noch erstellen.

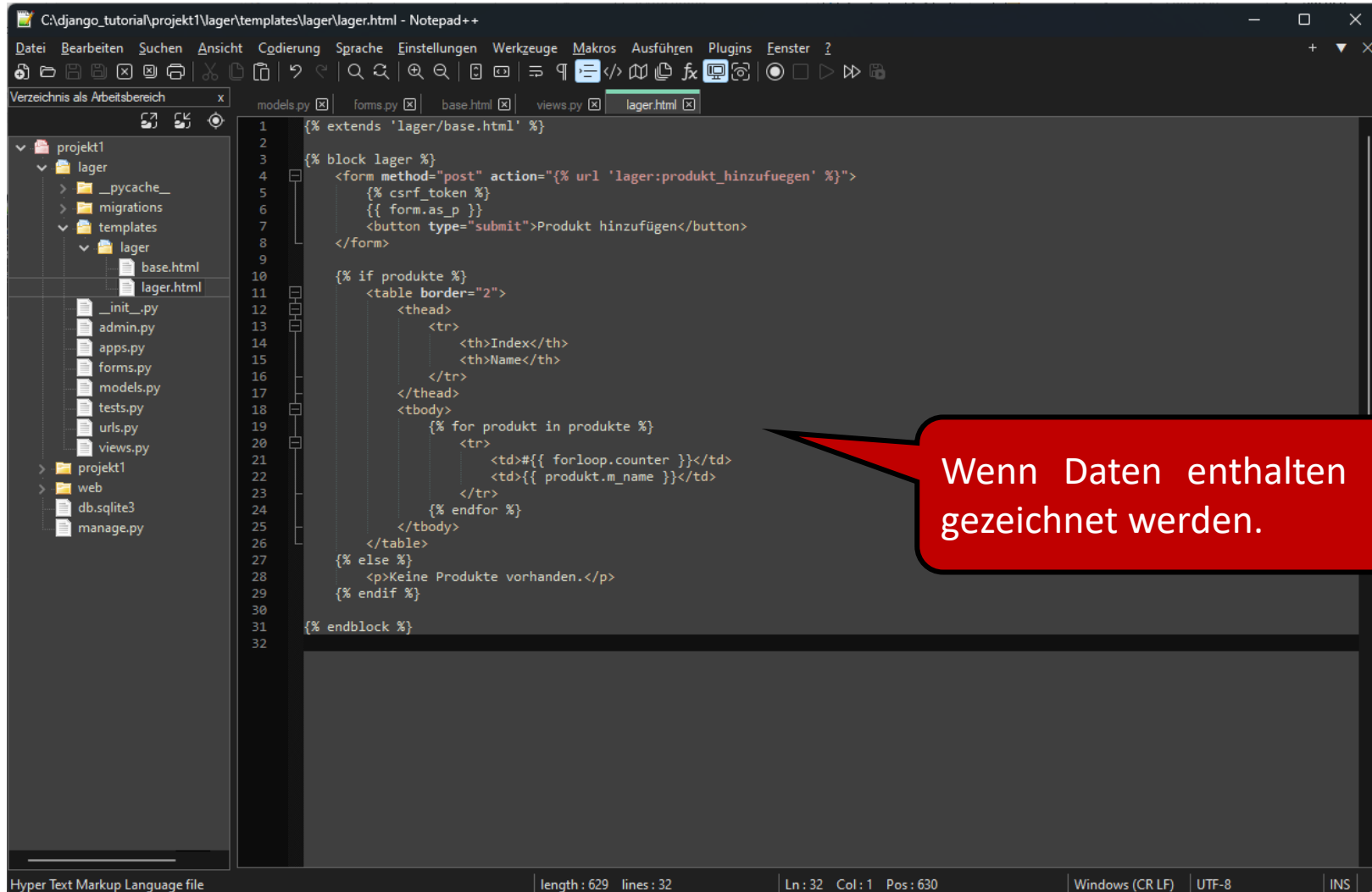
Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
5     {% csrf_token %}
6     {{ form.as_p }}
7     <button type="submit">Produkt hinzufügen</button>
8 </form>
9
10 {% if produkte %}
11 <table border="2">
12     <thead>
13         <tr>
14             <th>Index</th>
15             <th>Name</th>
16         </tr>
17     </thead>
18     <tbody>
19         {% for produkt in produkte %}
20             <tr>
21                 <td>#{{ forloop.counter }}</td>
22                 <td>{{ produkt.m_name }}</td>
23             </tr>
24         {% endfor %}
25     </tbody>
26 </table>
27 {% else %}
28 <p>Keine Produkte vorhanden.</p>
29 {% endif %}
30
31 {% endblock %}
32
```

Wir fragen mittels Django-Tag ab, ob die Variable produkte Daten beinhaltet.

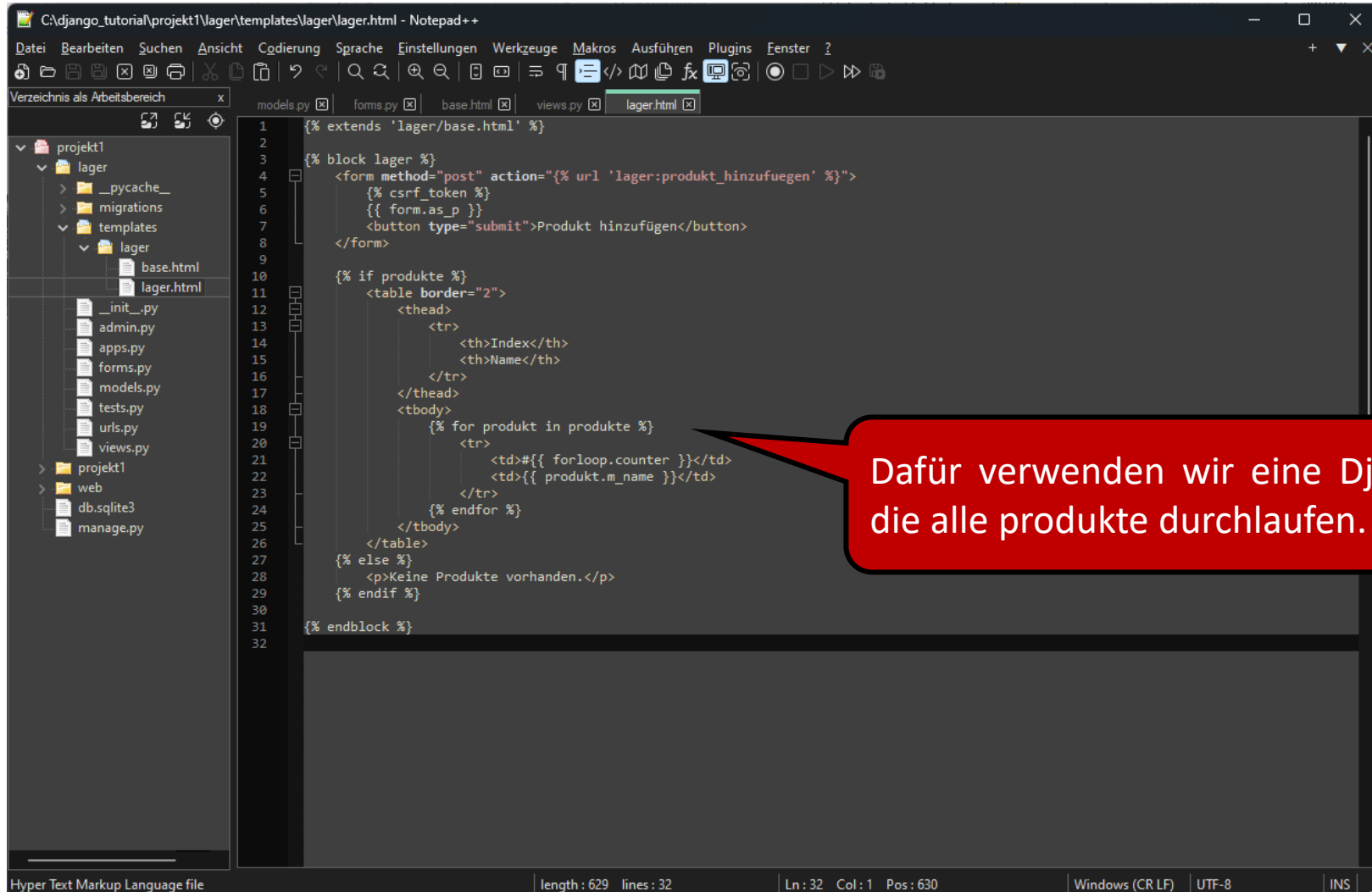
Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
5     {% csrf_token %}
6     {{ form.as_p }}
7     <button type="submit">Produkt hinzufügen</button>
8 </form>
9
10 {% if produkte %}
11 <table border="2">
12     <thead>
13         <tr>
14             <th>Index</th>
15             <th>Name</th>
16         </tr>
17     </thead>
18     <tbody>
19         {% for produkt in produkte %}
20             <tr>
21                 <td>#{{ forloop.counter }}</td>
22                 <td>{{ produkt.m_name }}</td>
23             </tr>
24         {% endfor %}
25     </tbody>
26 </table>
27 {% else %}
28 <p>Keine Produkte vorhanden.</p>
29 {% endif %}
30
31 {% endblock %}
32
```

Wenn Daten enthalten ist, soll eine Tabelle gezeichnet werden.

Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
5     {% csrf_token %}
6     {{ form.as_p }}
7     <button type="submit">Produkt hinzufügen</button>
8 </form>
9
10 {% if produkte %}
11 <table border="2">
12     <thead>
13         <tr>
14             <th>Index</th>
15             <th>Name</th>
16         </tr>
17     </thead>
18     <tbody>
19         {% for produkt in produkte %}
20             <tr>
21                 <td>#{{ forloop.counter }}</td>
22                 <td>{{ produkt.m_name }}</td>
23             </tr>
24         {% endfor %}
25     </tbody>
26 </table>
27 {% else %}
28 <p>Keine Produkte vorhanden.</p>
29 {% endif %}
30
31 {% endblock %}
32
```

Dafür verwenden wir eine Django-Tag Schleife die alle produkte durchlaufen.

Modelle und Form

← → ↻ ⓘ 127.0.0.1:8000/lager/

📖 Meine Bibliothek |...

Name:

Produkt hinzufügen

Index	Name
#1	Buch
#2	Computer
#3	Handy
#4	Schallplatte
#5	Kassette

Die Einträge aus der Datenbank werden auf diese Weise angezeigt. Wenn dies nicht wie erwartet funktioniert, könnte dies auf Probleme im Browser-Cache hindeuten.

Modelle und Form

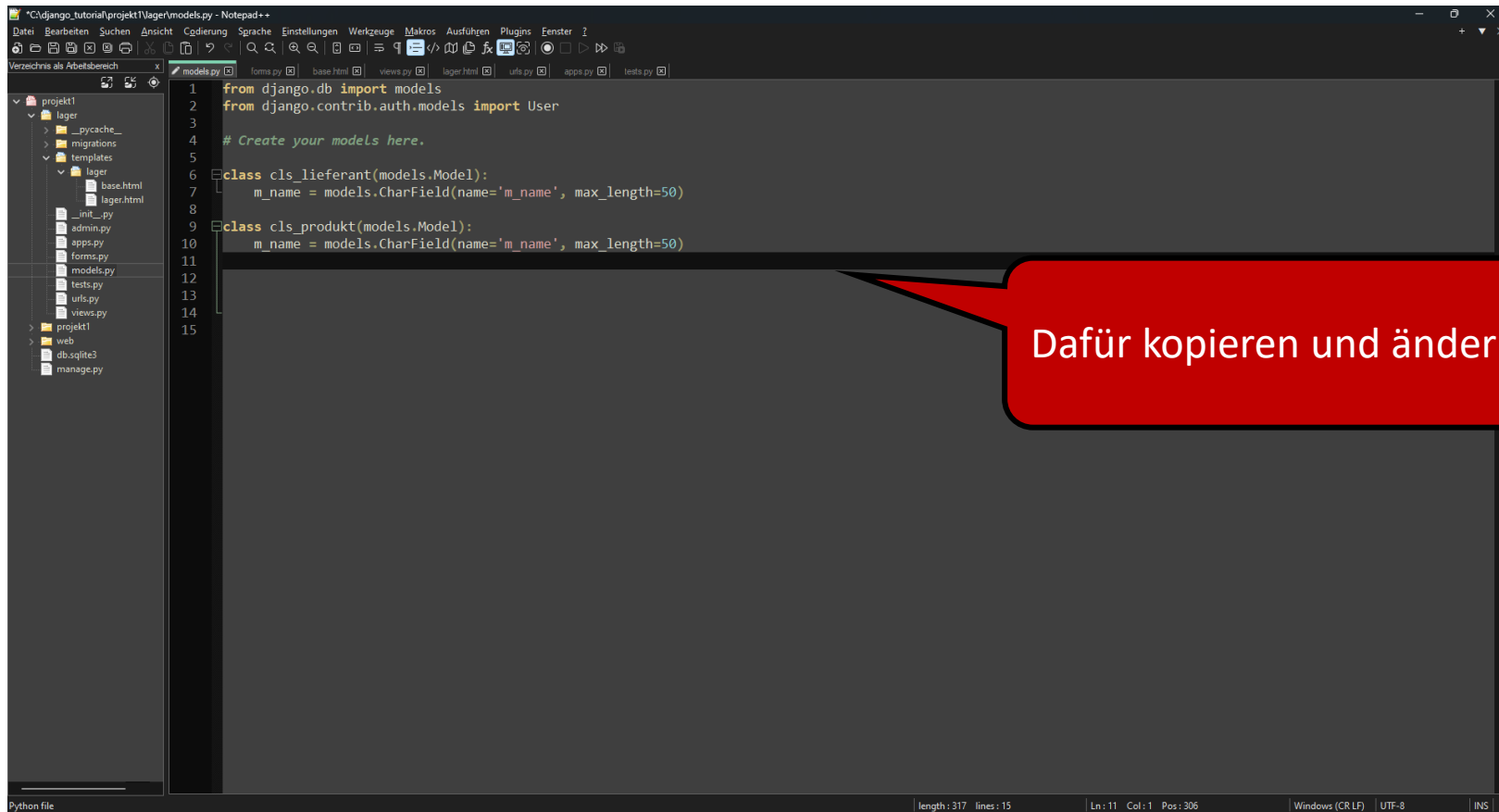
← → ↻ ⓘ 127.0.0.1:8000/lager/
📖 Meine Bibliothek |...

