I have worked on the mentioned project and have uploaded my code to the github at <https://github.com/tina-projects/deliveggie>. The application is working fine when run from my local machine, but after dockerizing, I am facing an issue where the console application is exiting because the queue connection is closed.

The github repository contains two branches, master and webapi. The master branch contains the application with rabbitmq for communication and the details are given below. The webapi branch contains the same application as a webapi. I started with webapi as I am not familiar with rabbitmq or mongodb and wanted to start with something I was familiar with.

**Application Components**

Angular application

* Folder Location – ui/DeliVeggie
* Build Command – docker build -t veggie-ui:0.0.1 .
* API Urls - src/app/veggie.service.ts
* Runs on localhost:4200

API

* Folder Location – DelVeggieAPI
* Build Command – docker build -t veggie-api:0.0.1
* Exposed APIs
  + GET /veggie – Gets a list of products
  + GET /veggie/{id} – Gets the specified product
* Rabbitmq URL - Controllers/VeggieController.cs

Console App

* Folder Location – DelVeggieConsoleApp
* Build Command – docker build -t veggie-console:0.0.1
* Rabbitmq URL - Program.cs
* Mongo URL - DAL/DbConnection.cs

Docker Compose

* Folder Location – compose
* Command – docker-compose up

Other Components (deployed in docker-compose)

* RabbitMQ (rabbitmq:3-management)
* MongoDB (mongo:latest)

**Application Flow**

1. Navigate to the UI application at localhost:4200
2. UI calls the /veggie API to get a list of products
3. In the API, a message is created with a correlationId and published to rabbitmq at application queue – veggie.request
4. Another queue <correlationId> is created and the API subscribes to that.
5. The console app, which is listening to the application queue, veggie.request, gets the correlationId
6. Console app queries the mongodb to get list of products through VeggieService class.
7. A singleton class, ProductService is used to get the Price Reduction data (This calls the DB only once for an application and the result is stored in memory).
8. Price reduction is applied to the list of products.
9. The final list is serialized and published to the <correlationId> queue.
10. The API gets the serialized data and deserialized into List of Veggie objects.
11. This list is sent to the UI for displaying.
12. On clicking a particular product, /veggie/{id} API is called
13. A similar process is done, with correlationId and productid sent to the console app.

Note:

As the APIs have to wait for response from the queue asynchronously, both the APIs are defined as async.

To be completed:

1. Unit tests for microservices
2. Kubernetes
3. Externalize connection strings