Search course

Purchase all 3 parts NOW ONLY $45

HTTP

Part 1, Chapter 5

« Authentication   WebSockets - Part One »

After users log in, they should be taken to a dashboard that displays an overview of their user-related data. Even though we plan to use WebSockets for user-to-user communication, we still have a use for run-of-the-mill HTTP requests. Users should be able to query the server for information about their past, present, and future trips. Up-to-date information is vital for understanding where users have travelled from and for planning where they are traveling to next.

Our HTTP-related tests capture these scenarios.

All Trips

First, let's add a feature to let users view all of the trips associated with their accounts. As an initial step, we'll allow users to see all existing trips; later on in this tutorial, we'll add better filtering.

Test

Add the following test case to the bottom of our existing tests in *server/trips/tests/test_http.py*:

```
# server/trips/tests/test_http.py

class HttpTripTest(APITestCase):
    def setUp(self):
        user = create_user()
        response = self.client.post(reverse('log_in'), data={
            'username': user.username,
            'password': PASSWORD,
        })
        self.access = response.data['access']

    def test_user_can_list_trips(self):
        trips = [
            Trip.objects.create(pick_up_address='A', drop_off_address='B'),
            Trip.objects.create(pick_up_address='B', drop_off_address='C')
        ]
        response = self.client.get(reverse('trip:trip_list'),
            HTTP_AUTHORIZATION=f'Bearer {self.access}'
        )
        self.assertEqual(status.HTTP_200_OK, response.status_code)
        exp_trip_ids = [str(trip.id) for trip in trips]
        act_trip_ids = [trip.get('id') for trip in response.data]
        self.assertCountEqual(exp_trip_ids, act_trip_ids)
```

Update the imports as well:

```
# server/trips/tests/test_http.py

from trips.models import Trip # new
```

Our test creates two trips and then makes a call to the *trip list* API, which should successfully return the trip data.

For now, the test should fail:

Feedback

```
(env)$ python manage.py test trips.tests
```

Error:

```
ImportError: cannot import name 'TripSerializer'
```

We have a lot of work to do in order to get the test passing.

Model

First, we need to create a model that represents the concept of a trip. Update the *server/trips/models.py* file as follows:

```
# server/trips/models.py

import uuid # new

from django.contrib.auth.models import AbstractUser
from django.db import models # new
from django.shortcuts import reverse # new


class User(AbstractUser):
    pass


class Trip(models.Model): # new
    REQUESTED = 'REQUESTED'
    STARTED = 'STARTED'
    IN_PROGRESS = 'IN_PROGRESS'
    COMPLETED = 'COMPLETED'
    STATUSES = (
        (REQUESTED, REQUESTED),
        (STARTED, STARTED),
        (IN_PROGRESS, IN_PROGRESS),
        (COMPLETED, COMPLETED),
    )

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    pick_up_address = models.CharField(max_length=255)
    drop_off_address = models.CharField(max_length=255)
    status = models.CharField(
        max_length=20, choices=STATUSES, default=REQUESTED)

    def __str__(self):
        return f'{self.id}'

    def get_absolute_url(self):
        return reverse('trip:trip_detail', kwargs={'trip_id': self.id})
```

Since a trip is simply a transportation event between a starting location and a destination, we included a pick-up address and a drop-off address. At any given point in time, a trip can be in a specific state, so we added a status to identify whether a trip is requested, started, in progress, or completed. Lastly, we need to have a consistent way to identify and track trips that is also difficult for someone to guess. So, we used a UUID for our `Trip` model.

Let's make a migration for our new model and run it to create the corresponding table.

