Django 8: Django REST Framework

IN608: Intermediate Application Development Concepts

Kaiako: Tom Clark & Grayson Orr

Last Session’s Content

- Security in Django
    - XSS protection
    - CSRF protection
    - SQL injection protection
    - Clickjacking protection
    - Secret key

Today’s Content

- Django REST framework
    - Permissions
    - Serialization
    - Viewsets
    - Routers
- GraphQL

Django REST Framework

Django REST Framework

- Install Django REST framework

```
# Windows
...\> pipenv install djangorestframework

# Linux or macOS
$ pipenv install djangorestframework
```

Django REST Framework

- View `Pipfile` & `Pipfile.lock`

```
[[source]]
name = "pypi"
url = "https://pypi.org/simple"
verify_ssl = true

[dev-packages]

[packages]
django = "*"
selenium = "*"
django-crispy-forms = "*"
djangorestframework = "*"

[requires]
python_version = "3.7"
```

Django REST Framework

- `mvt/settings.py`
- Resource: https://www.django-rest-framework.org/#installation

```
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'crispy_forms',
    'rest_framework',
]
```

Permissions

- `mvt/settings.py`
- Default permission policy may be set globally using the `DEFAULT_PERMISSION_CLASSES` setting
- `IsAdminUser` permission class will deny permission to any user unless `user.is_staff` is `True`
  - Suitable if you want your API to only be accessible to trusted admins
- `IsAuthenticated` permission class will deny permission to any unauthenticated user
- Resource: https://www.django-rest-framework.org/api-guide/permissions

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAdminUser',
    ]
}
```

Serialization

- In the `polls/` directory, create a file called `serializers.py`
- App structure:

```
polls/
    __init__.py
    admin.py
    apps.py
    forms.py
    migrations/
        __init__.py
    models.py
    serializers.py
    static/
        polls/
            style.css
    templates/
        polls/
            base.html
            detail.html
            index.html
            results.html
    tests.py
    urls.py
    views.py
```

Serialization

- `polls/serializers.py`
- `ModelSerializer`
  - Automatically generates a set of fields based on the model
  - Automatically generates validators for the serializer
  - Includes default implementations of `.create()` & `.update()`
- Resource: https://www.django-rest-framework.org/api-guide/serializers

```python
from rest_framework import serializers
from .models import User, Question, Choice

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'email', 'first_name', 'last_name']

class QuestionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Question
        fields = ['id', 'question_text', 'pub_date']

class ChoiceSerializer(serializers.ModelSerializer):
    class Meta:
        model = Choice
        fields = ['id', 'question', 'choice_text', 'votes']
```

Viewsets

- `polls/views.py`
- `from rest_framework import viewsets`
- `from .serializers import UserSerializer, QuestionSerializer, ChoiceSerializer`
- `ModelViewSet` - provides all actions
- `ReadOnlyModelViewSet` - only provides read-only actions, i.e, `.list()` & `.retrieve()`
- Resource: https://www.django-rest-framework.org/api-guide/viewsets

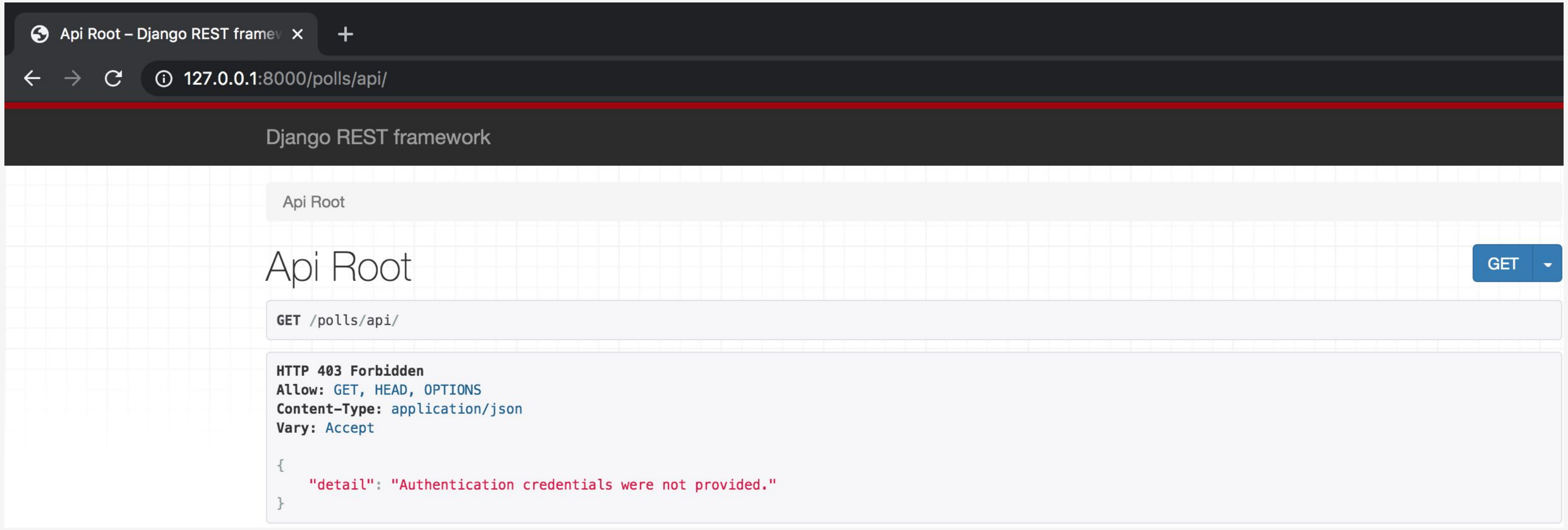
```python
class UserViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer

class QuestionViewSet(viewsets.ModelViewSet):
    queryset = Question.objects.all()
    serializer_class = QuestionSerializer

class ChoiceViewSet(viewsets.ModelViewSet):
    queryset = Choice.objects.all()
    serializer_class = ChoiceSerializer
```

Routers

- `polls/urls.py`
- `DefaultRouter()` - returns a response containing hyperlinks to all the views
- `register()` - prefix & viewset
- Resource: https://www.django-rest-framework.org/#installation

```
from django.contrib.auth import views as auth_views
from django.urls import include, path
from rest_framework import routers
from . import views

app_name = 'polls'

router = routers.DefaultRouter()
router.register(r'api/user', views.UserViewSet)
router.register(r'api/question', views.QuestionViewSet)
router.register(r'api/choice', views.ChoiceViewSet)

urlpatterns = [
    path('api/', include(router.urls), name='api'),
    path('accounts/signup/', views.SignupView.as_view(), name='signup'),
    path('accounts/login/', auth_views.LoginView.as_view(), name='login'),
    path('accounts/logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('', views.IndexView.as_view(), name='index'), # /polls/
    path('<int:pk>/', views.DetailView.as_view(), name='detail'), # /polls/2/
    path('<int:pk>/results/', views.ResultsView.as_view(), name='results'), # /polls/2/results/
    path('<int:question_id>/vote/', views.vote, name='vote'), # /polls/2/vote/
]

urlpatterns += router.urls
```

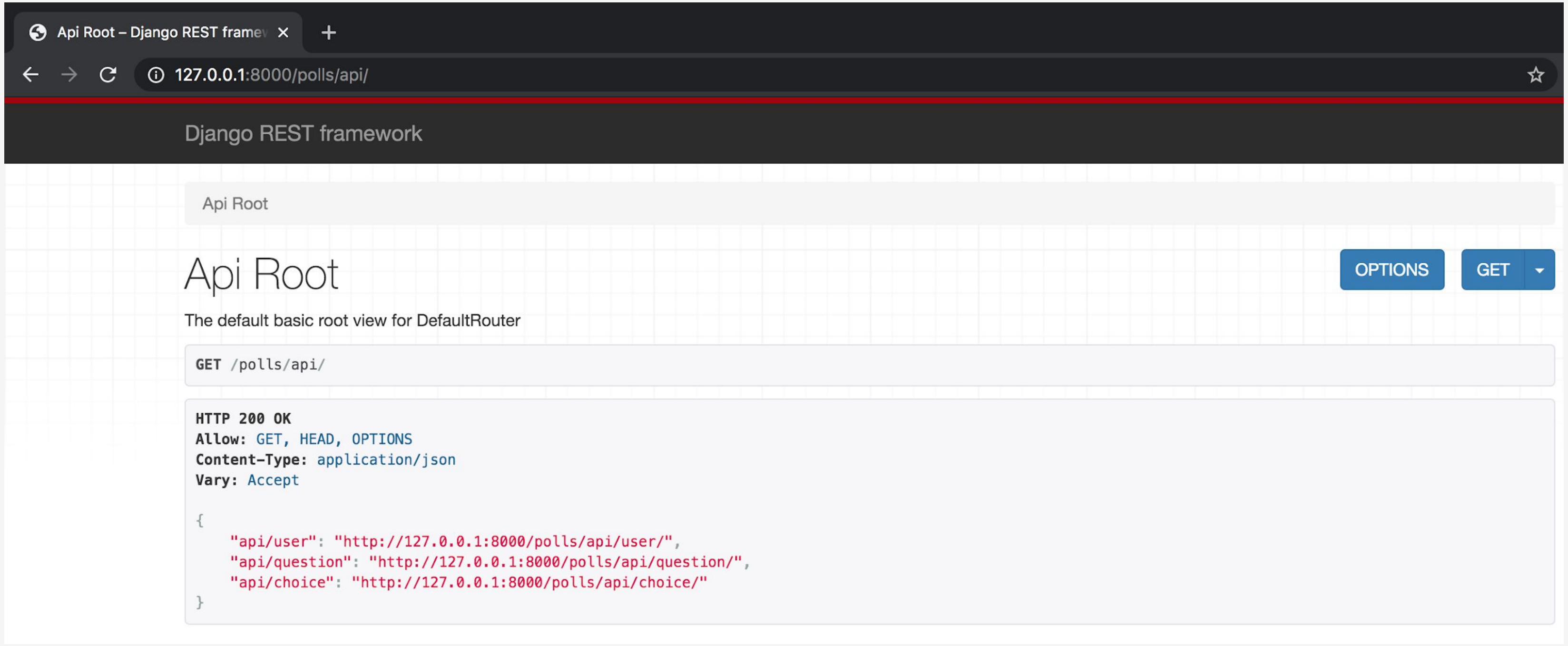
Django REST Framework



```
GET /polls/api/
```

```
HTTP 403 Forbidden
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "detail": "Authentication credentials were not provided."
}
```

Django REST Framework



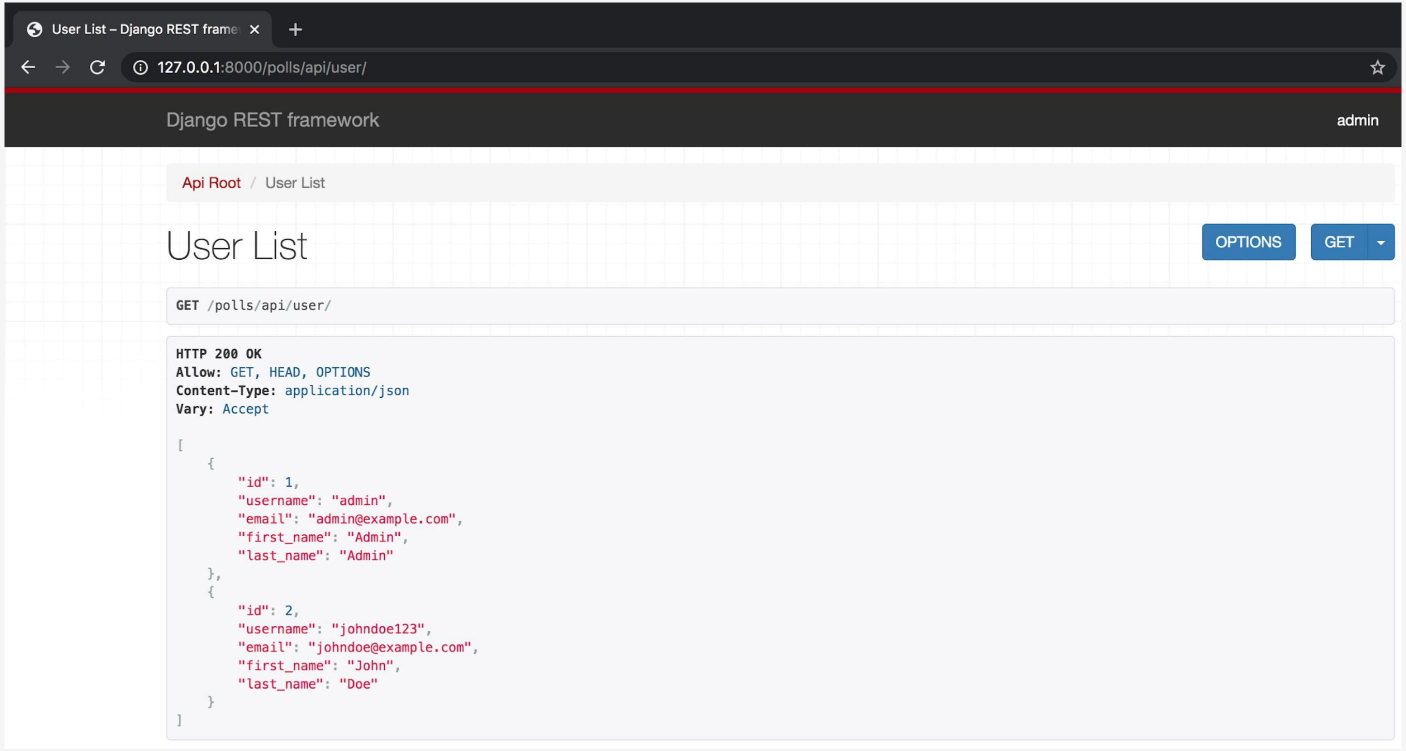
Api Root

The default basic root view for DefaultRouter