Name:

Produkt hinzufügen	
Index	Name
#1	Buch
#2	Computer
#3	Handy
#4	Schallplatte
#5	Kassette

Nun wollen wir noch ein Modell Lieferant hinzufügen. Sodass jedes Produkt einen Lieferanten besitzt.

Modelle und Form



The screenshot shows a Notepad++ window with the file path "C:\django_tutorial\projekt1\lager\models.py". The code defines two Django models: `cls_lieferant` and `cls_produk`. Both models inherit from `models.Model` and have a `m_name` field of type `CharField` with a maximum length of 50. A red callout bubble with a black border points to the model classes, containing the text "Dafür kopieren und ändern wir das Modell.".

```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4 # Create your models here.
5
6 class cls_lieferant(models.Model):
7     m_name = models.CharField(name='m_name', max_length=50)
8
9 class cls_produk(models.Model):
10     m_name = models.CharField(name='m_name', max_length=50)
11
12
13
14
15
```

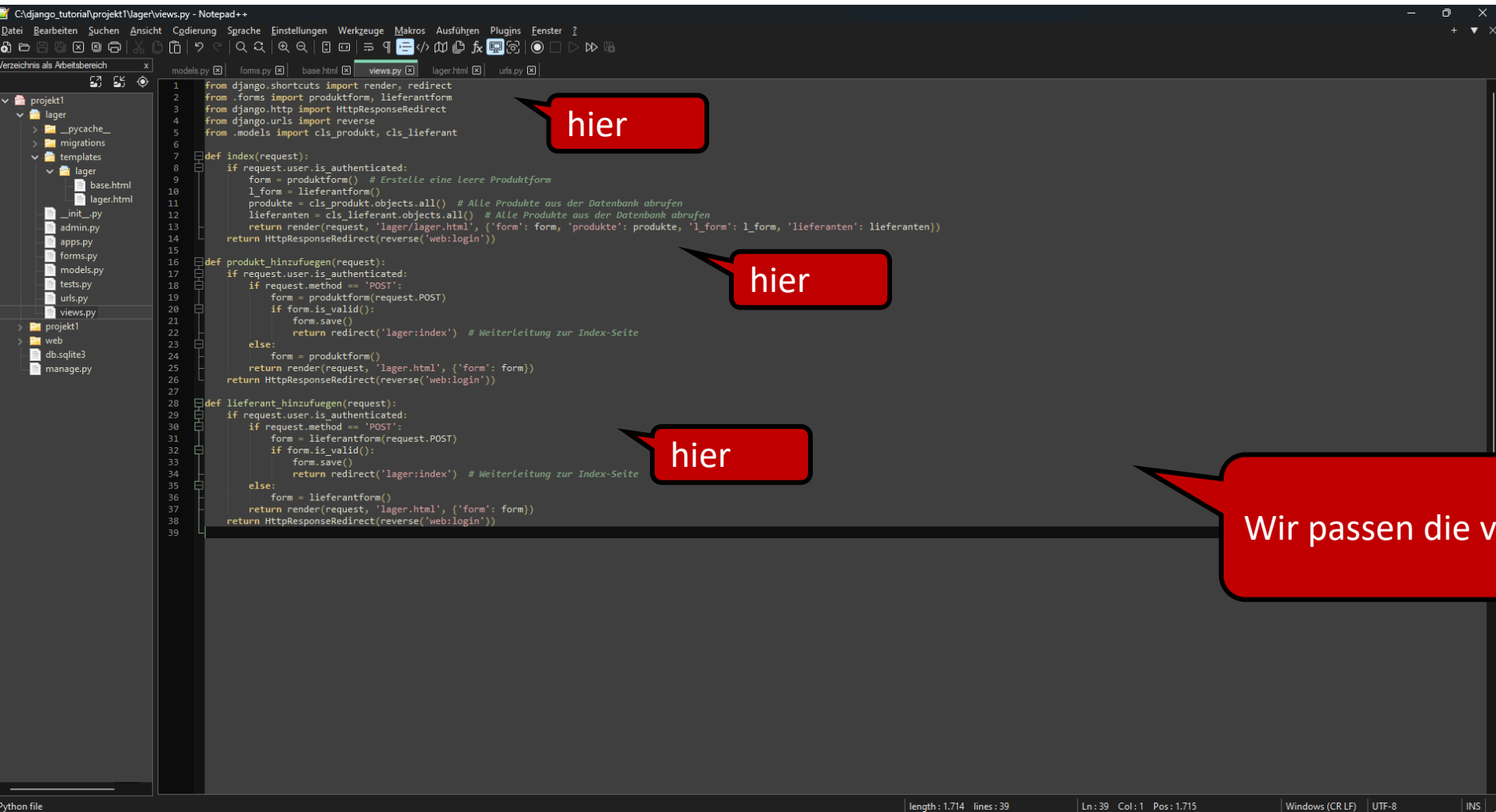
Dafür kopieren und ändern wir das Modell.

Modelle und Form

```
1 from django import forms
2 from .models import cls_produk, cls_lieferant
3
4 class produktform(forms.ModelForm):
5     class Meta:
6         model=cls_produk
7         fields=['m_name']
8         labels={'m_name': 'Name'}
9
10 class lieferantform(forms.ModelForm):
11     class Meta:
12         model=cls_lieferant
13         fields=['m_name']
14         labels={'m_name': 'Name'}
15
```

Dafür kopieren und ändern wir die Form.

Modelle und Form



The screenshot shows a Notepad++ window editing `C:\django_tutorial\projekt1\lager\views.py`. The left sidebar displays a project tree with folders like `lager`, `migrations`, and `templates`, and files like `base.html`, `lager.html`, `__init__.py`, `admin.py`, `apps.py`, `forms.py`, `models.py`, `tests.py`, `urls.py`, and `views.py`.

