

# DJANGO

Michał Niezgoda  
&  
Krystian Kiejno



# Czym jest Django?

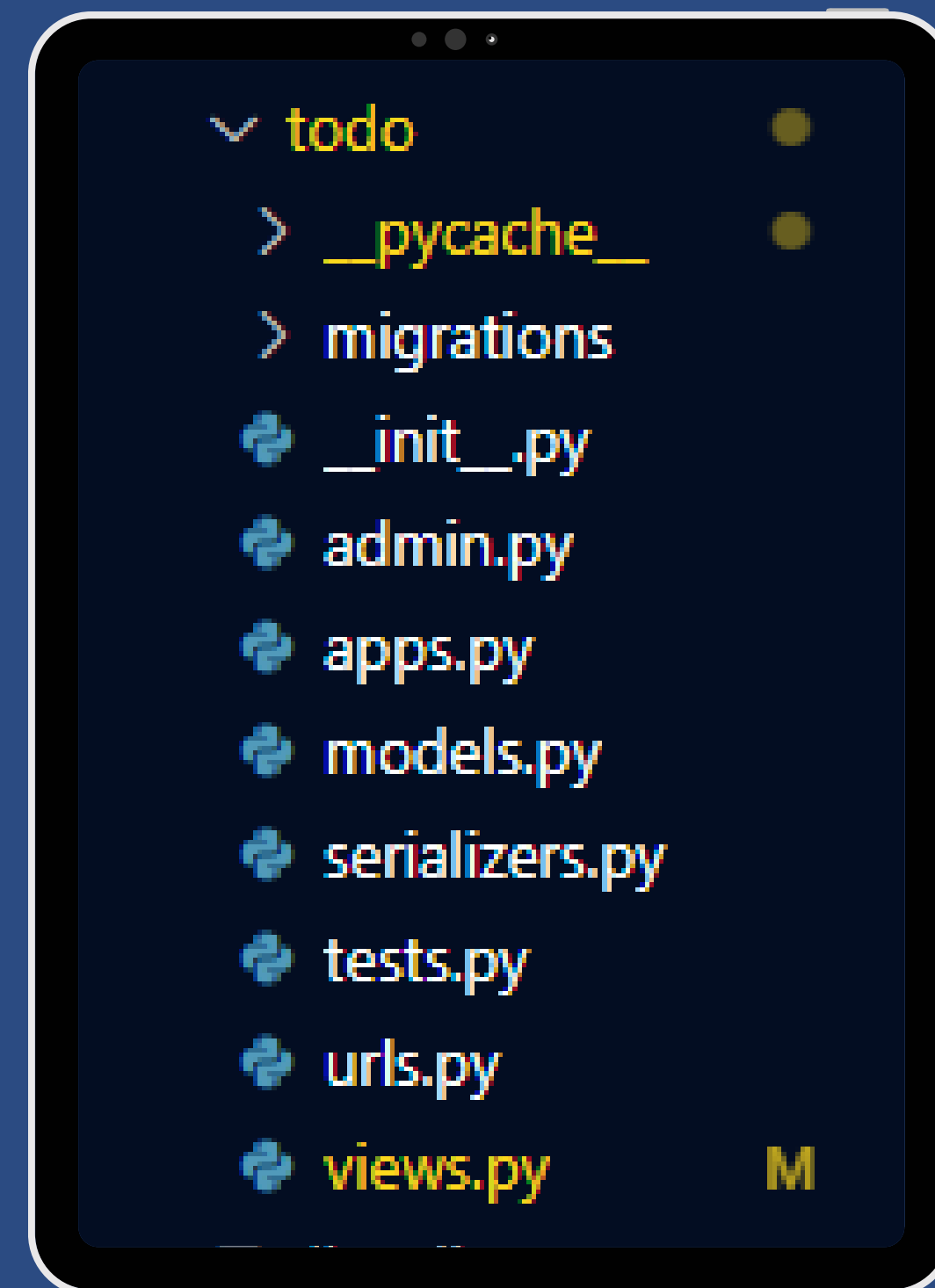
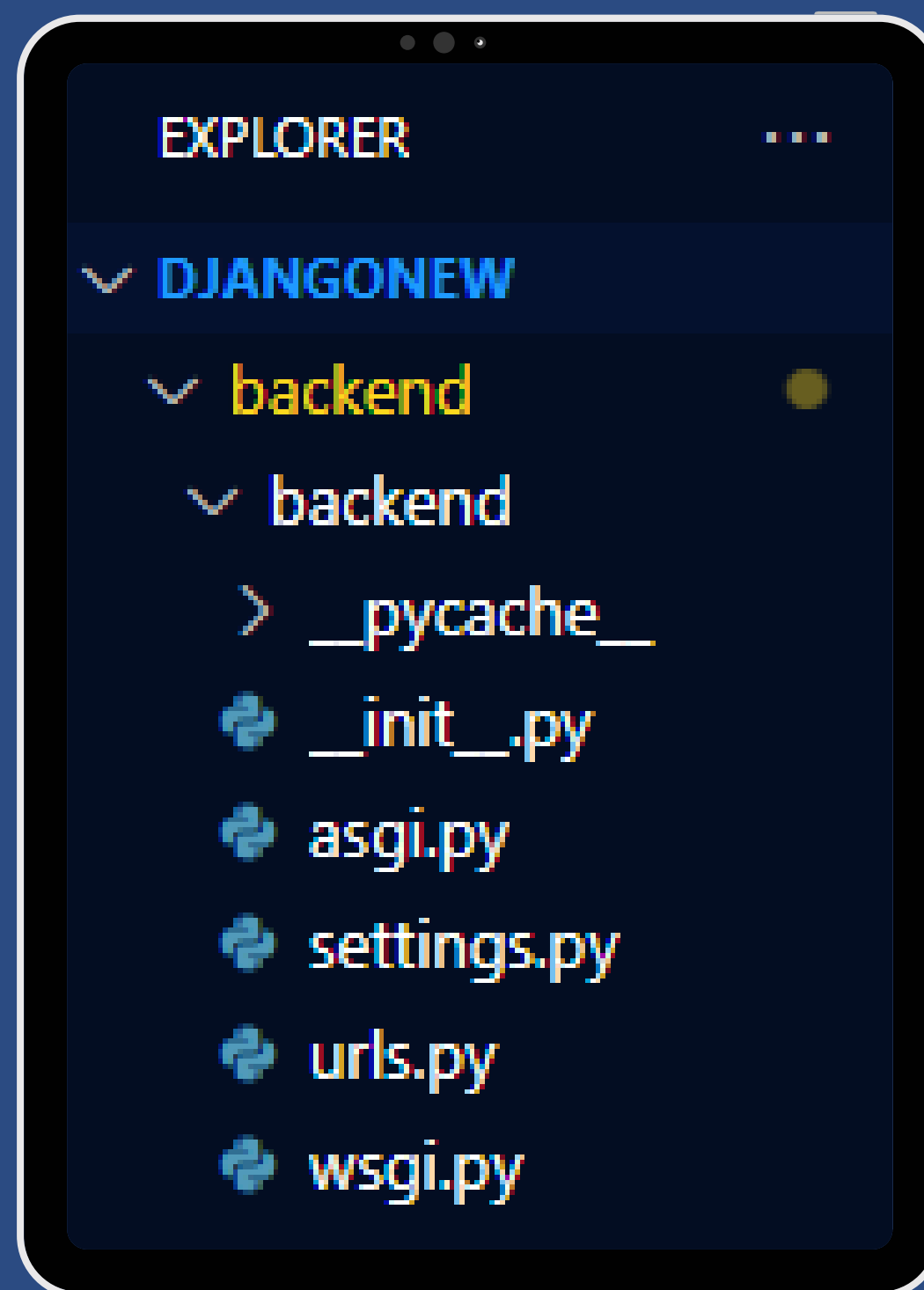
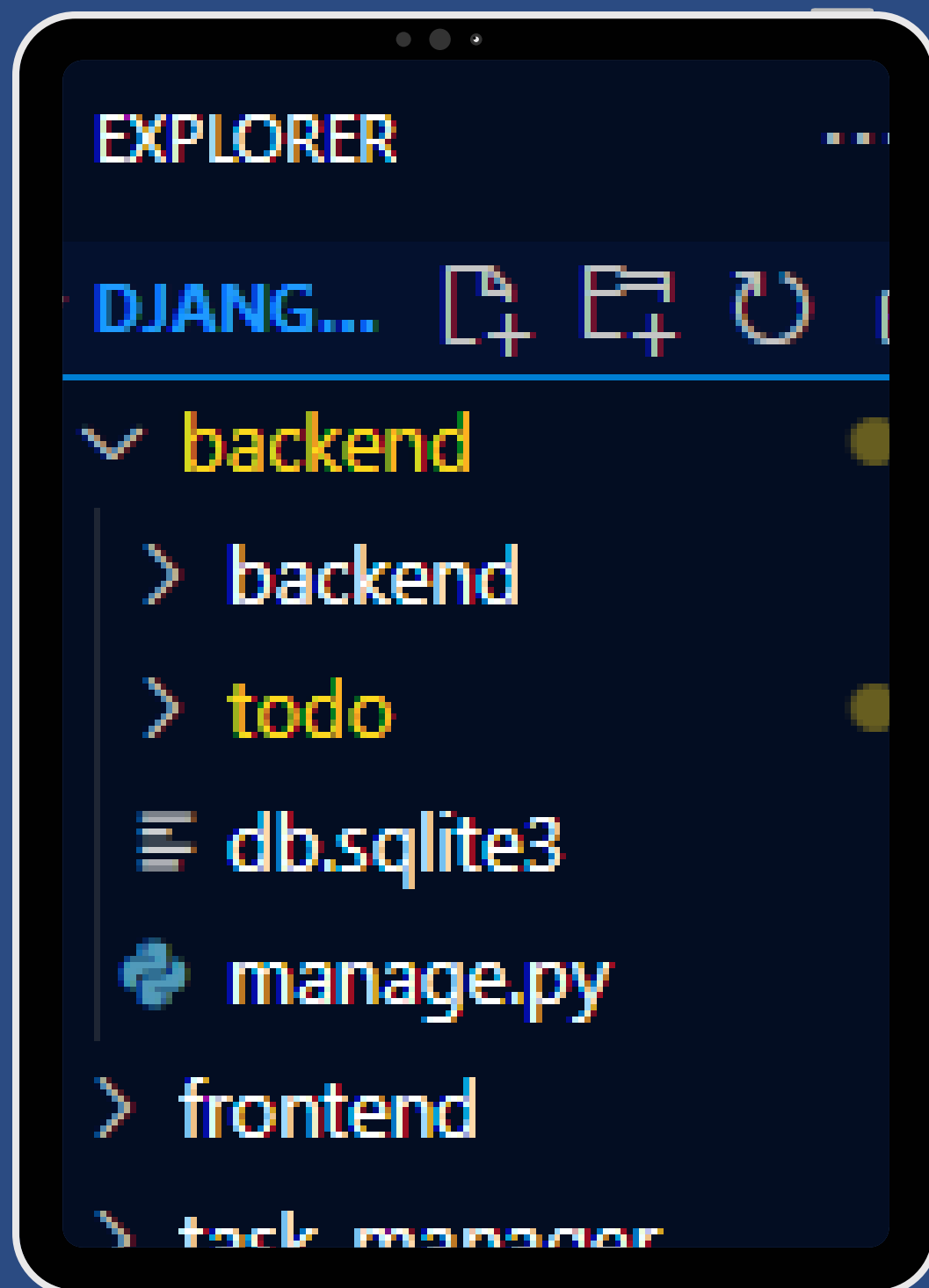
Open-Source framework przeznaczony do tworzenia aplikacji internetowych, napisany w Pythonie. Django realizuje wzorzec architektoniczny model-template-view(MTV). Powstał pod koniec 2003 roku.