```
GET /polls/api/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "api/user": "http://127.0.0.1:8000/polls/api/user/",
    "api/question": "http://127.0.0.1:8000/polls/api/question/",
    "api/choice": "http://127.0.0.1:8000/polls/api/choice/"
}
```

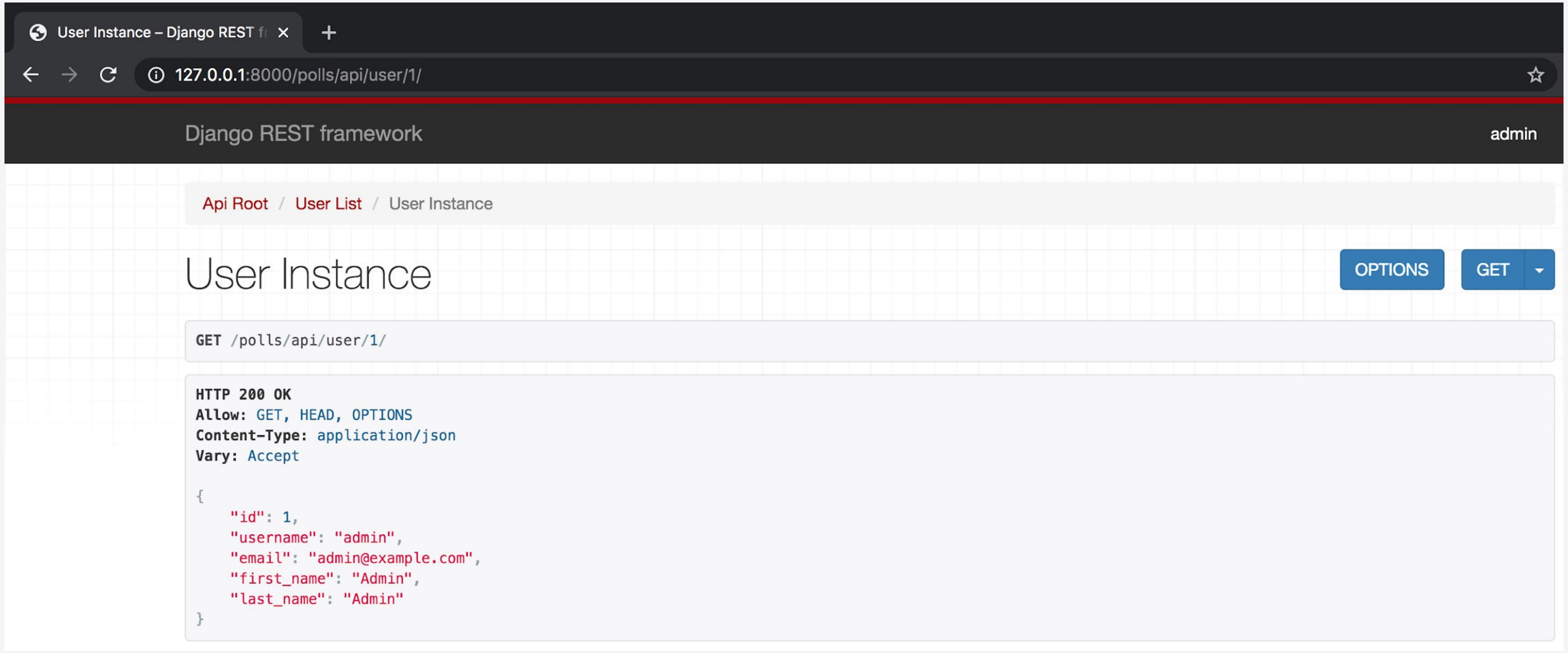
Django REST Framework



```
GET /polls/api/user/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "username": "admin",
        "email": "admin@example.com",
        "first_name": "Admin",
        "last_name": "Admin"
    },
    {
        "id": 2,
        "username": "johndoe123",
        "email": "johndoe@example.com",
        "first_name": "John",
        "last_name": "Doe"
    }
]
```

Django REST Framework

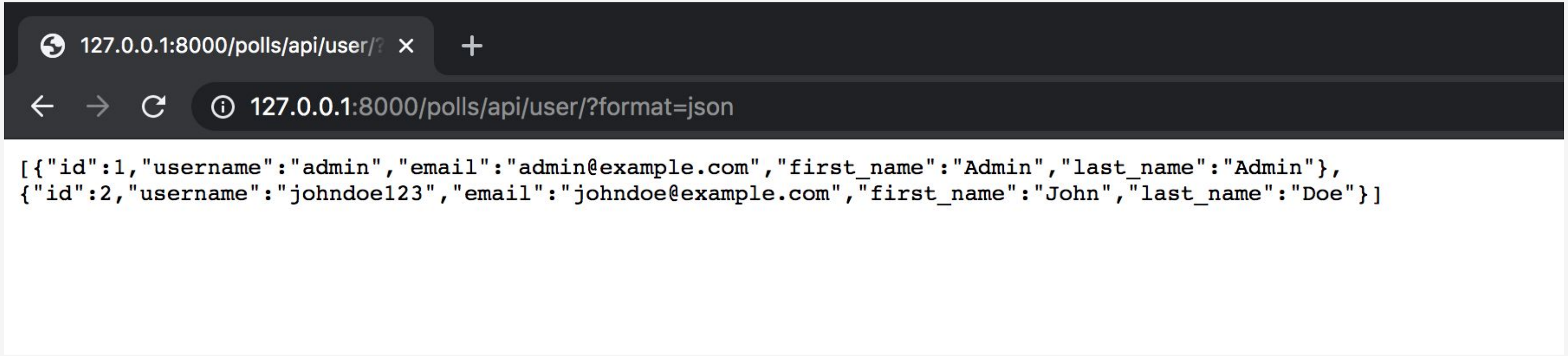

User Instance – Django REST fr
127.0.0.1:8000/polls/api/user/1/
Django REST framework
admin
Api Root / User List / User Instance
OPTIONS
GET

User Instance

```
GET /polls/api/user/1/
```

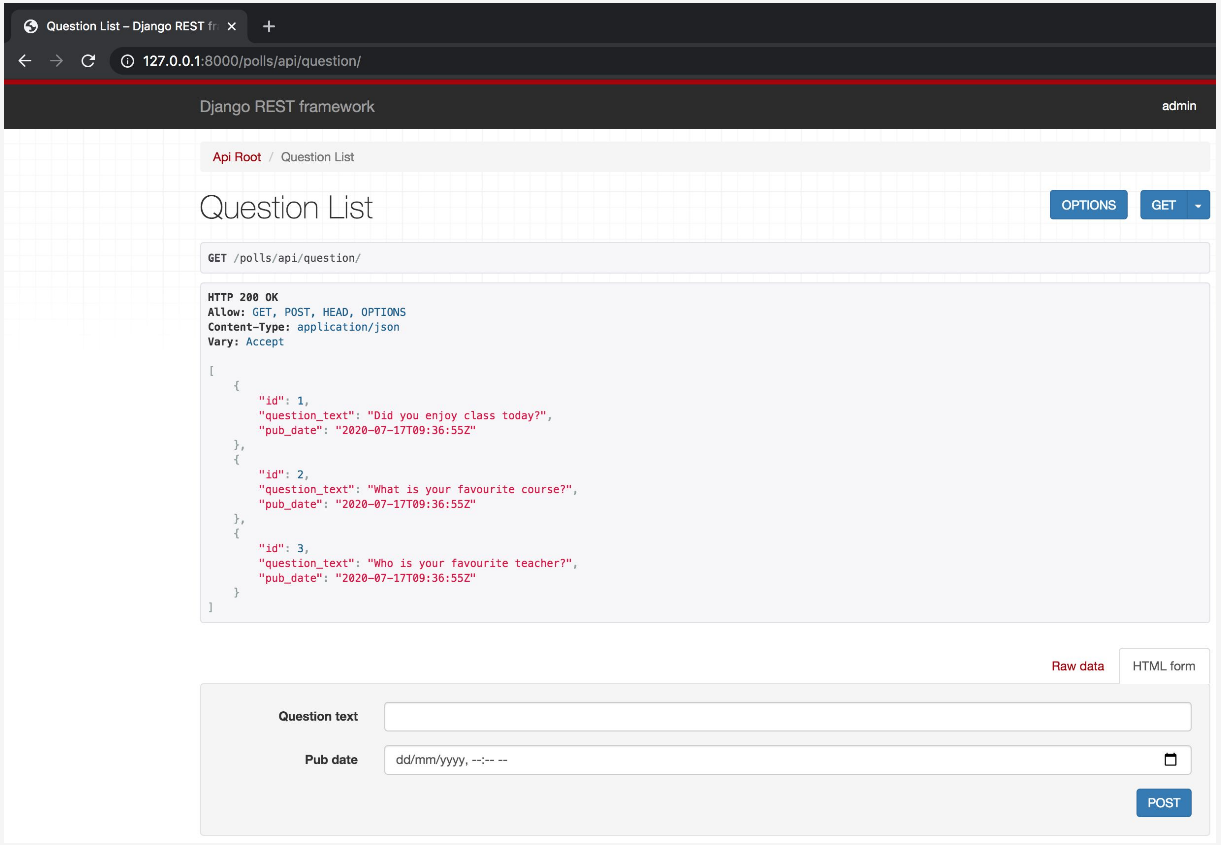
```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "username": "admin",
    "email": "admin@example.com",
    "first_name": "Admin",
    "last_name": "Admin"
}
```

Django REST Framework


127.0.0.1:8000/polls/api/user/?
127.0.0.1:8000/polls/api/user/?format=json

