Technical Test Backend

Created:	March 5th, 2020
Last updated:	May 04th, 2021
Author:	Stijn De Ketelaere
	Jeroen Vervaeke
	Peter De Jonghe

Table of Contents

1. Introduction **3**

2. Assignment **3**

2.1. Setting 3

2.2. Rest endpoint 3

2.3 Database 4

2.4. Requirements 4

2.4.1 Technology 4

2.4.2 Performance and efficiency 4

2.4.3 Scoring criteria 4

2.4.4 Final note 5

1. Introduction

This document contains the description of a technical test for backend development.
This document needs to be treated as confidential and cannot be distributed without the written consent of Proceedix.

2. Assignment

2.1. Setting

A client application fires log messages - in parallel - at a logging endpoint at a rate of 1500 messages per second. Log messages are sent in bulk (array of log messages).

2.2. Rest endpoint

A log message sent by the client application, has the following properties:

- a **log date**: the date/time that the log message was created (unix timestamp)
- **name** of the application that's logging the message
- a **message**: the actual text of the log message

See *swagger.yaml* for more details about the rest endpoint.

The actual text of the log message *may* start with "[<log level>]" followed by the log message itself. Example: "*[error] Could not connect to the database!*". The whitespaces on both ends of the message should be stripped, in the previous example this would turn " *Could not connect to the database!*" into "*Could not connect to the database!*"

The log level should be stripped from the message and saved in a separate column.
Allowed log levels are:

- Trace
- Debug
- Info
- Warn
- Error

When no log level is included in the message the log level is *Info*.

2.3 Database

The Web API stores the log messages in the postgres database (see `database.sql`).
This database contains 2 tables:

- **Application**: stores the names of all different applications
- **Logmessage**: stores all log messages

Typically log records are appended, and never updated or deleted.

2.4. Requirements

2.4.1 Technology

The **backend** should be written using one of the following languages:

- C# (dotnet core)
- Rust

The **database** that the backend is talking to is a PostgreSql database.

It should be **easy** to **deploy** the application locally. The project should contain a *Dockerfile* and a *docker-compose.yaml* file. The Docker container should be configured to run on a Linux system.
Typing "*docker-compose up*" should be sufficient for us (the reviewers) to test the application.

In case the candidate did not manage to set up the docker-compose file as requested above then the test should contain clear guidelines on how to run the application. Imagine that these guidelines are the manual for the devops team to deploy the application.
This documentation is not required when you were able to provide a proper docker-compose file.

2.4.2 Performance and efficiency

The application should be written as performant as possible.

2.4.3 Scoring criteria

The scoring criteria include (but are not limited to):

- Code style
    - Consistency
    - Documentation (comments)
    - Readability
    - Resilience
- Code architecture
    - Clean separation of concerns
    - Testability (is this code easy to use in unit tests? Unit tests are not required)

- Implementation of the docker file
    - Does it actually work and run the application (and its dependencies)?
- Performance
    - Did the candidate consider the 1500 requests / second requirement?

2.4.4 Final note

Wherever the assignment lacks clear instructions or is incomplete, the candidate is free to fill in the voids in any way he/she deems appropriate. In this case, please provide proper documentation with the motivation of your choices.