This project is simulation microservice which is used from a web app . Client writes a task to the queue and a task is consumed and simulated by the worker. The tasks processed by the worker are kept in a table in SQLite by updating their durations.

While designing this project, I thought that I needed an entity for each execution of the simulate method and for this reason I created a **MyTask** class. In this class I wanted to create *Id*, *TaskStatus* and *TaskUpdatedTime* members. Id is a Guid, TaskStatus is an enum and TaskUpdatedTime is a Datetime variable.

My goal here is to run the Simulate method by the Worker, when the client sends a request, I created a taskId to the Queue and a record with the information of this taskId to the database. Then, when a data is placed in the Queue subscribed by the Worker, it is automatically triggered, allowing the **Simulate** method to be simulated by the **Worker** in a non-blocking and asynchronous manner.

There are 3 different endpoints in my **SimulateController** class. These are **start**, **checkProgress** and **results** endpoints. **start** endpoint allows a task to be simulated. When the **start** method is called by the client, a taskId is written to the queue and at the same time a record is created in the database with the status “**InQueue**”. After the taskId is written to the queue, the subscribed Worker takes this taskId and runs the Simulate method as a different thread. Before running the Simulate method, it updates the database record of this taskId as “**InProgress**” and “**TaskUpdatedTime**” according to the current time. And then the Simulate method runs. After a certain period, the Simulate method updates the status of the related taskId in the database as **“Done”** and the “**TaskUpdatedTime**” according to the current time. We can see the time the task has finished. While these processes are in progress, the client can call the **checkProgress** and **results** endpoints; with the checkProgress endpoint we can query the information of all tasks and with the **results** endpoint we can query only the information of the tasks whose status is “**Done**”. In parallel, the worker will continue to simulate tasks and update the status of these tasks in the database.

I used the **publisher/subscriber** method to manage the queue and I used **RabbitMQ** in my design. I used **SQLite** as a database to make it easy to keep the information of the tasks.  
  
If I had more time, I would like to research in more detail and handle edge cases with best practices. In this case, my project would be running very effectively and without any crashes.

Since I designed my project this way, I used RabbitMQ.Client and Data.SQlite libraries. I used EntityFramework libraries for my database operations. I did not want to write raw queries for DB operations and also used xUnit and Moq libraries for testing.

Design:  
  