The main editor displays the following Python code:

```
1 from django.shortcuts import render, redirect
2 from .forms import produktform, lieferantform
3 from django.http import HttpResponseRedirect
4 from django.urls import reverse
5 from .models import cls_produkt, cls_lieferant
6
7 def index(request):
8     if request.user.is_authenticated:
9         form = produktform() # Erstelle eine Leere Produktform
10        l_form = lieferantform()
11        produkte = cls_produkt.objects.all() # Alle Produkte aus der Datenbank abrufen
12        lieferanten = cls_lieferant.objects.all() # Alle Lieferanten aus der Datenbank abrufen
13        return render(request, 'lager/lager.html', {'form': form, 'produkte': produkte, 'l_form': l_form, 'lieferanten': lieferanten})
14    return HttpResponseRedirect(reverse('web:login'))
15
16 def produkt_hinzufuegen(request):
17     if request.user.is_authenticated:
18         if request.method == 'POST':
19             form = produktform(request.POST)
20             if form.is_valid():
21                 form.save()
22                 return redirect('lager:index') # Weiterleitung zur Index-Seite
23         else:
24             form = produktform()
25             return render(request, 'lager.html', {'form': form})
26     return HttpResponseRedirect(reverse('web:login'))
27
28 def lieferant_hinzufuegen(request):
29     if request.user.is_authenticated:
30         if request.method == 'POST':
31             form = lieferantform(request.POST)
32             if form.is_valid():
33                 form.save()
34                 return redirect('lager:index') # Weiterleitung zur Index-Seite
35         else:
36             form = lieferantform()
37             return render(request, 'lager.html', {'form': form})
38     return HttpResponseRedirect(reverse('web:login'))
39
```

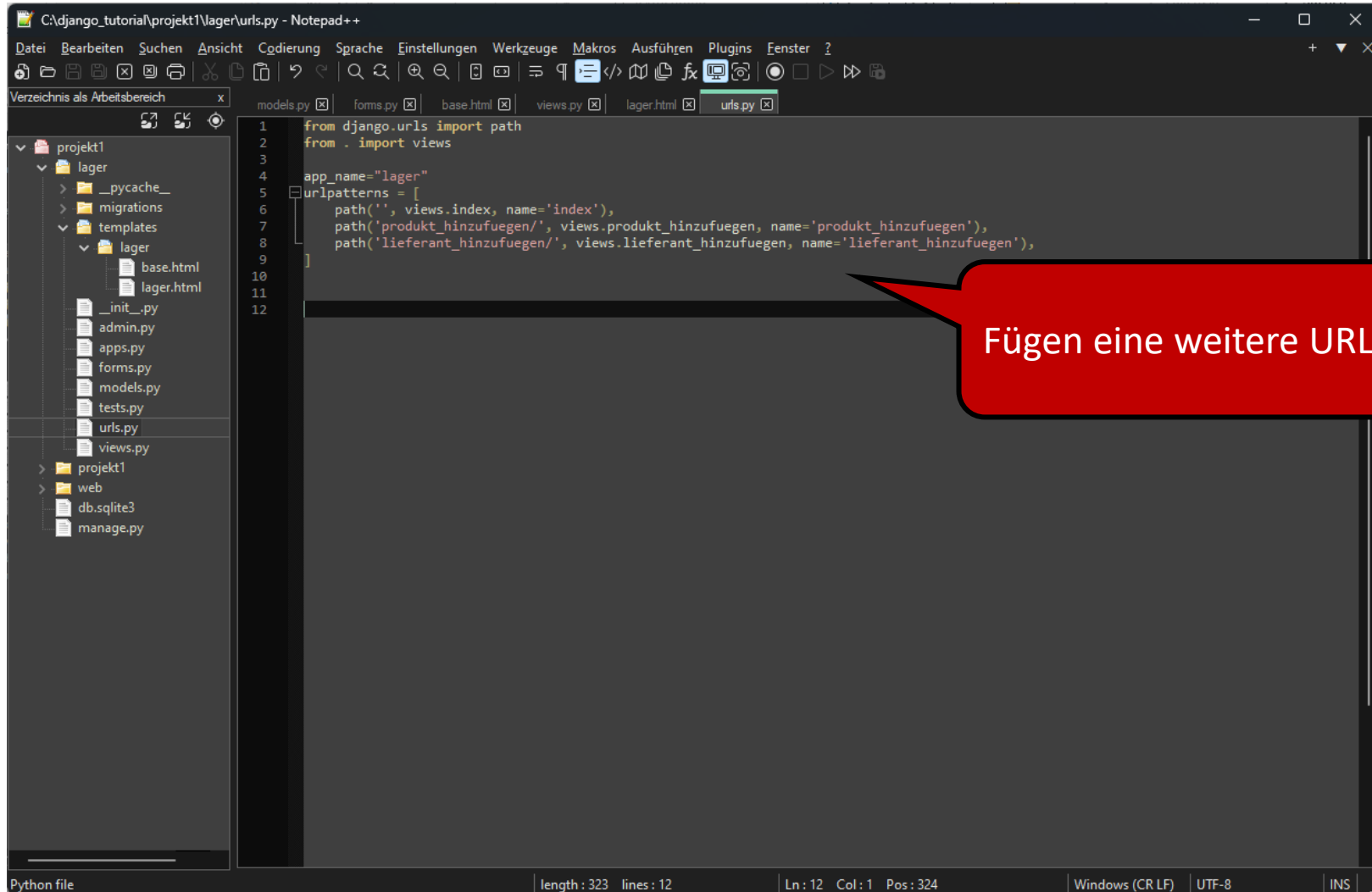
Three red callout boxes with white text are overlaid on the code:

- The first box, labeled "hier", points to line 13.
- The second box, labeled "hier", points to line 22.
- The third box, labeled "hier", points to line 34.

A larger red callout box with white text is positioned at the bottom right, stating: "Wir passen die view.py an."

