# Vet IT – Test

## Objective

Creating an API for use in an Angular 11 application using the provided database as a data source.

* The API should contain a call to select all active and non-deleted products. These should be ordered by newest created first.
* The API should contain a call to select all dangerous drugs whilst excluding those that are deleted or inactive from the list.
* The API Should allow a call to update a product description and return a success message or error message depending on whether the update has worked.
* Bonus – Get Swagger up and running.

**Please note**: This document have been written informally to reflect my thought process as if I was writing a diary. This by no means is a true reflection of how I would document a professional piece of work.

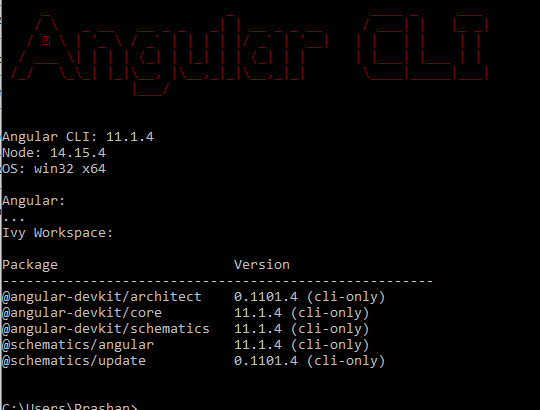
## How I went about this

### 1.Looked at the database

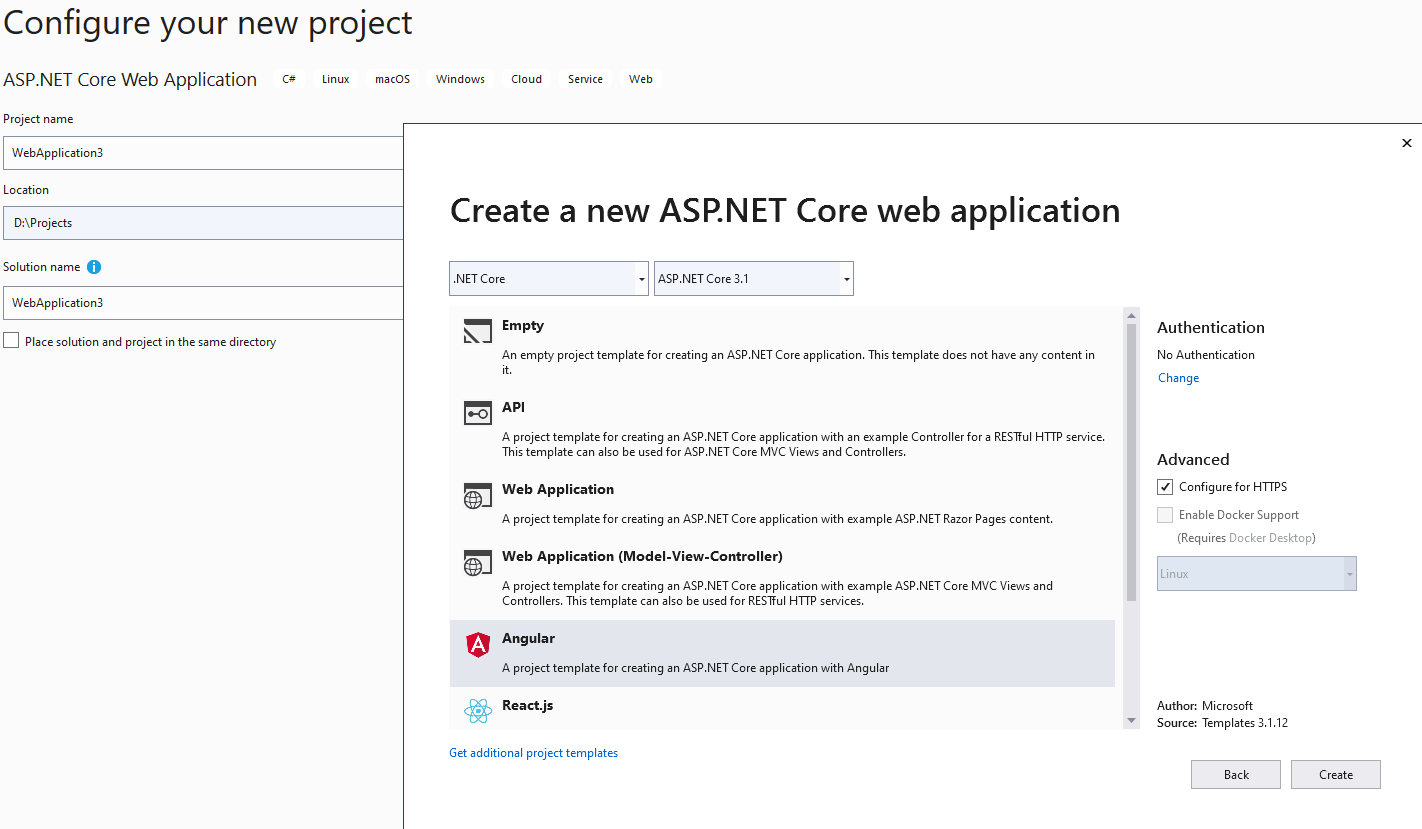
This was the first thing I did to understand what fields are there and how many records.