```
[{"id":1,"username":"admin","email":"admin@example.com","first_name":"Admin","last_name":"Admin"},
{"id":2,"username":"johndoe123","email":"johndoe@example.com","first_name":"John","last_name":"Doe"}]
```

Django REST Framework


Question List – Django REST fr
127.0.0.1:8000/polls/api/question/
Django REST framework
admin
Api Root / Question List

Question List

```
GET /polls/api/question/
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "question_text": "Did you enjoy class today?",
        "pub_date": "2020-07-17T09:36:55Z"
    },
    {
        "id": 2,
        "question_text": "What is your favourite course?",
        "pub_date": "2020-07-17T09:36:55Z"
    },
    {
        "id": 3,
        "question_text": "Who is your favourite teacher?",
        "pub_date": "2020-07-17T09:36:55Z"
    }
]
```

Question text	
Pub date	dd/mm/yyyy, --:-- --

GraphQL

GraphQL

- A query language for your API
- A server-side runtime for executing queries
- Does not use any specific database or storage engine
- Resources:
  - https://graphql.org/learn
  - https://graphene-python.org

GraphQL

- Install Graphene Django

```
# Windows
...\> pipenv install graphene-django

# Linux or macOS
$ pipenv install graphene-django
```

GraphQL

- View `Pipfile` & `Pipfile.lock`

```
[[source]]
name = "pypi"
url = "https://pypi.org/simple"
verify_ssl = true

[dev-packages]

[packages]
django = "*"
selenium = "*"
django-crispy-forms = "*"
djangorestframework = "*"
graphene-django = "*"

[requires]
python_version = "3.7"
```

GraphQL

- `mvt/settings.py`
- Resource: https://docs.graphene-python.org/projects/django/en/latest/installation

```
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'crispy_forms',
    'rest_framework',
    'graphene_django',
]
```

GraphQL

- `polls/urls.py`
- `from graphene_django.views import GraphQLView`
- `path('graphql/', GraphQLView.as_view(graphiql=True))`
- GraphQL is accessed through a single URL
- Requests to this URL are handled by Graphene’s `GraphQLView` view
- `graphiql=True` - serves as the GraphQL endpoint

GraphQL

- In the `polls/` directory, create a file called `schema.py`
- App structure:

```
polls/
    __init__.py
    admin.py
    apps.py
    forms.py
    migrations/
        __init__.py
    models.py
    schema.py
    serializers.py
    static/
        polls/
            style.css
    templates/
        polls/
            base.html
            detail.html
            index.html
            results.html
    tests.py
    urls.py
    views.py
```

GraphQL

- polls/schema.py

