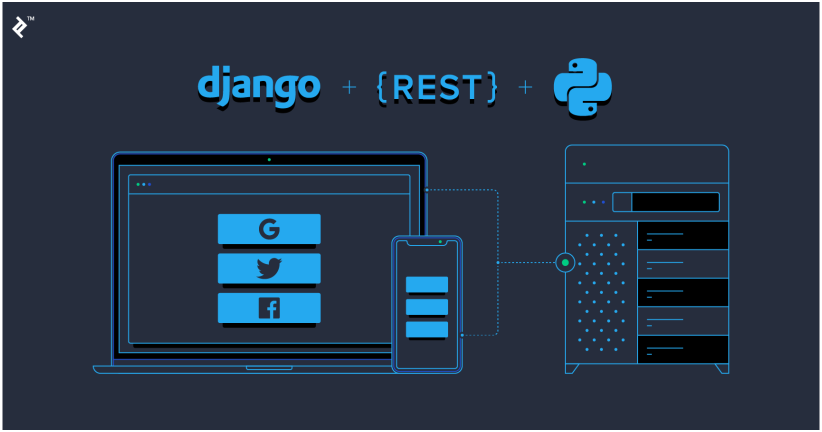
Django

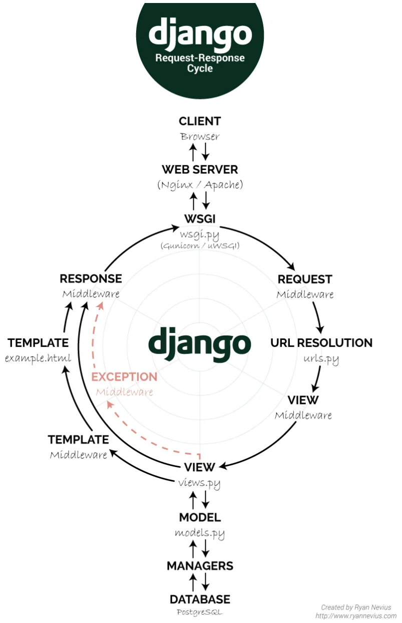
Jan Zbrocki

Uniwersytet Wrocławski

16.10.2025


django + {REST} +

Request-Response Cycle



Client sends request

It all begins when a client (typically a web browser) sends a request to your Django server. This request could be anything from visiting a web page to submitting a form. These requests are made using HTTP methods like:

- GET: Used to request data from the server (e.g., a web page)

Client sends request

It all begins when a client (typically a web browser) sends a request to your Django server. This request could be anything from visiting a web page to submitting a form. These requests are made using HTTP methods like:

- GET: Used to request data from the server (e.g., a web page)
- POST: Used to send data to the server (e.g., submitting a form)

Client sends request

It all begins when a client (typically a web browser) sends a request to your Django server. This request could be anything from visiting a web page to submitting a form. These requests are made using HTTP methods like:

- GET: Used to request data from the server (e.g., a web page)
- POST: Used to send data to the server (e.g., submitting a form)
- DELETE: Used to delete data from a server

Client sends request

It all begins when a client (typically a web browser) sends a request to your Django server. This request could be anything from visiting a web page to submitting a form. These requests are made using HTTP methods like:

- GET: Used to request data from the server (e.g., a web page)
- POST: Used to send data to the server (e.g., submitting a form)
- DELETE: Used to delete data from a server
- PUT: Used to actualize data on server

Web Server - native

Deafalt django server can be started with command:

```
python manage.py runserver
```

It creates developer server on port 8000 on localhost and it's perfect for debugging. But it's not scalable, not safe and it is single threaded.

Web Server - Nginx

Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. Released in 2004. Nginx is free and open-source software. A large fraction of web servers use Nginx, often as a load balancer.

WSGI

The **Web Server Gateway Interface** is a simple calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language.

Examples:

- Native django WSGI, It is single threaded :(

WSGI

The **Web Server Gateway Interface** is a simple calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language.
Examples:

- Native django WSGI, It is single threaded :(
- Gunicorn (natively supports Django), Faster

Request

Once the request hits the server, Django picks it up. This is where Django's powerful architecture comes into play. The request is wrapped into an HttpRequest object, which contains all the details about the request such as:

- Request Method: GET, POST, etc.

Request

Once the request hits the server, Django picks it up. This is where Django's powerful architecture comes into play. The request is wrapped into an HttpRequest object, which contains all the details about the request such as:

- Request Method: GET, POST, etc.
- URL: The path the user requested.

Request

Once the request hits the server, Django picks it up. This is where Django's powerful architecture comes into play. The request is wrapped into an HttpRequest object, which contains all the details about the request such as:

- Request Method: GET, POST, etc.
- URL: The path the user requested.
- Headers: Information like cookies, user agent, etc.

Request

Once the request hits the server, Django picks it up. This is where Django's powerful architecture comes into play. The request is wrapped into an HttpRequest object, which contains all the details about the request such as:

- Request Method: GET, POST, etc.
- URL: The path the user requested.
- Headers: Information like cookies, user agent, etc.
- Body: For POST requests, this contains form data or file uploads.

Url routing

After the request is encapsulated in the HttpRequest object, it's passed through Django's URL routing system. Django looks for a matching URL pattern in your urls.py file. If it finds one, it sends the request to the corresponding view function. from django.urls import path from . import views

```
urlpatterns = [
    path('home/', views.home_view, name='home'),
]
```

View

This is where logic for request starts. The view receives the HttpRequest object, processes the data, interacts with the database if needed, and prepares a response.

Middleware

Before Django sends the response back to the client, it passes the request and response through several middleware components. Middleware is a series of hooks that can process requests and responses globally.

- Modify the request before it reaches the view.

Middleware

Before Django sends the response back to the client, it passes the request and response through several middleware components. Middleware is a series of hooks that can process requests and responses globally.

- Modify the request before it reaches the view.
- Modify the response before it's sent to the client.

Middleware

Before Django sends the response back to the client, it passes the request and response through several middleware components. Middleware is a series of hooks that can process requests and responses globally.

- Modify the request before it reaches the view.
- Modify the response before it's sent to the client.
- Handle authentication, logging, or session management.

Creating Django project

- As every other package django can be installed with packege manager:

  ```
  pip install django
  ```

Creating Django project

- As every other package django can be installed with packege manager:

  ```
  pip install django
  ```

- Create project:

  ```
  django-admin startproject <name>
  ```

Creating Django project

- As every other package django can be installed with packege manager:

  ```
  pip install django
  ```

- Create project:

  ```
  django-admin startproject <name>
  ```

- Create first app in the project:

  ```
  python3 manage.py startapp <app name>
  ```

  and add it to INSTALLED_APPS in settings.py

View

```python
def detail(request, question_id):
    return HttpResponse(
        "You're looking at question %s."
        % question_id
    )


def results(request, question_id):
    response = "You're looking at \
the results of question %s."
    return HttpResponse(response % question_id)


def vote(request, question_id):
    return HttpResponse(
        "You're voting on question %s."
        % question_id
    )
```

View

```python
class UserViewSet(viewsets.ViewSet):
    def list(self, request):
        queryset = User.objects.all()
        serializer = UserSerializer(
            queryset,
            many=True,
        )
        return Response(serializer.data)

    def retrieve(self, request, pk=None):
        queryset = User.objects.all()
        user = get_object_or_404(queryset, pk=pk)
        serializer = UserSerializer(user)
        return Response(serializer.data)
```

View

```
class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer

    @action(detail=True, methods=['post'])
    def set_password(self, request, pk=None):
        user = self.get_object()
        serializer = PasswordSerializer(
            data=request.data
        )
        if serializer.is_valid():
            user.set_password(
                serializer.validated_data['password']
            user.save()
            return Response({
                'status': 'password␣set'})
        else:
            return Response(serializer.errors,
                status=status.HTTP_400_BAD_REQUEST)
```

View

```python
class SomeViewSet(viewsets.ModelViewSet):
    def some_action(self, request, pk=None):
        serializer = SomeSerializer(
            data=request.data
        )
        if serializer.is_valid():
            service = SomeService(
                serializer.validated_data
            )
            return Response(
                status=status.HTTP_200_OK
            )
        else:
            return Response(
                serializer.errors,
                status=status.HTTP_400_BAD_REQUEST
            )
```

Models

- In Django we have "all in"package so among other things we have full DB handling implemented.

Models

- In Django we have "all in"package so among other things we have full DB handling implemented.
- To handle DB tables we have Models (very similar to SQLAlchemy) which have class structures.

Models

- In Django we have "all in"package so among other things we have full DB handling implemented.
- To handle DB tables we have Models (very similar to SQLAlchemy) which have class structures.
- To handle queries to DB we have Django ORM with all its pros and cons.

Models

```python
from django.db import models
class Reporter(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    email = models.EmailField()

    def __str__(self):
        return f"{self.first_name}␣{self.last_name}"
class Article(models.Model):
    headline = models.CharField(max_length=100)
    pub_date = models.DateField()
    reporter = models.ForeignKey(Reporter, on_delete=
    def __str__(self):
        return self.headline
    class Meta:
        ordering = ["headline"]
```

Models

In Django we have migrations. These are representation of layers that were made via models to create DB.

- python manage.py makemigrations

Models

In Django we have migrations. These are representation of layers that were made via models to create DB.

- python manage.py makemigrations
- python manage.py migrate

Models

In Django we have migrations. These are representation of layers that were made via models to create DB.

- python manage.py makemigrations
- python manage.py migrate
- python manage.py showmigrations

Django ORM

To work on queries Django implements **QuerySet**. QuerySet is result of most database queries.

How can we get all articles from **Article** model?

Django ORM

To work on queries Django implements **QuerySet**. QuerySet is result of most database queries.

How can we get all articles from **Article** model?
Simply:

```
Article.objects.all()
```

Django ORM

Let's see some QuerySet refinement methods

- `filter()`
  `f.e. Reporter.objects.filter(first_name="Jan")`
  `or Article.objects.filter(reporter__first_name="Jan")`

Django ORM

Let's see some QuerySet refinement methods

- `filter()`
  f.e. `Reporter.objects.filter(first_name="Jan")`
  or `Article.objects.filter(reporter__first_name="Jan")`
- `exclude()`

Django ORM

Let's see some QuerySet refinement methods

- `filter()`
  f.e. `Reporter.objects.filter(first_name="Jan")`
  or `Article.objects.filter(reporter__first_name="Jan")`
- `exclude()`
- `annotate()`

Django ORM

Let's see some QuerySet refinement methods

- `filter()`
  `f.e. Reporter.objects.filter(first_name="Jan")`
  `or Article.objects.filter(reporter__first_name="Jan")`
- `exclude()`
- `annotate()`
- `order_by()`

Django ORM

Let's see some QuerySet refinement methods

- `filter()`
  f.e. `Reporter.objects.filter(first_name="Jan")`
  or `Article.objects.filter(reporter__first_name="Jan")`
- `exclude()`
- `annotate()`
- `order_by()`
- `values()`

Django ORM

Let's see some QuerySet refinement methods

- `filter()`
  `f.e. Reporter.objects.filter(first_name="Jan")`
  `or Article.objects.filter(reporter__first_name="Jan")`
- `exclude()`
- `annotate()`
- `order_by()`
- `values()`
- `values_list()`

Django ORM

Let's see some QuerySet refinement methods

- `filter()`
  f.e. `Reporter.objects.filter(first_name="Jan")`
  or `Article.objects.filter(reporter__first_name="Jan")`
- `exclude()`
- `annotate()`
- `order_by()`
- `values()`
- `values_list()`
- `get()`

Django ORM

Let's see how to do DB changes

- `create()`

  ```
  f.e.
  p = Person.objects.create(
      first_name="Bruce",
      last_name="Springsteen",
  )
  same as
  p = Person(
      first_name="Bruce",
      last_name="Springsteen",
  )
  p.save(force_insert=True)
  ```

Django ORM

Let's see how to do DB changes

- `create()`

  ```
  f.e.
  p = Person.objects.create(
      first_name="Bruce",
      last_name="Springsteen",
  )
  same as
  p = Person(
      first_name="Bruce",
      last_name="Springsteen",
  )
  p.save(force_insert=True)
  ```
- `update()`

Django ORM

Let's see how to do DB changes

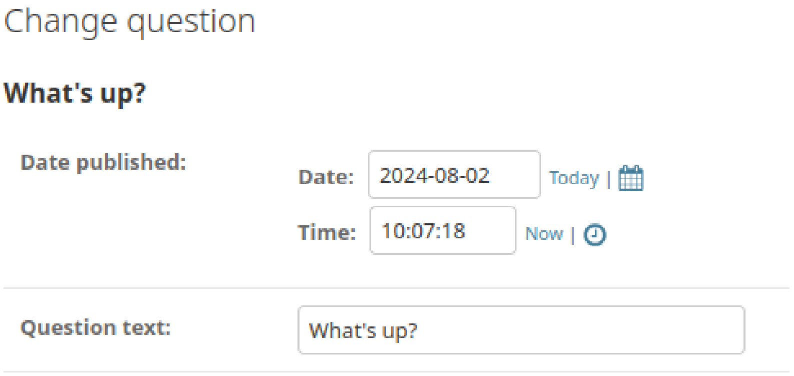
- `create()`

  ```
  f.e.
  p = Person.objects.create(
      first_name="Bruce",
      last_name="Springsteen",
  )
  same as
  p = Person(
      first_name="Bruce",
      last_name="Springsteen",
  )
  p.save(force_insert=True)
  ```
- `update()`
- `delete()`