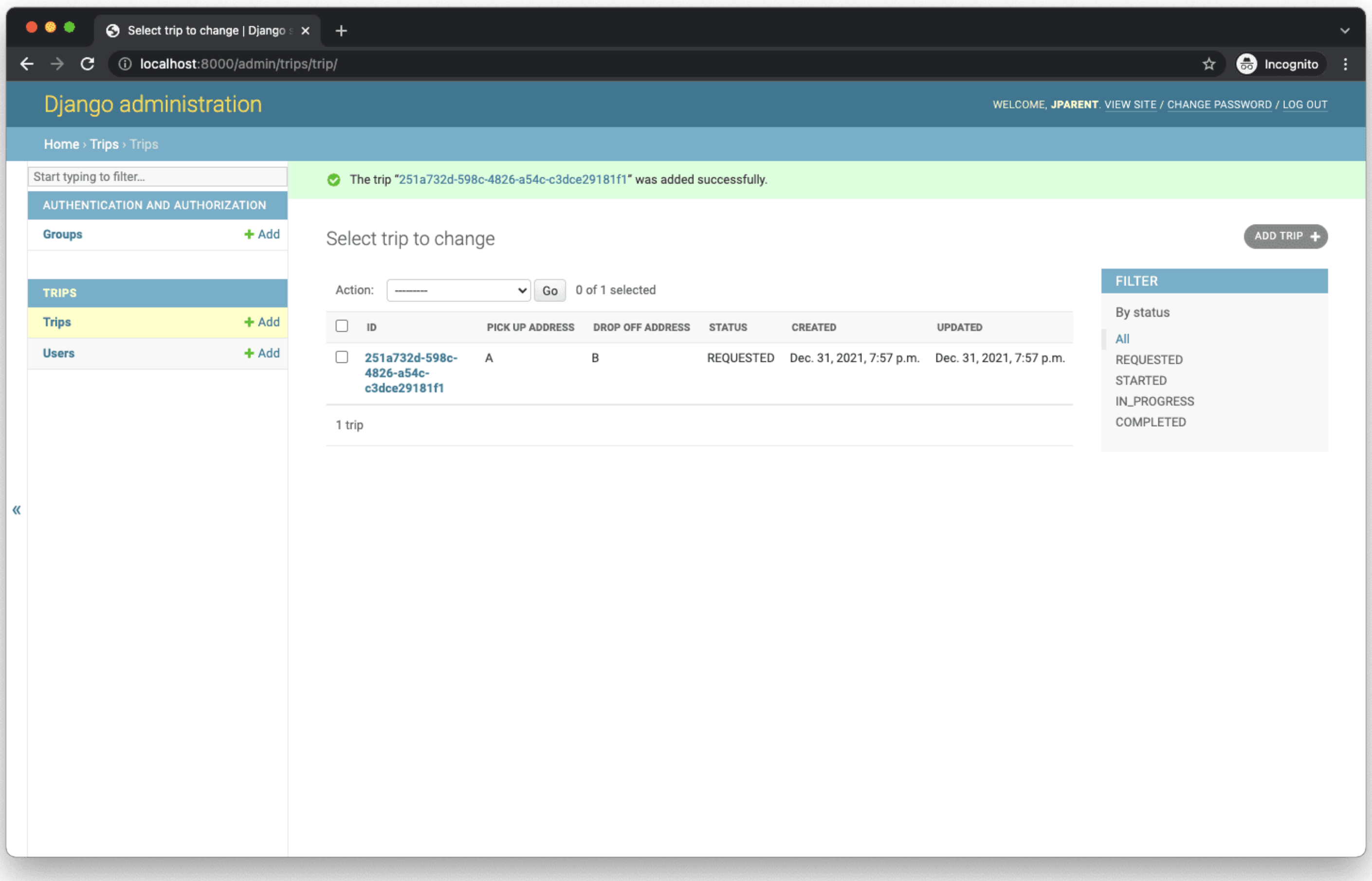
```
(env)$ python manage.py makemigrations
(env)$ python manage.py migrate
```

Admin

Now that our database has a `Trip` table, let's set up the corresponding admin page. Open *server/trips/admin.py* and register a `TripAdmin`:

```python
# server/trips/admin.py

from django.contrib import admin
from django.contrib.auth.admin import UserAdmin as DefaultUserAdmin

from .models import Trip, User # changed


@admin.register(User)
class UserAdmin(DefaultUserAdmin):
    pass


# new
@admin.register(Trip)
class TripAdmin(admin.ModelAdmin):
    fields = (
        'id', 'pick_up_address', 'drop_off_address', 'status', 'created', 'updated',
    )
    list_display = (
        'id', 'pick_up_address', 'drop_off_address', 'status', 'created', 'updated',
    )
    list_filter = (
        'status',
    )
    readonly_fields = (
        'id', 'created', 'updated',
    )
```

Visit the admin page and add a new `Trip` record. You should see something similar to:


Select trip to change | Django
localhost:8000/admin/trips/trip/
Incognito
Django administration
WELCOME, JPARENT. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Trips › Trips
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
TRIPS
Trips + Add
Users + Add
ADD TRIP +
Action: --------- Go
FILTER
By status
All
REQUESTED
STARTED
IN_PROGRESS
COMPLETED

The trip "251a732d-598c-4826-a54c-c3dce29181f1" was added successfully.

Select trip to change

0 of 1 selected

ID	PICK UP ADDRESS	DROP OFF ADDRESS	STATUS	CREATED	UPDATED
251a732d-598c-4826-a54c-c3dce29181f1	A	B	REQUESTED	Dec. 31, 2021, 7:57 p.m.	Dec. 31, 2021, 7:57 p.m.