The status bar at the bottom indicates: length: 1.714 lines: 39 | Ln: 39 Col: 1 Pos: 1.715 | Windows (CR LF) UTF-8 | INS

Modelle und Form



C:\django_tutorial\projekt1\lager\urls.py - Notepad++

Verzeichnis als Arbeitsbereich

projekt1

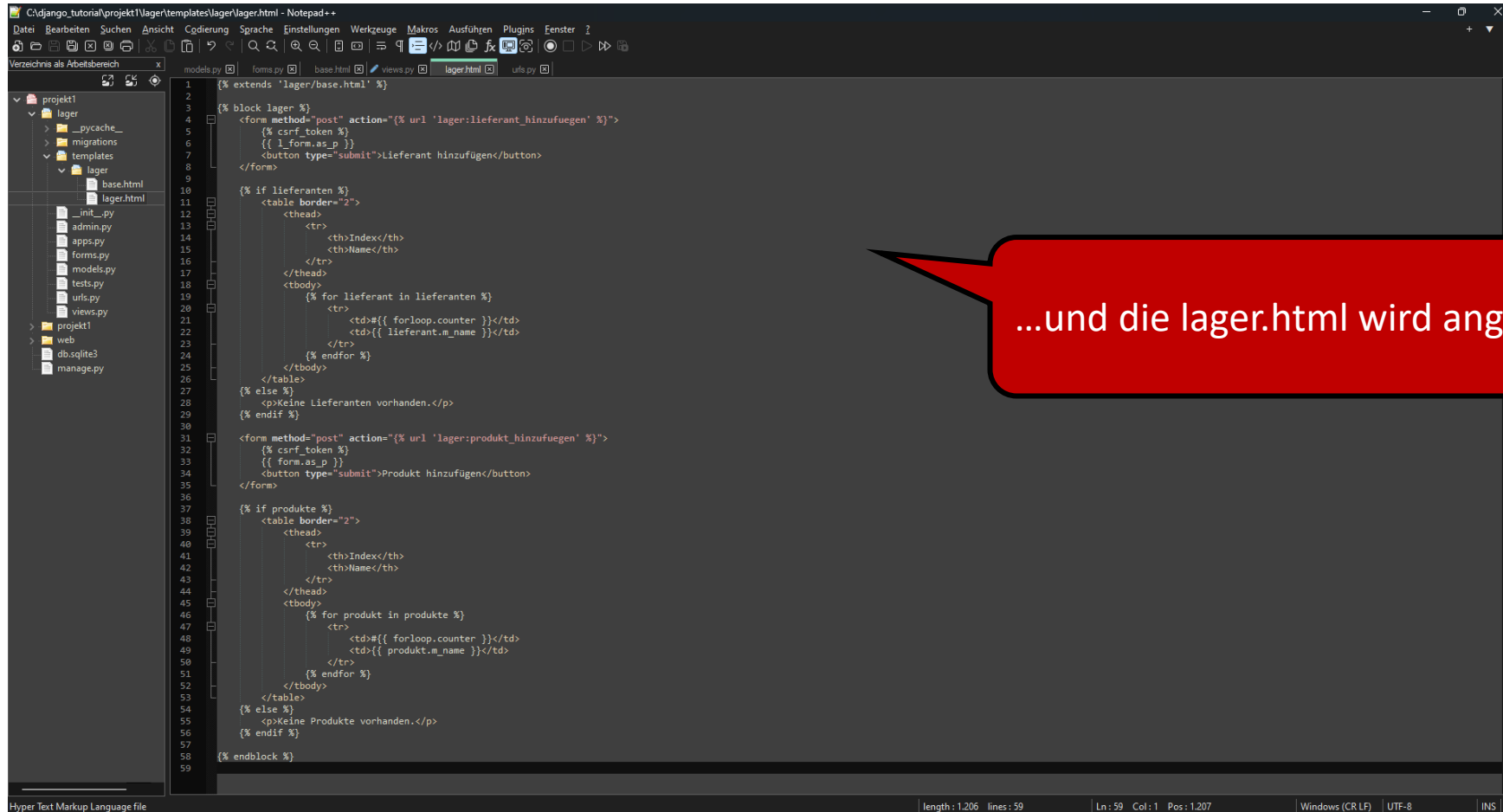
- lager
 - __pycache__
 - migrations
 - templates
 - lager
 - base.html
 - lager.html
 - __init__.py
 - admin.py
 - apps.py
 - forms.py
 - models.py
 - tests.py
 - urls.py
 - views.py
- projekt1
- web
 - db.sqlite3
 - manage.py

```
1 from django.urls import path
2 from . import views
3
4 app_name="lager"
5 urlpatterns = [
6     path('', views.index, name='index'),
7     path('produkt_hinzufuegen/', views.produkt_hinzufuegen, name='produkt_hinzufuegen'),
8     path('lieferant_hinzufuegen/', views.lieferant_hinzufuegen, name='lieferant_hinzufuegen'),
9 ]
10
11
12
```

Fügen eine weitere URL hinzu...

Python file | length: 323 lines: 12 | Ln: 12 Col: 1 Pos: 324 | Windows (CR LF) UTF-8 | INS

Modelle und Form



```
1 {% extends 'lager/base.html' %}
2
3 {% block lager %}
4 <form method="post" action="{% url 'lager:lieferant_hinzufuegen' %}">
5   {% csrf_token %}
6   {{ l_form.as_p }}
7   <button type="submit">Lieferant hinzufügen</button>
8 </form>
9
10 {% if lieferanten %}
11 <table border="2">
12   <thead>
13     <tr>
14       <th>Index</th>
15       <th>Name</th>
16     </tr>
17   </thead>
18   <tbody>
19     {% for lieferant in lieferanten %}
20     <tr>
21       <td>{{ forloop.counter }}</td>
22       <td>{{ lieferant.m_name }}</td>
23     </tr>
24     {% endfor %}
25   </tbody>
26 </table>
27 {% else %}
28 <p>Keine Lieferanten vorhanden.</p>
29 {% endif %}
30
31 <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
32   {% csrf_token %}
33   {{ form.as_p }}
34   <button type="submit">Produkt hinzufügen</button>
35 </form>
36
37 {% if produkte %}
38 <table border="2">
39   <thead>
40     <tr>
41       <th>Index</th>
42       <th>Name</th>
43     </tr>
44   </thead>
45   <tbody>
46     {% for produkt in produkte %}
47     <tr>
48       <td>{{ forloop.counter }}</td>
49       <td>{{ produkt.m_name }}</td>
50     </tr>
51     {% endfor %}
52   </tbody>
53 </table>
54 {% else %}
55 <p>Keine Produkte vorhanden.</p>
56 {% endif %}
57
58 {% endblock %}
```

...und die lager.html wird angepasst.

Modelle und Form

← → ↻ ⓘ 127.0.0.1:8000/lager/

📖 Meine Bibliothek |...

Name:

Lieferant hinzufügen

Index	Name
#1	Picard

Name:

Produkt hinzufügen

Index	Name
#1	Buch
#2	Computer
#3	Praktisch
#4	Schallplatte
#5	Kassette

Nach einen neuen *makemigrations* und *migrate*, sieht das Ergebnis wie hier aus.

Modelle und Form

← → ↻ ⓘ 127.0.0.1:8000/lager/

📖 Meine Bibliothek |...