```python
from graphene import Field, Int, List
from graphene_django.types import DjangoObjectType
from .models import Question, Choice

class QuestionType(DjangoObjectType):
    class Meta:
        model = Question

class ChoiceType(DjangoObjectType):
    class Meta:
        model = Choice

class Query:
    question = Field(QuestionType, id=Int())
    all_questions = List(QuestionType)
    choice = Field(ChoiceType, id=Int())
    all_choices = List(ChoiceType)

    def resolve_question(self, int, **kwargs):
        id = kwargs.get('id')
        if id is not None:
            return Question.objects.get(pk=id)
        return None

    def resolve_all_questions(self, info, **kwargs):
        return Question.objects.all()

    def resolve_choice(self, int, **kwargs):
        id = kwargs.get('id')
        if id is not None:
            return Choice.objects.get(pk=id)
        return None

    def resolve_all_choices(self, info, **kwargs):
        return Choice.objects.select_related('question').all()
```

GraphQL

- `DjangoObjectType`
  - Converts a Django model into a `ObjectType`
  - By default, displays all fields in a Django model
  - Use `fields` or `exclude` to display a subset of fields in a Django model
  - Strongly recommended that you explicitly set all fields using the `fields` attribute
- `graphene_django.types`
- Resources:
  - https://docs.graphene-python.org/projects/django/en/latest/queries
  - https://docs.graphene-python.org/en/latest/types/scalars

GraphQL

- `mvt/settings.py`

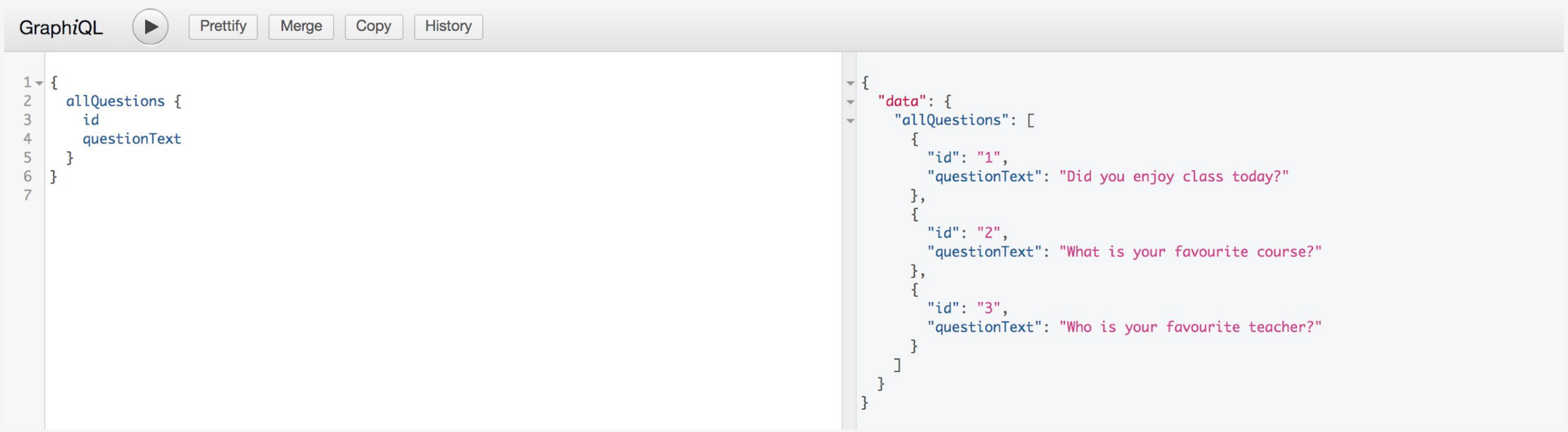
```
GRAPHENE = {
    'SCHEMA': 'mvt.schema.schema'
}
```

GraphQL

- In the `mvt/` directory, create a file called `schema.py`
- `ObjectType` - defines the relationship between `Fields` in your `Schema` & how its data is retrieved
- `Schema` - defines the types & relationships between `Fields` & your API