# Co oferuje Django?



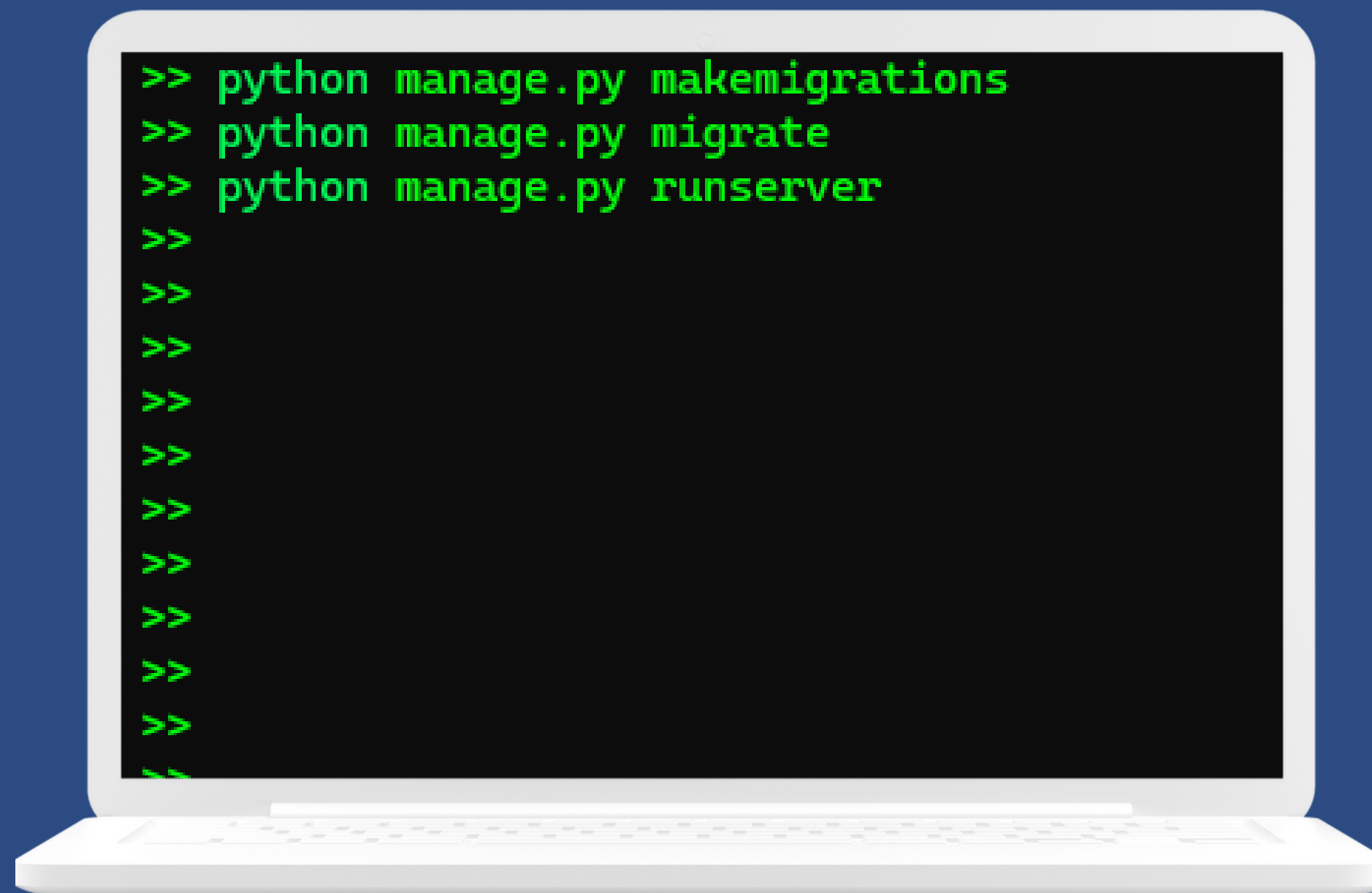
- Uporządkowana struktura plików
- Prosta implementacja z użyciem pythona
- Wiele wbudowanych pakietów
- Panel administratora
- Możliwość użycia różnych rodzajów baz danych
- ORM - Object-Relational Mapping

# Struktura plików w Django



# Manage.py

Skrypt służący do zarządzania projektem Django, takim jak uruchamianie serwera deweloperskiego, synchronizowanie bazy danych, itp.



# wsgi.py

Plik służący do obsługi aplikacji przez serwery WWW.

# admin.py

Plik zawierający konfigurację panelu administracyjnego Django dla danej aplikacji.

# urls.py

Plik zawierający mapowanie URL-i dla całego projektu.

# views.py

Plik zawierający widoki (funkcje lub klasy, które obsługują żądania HTTP i zwracają odpowiedzi).

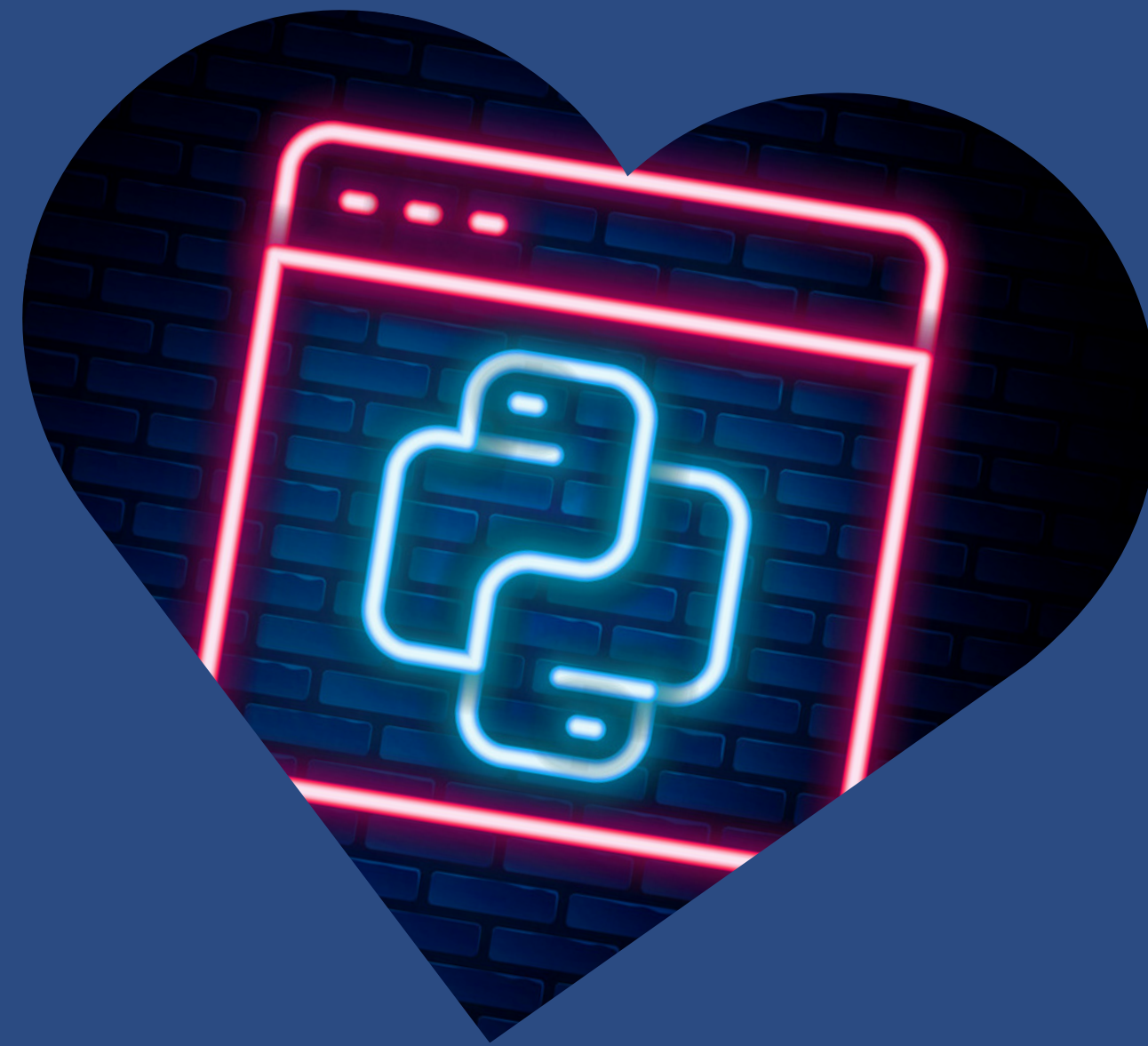
# Django wady i zalety

# Django zalety

## Napisany w Pythonie

Napisanie Django w Pythonie przynosi szereg korzyści.

Python jest językiem programowania znakomicie przystosowanym do tworzenia aplikacji webowych, ze względu na swoją przejrzystą i zwięzłą składnię. To sprawia, że kod napisany w Django jest łatwy do zrozumienia, co ułatwia zarówno rozwój, jak i utrzymanie projektów.





