**Taina Tech Test**

1. I decided to structure the application in 4 projects. (The UI, Business Object, the Web API and Data Access).
   1. The UI was implemented using AngularJS and Bootstrap which are light weight client-side responsive technology (Keeping performance and good user experience in mind).
   2. The business object holds the foundation objects that can be used across the entire application. This give the flexibility when coming to scaling or changing the objects.
   3. I have used RESTful web API (Keeping scalability and performance in mind) and also making keeping in line with industry standard.
   4. Data access uses interface method calls so it is flexible enough for extension (Scalability and Robustness) should the data source change in future.
2. I have pushed as much of the validation unto the user input screens and I have also created a Log of any missed errors that could occur on the server side. The logs are created on a daily bases (distinct log for each day) and the “Call Stack” of any potential error is logged so as to make debugging easier. Also used a “try catch log” at point where errors are anticipated. I have tried to ensure the usage of functionalities that would invoke the cleaning up of memory (e.g. “using” at point of writing to log).
3. I have some meaningful messages reported to the user and in some cases encourage the user to try again in reasonable time intervals. Also, this type of issues is properly logged so preventive measures can be thought of to ensure it does not repeat itself in future. Also, the thought of having the service on a distributed network using all the benefit of such topology to guarantee service at any time.
4. The application is flexible enough to be scaled and the tech chosen would allow for this. The structure is such that individual component of the application can be replaced with other application doing similar functionalities. (i.e. UI project can be replaced with another built with ReactJS if the Web API are consumed as expected OR some other database access can be introduced if it implements the Data Access Interface methods). Also, individual components can be integrated to other sub-application build with some other technology.
5. Implementation of client side caching and a queue system while processing the task asynchronously to help with traffic load on the site. The data access also implemented natively so as to remove any dependence on libraries or the extra budding on entity framework which would affect performance. I have used an identity key as the primary day for my table to improve performance but I feeling the table could be further normalize to reap the benefit of good database structure which would further improve performance. Lastly, the compression of dynamic content can be enable to further reduce the size of the result of the APIs which would further improve performance.
6. The user interface has some filtering and also plans for paging to cope with the huge number of data the application would handle. The forms are only allowed processing when/if all validations have been satisfied. Clear information on the mandatory fields which are highlighted.

Plans for Scaling:

I would recommend using a cloud service from the numerous reliable companies offering these services. This would give the benefit of being able to concentrate on how the Taina’s business offerings can be improved rather than worry about the means because the companies offering these services ensure they go the extra mile to guarantee great service. There is also the advantage of being able to quickly set up needed resource, scale up or down on already subscribed resource to meet immediate needs. Having said that, my scaling ideas which I eluded to earlier are as follows.

1. There could be multiple database instance serving the application (mirroring) such that if an instance is down there would still be other that would continue to serve the application.
2. Introduction of a robust queueing mechanism so that write activities are managed well for the asynchronous messages(threads)
3. A robust caching system both on the client machine and server would be introduced to help with reducing the read load on the application database. This is thought of since the application is not one which is changing frequently. Even in that case, a reasonable refresh time with the caching would still be beneficial.
4. Depending on the amount of traffic, the resource allocation could be scaled up and down to ensue flexibility while also keeping cost in check. This are available from most cloud service providers.

**I am aware there are still a lot of areas which I would need to improve upon but rest assured I am an intelligent tenacious young man who is willing to sacrifice and strive hard to reach my goal which is to be a reputable software architect and IT consultant who is an asset to my prospective organization.**