When you're working with JWT Token Authentication and you want to be a well behaved software developer and build out your Unit Tests (in this case we're using `phpunit`), it can be somewhat challenging to tackle out of the box. My solution here uses an inherited "service" that generates the token that can be used in any subsequent requests:

```
<?php namespace LitwickiBundleCoreBundleTesting;

use GuzzleHttpClient;

class LitwickiTest extends PHPUnit_Framework_TestCase
{
    public function authorize($username = 'litwicki', $password =
'Password1!')
    {
        $client = new Client('http://api.litwicki.dev/api/v1', array(
            'request.options' => array(
                'exceptions' => FALSE,
            )
        ));

        $data = array(
            'username' => $username,
            'password' => $password
        );

        $request =
$client->post('http://api.litwicki.dev/api/v1/auth', null, $data);
        $response = $request->send();

        $json = $response->getBody(TRUE);
```

```
        $body = json_decode($json, TRUE);

        if(isset($body['token'])) {
            return $body['token'];
        }

        return FALSE;
    }
}
```

In the interest of adhering to the DRY principle, this class exists specifically for the purpose of being called in every unit test as a means of authorizing each request (see below).

```
<?php namespace TestsApiController;

use GuzzleHttpClient;
use LitwickiBundleCoreBundleTestingLitwickiTest;
use RhumsaaUuidUuid;

class UserTest extends LitwickiTest
{

    public function testUserRoute()
    {
        $client = new Client('http://api.litwicki.dev/api/v1', array(
            'request.options' => array(
                'exceptions' => false,
            )
        ));

        $token = $this->authorize();
```

```
        $url = 'http://api.litwicki.dev/api/v1/users';

        $request = $client->get($url);
        $request->addHeader('Authorization', sprintf('Bearer %s',
$token));
        $response = $request->send();

        $json = $response->getBody(true);
        $body = json_decode($json, true);

        $this->assertEquals(200, $response->getStatusCode());

    }

    public function testUserCreate()
    {

        $token = $this->authorize();

        $faker = FakerFactory::create('en_EN');

        $data = array(
            'first_name' => $faker->firstName,
            'last_name' => $faker->lastName,
            'email' => $faker->email,
            'username' => md5(time()),
            'signature' => $faker->text(100),
            'password' => 'Password1!'
        );

        $url = 'http://api.litwicki.dev/api/v1/signup';
```

```
        $client = new Client($url, array(
            'request.options' => array(
                'exceptions' => false,
            )
        ));

        $request = $client->post($url, null, json_encode($data));
        $request->addHeader('Authorization', sprintf('Bearer %s',
$token));
        $response = $request->send();

        $json = $response->getBody(true);
        $body = json_decode($json, true);

        $this->assertEquals(200, $response->getStatusCode());

    }

}
```

You may want to expand upon this to do some additional authorization or validation on the `$token` response, but in my use case I wanted the request to fail spectacularly if the `$token` were FALSE, so I've left it as such.