Technologies:

Although REST messaging can be realized over any protocol, this Web API was exposed using HTTP (RFC 2616). Standard HTTP response codes; along with response body content were used to pass appointment data to the client or transmit additional error information.

The base platform environment for this exercise was built on XAMPP v3.2.1.

The RESTful API was built on top of Laravel 5 : <http://laravel.com/> ; Laravel 5 was installed using Composer. Composer is used to manage the PHP dependencies <https://getcomposer.org/> . Laravel 5 was chosen because it is written with PHP, has a nice MVC and a routes abstractions Both help make the creation of web api’s elegant.

When working with Laravel 5 the Migration, Model, Controller, and new Request object stubs were created using php artisan. php artisan also has an interface to print the routes …

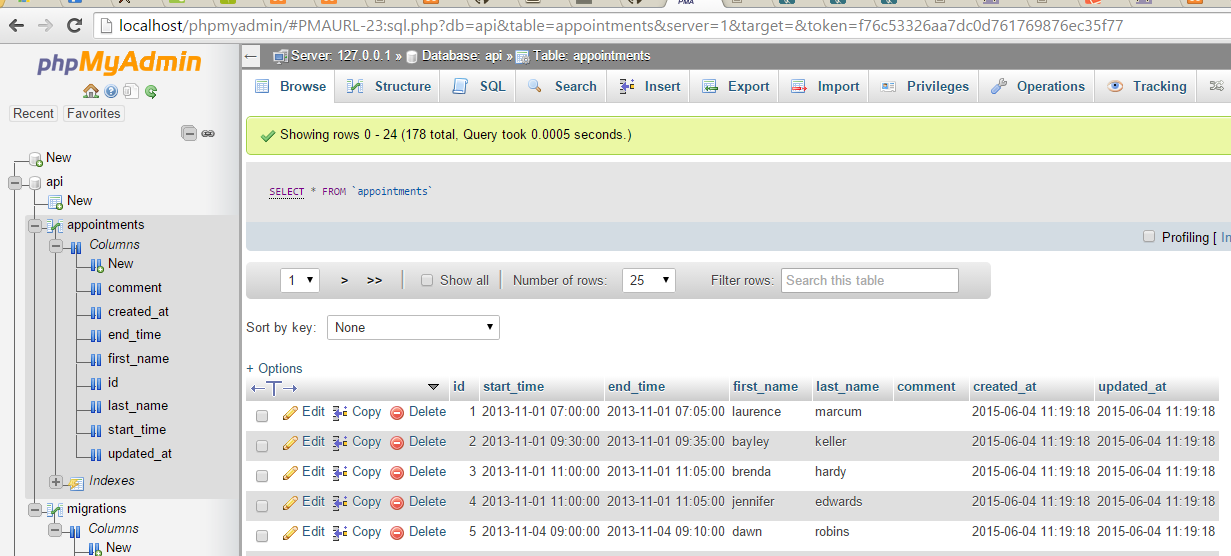


Sublime Text 3 was used as the editor. It integrates nicely with PHP. It also has a file browser that is very helpful working with the Laravel 5 framework.

GIT and GitHub were both used for versioning the software as I was working on it.

phpMyAdmin was used to interact with a database, named api, that contains one table for the appointments.

The API’s were tested with the Postman Chrome extension. There is something to watch out for. When using Postman, the “send” format must be x-www-form-urlencoded. If this is not set both PUT and PATCH will not work.



MySQL was chosen for this exercise because of the tight integration with php.

API Versioning:

Ideally, API’s should be versioned. I was using a local XAMPP installation and did not create the web api with a version number as part of the URI. A version scheme like [www.mycompany.com/api/v1/appointments](http://www.mycompany.com/api/v1/appointments) would have been ideal.