DjangoAdmin

```python
from django.contrib import admin

from .models import Question


class QuestionAdmin(admin.ModelAdmin):
    fields = ["pub_date", "question_text"]


admin.site.register(Question, QuestionAdmin)
```

DjangoAdmin



Change question

What's up?

Date published:	**Date:** 2024-08-02	Today |
	Time: 10:07:18	Now |
Question text:	What's up?	

Django test

To run tests we type command:

```
python manage.py test polls
```

Output:

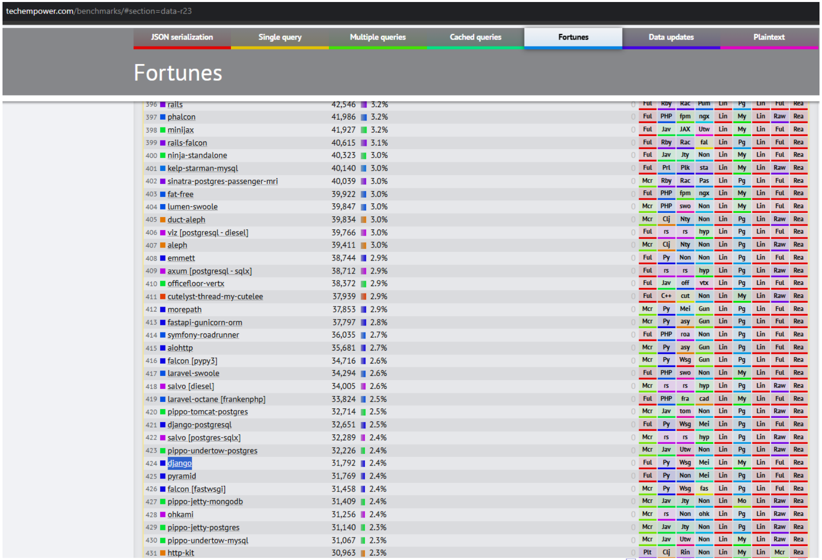
```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
==============================================================================
FAIL: test_was_published_recently_with_future_question (polls.te
------------------------------------------------------------------------------
Traceback (most recent call last):
  File "/path/to/djangotutorial/polls/tests.py", line 16, in tes
    self.assertIs(future_question.was_published_recently(), Fals
AssertionError: True is not False

------------------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
Destroying test database for alias 'default'...
```

Django test

```python
import datetime

from django.test import TestCase
from django.utils import timezone

from .models import Question

class QuestionModelTests(TestCase):
    def test_was_published_recently_with_future_questi
        time = (
            timezone.now() +
            datetime.timedelta(days=30)
        )
        future_question = Question(pub_date=time)
        self.assertIs(
            future_question.was_published_recently(),
            False
        )
```

Django in benchmarks

`https://www.techempower.com/benchmarks/#section=data-r23`



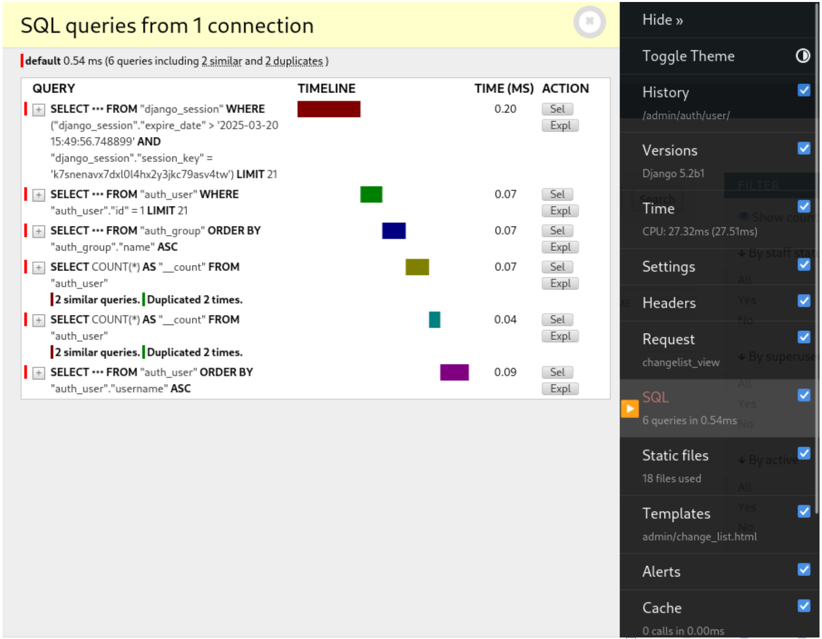
techempower.com/benchmarks/#section=data-r23

JSON serialization | Single query | Multiple queries | Cached queries | Fortunes | Data updates | Plaintext

Fortunes

396	🟪 rails	42,546	3.2%	0	Ful	Rby	Rac	Pum	Lin	Pg	Lin	Ful	Rea
397	🟦 phalcon	41,986	3.2%	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
398	🟩 minijax	41,927	3.2%	0	Ful	Jav	JAX	Utw	Lin	My	Lin	Ful	Rea
399	🟪 rails-falcon	40,615	3.1%	0	Ful	Rby	Rac	fal	Lin	Pg	Lin	Ful	Rea
400	🟩 ninja-standalone	40,323	3.0%	0	Ful	Jav	Jty	Non	Lin	My	Lin	Ful	Rea
401	🟦 kelp-starman-mysql	40,140	3.0%	0	Ful	Prl	Plk	sta	Lin	My	Lin	Raw	Rea
402	🟪 sinatra-postgres-passenger-mri	40,039	3.0%	0	Mcr	Rby	Rac	Pas	Lin	Pg	Lin	Ful	Rea
403	🟦 fat-free	39,922	3.0%	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Ful	Rea
404	🟦 lumen-swoole	39,847	3.0%	0	Mcr	PHP	swo	Non	Lin	My	Lin	Ful	Rea
405	🟧 duct-aleph	39,834	3.0%	0	Mcr	Clj	Nty	Non	Lin	Pg	Lin	Raw	Rea
406	🟪 viz [postgresql - diesel]	39,766	3.0%	0	Ful	rs	rs	hyp	Lin	Pg	Lin	Ful	Rea
407	🟧 aleph	39,411	3.0%	0	Mcr	Clj	Nty	Non	Lin	Pg	Lin	Raw	Rea
408	🟦 emmett	38,744	2.9%	0	Ful	Py	Non	Non	Lin	Pg	Lin	Ful	Rea
409	🟪 axum [postgresql - sqlx]	38,712	2.9%	0	Ful	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
410	🟩 officefloor-vertx	38,372	2.9%	0	Ful	Jav	off	vtx	Lin	Pg	Lin	Ful	Rea
411	🟥 cutelyst-thread-my-cutelee	37,939	2.9%	0	Ful	C++	cut	Non	Lin	My	Lin	Raw	Rea
412	🟦 morepath	37,853	2.9%	0	Mcr	Py	Mei	Gun	Lin	Pg	Lin	Ful	Rea
413	🟦 fastapi-gunicorn-orm	37,797	2.8%	0	Mcr	Py	asy	Gun	Lin	Pg	Lin	Ful	Rea
414	🟦 symfony-roadrunner	36,035	2.7%	0	Ful	PHP	roa	Non	Lin	Pg	Lin	Ful	Rea
415	🟦 aiohttp	35,681	2.7%	0	Mcr	Py	asy	Gun	Lin	Pg	Lin	Ful	Rea
416	🟦 falcon [pypy3]	34,716	2.6%	0	Mcr	Py	Wsg	Gun	Lin	Pg	Lin	Ful	Rea
417	🟦 laravel-swoole	34,294	2.6%	0	Ful	PHP	swo	Non	Lin	My	Lin	Ful	Rea
418	🟪 salvo [diesel]	34,005	2.6%	0	Mcr	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
419	🟦 laravel-octane [frankenphp]	33,824	2.5%	0	Ful	PHP	fra	cad	Lin	My	Lin	Ful	Rea
420	🟩 pippo-tomcat-postgres	32,714	2.5%	0	Mcr	Jav	tom	Non	Lin	Pg	Lin	Raw	Rea
421	🟦 django-postgresql	32,651	2.5%	0	Ful	Py	Wsg	Mei	Lin	Pg	Lin	Ful	Rea
422	🟪 salvo [postgres-sqlx]	32,289	2.4%	0	Mcr	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
423	🟩 pippo-undertow-postgres	32,226	2.4%	0	Mcr	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea
424	🟦 django	31,792	2.4%	0	Ful	Py	Wsg	Mei	Lin	My	Lin	Ful	Rea
425	🟦 pyramid	31,769	2.4%	0	Ful	Py	Non	Mei	Lin	Pg	Lin	Ful	Rea
426	🟦 falcon [fastwsgi]	31,458	2.4%	0	Mcr	Py	Wsg	fas	Lin	Pg	Lin	Ful	Rea
427	🟩 pippo-jetty-mongodb	31,409	2.4%	0	Mcr	Jav	Jty	Non	Lin	Mo	Lin	Raw	Rea
428	🟪 ohkami	31,256	2.4%	0	Mcr	rs	Non	ohk	Lin	Pg	Lin	Raw	Rea
429	🟩 pippo-jetty-postgres	31,140	2.3%	0	Mcr	Jav	Jty	Non	Lin	Pg	Lin	Raw	Rea
430	🟩 pippo-undertow-mysql	31,067	2.3%	0	Mcr	Jav	Utw	Non	Lin	My	Lin	Raw	Rea
431	🟧 http-kit	30,963	2.3%	0	Plt	Clj	Rin	Non	Lin	My	Lin	Mcr	Rea

