

|  |  |
| --- | --- |
| **Details** | |
| **Full Names** | Hugin Antony Ponnusamy |
| **Email Address** | HuginAntony@gmail.com |
| **Public GIT Repo** | https://github.com/HuginAntony/OsirisTest |

**\*\* Please note that you are not being tested against the technologies being used in this solution, rather against the SOLID principles of software development along with understanding of software development practices.**

In the solution provided you are given three applications which simulate an event driven process. The projects needed to be executed in the below sequence in order for the solution to work as “expected”.

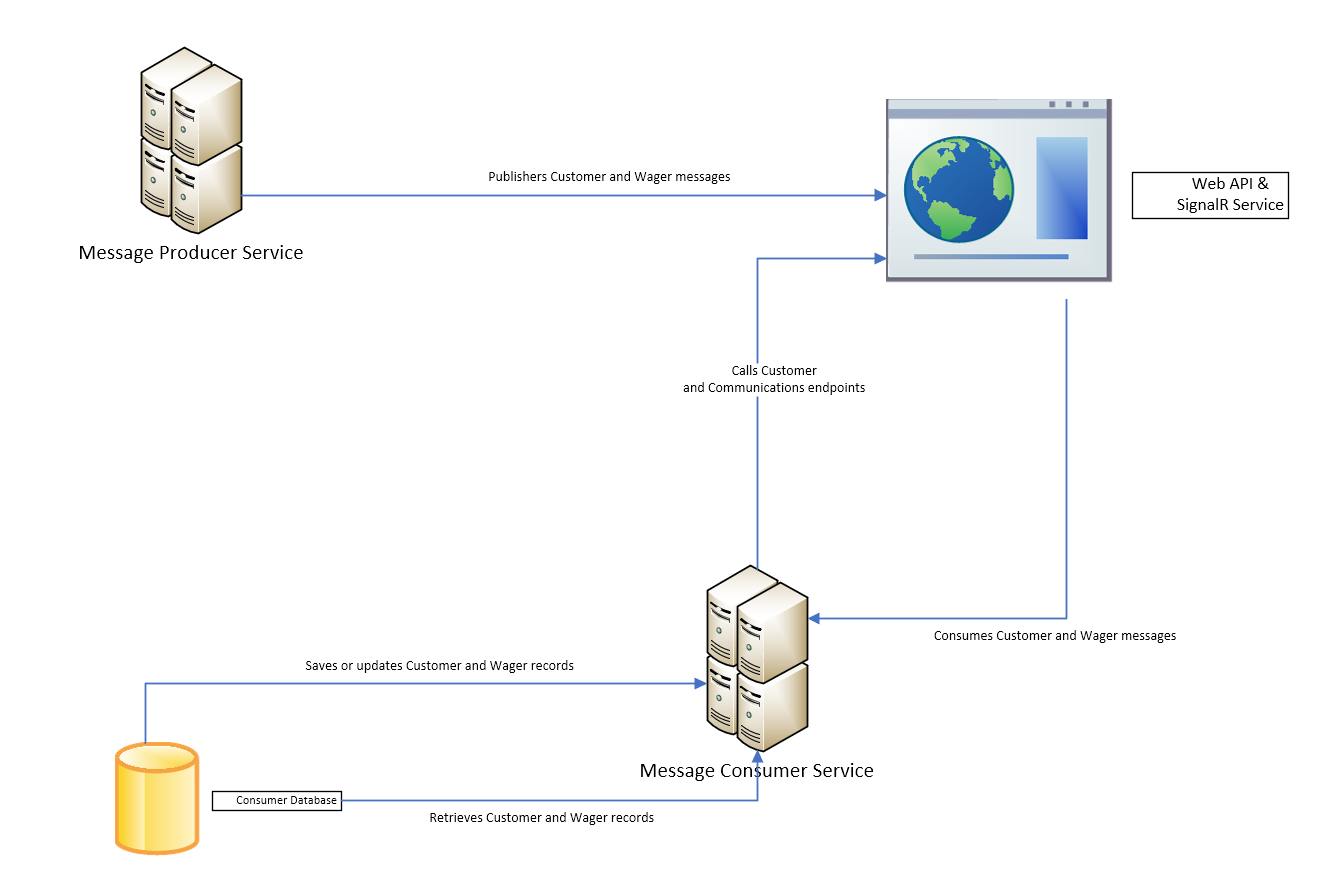
* **OsirisTest.WebApi**
  + Microsoft .Net Core Web API which exposes both the SignalR publish functionality along with standard web controller methods. Think of this as a 3rd Party API.
* **OsirisTest.Service.Publisher**
  + Microsoft windows service that generates customer and wager events using basic SignalR. Think of this as a 3rd Party event publisher.
* **OsirisTest.Service.Consumer**
  + Microsoft windows service responsible for processing the messages being received from the publisher.

This test is made up of a short theory section along with a practical section which requires the candidate to:

* Fix solution and make it build.
* Refactor where necessary to follow best coding practices.
* Add additional logic based on requirements that will be stated later.

Theory

1. Identify three different design patterns used in this solution.
   1. Template method. The parent class BaseConsumer defines abstract methods and properties that the child classes CustomerConsumer and WagerConsumer override with their own implementation.
   2. Command pattern. The API controllers CommunicationsController and CustomerController use the RequestHandler method. This method allows different commands in the form of a Func to be passed in.
   3. Singleton pattern. TopShelf is using singletons when running the application as a windows service. The publisher project also uses a singleton for the WagerProducer.
2. Identify three examples of SOLID principles being used in this solution.
   1. Dependency inversion principle. The application uses dependency injection in all constructors.
   2. Liskov substitution principle. Classes such as CustomerConsumer and WagerConsumer are child classes of BaseConsumer.
   3. Single responsibility principle. Classes such as CustomerConsumer and CustomerProducer have a single responsibility that only consumes customer messages or produces customer messages.
3. Create a high-level architectural diagram of the solution.



Practical

1. Identify and fix all errors such that the solution builds and runs with no issues.
2. Create database schema from provided schema file.

\*\* Please note schema is using T-SQL but you may use whichever SQL technology you are comfortable with.

* 1. \OsirisTest\OsirisTest.Utilities\DataAccess\
  2. Schema.sql

1. Implement custom or inject console logger for all three applications.
2. Find all TODOs in the solution and implement the logic as required.
3. Create a SQL view/dataset report showing the total number and sum of valid and invalid wagers. \*\*Note the above result should be grouped per customer, per valid status.

Example screen shot:

