Senior QA Test Exercise

## Application Notes:

* **Sender**:
  + Who pickups up messages from the producer and simulates sending messages by waiting a random period time distributed around a configurable mean.
  + The sender also has a configurable failure rate.
* **Producer**:
  + Generates a configurable number of messages (default 1000) to random phone numbers.
  + Each message contains up to 100 random characters.
* **Monitor**:
  + Displays the following:
    - Number of messages sent so far
    - Number of message failed so far
    - Average time per message so far
  + Updates the report event N seconds (configurable)

## First Task:

* Install the application in your development environment and treat it as a black box. You can modify the config file to test different configurations and parameters.
  + Running the application in a docker container to abstract it from my regular PC and made to be able to run almost anywhere.
  + Pre-requisites:
    - [Docker Desktop/ Docker](https://docs.docker.com/get-docker/) installed
    - [Rust](https://rustup.rs/.) installed
    - [Python](https://www.python.org/downloads/) installed
  + Building the application with Dockerfile and docker-compose:
    - docker compose build --force-rm --pull
    - docker compose up -d
    - docker compose logs -f

## Second Task:

* Write a test plan for the application (Steps have been recorded in the automated tests)
  + **Monitor**:
    - Reporting interval
      * Verify the reporting interval is reporting as expected
      * With the following parameters: -1, 1, 2 (default), 30. Anything higher will be just waiting for a longer period of unnecessary time unless a certain higher time was required.
    - Successful Messages:
      * Verify the number of successful messages sent so far and that they end up sending the configured num\_messages
      * With the following parameters: -1, 0, 1000 (default), 10000.
    - Failed Messages:
      * Verify the number of failed messages sent so far based on the failure\_rate
      * With the following parameters: -1, 0, 0.5, 1.0
    - Mean\_time:
      * Verify the expected rate based on mean\_time
      * With the following parameters: -1, 0, 0.5, 1.0
  + **Sender**:
    - This will have to be validated against the monitor reporting as the sender does not have a reported exposed port or UI to view the information sent from it from a black box perspective.
    - Number of messages sent to customer (default: 1000)
  + **Producer**
    - There is nothing to be able to see here from a black box perspective so there is not much to test. Everything from the producer depends on what the sender is reporting to the monitor.
* Are there any missing or ambiguous requirements that need to be clarified?
  + The `failure\_rate` and `mean\_time` configuration are ambiguous where:
    - `mean\_time`: A sender picks up messages from the producer and simulates sending messages by waiting a random period time distributed around a configurable mean.
    - `failure\_rate`: The sender also has a configurable failure rate.
  + In the README the `num\_messages` are based on the producer, however from testing the sender is what seems to be using that configured limit
    - What happens when a transmission fails does not seem very clear. Does it retry the same message? or does it keep going and the message is lost?

## Third Task

* Automate three or more tests from your test plan with any test framework of your choice to test the application.
  + The automation has been pushed in as a zipped file of the entire repository and changes made, which can later be used as a docker container possibly or just a direct test script in the CI/CD pipeline because of the nature of how it was automated.
* Provide a brief description of the implementation (language/tools used, etc).
  + Python:
    - Python offers a lot of flexibility with being able to run a test against a lot of different systems with less setup time and great support.
    - Rust should be given a more in-depth look to see if it would be able to do the same as python in this case to keep it simple and programming context switching to a minimum.
  + Docker:
    - Docker allows applications to run in various environments with reproducible testing scenarios as long as Docker can be installed. This also gives us the ability to use within a CI/CD pipeline without installing another dependency to keep up-to-date within the pipeline.
  + VSCode:
    - VSCode provides a variety of uses with extensions for all the different ways I can use/read different programming languages and other third party tools.

## Fourth Task

* Is there anything missing in the output or configuration to be able to effectively test the system?
  + Debug logs do not report the details of the message which is supposed to be 100 random characters. Enabling Trace on RUST\_LOG does not seem to log the messages either. This prevents me from validating that the messages we are getting are what is expected.
  + Being able to get the logs in a more programmatic order such as json logs or via an API for easier automation.

## Fifth Task

* Can you find any bugs? (Just look out for errors as we are testing/writing automation)
  + Bugs are reported in the `test/Bug Reports` folder in the zipped repository