# Messaging App Trade offs

## Database

With the time given I went with the standard and basic implementation of SQL server. Now for this kind of application I would have gone with something like a Document DB like MongoDB for the reason of I’m maxing the length of the message to a char of 250. If I had used something like mongo, I would have just been able to store the entire JSON blob into mongo which will increase the speed of retrievals.

I created an interface that everything uses from the service layer. So, at a later stage to implement a MongoDB solution would be a lot easier as all that would be required is create another Repository implementation and then swap the new MongoDB one out for the SQL implementation.

## Queuing

As we are saving and updating values in the DB from an HTTP request sometimes those requests can fail. I would have liked to implement some sort of queuing aspect for the Creating, Updating and Deleting of the messages.

Reason for this is that with the queue aspect if anything fails the queue will automatically retry 3 times and if fails will go to the DLQ which then someone can just retry the message until it goes through. This will also help in if the connection to the DB goes down and say 100 requests all failed while the DB went down. Once the DB is back up then we would be able to retry all those messages.

With implementing the queue in this way there will be a slight disconnect from when the user saves/deletes/updates a message from when they can see it being actioned on the portal. If the UI/UX is done properly though then this issue will be slightly less impacted.

## Open API/Swagger Spec

With any API it is always wise to have either an Open API yaml/redoc or Swagger Spec for other developers to use in case they need to integrate into the API you have just built. If I had more time, I would have added this. As a secondary option I attached the Postman Collection I used to test.