Then using SQL to see if I could satisfy the two get calls in the objectives. Looking at the fields in question I could create a simple statement that would filter out what I don’t need. E.g. select \* from [VetITTest].[dbo].[Product] where DeleteDate is null or DeleteDate >= GETDATE()

### 2. Installed the latest angular globally.



### 3. Created a new project.



Selecting angular added the client app so I didn’t have to do this manually.

##### Issues.

Even though I installed the latest Angular, when checking package.json, the dependancies were on 8.2.12. If I tried to update this to the latest, npm would raise errors with other packages and the program would not build correctly, so I left this as it is.

### 4. Started where I felt comfortable.

With the demo project VS provides, it was a good place to get swagger up and running first.

1. Do this, I first installed the nuget package ‘swashbuckler.AspNetCore’.
2. Then added services.AddSwaggerGen(); to Startup. ConfigureServices()
3. And then

app.UseSwagger( c => { c.SerializeAsV2 = true; }) ;

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json","VetIt"));

to Configure.

Now I need to start thinking about connecting to the data source and writing up the api. The approach I went for was a database first entity framework setup. I found by scaffolding, I would get the model and dbcontext I need quickly. At this point I amending ConfigureServices to include the newly created dbcontext and added the connection string to appsettings

Now I need a controller. I wanted to a keep generic get all products method for refreshing data but I really needed three new methods:

1. GetActiveAndNonDeletedProducts
2. GetActiveDangerousDrugs
3. PutProduct – Updates the description

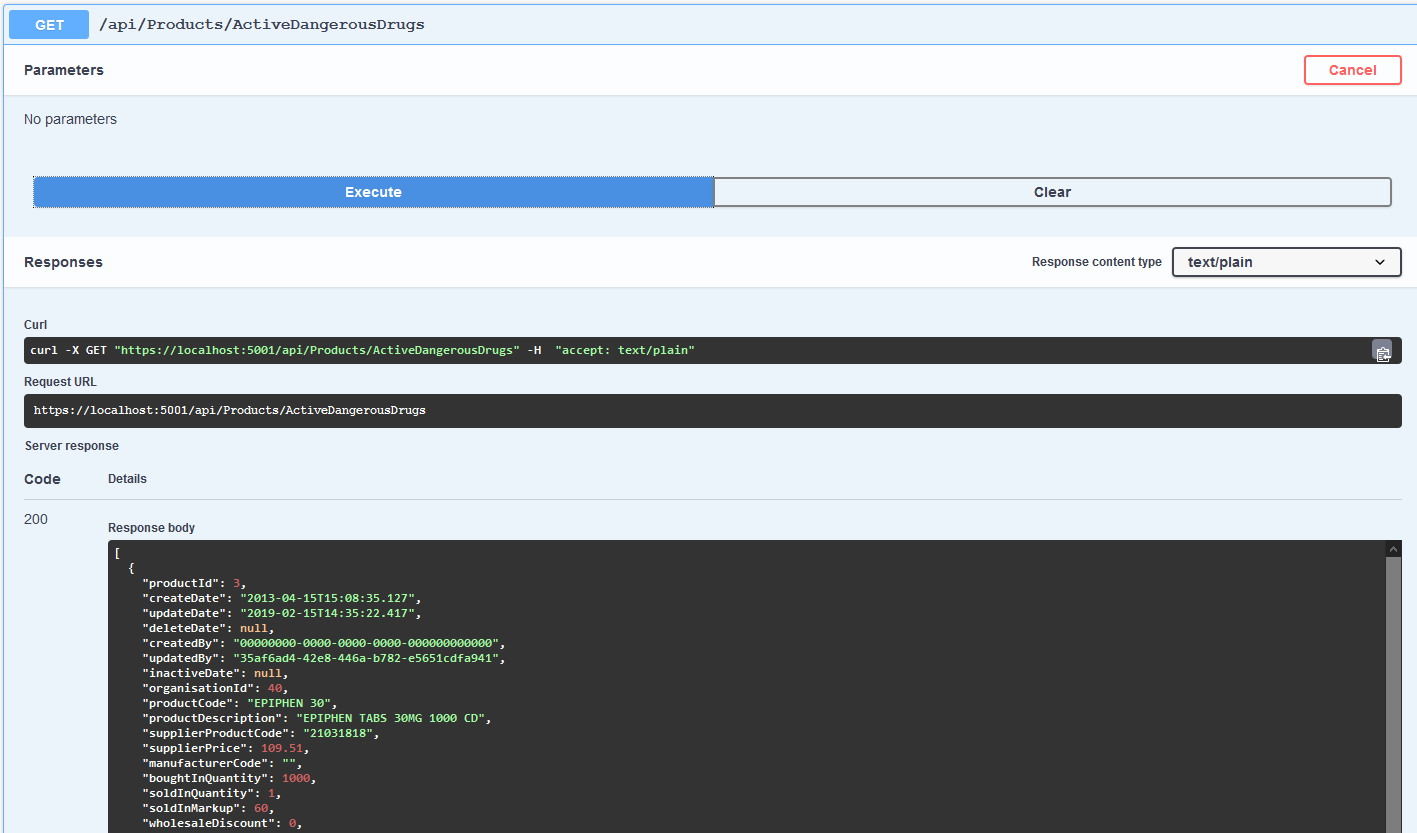
So, I would go on to create their signature with no logic at the moment and then create a new unit test project.

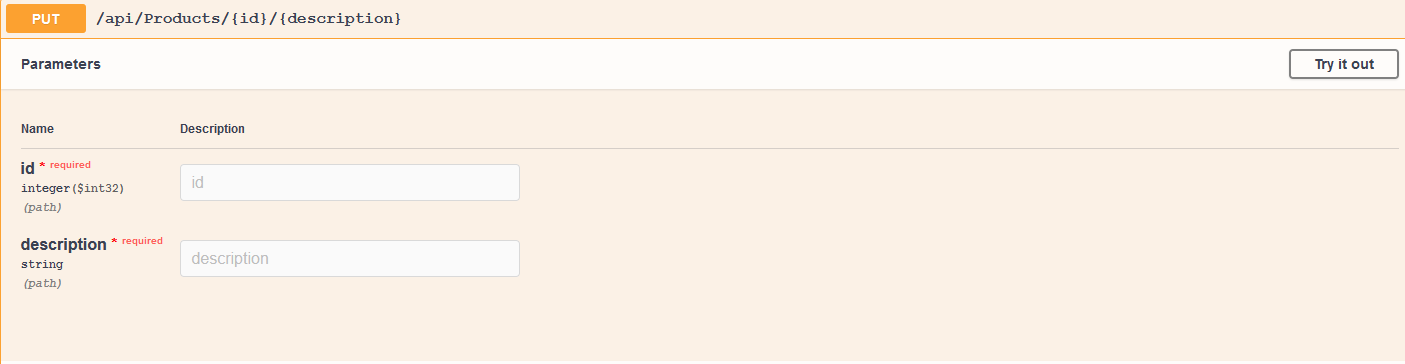
So, the unit test project I really just want to cover the three methods and cover as much as possible in terms of what I should be expecting based on certain data or input. I managed 25 tests between the methods and decided that was enough coverage to get me going.

Method logic – Wanted it keep it simple. Used linq on the dbcontext to get the same results as a SQL statement would do. If I didn’t get anything back from the linq query, then I would return a 204 No content. Reason why, I found out later on when creating the UI that a 204 came back from the server whereas a NotFound 404 wouldn’t and I wanted something to come back so I could give the user some context. With the update method I just need to two pieces of info; the primary key and description. Again, kept it simple. Find the product and change the description, save the context and raise errors where needed.