Django toolbar



SQL queries from 1 connection

default 0.54 ms (6 queries including 2 similar and 2 duplicates)

QUERY	TIMELINE	TIME (MS)	ACTION
SELECT ••• FROM "django_session" **WHERE** ("django_session"."expire_date" > '2025-03-20 15:49:56.748899' **AND** "django_session"."session_key" = 'k7snenavx7dxl0l4hx2y3jkc79asv4tw') **LIMIT** 21	🟥	0.20	Sel Expl
SELECT ••• FROM "auth_user" **WHERE** "auth_user"."id" = 1 **LIMIT** 21	🟩	0.07	Sel Expl
SELECT ••• FROM "auth_group" **ORDER BY** "auth_group"."name" **ASC**	🟦	0.07	Sel Expl
SELECT COUNT(*) **AS** "__count" **FROM** "auth_user" **2 similar queries.** **Duplicated 2 times.**	🟨	0.07	Sel Expl
SELECT COUNT(*) **AS** "__count" **FROM** "auth_user" **2 similar queries.** **Duplicated 2 times.**	🟦	0.04	Sel Expl
SELECT ••• FROM "auth_user" **ORDER BY** "auth_user"."username" **ASC**	🟪	0.09	Sel Expl

Hide »

Toggle Theme

History ✓
/admin/auth/user/

Versions ✓
Django 5.2b1

Time ✓
CPU: 27.32ms (27.51ms)