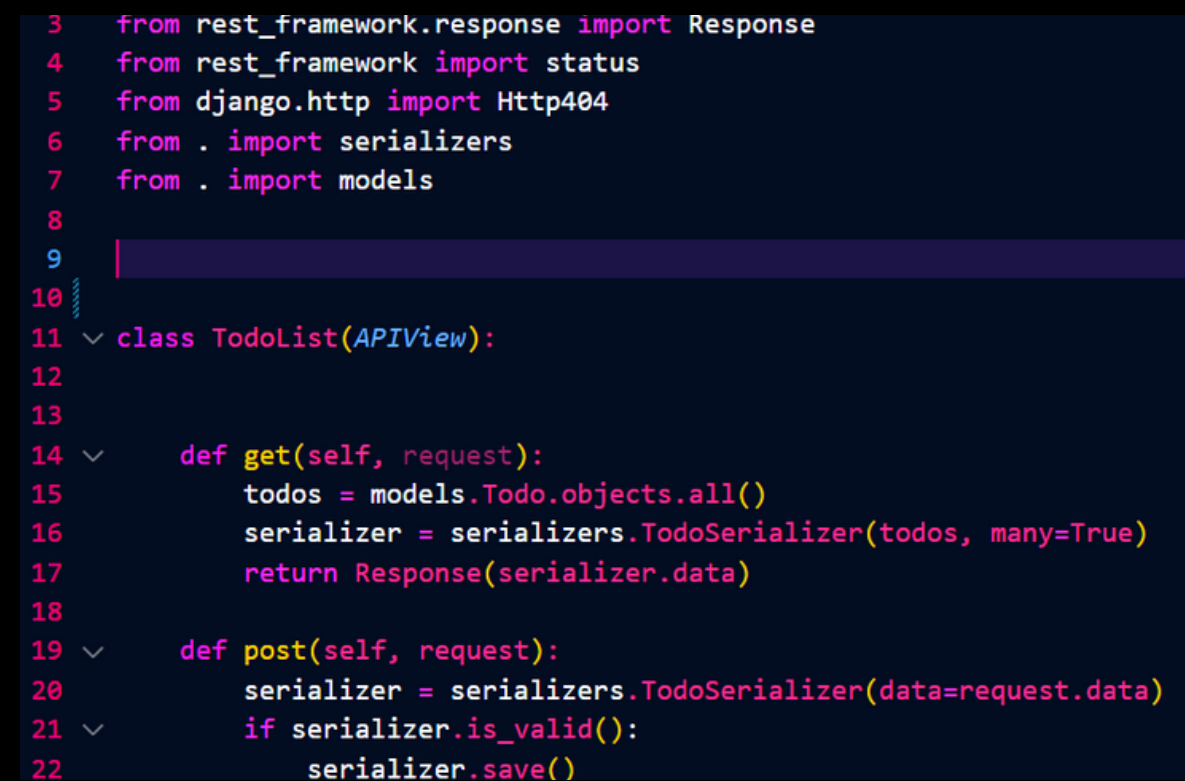
# Django zalety

## Batteries included

Batteries included (slang), w użyteczności produktu oznacza, że produkt jest dostarczany wraz ze wszystkimi możliwymi częściami wymaganymi do pełnej użyteczności.

### Pakiety zapewniające kompletność:

- Pakiety baz danych np. Postgres
- Pakiety uwierzytelniania
- Pakiety interfejsu administratora

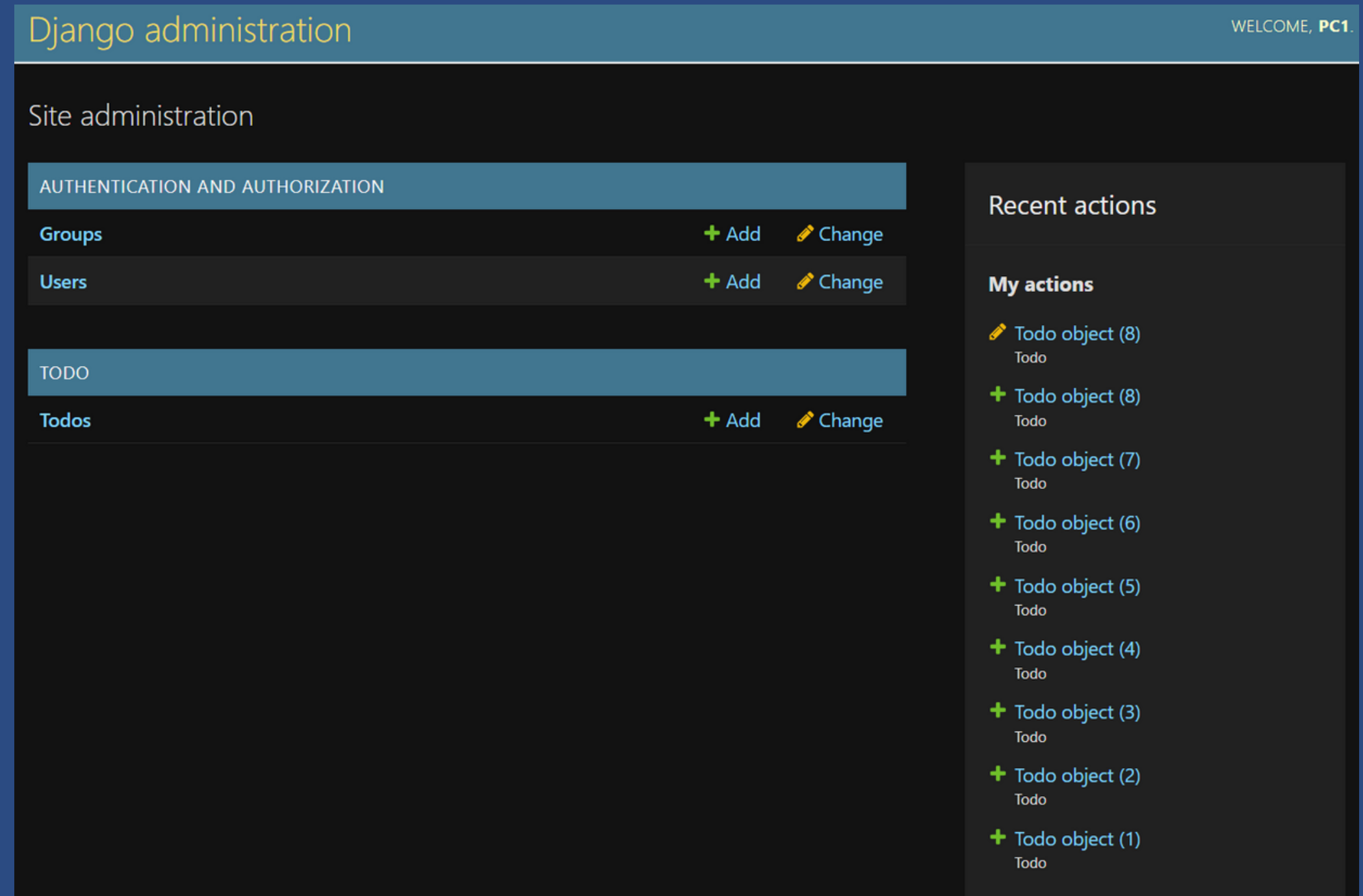


```
3 from rest_framework.response import Response
4 from rest_framework import status
5 from django.http import Http404
6 from . import serializers
7 from . import models
8
9
10
11 class TodoList(APIView):
12
13
14     def get(self, request):
15         todos = models.Todo.objects.all()
16         serializer = serializers.TodoSerializer(todos, many=True)
17         return Response(serializer.data)
18
19     def post(self, request):
20         serializer = serializers.TodoSerializer(data=request.data)
21         if serializer.is_valid():
22             serializer.save()
```

# Django zalety

## Wbudowany interfejs administratora

Django zapewnia **wbudowany interfejs administratora**, który znacznie ułatwia programistom obsługę bazy użytkowników poprzez zapewnienie kontroli dostępu bez konieczności pisania kodu administratora od zera.