Testing the methods – Once the logic was done and test are passing. I load up swagger docs to test the api

e.g.





##### Issues

Swagger would crash the browser when try to get all the data i.e. 10000 rows or even active data (around 9555 rows). So, I would end up using Postman to test these endpoints.

### 5.UI

This was my first time creating a client side project using angular. So before starting this, I did watch quite a few YouTube videos and read some guides to help me understand the ins and outs of it all.

What I wanted to achieve before starting was a page showing all products and then links to other pages to call its corresponding endpoint. That idea changed as I watched more videos and actually starting experimenting with what I could do and this was all a part of the learning curve for me.

Eventually what I decided for was to have one main page to show all products with buttons that would change what data was shown. And a second page for editing.

How I went about this – first I need to create the pages using the command ng – g component product and product/edit. I would also need a model for product as it is on the server side. Ng g class product –type model and finally I would need a service class which be talk to the server ng – s product product

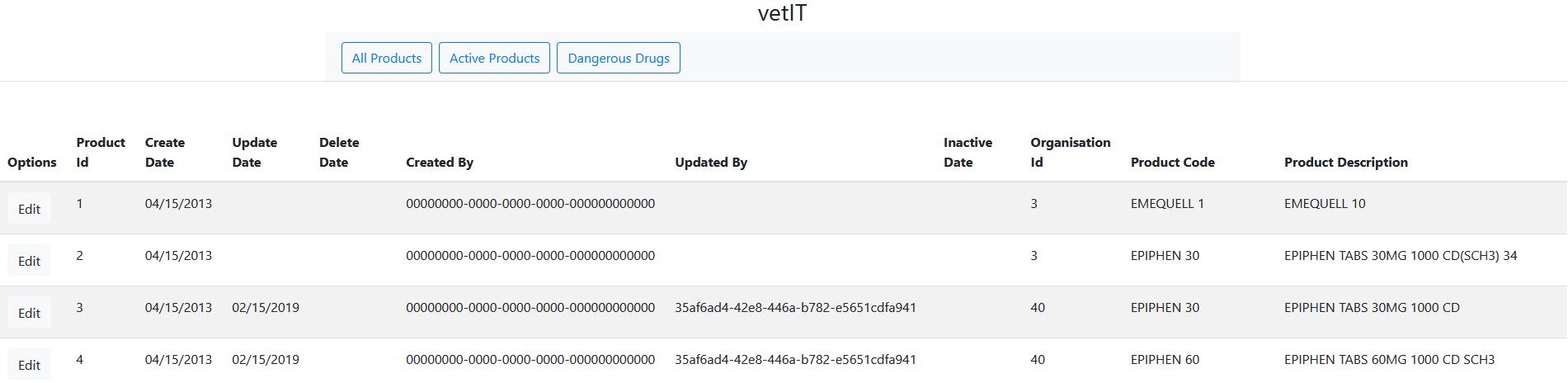
Main page – Kept it simple showing the data in the table. I made use of ngIf to display a loading message whilst the data is not ready and then ngFor for looping through the list of products. Whilst watching a video I got an idea to include bootstrap and to use a modal class to add another button that would hook up to the second page.

Challenges

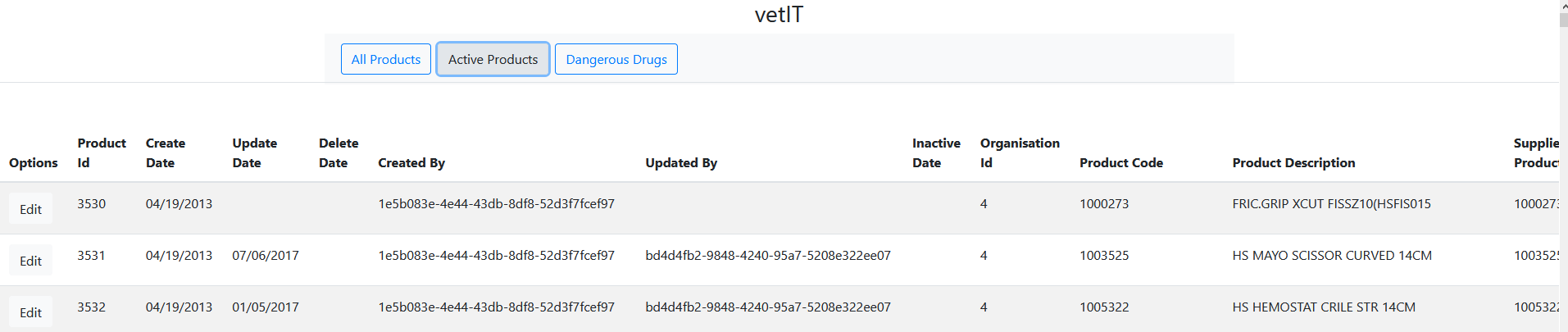
The biggest challenged I faced was how do I change the data on the main page to match the data received by one of the GET calls. This is where I learnt that I could use the service component to share this data across by using BehaviorSubject on the product list and having a ‘changeProductList’ method that allowed components to change that list. On the main component I just had to make sure that I am subscribed to the current list.