Notes about Date and Time:

Date and Time is not a type covered as part of the JSON RFC. <http://www.rfc-editor.org/rfc/rfc7159.txt> . Within JSON Date/Times can be represented as strings (there may be other custom ways). Representing date times as strings is problematic as they are normally the product of some serialization. This means that date / time representation can be implantation dependent. For example (in C#):

DateTime dateValue3 = new DateTime(2013, 11, 6, 21, 15, 07);

Console.WriteLine("{0:M/d/yy H:mm:ss}", dateValue3);

Produces 11/6/13 21:15:07 (which matches the format that is in the challenge)

Furthermore, there could be culture issues with representing date/times as strings. For example, applying a “Short Date (d) Format specifier in C# “ for 2009-06-15T13:45:30 will produce 6/15/2009 (en-US); while the same time value will produce 15/06/2009 (fr-FR).

There are other formatting issues as well for example “7 AM” could be represented as 07:00 or 07:00:00.

It is recommended that date/time values that will be consumed from different clients need to be stored in Unix Time format or ISO8601 that has explicit timezone information.

**For this challenge it is assumed that the date/time is not shared and the user has the same culture and same time zone etc.**

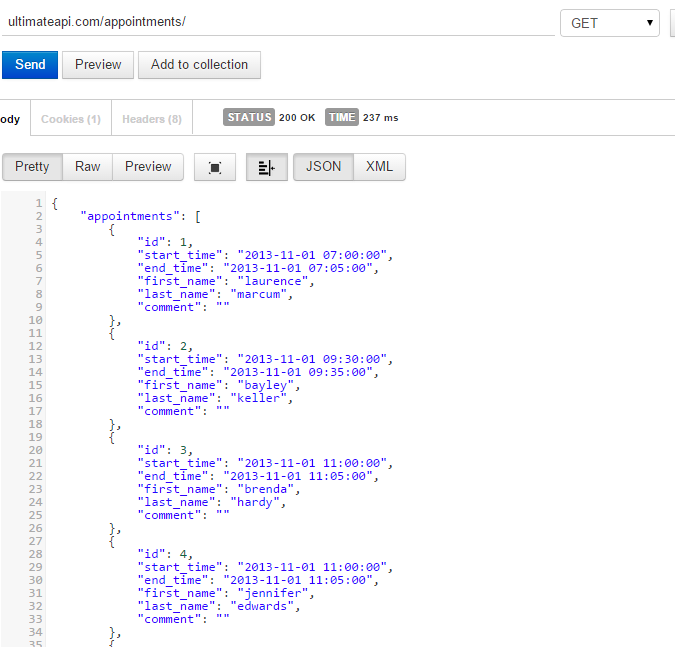
**When the data was imported into the database I manually modified the supplied dates to be in the**

**yyy-mm-dd hh:mm:ss format that is easily used with MySql.**

Sample APIs and Results:

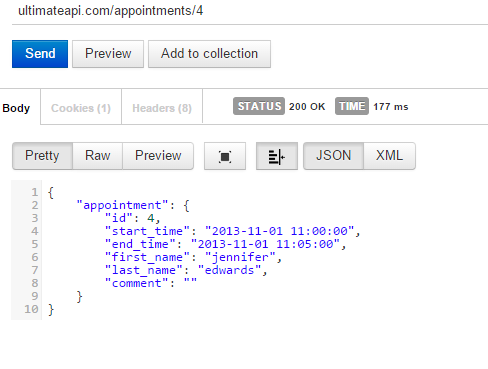
Get all appointments

GET ultimateapi.com/appointments/



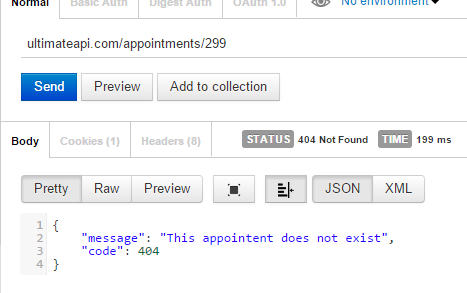
Get a single appointment

GET ultimateapi.com/appointments/4



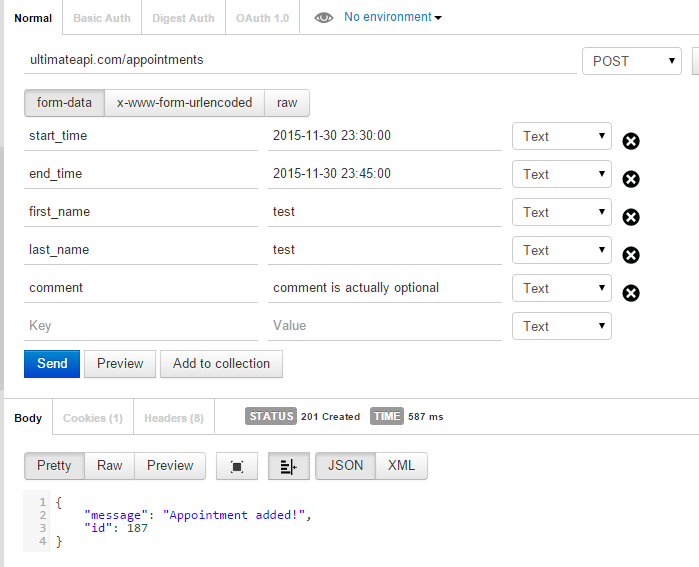
GET an appointment that does not exist

GET ultimateapi.com/appointments/299

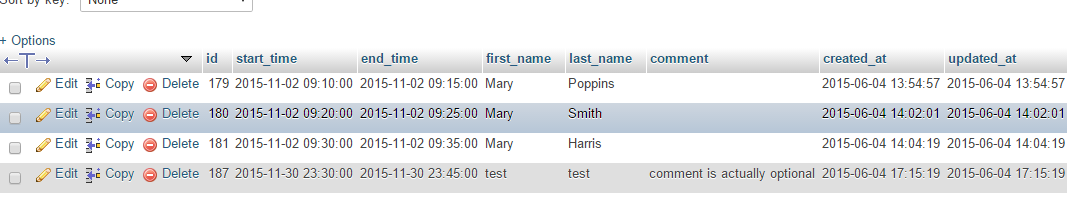


Create a new Appointment

POST ultimateapi.com/appointments

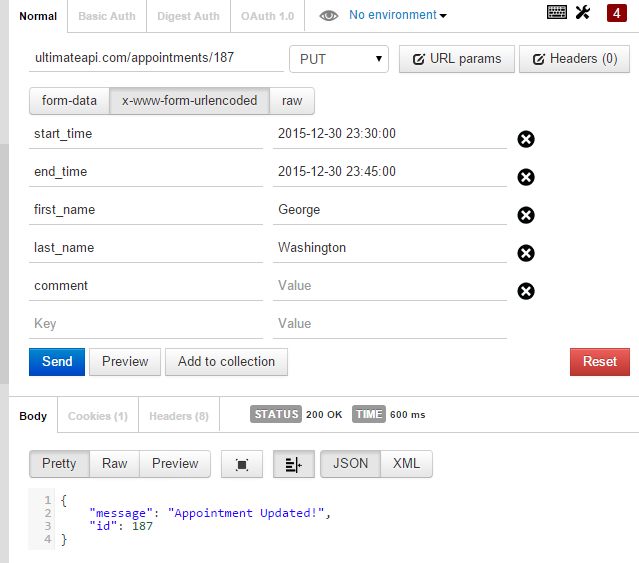


phpMyAdmin shows



Update to a record

PUT/PATCH ultimateapi.com/appointments/187

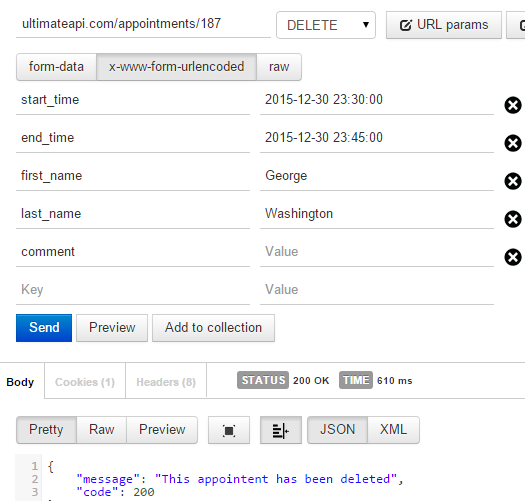


phpMyAdmin shows that the Record for id 187 was updated.



Delete a record.

DELETE ultimateapi.com/appointments/187



phpMyAdmin shows that resource 187 is gone:

