

G-Research Take-Home Assessment Report

Project Overview

Task Description

A car rental application logs a start event when a rental car is collected by the customer, and a corresponding end event when the car is returned. Each rental session is limited to 24 hours. The application writes the events into an output file as JSON records, where each record represents either the start or end of a rental session. The file contains start and end records for multiple rental sessions.

Objectives

Using the logged information in an output file, the objective of this project is to develop an application that parses the output file and generates a single summary record for each session in the format prescribed in the project documentation.

Approach

Technical Choices

The project will use Python 3.11 as the programming language. For this project, PyCharm will be used as an Integrated Development Environment (IDE) due to its seamless integration with GitHub through its Git functionality.

Algorithm and Data Structures

The code reads JSON data from a text file to populate a dictionary called 'db', keyed by an 'id' and containing 'Summary' objects as values. The 'Summary' objects hold information like start and end timestamps, duration, and flags for late receipt and damaged return'. The main logic uses a straightforward iteration over items in the dictionary to write them into a tab-separated text file, using and evaluating Unix timestamps in the process. Local timezone-formatted dates could also be used in place of timestamps. The use of a dictionary for quick look-up and the in-place modification of 'Summary' objects make the approach efficient.

Issues and Challenges

Problems Encountered

The primary challenge I encountered was interpreting the project requirements based on the provided test cases. While the documentation effectively laid out the context of the task, inconsistencies in the example output files made parsing the data confusing.

Sample events

```
[{
  "type": "START",
  "id": "ABC123",
  "timestamp": "1681722000",
  "comments": "No issues - brand new and shiny!"
},
{
  "type": "END",
  "id": "ABC123",
  "timestamp": "1681743600",
  "comments": "Car is missing both front wheels!"
},
{
  "type": "START",
  "id": "ABC456",
  "timestamp": "1680343200",
  "comments": "Small dent on passenger door"
},
{
  "type": "END",
  "id": "1680382800",
  "timestamp": "0123499",
  "comments": ""
}]
```

For instance, the first record, indicated by a red number 1, adheres to the expected data format for its corresponding 'id'.

However, the record near the red number 2 displays an 'id' that appears to be in a timestamp format rather than a standard 'id'. Moreover, the timestamp for that particular record is not coherent or easily understandable.

Solutions and Workarounds

I operated under the assumption that the timestamp in the final entry was erroneously placed. Consequently, I switched it with the 'id' and assigned the 'id' of the third record as the rental session id. I also generated a new test data file to account for consistency and various edge cases.

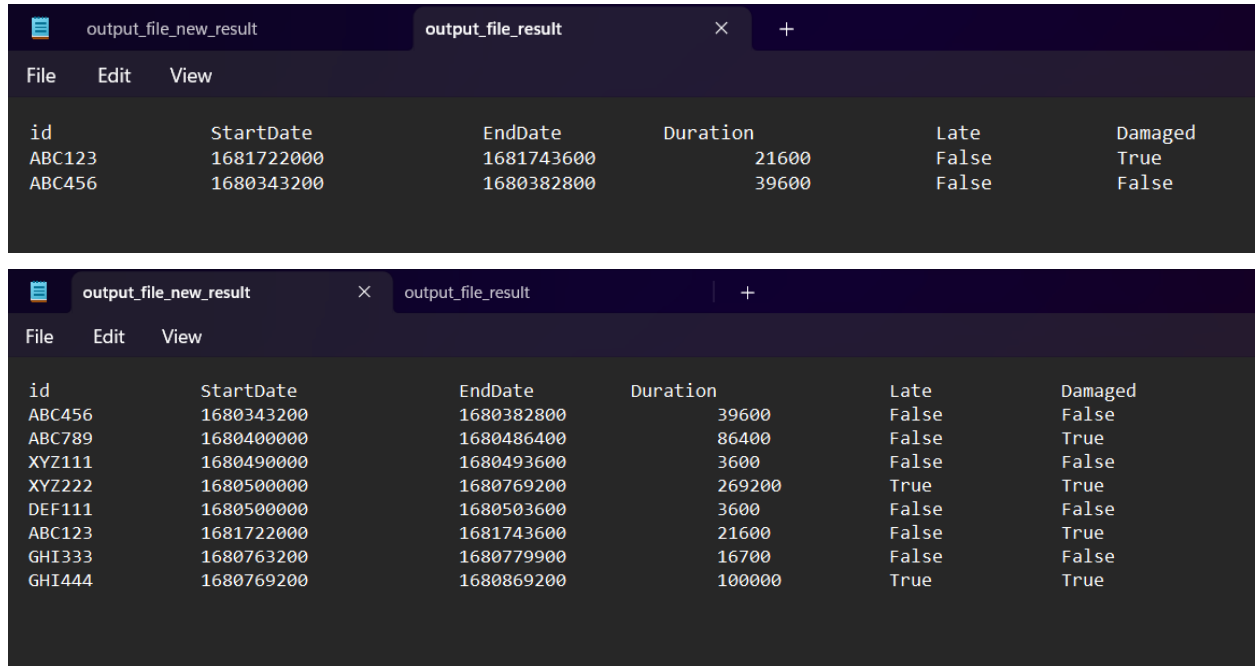
Testing Strategy

Test Cases

I generated a fresh text file named '**output_file_new.txt**' to accommodate various edge cases. These include scenarios such as multiple rental sessions initiating simultaneously, overlapping rental sessions, and sessions that exceed the 24-hour limit. I've made the assumption that the car rental application logging this data is logically consistent—meaning rental session IDs will be unique, records will arrive sorted by timestamp, and no session will lack an ending record, among other things.

Test Results

There are two test data files in the project folder. One file was provided in the documentation. The other file is synthetically generated. These are the results my code gives upon processing the data in the two files. The output generated files can be identified by their corresponding test files with ‘_result’ added at the end.



The image displays two screenshots of a code editor window showing test results. The top screenshot shows the output for 'output_file_new_result' with two rows of data. The bottom screenshot shows the output for 'output_file_result' with ten rows of data.

id	StartDate	EndDate	Duration	Late	Damaged
ABC123	1681722000	1681743600	21600	False	True
ABC456	1680343200	1680382800	39600	False	False

id	StartDate	EndDate	Duration	Late	Damaged
ABC456	1680343200	1680382800	39600	False	False
ABC789	1680400000	1680486400	86400	False	True
XYZ111	1680490000	1680493600	3600	False	False
XYZ222	1680500000	1680769200	269200	True	True
DEF111	1680500000	1680503600	3600	False	False
ABC123	1681722000	1681743600	21600	False	True
GHI333	1680763200	1680779900	16700	False	False
GHI444	1680769200	1680869200	100000	True	True

Potential Improvements

Instead of simply generating a text file with the data, it is also easy to set up code to store the output data to a MySQL/SQLite3 database but I didn't feel the need to do so because of its complexity in testing.