1 trip

Serializer

Like the user data, we need a way to serialize the trip data to pass it between the client and the server, so add a new serializer to the bottom of the *server/trips/serializers.py* file:

```python
# server/trips/serializers.py

class TripSerializer(serializers.ModelSerializer):
    class Meta:
        model = Trip
        fields = '__all__'
        read_only_fields = ('id', 'created', 'updated',)
```

Add at the top the import:

```
from .models import Trip
```

By identifying certain fields as "read only", we can ensure that they will never be created or updated via the serializer. In this case, we want the server to be responsible for creating the `id`, `created`, and `updated` fields.

View

Add the `TripView` to *server/trips/views.py*:

```
# server/trips/views.py

class TripView(viewsets.ReadOnlyModelViewSet):
    permission_classes = (permissions.IsAuthenticated,)
    queryset = Trip.objects.all()
    serializer_class = TripSerializer
```

As you can see, our `TripView` is incredibly basic. We leveraged the DRF ReadOnlyModelViewSet to support our trip list and trip detail views. For now, our view will return all trips. Note that a user needs to be authenticated in order to access this API.

Update the imports like so:

```
# server/trips/views.py

from django.contrib.auth import get_user_model
from rest_framework import generics, permissions, viewsets # changed
from rest_framework_simplejwt.views import TokenObtainPairView

from .models import Trip # new
from .serializers import LogInSerializer, TripSerializer, UserSerializer # changed
```

URLs

Include the trip-specific URL configuration in the main URLs file, *server/taxi/urls.py*:

```
# server/taxi/urls.py

from django.contrib import admin
from django.urls import include, path # changed
from rest_framework_simplejwt.views import TokenRefreshView

from trips.views import SignUpView, LogInView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/sign_up/', SignUpView.as_view(), name='sign_up'),
    path('api/log_in/', LogInView.as_view(), name='log_in'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('api/trip/', include('trips.urls', 'trip',)), # new
]
```

Then, add our first trip-specific URL, which enables our `TripView` to provide a list of trips. Create a *server/trips/urls.py* file and populate it as follows:

```
# server/trips/urls.py

from django.urls import path

from .views import TripView

app_name = 'taxi'

urlpatterns = [
    path('', TripView.as_view({'get': 'list'}), name='trip_list'),
]
```

> If curious, you can read more about the need to set `app_name` here.

Run the tests again:

```
(env)$ python manage.py test trips.tests
```

Single Trip

Our next, and last, HTTP test covers the trip detail feature. With this feature, users are able to retrieve the details of a trip identified by its primary key (UUID) value.

Add the following test to `HttpTripTest` in *server/trips/tests/test_http.py*:

```
# server/trips/tests/test_http.py

def test_user_can_retrieve_trip_by_id(self):
    trip = Trip.objects.create(pick_up_address='A', drop_off_address='B')
    response = self.client.get(trip.get_absolute_url(),
        HTTP_AUTHORIZATION=f'Bearer {self.access}'
    )
    self.assertEqual(status.HTTP_200_OK, response.status_code)
    self.assertEqual(str(trip.id), response.data.get('id'))
```

Here, we leveraged the use of the handy `get_absolute_url` function on our `Trip` model to identify the location of our `Trip` resource. We added asserts that get the serialized data of a single trip and a success status.

Of course, we create a failing test to begin:

```
(env)$ python manage.py test trips.tests
```

Error:

```
django.urls.exceptions.NoReverseMatch: Reverse for 'trip_detail' not found.
'trip_detail' is not a valid view function or pattern name.
```

Update the `Tripview` in *server/trips/views.py*, like so:

```
# server/trips/views.py

class TripView(viewsets.ReadOnlyModelViewSet):
    lookup_field = 'id' # new
    lookup_url_kwarg = 'trip_id' # new
    permission_classes = (permissions.IsAuthenticated,)
    queryset = Trip.objects.all()
    serializer_class = TripSerializer
```

Supporting our new functionality is as easy as adding two variables to our `TripView`:

1. The `lookup_field` variable tells the view to get the trip record by its `id` value.
2. The `lookup_url_kwarg` variable tells the view what named parameter to use to extract the `id` value from the URL.

Add the URL to *server/trips/urls.py*:

```
# server/trips/urls.py

from django.urls import path

from .views import TripView

app_name = 'taxi'

urlpatterns = [
    path('', TripView.as_view({'get': 'list'}), name='trip_list'),
    path('<uuid:trip_id>/', TripView.as_view({'get': 'retrieve'}), name='trip_detail'),  # new
]
```

We identified a `trip_id` in our URL configuration, which should be a UUID.

Ensure the tests pass:

```
(env)$ python manage.py test trips.tests
```

« Authentication   WebSockets - Part One »

✓ Mark as Completed