# Django zalety

## Różnorodność baz danych

Bazy danych, które oferuje:

- SQLite
- PostgreSQL
- MySQL
- Oracle
- Microsoft SQL Server
- MongoDB



# Django zalety

## Bezpieczeństwo

Django, jako framework, ma wbudowanych wiele funkcji bezpieczeństwa. Posiada system uwierzytelniania użytkowników, który chroni przed atakami typu Clickjacking, XSS, CSRF lub SQL injection. Nie ma konieczności ręcznego wdrażania środków bezpieczeństwa, ponieważ są one dostarczane wraz z frameworkiem.

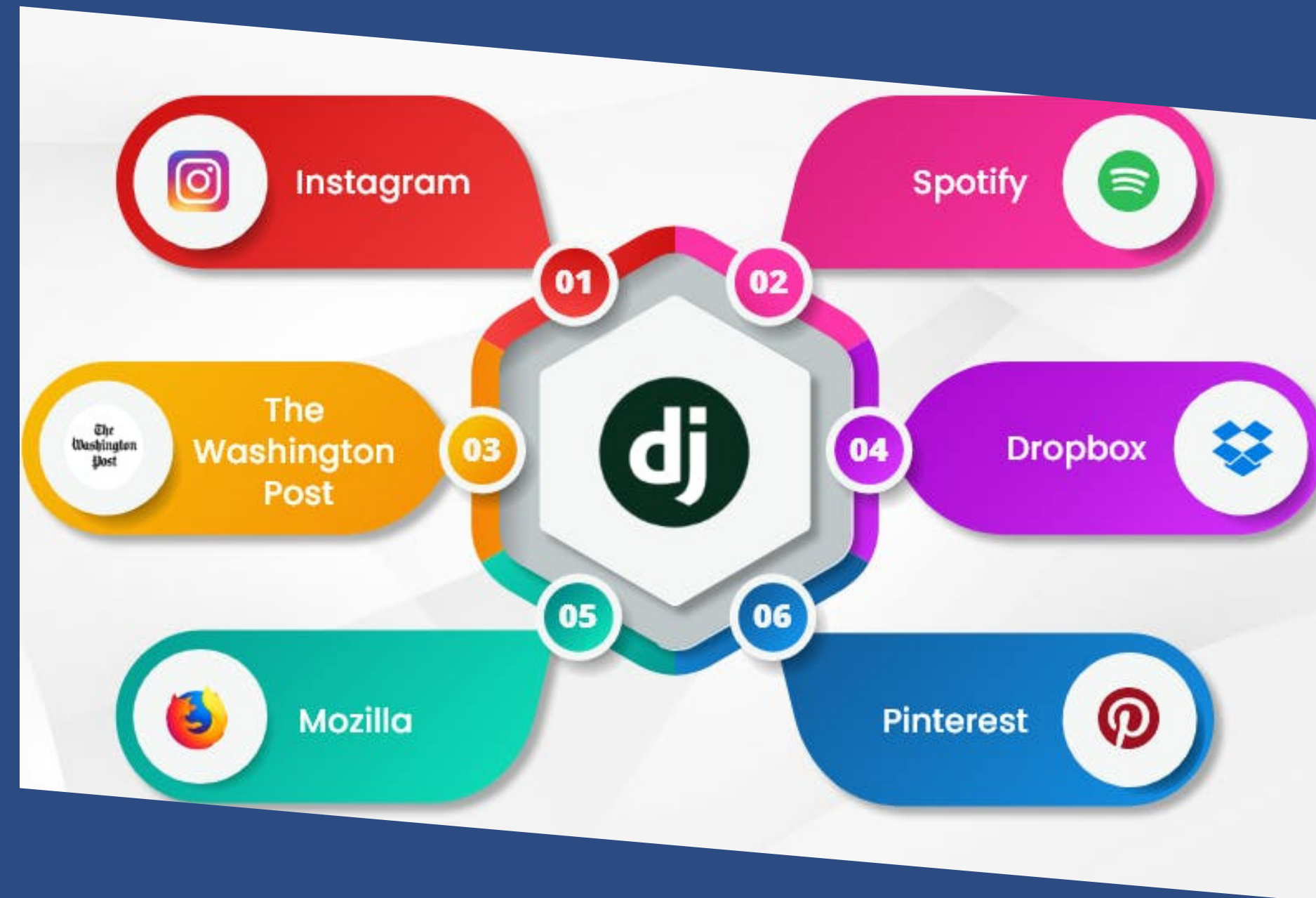




# Django zalety

## Skalowalność

Pozwala on na podejmowanie wielu różnych działań dotyczących skalowalności, takich jak uruchamianie oddzielnych serwerów dla bazy danych, mediów i samej aplikacji, a nawet korzystanie z klastrowania lub równoważenia obciążenia w celu dystrybucji aplikacji na wielu serwerach.



# Django zalety

## Zgodne z KISS i DRY

Django podąża za zasadą DRY (**Don't Repeat Yourself**), co oznacza, że możesz zastąpić często powtarzające się wzorce programistyczne abstrakcjami lub użyć normalizacji danych. W ten sposób można uniknąć redundancji i błędów.

Ponadto ponowne wykorzystanie kodu upraszcza tworzenie strony internetowej, dzięki czemu można skupić się na kodowaniu unikalnych funkcji.