Name:

Lieferant hinzufügen

Index	Name
#1	Picard

Name:

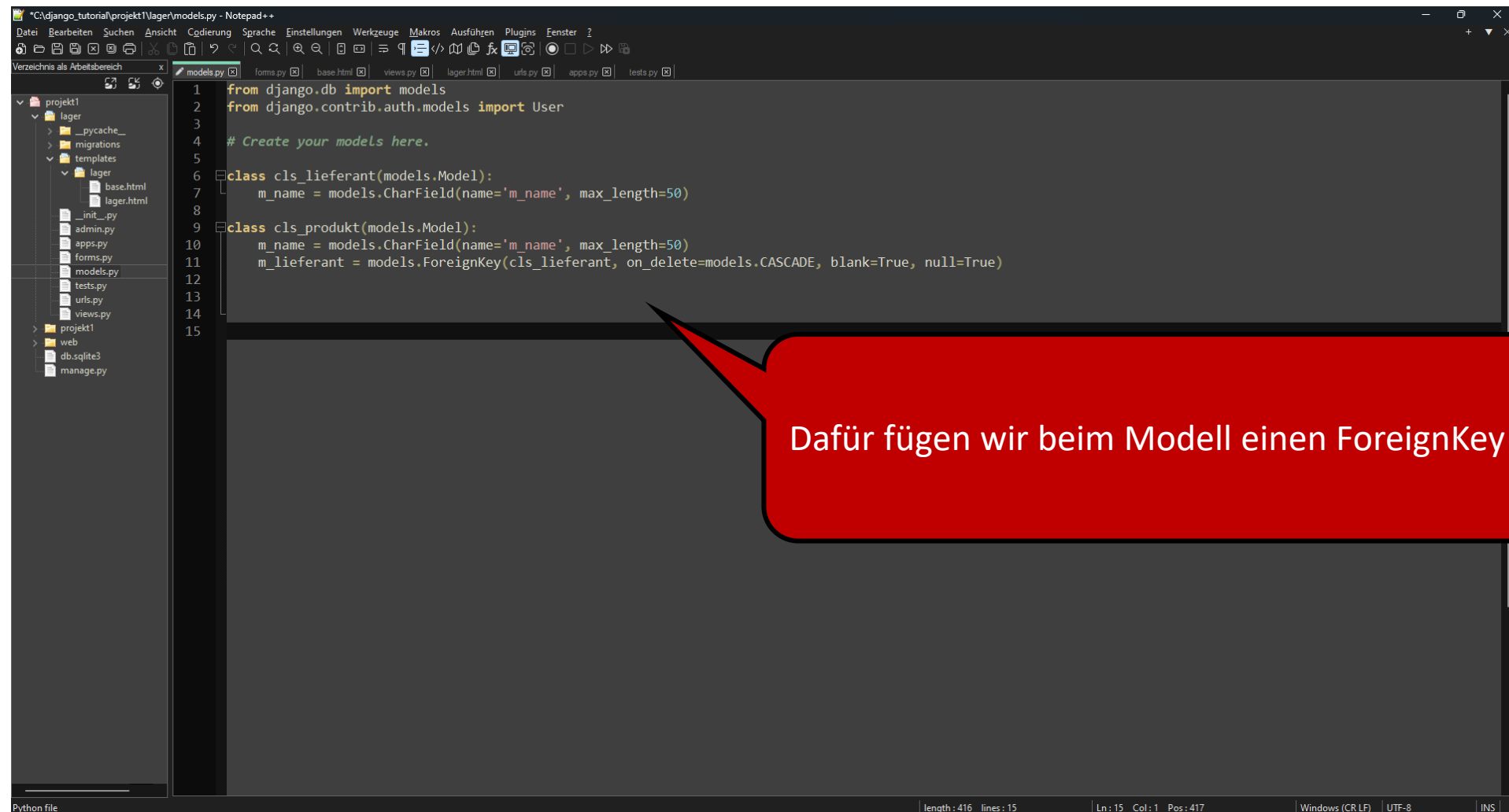
Produkt hinzufügen

Index	Name
#1	Buch
#2	Computer
#3	Praktisch
#4	Schallplatte
#5	Kassette

Nach einen neuen *makemigrations* und *migrate*, sieht das Ergebnis wie hier aus.

Nun wollen wir noch einen Verweis einbauen. Dies nennt sich ForeignKey.

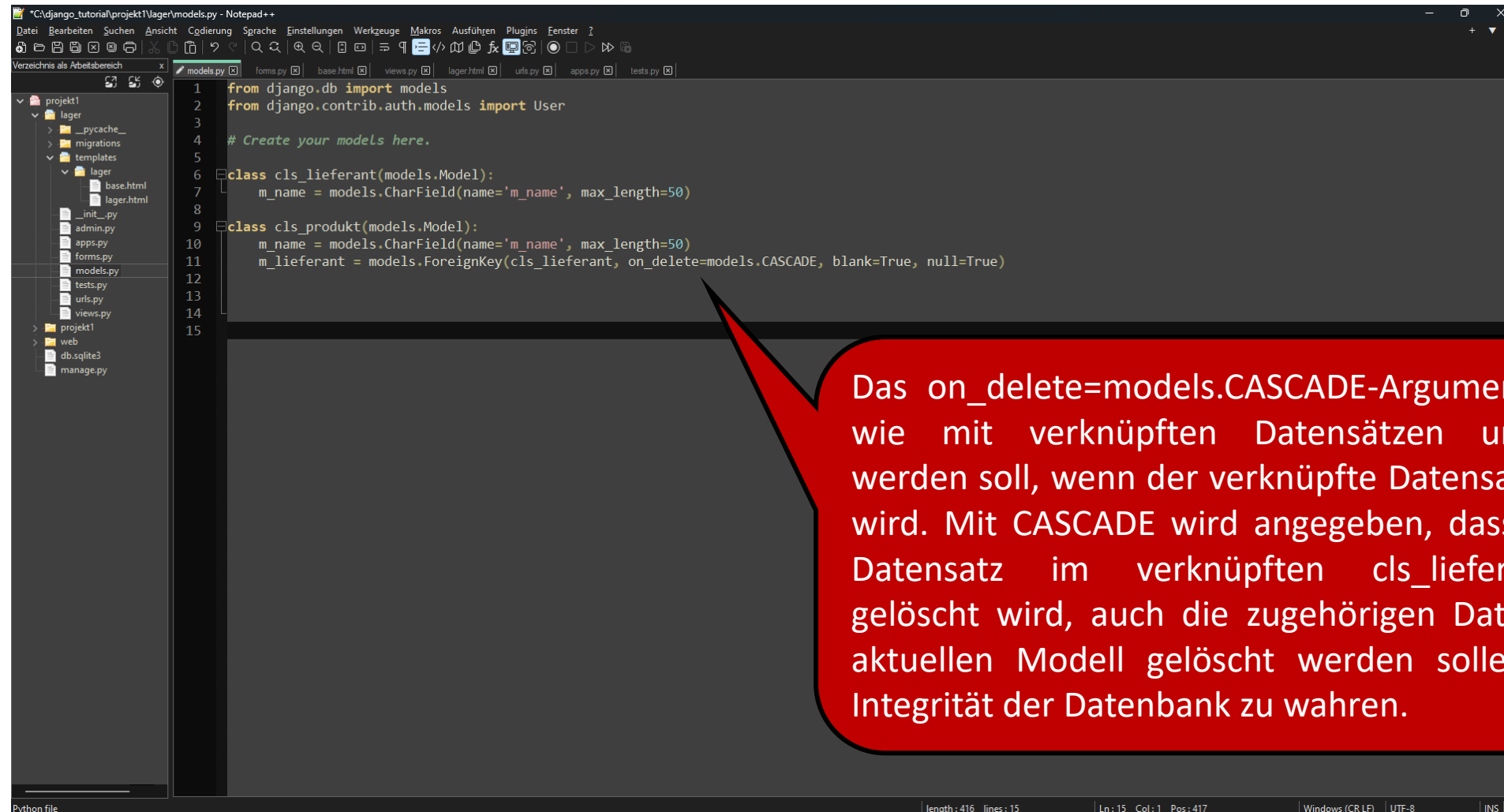
Modelle und Form



```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4 # Create your models here.
5
6 class cls_lieferant(models.Model):
7     m_name = models.CharField(name='m_name', max_length=50)
8
9 class cls_produk(models.Model):
10     m_name = models.CharField(name='m_name', max_length=50)
11     m_lieferant = models.ForeignKey(cls_lieferant, on_delete=models.CASCADE, blank=True, null=True)
12
13
14
15
```

Dafür fügen wir beim Modell einen ForeignKey hinzu.

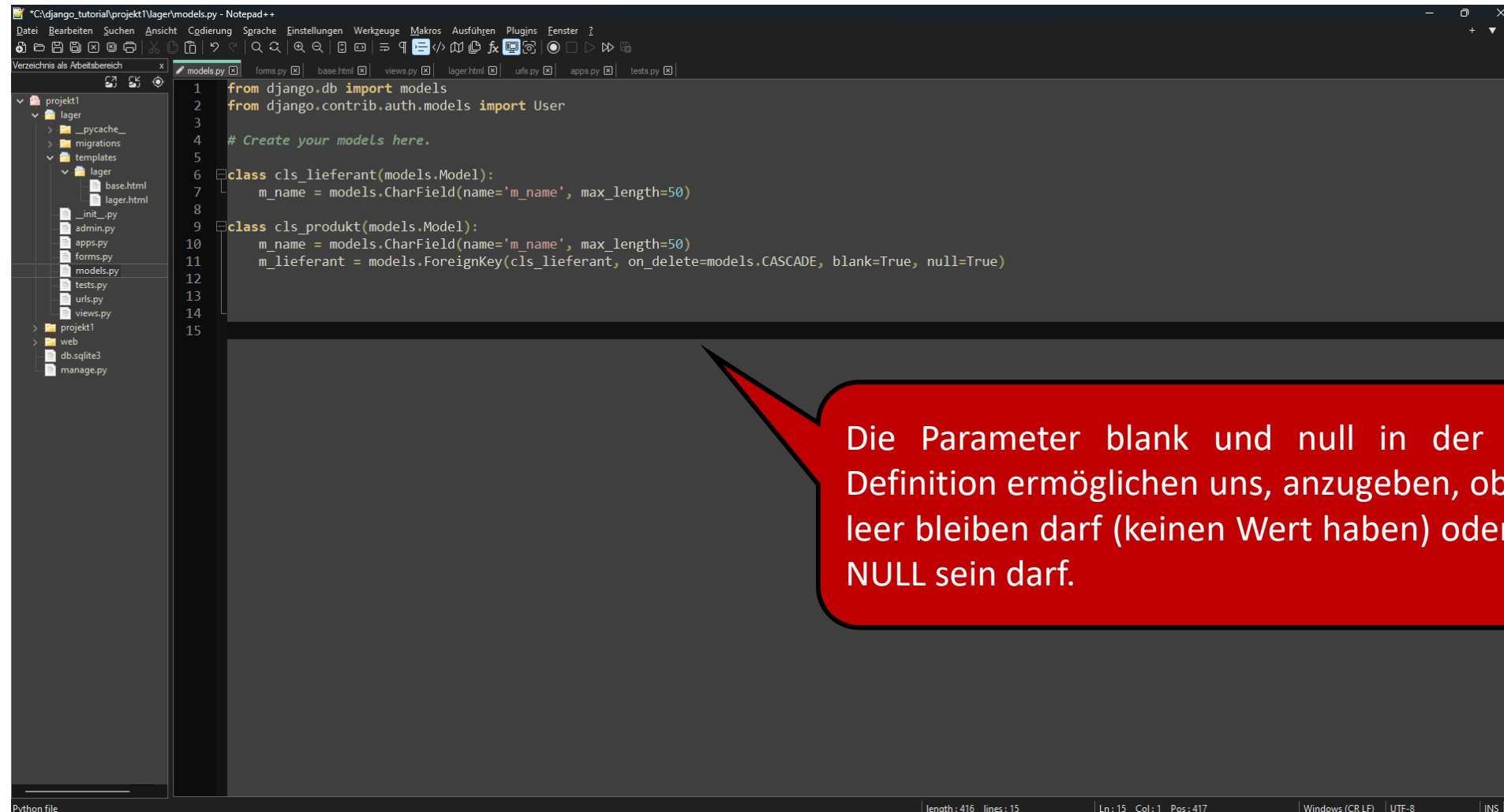
Modelle und Form



```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4 # Create your models here.
5
6 class cls_lieferant(models.Model):
7     m_name = models.CharField(name='m_name', max_length=50)
8
9 class cls_produk(t(models.Model):
10     m_name = models.CharField(name='m_name', max_length=50)
11     m_lieferant = models.ForeignKey(cls_lieferant, on_delete=models.CASCADE, blank=True, null=True)
12
13
14
15
```

Das `on_delete=models.CASCADE`-Argument gibt an, wie mit verknüpften Datensätzen umgegangen werden soll, wenn der verknüpfte Datensatz gelöscht wird. Mit `CASCADE` wird angegeben, dass wenn ein Datensatz im verknüpften `cls_lieferant`-Modell gelöscht wird, auch die zugehörigen Datensätze im aktuellen Modell gelöscht werden sollen, um die Integrität der Datenbank zu wahren.

Modelle und Form



```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4 # Create your models here.
5
6 class cls_lieferant(models.Model):
7     m_name = models.CharField(name='m_name', max_length=50)
8
9 class cls_produk(models.Model):
10     m_name = models.CharField(name='m_name', max_length=50)
11     m_lieferant = models.ForeignKey(cls_lieferant, on_delete=models.CASCADE, blank=True, null=True)
12
13
14
15
```

Die Parameter blank und null in der ForeignKey-Definition ermöglichen uns, anzugeben, ob dieses Feld leer bleiben darf (keinen Wert haben) oder ob es auch NULL sein darf.

Modelle und Form

```
25     </tbody>
26   </table>
27   {% else %}
28   <p>Keine Lieferanten vorhanden.</p>
29   {% endif %}
30
31   <form method="post" action="{% url 'lager:produkt_hinzufuegen' %}">
32     {% csrf_token %}
33     {{ form.as_p }}
34     <button type="submit">Produkt hinzufügen</button>
35   </form>
36
37   {% if produkte %}
38   <table border="2">
39     <thead>
40       <tr>
41         <th>Index</th>
42         <th>Name</th>
43         <th>Lieferant</th>
44       </tr>
45     </thead>
46     <tbody>
47       {% for produkt in produkte %}
48       <tr>
49         <td>#{{ forloop.counter }}</td>
50         <td>{{ produkt.m_name }}</td>
51         <td>{{ produkt.m_lieferant.m_name }}</td>
52       </tr>
53       {% endfor %}
54     </tbody>
55   </table>
56   {% else %}
57   <p>Keine Produkte vorhanden.</p>
58   {% endif %}
59
60 {% endblock %}
61
```

hier

Für die korrekte Ausgabe, passen wir noch die lager.html an. Indem wir die Tabellen Elemente hinzufügen.