Settings ✓

Headers ✓

Request ✓
changelist_view

SQL ✓
6 queries in 0.54ms

Static files ✓
18 files used

Templates ✓
admin/change_list.html

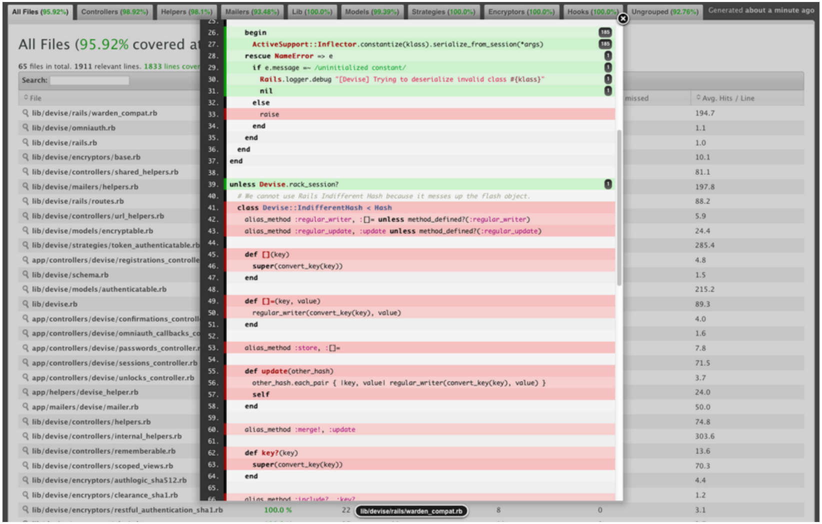
Alerts ✓

Cache ✓
0 calls in 0.00ms

Django additional concepts - Aggregation

```
class Book(models.Model):

    name = models.CharField(max_length=300)

    pages = models.IntegerField()

    price = models.DecimalField(max_digits=10, decima

    rating = models.FloatField()

    authors = models.ManyToManyField(Author)

    publisher = models.ForeignKey(Publisher, on_delete

    pubdate = models.DateField()


Book.objects.aggregate(Max("price", default=0))
```

Django additional concepts - Coverage report



All Files (95.92%) | Controllers (98.92%) | Helpers (98.1%) | Mailers (93.48%) | Lib (100.0%) | Models (99.39%) | Strategies (100.0%) | Encryptors (100.0%) | Hooks (100.0%) | Ungrouped (92.76%) | Generated about a minute ago

All Files (95.92% covered at

65 files in total. 1911 relevant lines. 1833 lines cover

Search:

File	Avg. Hits / Line
lib/devise/rails/warden_compat.rb	194.7
lib/devise/omniauth.rb	1.1
lib/devise/rails.rb	1.0
lib/devise/encryptors/base.rb	10.1
lib/devise/controllers/shared_helpers.rb	81.1
lib/devise/mailers/helpers.rb	197.8
lib/devise/rails/routes.rb	88.2
lib/devise/controllers/url_helpers.rb	5.9
lib/devise/models/encryptable.rb	24.4
lib/devise/strategies/token_authenticatable.rb	285.4
app/controllers/devise/registrations_controlle	4.8
lib/devise/schema.rb	1.5
lib/devise/models/authenticatable.rb	215.2
lib/devise.rb	89.3
app/controllers/devise/confirmations_controll	4.0
app/controllers/devise/omniauth_callbacks_co	1.6
app/controllers/devise/passwords_controller.r	7.8
app/controllers/devise/sessions_controller.rb	71.5
app/controllers/devise/unlocks_controller.rb	3.7
app/helpers/devise_helper.rb	24.0
app/mailers/devise/mailer.rb	50.0
lib/devise/controllers/helpers.rb	74.8
lib/devise/controllers/internal_helpers.rb	303.6
lib/devise/controllers/rememberable.rb	13.6
lib/devise/controllers/scoped_views.rb	70.3
lib/devise/encryptors/authlogic_sha512.rb	4.4
lib/devise/encryptors/clearance_sha1.rb	1.2

lib/devise/encryptors/restful_authentication_sha1.rb 100.0 % 22 8 0 3.1

```
    begin                                                                          185
      ActiveSupport::Inflector.constantize(klass).serialize_from_session(*args)    185
    rescue NameError => e                                                          1
      if e.message =~ /uninitialized constant/                                     1
        Rails.logger.debug "[Devise] Trying to deserialize invalid class #{klass}" 1
        nil                                                                        1
      else
        raise
      end
    end
  end
end

unless Devise.rack_session?                                                        1
  # We cannot use Rails Indifferent Hash because it messes up the flash object.
  class Devise::IndifferentHash < Hash
    alias_method :regular_writer, :[]= unless method_defined?(:regular_writer)
    alias_method :regular_update, :update unless method_defined?(:regular_update)

    def [](key)
      super(convert_key(key))
    end

    def []=(key, value)
      regular_writer(convert_key(key), value)
    end

    alias_method :store, :[]=

    def update(other_hash)
      other_hash.each_pair { |key, value| regular_writer(convert_key(key), value) }
      self
    end

    alias_method :merge!, :update

    def key?(key)
      super(convert_key(key))
    end

    alias_method :include?, :key?
```

lib/devise/rails/warden_compat.rb

Django additional concepts

- Raw SQL

Django additional concepts

- Raw SQL
- Database denormalization

Django additional concepts

- Raw SQL
- Database denormalization
- PostgreSQL

Django additional concepts

- Raw SQL
- Database denormalization
- PostgreSQL
- Signals

Django additional concepts

- Raw SQL
- Database denormalization
- PostgreSQL
- Signals
- Django debug toolbar

Django

These are some basic and some more advanced concepts of Django.

Any questions?