# Straddle Technical Challenge

## Solution Summary

The solution consists of an Api and a worker service. The Api exposes endpoints for creating, retrieving, and cancelling payments. The worker service simulates sending the payments to the bank and checking the status of the payment after 60 seconds. You can cancel a payment up to the point that it’s status changes to processing. There is a history table in the database the logs all the activity relating to the payment.

## Technical Summary

* The solution is built using clean architecture and SOLID principles. The solution is structured to reflect that.
* Security is implemented with Azure AD OAuth.
* Data is stored in an SQL database.
* Repository pattern is used for accessing all data in the db via Entity Framework.
  + I use a NuGet package DataRepositoryCore that provides a set of generic read/update repositories over entity framework. This is a package written by me and has been used in multiple production applications. Source is [here](https://github.com/Russ256/DataRepositoryCore)
* MediatR is used to decouple application commands execution from requests.
* Unit of work pattern implemented via MediatR pipeline to ensure database transaction integrity.
* Azure Service Bus is used for messaging.
  + Outbox pattern implemented to delay sending of messages from the commands to the unit of work.
* Command validation is implemented via the MediatR pipeline and uses FluentValidation and validator classes.
* There are some unit tests for the application commands using Moq and FluentAssertions.
* The worker service processes two messages, the first simulates sending the payment to the bank, the second to simulate checking the progress of the payment after 60 seconds and update the database.

## Setup

If you want to run locally you will need to set up the following:

* An azure app registration with a web platform and a secret added. These will need to be added to the appsettings.json. The web platform will need callback of <https://localhost:7033/signin-oidc> and for postman <https://oauth.pstmn.io/v1/callback>
* An Azure service bus on standard tier with the following topics (the current solution settings point to a service bus in my subscription which should work):
  + Topic: payment\_request with a subscription called process
  + Topic: payment\_processing with a subscription call progress\_check
* A SQL database. The database project is setup to publish and run on the default local instance by default
* There is a postman collection in the solution folder you can update with your app registration to get tokens.

## Scalability

As this is a REST based API it can be easily scaled up and run on multiple instances based on demand. Either using the standard app service scaling or by using containers and Kubernetes. There same can be applied to the worker service to increase the number of messages processors and so the throughput.

### Sensitive Data

To protect sensitive data stored in SQL server I would use the encrypted columns feature. This ensures only authorized users can access the data.

I would also ensure the services and the database are running on a separate private vnet using private link to ensure no data travels across public networks.

In addition, you could also turn on always encrypted on the database to ensure the data transmitted across the network is always protected, although not necessary if running on a private vnet.