### 6. End Result

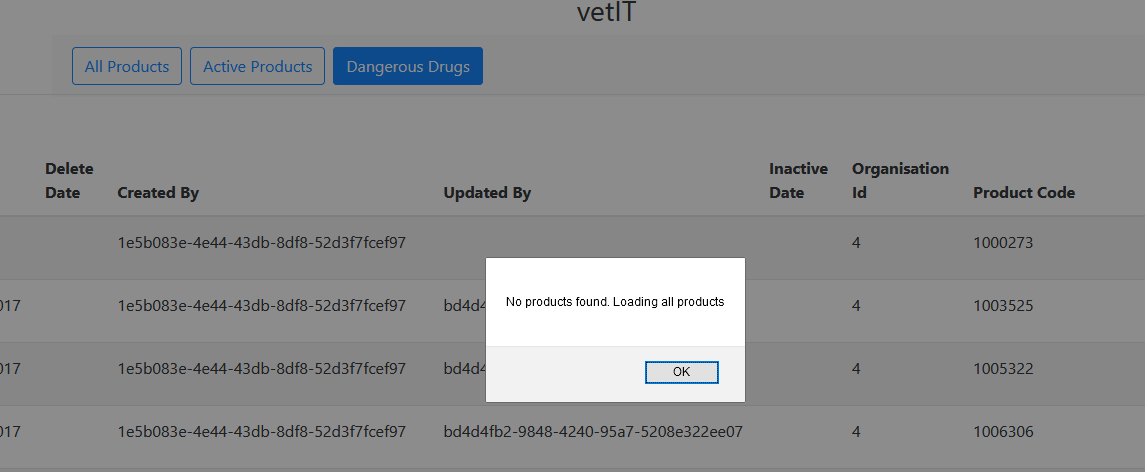
Main Page



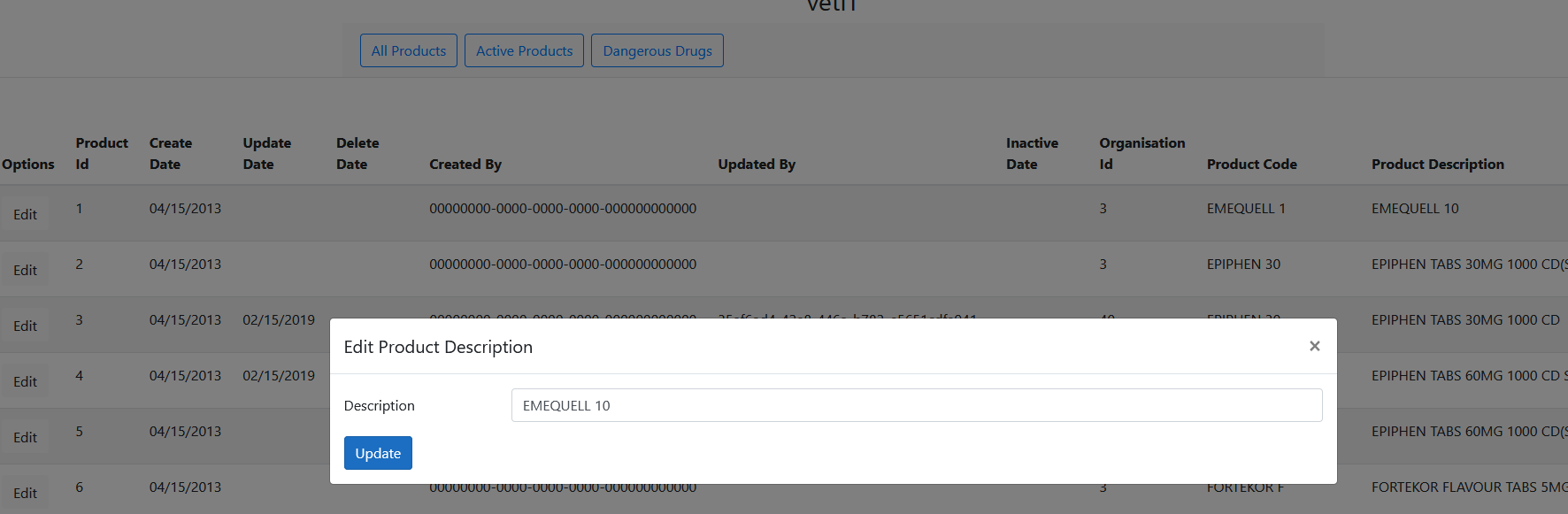
Active products



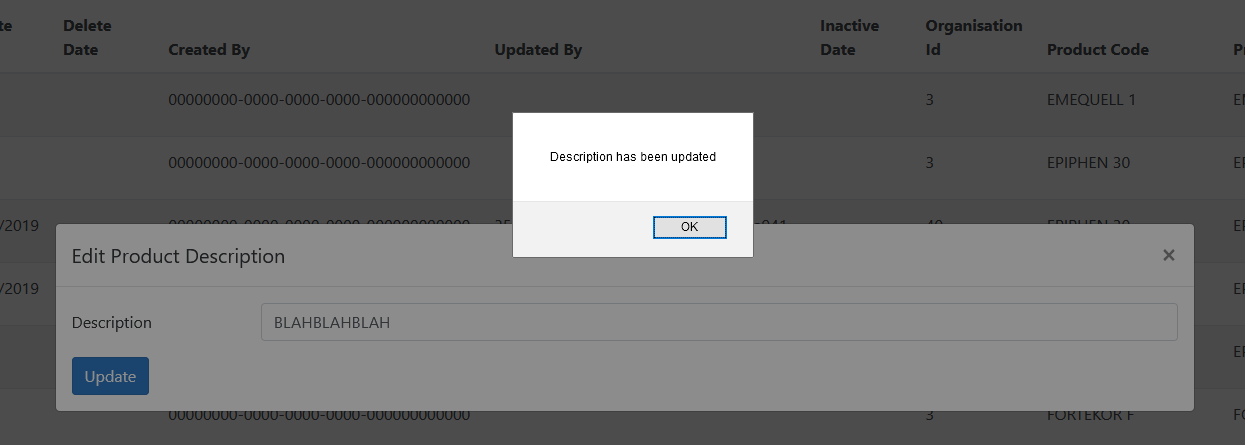
Dangerous Drugs – As there are no records and alert box is show



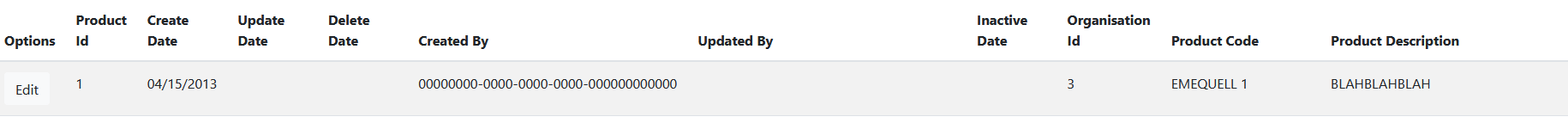
Edit Description



Successful update



Once the user closed off the edit box the update record is shown



#### Room for improvement

The project was a fantastic way to show something of things I could do and as well as try new stuff like angular. I do feel I there is a lot of places where I could explorer different possibilities.

1. Store procedures – I would have liked to try out if store procedure could have been a faster approach of receiving the data. The database has 10000 rows and on a get all call dbcontext took any around 400ms to 650ms.
2. Rendering less or splitting the data up on client side – It takes too long for my liking at the moment to render a page and I think its due to the amount of data I am outputting in the table. I would have liked to split it up or just not show as much.
3. Render the page for a single record after an update – right now it updates the entire table which is not ideal since we only edit for row at a time.
4. The PUT endpoint - Could use some sort of validation of the description to ensure its not malicious.