Modelle und Form

← → ↻ ⓘ 127.0.0.1:8000/lager/

📖 Meine Bibliothek |...

Name:

Keine Lieferanten vorhanden.

Name:

Lieferant: ▼

Keine Produkte vorhanden.

Wir löschen die Datenbank und legen diese neu an.

Modelle und Form

← → ↻ ⓘ 127.0.0.1:8000/lager/

📖 Meine Bibliothek |...

Name:

Lieferant hinzufügen

Index	Name
#1	Picard
#2	Sisko

Name:

Lieferant: ▼

Produkt hinzufügen

Keine Produkte vorhanden.

Anschließend legen wir ein paar Lieferanten an...

Modelle und Form

← → ↻ ⓘ 127.0.0.1:8000/lager/

📖 Meine Bibliothek |...

Name:

Lieferant hinzufügen

Index	Name
#1	Picard
#2	Sisko

Name:

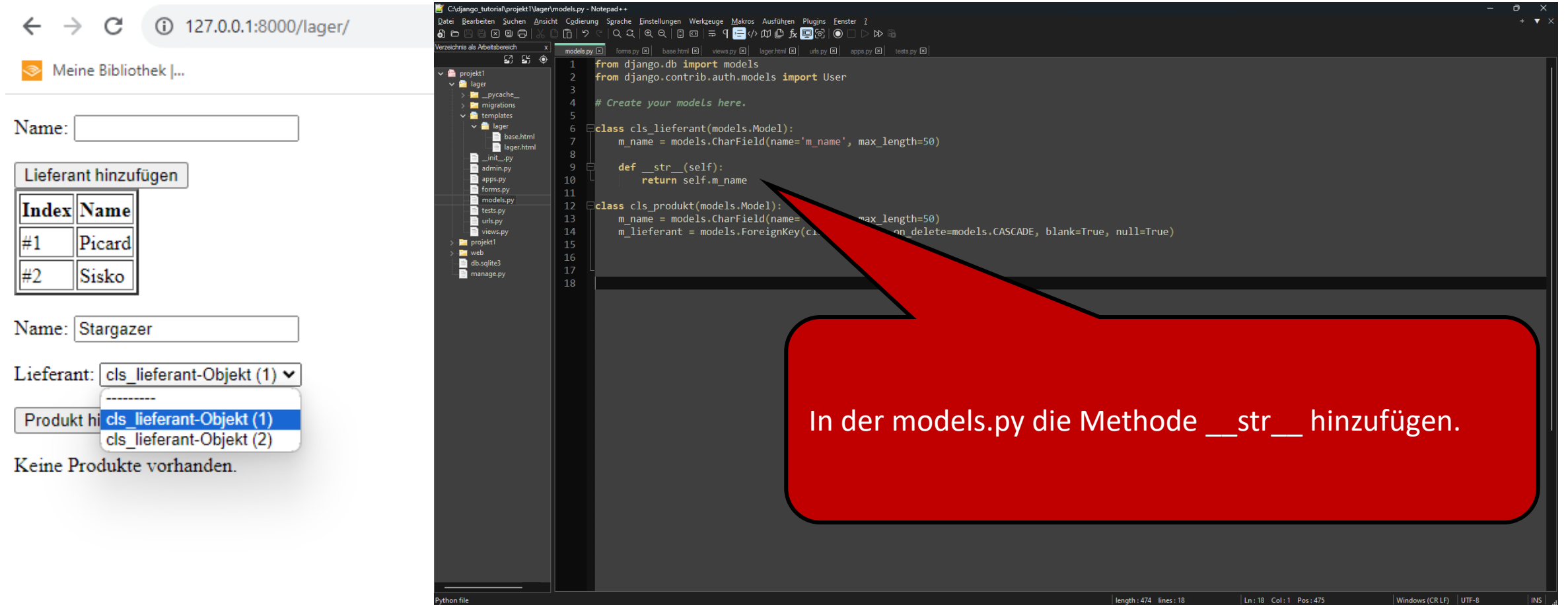
Lieferant:

Produkt hi

Keine Produkte vorhanden.

Leider werden uns die Lieferanten nicht als Text angezeigt. Dies können wir ändern, indem wir...

Modelle und Form



The screenshot displays a Django web application interface on the left and its corresponding Python code in a text editor on the right.

Web Application Interface (Left):

- URL: 127.0.0.1:8000/lager/
- Meine Bibliothek [...]
- Name:
- Lieferant hinzufügen
- Table:

Index	Name
#1	Picard
#2	Sisko

Name: Stargazer

Lieferant: cls_lieferant-Objekt (1) ▼

Produkt hi: cls_lieferant-Objekt (1)
cls_lieferant-Objekt (2)

Keine Produkte vorhanden.

Python Code (Right):

```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4 # Create your models here.
5
6 class cls_lieferant(models.Model):
7     m_name = models.CharField(name='m_name', max_length=50)
8
9     def __str__(self):
10         return self.m_name
11
12 class cls_produkkt(models.Model):
13     m_name = models.CharField(name='m_name', max_length=50)
14     m_lieferant = models.ForeignKey(cls_lieferant, on_delete=models.CASCADE, blank=True, null=True)
15
16
17
18
```

In der models.py die Methode `__str__` hinzufügen.

Modelle und Form

← → ↻ ⓘ 127.0.0.1:8000/lager/

📖 Meine Bibliothek |...

Name:

Lieferant hinzufügen

Index	Name
#1	Picard
#2	Sisko

Name:

Lieferant:

Produkt hinzufügen

Index	Name	Lieferant
#1	Stargazer	Picard
#2	Enterprise	Picard
#3	Defiant	Sisko

Damit können wir jetzt den ForeignKey verwenden.

Modelle und Form

← → ↻ ⓘ 127.0.0.1:8000/lager/

📖 Meine Bibliothek |...

Name:

Lieferant hinzufügen

Index	Name
#1	Picard
#2	Sisko

Name:

Lieferant:

Produkt hinzufügen

Index	Name	Lieferant
#1	Stargazer	Picard
#2	Enterprise	Picard
#3	Defiant	Sisko

Neben dem ForeignKey gibt es auch das ManyToMany-Feld, das es ermöglicht, mehrere Beziehungen zu verwalten.