KISS oznacza "**Keep It Simple, Stupid**", pośród wielu jego odmian. W Django oznacza to prosty, czytelny i zrozumiały kod. Na przykład, metody nie powinny być dłuższe niż 40-50 linii.

```
def get_object(self, pk):  
    try:  
        return models.TODO.objects.get(pk=pk)  
    except models.TODO.DoesNotExist:  
        raise Http404
```

```
def delete(self, request, pk):  
    todo = self.get_object(pk)  
    todo.delete()  
    return Response(status=status.HTTP_204_NO_CONTENT)
```

```
def patch(self, request, pk):  
    todo = self.get_object(pk)  
    serializer = serializers.TODOSerializer(todo, data=request.data, partial=True)  
    if serializer.is_valid():  
        serializer.save()  
        return Response(serializer.data)  
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

# Django wady

## Monolityczny

Django nie zapewnia dużej swobody, jeśli chodzi o architekturę projektu, Django ma swój własny sposób robienia rzeczy. Jeśli deweloper próbuje zaimplementować coś poza strukturą plików, to nie ma możliwości, by Django uzyskało dostęp do tego pliku.

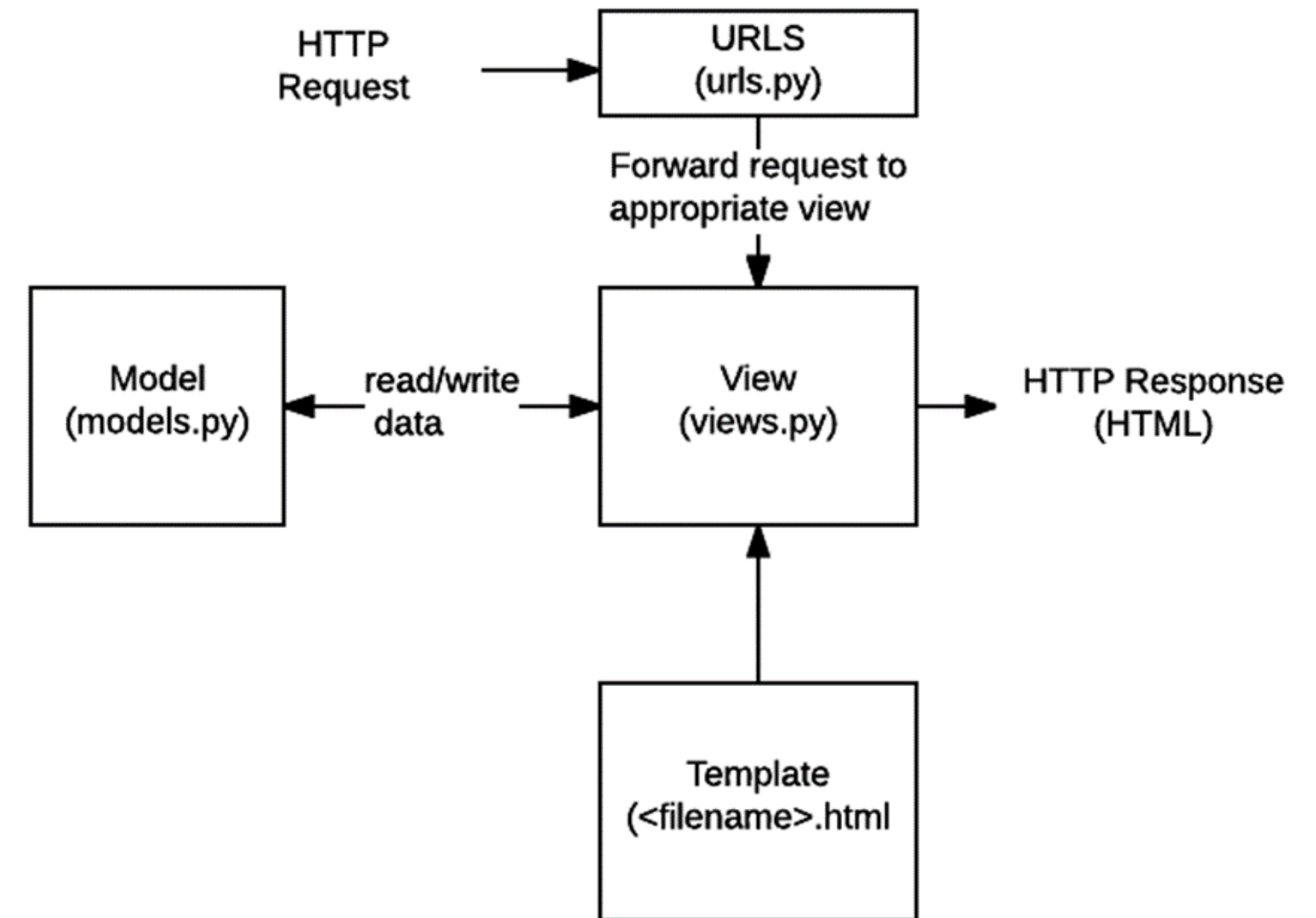


# Model MVT

## MVT - Model,View,Template

### Co to MVT?

MVT wzorec projektowy oprogramowania do tworzenia aplikacji internetowych. Jest to zbiór trzech ważnych komponentów: Modelu, Widoku i Szablonu. Model pomaga obsługiwać bazę danych. Jest to warstwa dostępu do danych, która obsługuje dane.





# Model

Model działa jako interfejs danych. Jest on odpowiedzialny za utrzymanie danych. To logiczna struktura danych stojąca za całą aplikacją i jest reprezentowana przez bazę danych (zazwyczaj relacyjne bazy danych, takie jak MySql, Postgres).



# Model w kodzie

models.py X

backend > todo > models.py > ...

```
1  from django.db import models
2
3
4
5  class Todo(models.Model):
6      body = models.CharField(max_length=300)
7      content = models.TextField(max_length=1000, default=" ")
8      completed = models.BooleanField(default=False)
9      updated = models.DateTimeField(auto_now=True)
10     created = models.DateTimeField(auto_now_add=True)
11
12
```

## Todo List

GET /api/todo/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
[
  {
    "id": 43,
    "body": "task 1",
    "completed": false,
    "content": "",
    "updated": "2024-04-07T18:32:40.888828Z",
    "created": "2024-04-07T18:32:40.888828Z"
  },
  {
    "id": 44,
    "body": "task 2",
    "completed": false,
    "content": "zawartość content",
    "updated": "2024-04-07T18:32:59.366442Z",
    "created": "2024-04-07T18:32:44.056320Z"
  }
]
```