```python
from graphene import ObjectType, Schema
from polls.schema import Query

class Query(Query, ObjectType):
    pass

schema = Schema(query=Query)
```

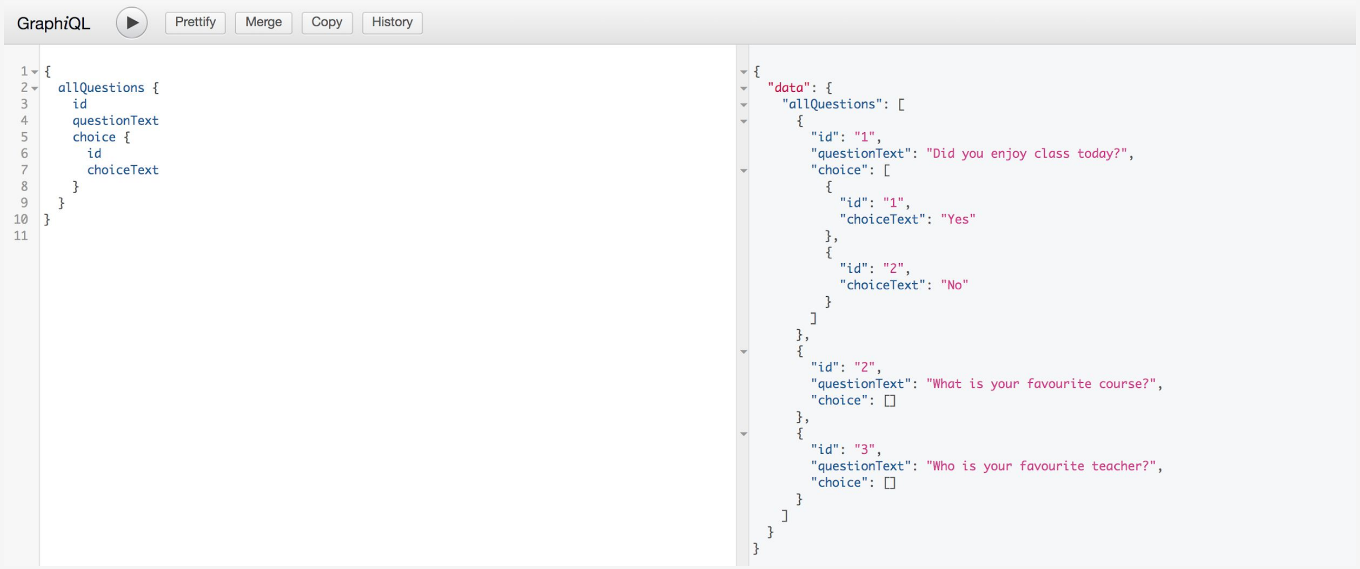
GraphQL



```
{
  allQuestions {
    id
    questionText
  }
}
```

```
{
  "data": {
    "allQuestions": [
      {
        "id": "1",
        "questionText": "Did you enjoy class today?"
      },
      {
        "id": "2",
        "questionText": "What is your favourite course?"
      },
      {
        "id": "3",
        "questionText": "Who is your favourite teacher?"
      }
    ]
  }
}
```

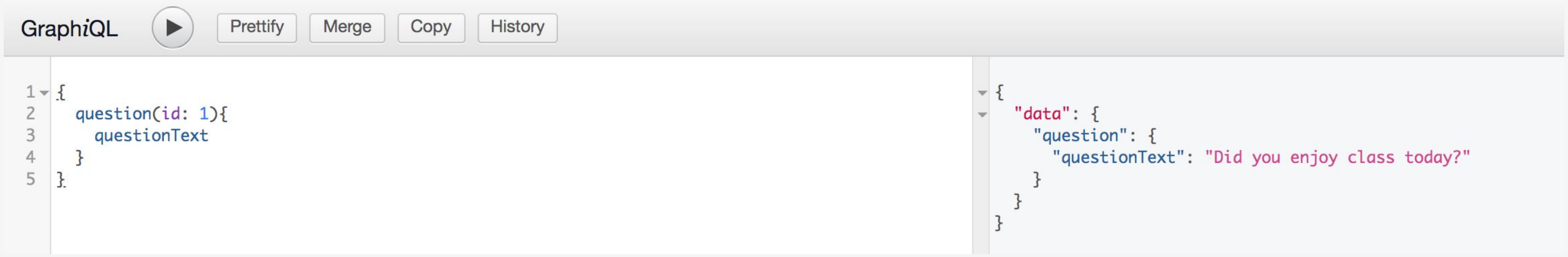
GraphQL



```
{
  allQuestions {
    id
    questionText
    choice {
      id
      choiceText
    }
  }
}
```

```
{
  "data": {
    "allQuestions": [
      {
        "id": "1",
        "questionText": "Did you enjoy class today?",
        "choice": [
          {
            "id": "1",
            "choiceText": "Yes"
          },
          {
            "id": "2",
            "choiceText": "No"
          }
        ]
      },
      {
        "id": "2",
        "questionText": "What is your favourite course?",
        "choice": []
      },
      {
        "id": "3",
        "questionText": "Who is your favourite teacher?",
        "choice": []
      }
    ]
  }
}
```

GraphQL



```
{
  question(id: 1){
    questionText
  }
}
```

```
{
  "data": {
    "question": {
      "questionText": "Did you enjoy class today?"
    }
  }
}
```