Pixel Tech Test Log

This log documents my actions and what I have learned while undertaking this tech test. Note that at the time of beginning this project I had no working knowledge of Laravel.

04/08/2021

Found out about this task fairly late in the day, struggled for a while to set up Laravel and barked up quite a few wrong trees (using docker), but eventually got the project running using Laragon.

I'm using VS code with some appropriate plugins to as my editor.

05/08/2021

Following an online tutorial to get an understanding of Laravel.
Got an understanding of routes for pages (not strictly necessary for the task but I'd have to learn it eventually anyway)
spent a *long* time trying to get the database to work, I now finally have a MySQL server running and migrated to it. Tomorrow is when I'll actually begin implementing the API.

06/08/2021

Seeded the database, CommentSeeder was not being called from DatabaseSeeder.php, which I presume was either a mistake or part of the test, I included it.

Learned about how Laravel automatically creates connections to the database using the models, such as the blogs table is accessible through the Blog model. To run queries on the blogs table I use the Blog model, for comments table it's the Comment model etc..

I installed another VS code plugin called REST client, which allows me to test the API being implemented, the file is uses is under the root directory and is called "test.http".

07/08/2021

Having issues with the Model. For some reason I can access the models and run queries just fine from some files (test.blade.php for example), but for some reason the model classes cannot be found from API.php. I've tried dumping the autoloader, as well as a couple other suggested solutions, but none worked.
Moments after writing that segment I solved the issue, my inclusion "use app/Models/Blog" actually needed to be "use App/Models/Blog". I tried it out after seeing the A had been capitalised on an example. The real file structure does not have a capitalised A so I have no idea why this now works, but I'm glad that it does.

API assumptions

GET blogs – for this I assume it gets all the blogs and all their contents, so a simple Blog::All() is sufficient.

GET single blog with comments: The API will have the ID of the blog passed as a query string, if no ID was passed it will return an empty JSON object.

POST comment: All the post details need to be validated, meaning that title, name, email, comment and blog_id need to be present, sent as JSON. The blog_id needs to be further validated to check that a blog with that ID exists.

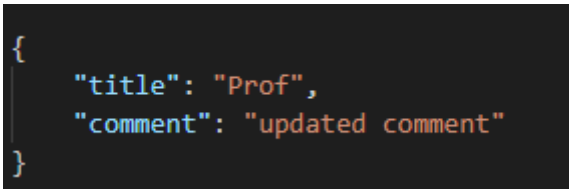
08/08/2021

PUT comment – any detail aside from blog_id should be updatable.

After doing the PUT comment I created an alternate implementation of GET single blog with comments, this is because I learned how to pass in information through the URL instead of with query strings (blog/1 rather than blog? id=1)

DELETE comment – should just take an ID and delete the corresponding entry (if it exists)

I went back and made an alternative PUT comment route as well. The alternate version does not require all the comment fields to be sent (title, name, email and comment), instead you only send the data that needs to be changed (see below).



That JSON data will mean the title and comment will be updated, but the name and email will be left alone.
This implementation returns an error when no fields are provided.

I learned a lot through doing this project, both about how to set-up these technologies (Laravel and MySQL servers), how to utilise the framework and the workflow involved in creating and testing an API. I still have a lot of learn about how Laravel works and how I can best utilise it, but for a demonstration I hope this proves that I am able to learn and persevere past challenges on my own, and that will be even better when I'm part of a team and have access to guidance from people with much more experience than me.

10/08/2021

Received feedback, I was informed that I did the API logic in the Routes file, whereas they should have been in a controller. This is something I missed because of my unfamiliarity with Laravel, but after a short bit of research refactoring the code into the proper form was quite easy.

The logic has been abstracted into CommentApiController and BlogApiController.
There were some decisions to be made, one was should the Blog controller utilise the Comment model, or should it request the data from the Comment controller? I did not find

much to support either choice, so I made my own, which was that it can use both controllers. The reason I made that decision is because if one controller relied on the other in order to function, it creates dependencies, whereas they should be able to work completely independently of each other.