# View

View to interfejs użytkownika, czyli to co jest widzione w przeglądarce podczas renderowania strony internetowej. Jest on reprezentowany przez HTML/CSS/Javascript.

```
from django.shortcuts import render
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from django.http import Http404
from . import serializers
from . import models

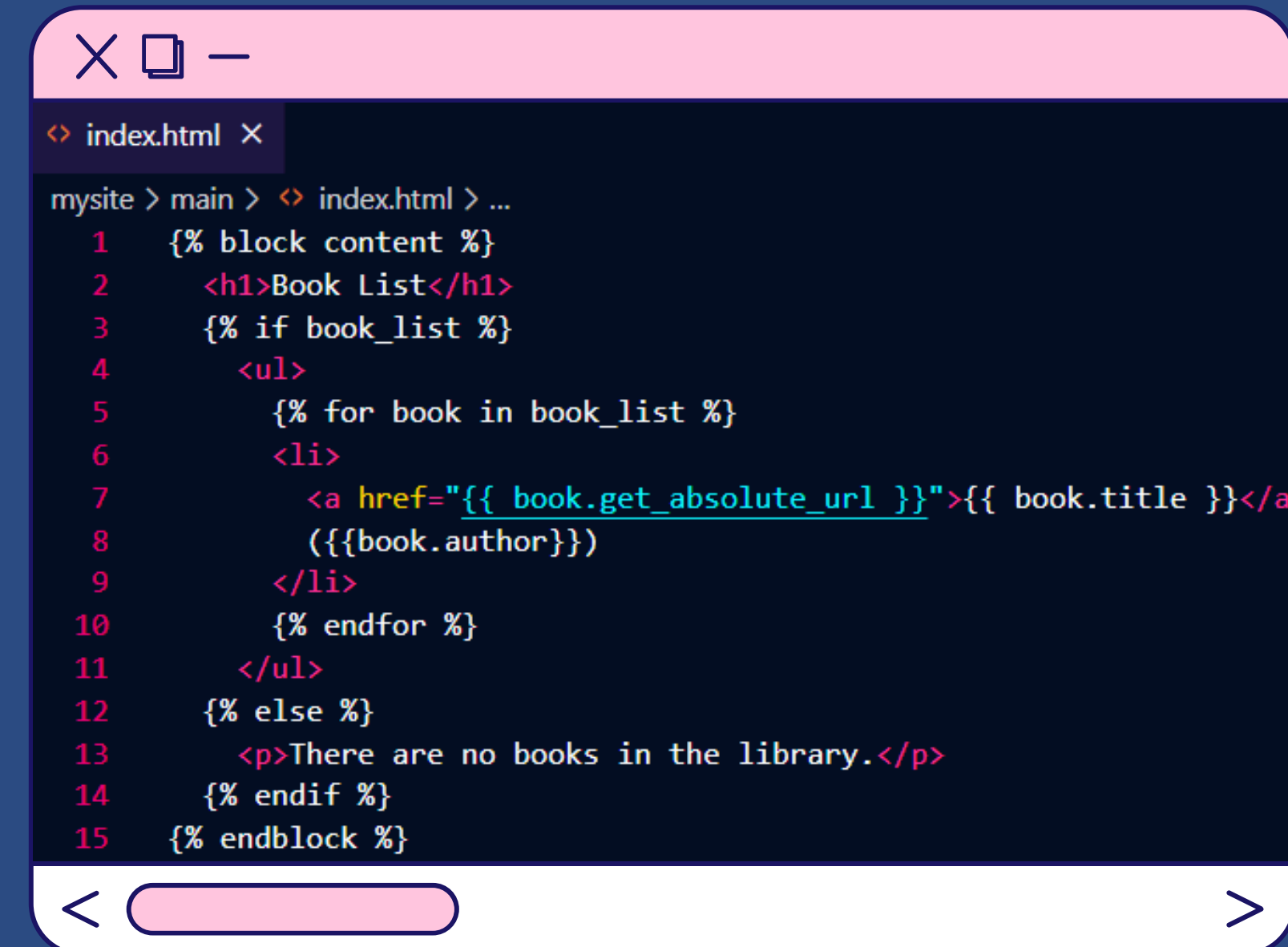
class TodoList(APIView):

    def get(self, request):
        todos = models.Todo.objects.all()
        serializer = serializers.TodoSerializer(todos, many=True)
        return Response(serializer.data)

    def post(self, request):
        serializer = serializers.TodoSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

# Template

W Django, szablony (templates) są używane do tworzenia interfejsu użytkownika. Są one plikami HTML z dodatkowymi znacznikami i filtrowaniami Django Template Language (DTL), które pozwalają na dynamiczne generowanie zawartości w oparciu o dane dostarczone z widoków.



```
index.html X
mysite > main > <> index.html > ...
1 {% block content %}
2   <h1>Book List</h1>
3   {% if book_list %}
4     <ul>
5       {% for book in book_list %}
6         <li>
7           <a href="{{ book.get_absolute_url }}">{{ book.title }}</a>
8             ({{book.author}})
9         </li>
10      {% endfor %}
11    </ul>
12  {% else %}
13    <p>There are no books in the library.</p>
14  {% endif %}
15 {% endblock %}
```

# Django template language

Jest to to język szablonów używany w frameworku Django do generowania dynamicznych treści HTML.

```
home.html x
8 {% if message_list %}
9 <table class="message_list">
10     <thead>
11         <tr>
12             <th>Date</th>
13             <th>Time</th>
14             <th>Message</th>
15         </tr>
16     </thead>
17     <tbody>
18         {% for message in message_list %}
19         <tr>
20             <td>{{ message.log_date | date:'d M Y' }}</td>
21             <td>{{ message.log_date | date:'H:i:s' }}</td>
22             <td>
```

# Użycie DTL'a

## Zmienne

hello.html X

mysite > main > hello.html > html

```
1 <html>
2     <body>
3         Hello World!!!<p>Today is {{today}}</p>
4     </body>
5 </html>
```

```
9 def hello(request):
10     today = datetime.datetime.now().date()
11     return render(request, "hello.html", {"today" : today})
```

# Tag if & Tag for

<> hello.html X

mysite > main > <> hello.html > ...

```
1  <html>
2    <body>
3
4      Hello World!!!<p>Today is {{today}}</p>
5      We are
6      {% if today.day == 1 %}
7
8      the first day of month.
9      {% elif today.day == 30 %}
10
11     the last day of month.
12     {% else %}
13
14     I don't know.
15     {%endif%}
16
17   </body>
18 </html>
```

<> hello.html X

mysite > main > <> hello.html > ...

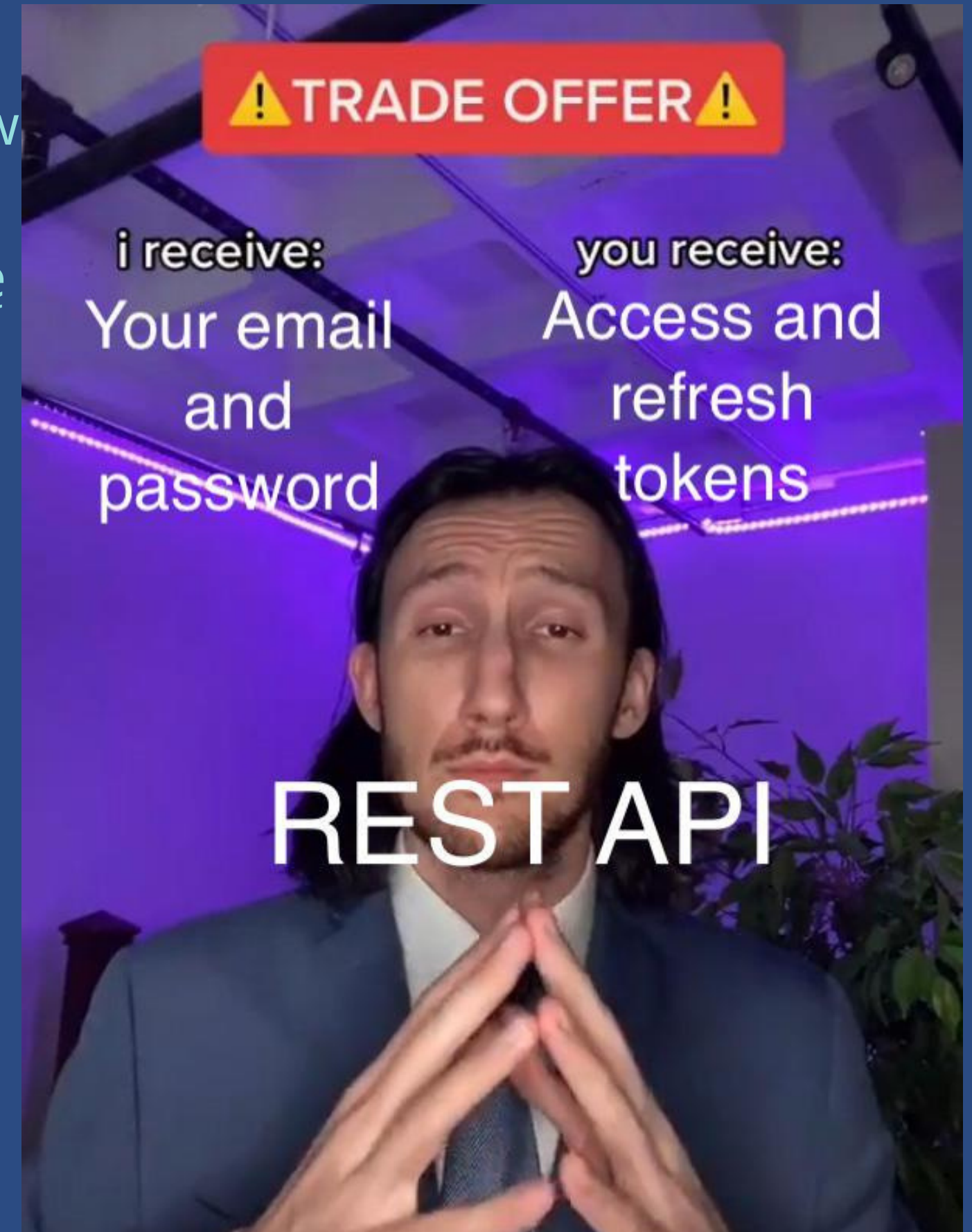
```
1  <html>
2    <body>
3
4      Hello World!!!<p>Today is {{today}}</p>
5      We are
6      {% if today.day == 1 %}
7
8      the first day of month.
9      {% elif today.day == 30 %}
10
11     the last day of month.
12     {% else %}
13
14     I don't know.
15     {%endif%}
16
17     <p>
18       {% for day in days_of_week %}
19         {{day}}
20       </p>
21
22     {% endfor %}
23
24   </body>
25 </html>
```

```
9  def hello(request):
10     today = datetime.datetime.now().date()
11
12     daysOfWeek = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
13     return render(request, "hello.html", {"today" : today, "days_of_week" : daysOfWeek})
```



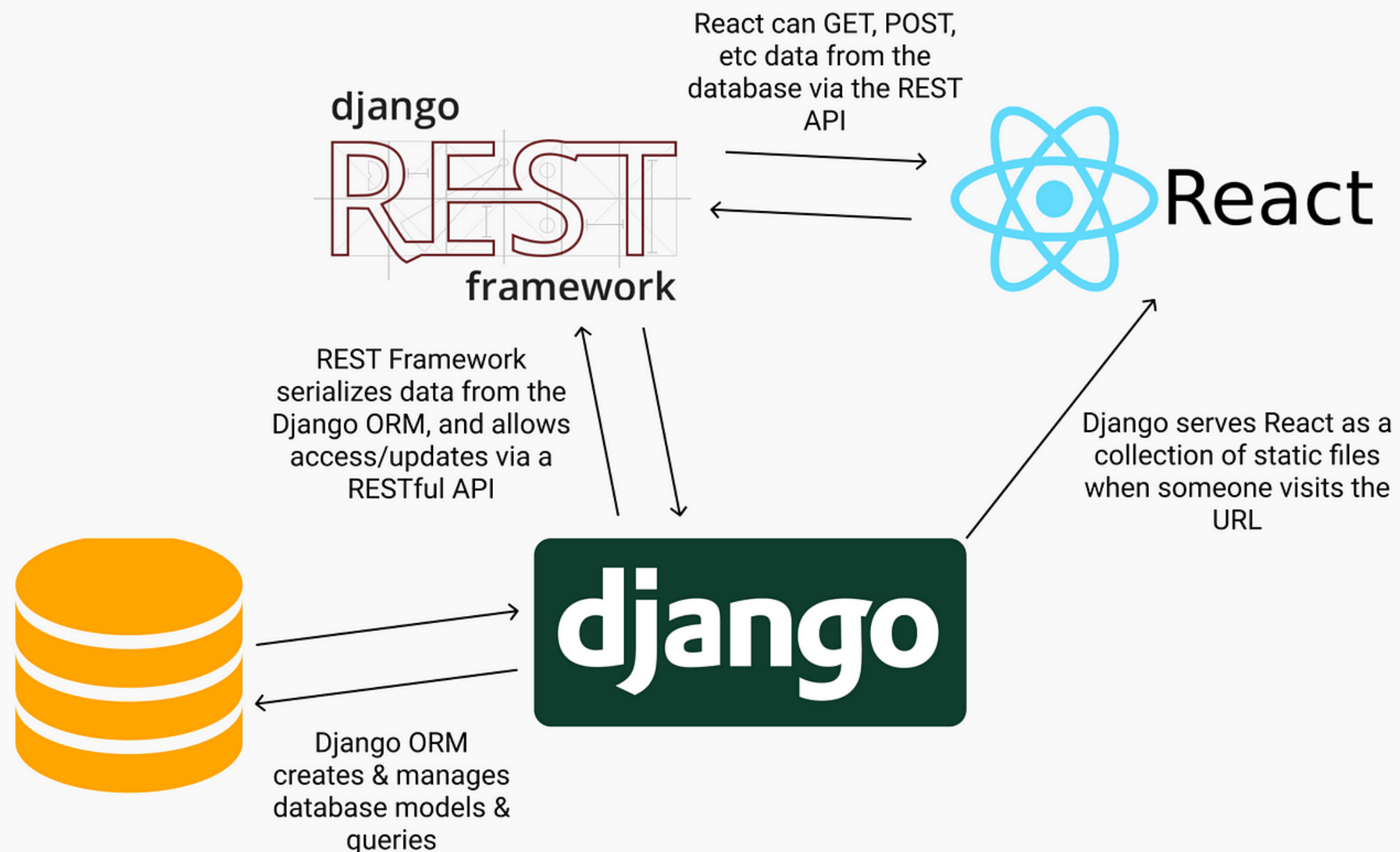
# Django REST framework

Django REST to specjalistyczny framework Python typu open source, który umożliwia budowę zaawansowanych interfejsów API Web. Posiada szereg funkcjonalnych narzędzi o budowie modułowej, które zapewniają wysoką elastyczność i swobodę w dostosowywaniu architektury aplikacji. Dzięki temu programiści mogą budować zarówno proste w obsłudze interfejsy API, jak i skomplikowane systemy REST (Representational State Transfer), które umożliwiają komunikację pomiędzy API a aplikacją czy witryną internetową. Jest to oprogramowanie stosunkowo proste w obsłudze, a jego główną ideą jest podzielenie modelu reprezentacji transferu (np.: Jason, XML) oraz szybkich widoków zbudowanych na klasach, które w łatwy sposób można dostosować do potrzeb użytkownika i rozszerzyć ich funkcjonalności.





# Współpraca backendu z frontendem



# Dziękujemy za uwagę

