**Push Notifications Compressor**

**1. Problem Definition**

Some mobile apps like komoot provide the user with push notifications of their friends’ updates like status posted … etc.

Although push notifications are useful, sending to many notifications can be annoying.

**2. Proposed Solutions**

1. **Solution Idea:**

To avoid such scenario in which the user receives a lot of annoying notifications, we came up with the idea of bundling notifications to reduce the amount of notifications we send.   
That means we need to wait a bit to collect notifications until we can send those bundles. But we also know how important it is to send notifications as soon as possible; users want to know about friend’s updates as soon as possible and start talking about it.  
The goal is

1. **Solution Requirements/Restrictions:**

* To send no more than 4 notifications a day to a user (should happen only few times)
* To keep the sending delay minimal as possible.

1. **Solution Criteria (what is guaranteed):**

* The user will get all the notifications that happened today, today.
* The user will not get more than 5 push notifications per day.
* The average maximum delay between the update release (a user posted something), and receiving a push notification is 4 hours.
* The delay between the update release and receiving push notification with the updates will not be more than 8 hours ever.

1. **The Idea Behind:**

If we splitted the day into 5 sections.



- The first section starts at 00:00, ends at 04:00.  
- The second section starts at 04:00, ends at 08:00.  
- The third section starts at 08:00, ends at 12:00.  
- The fourth section starts at 12:00, ends at 16:00.  
- The fifth section starts at 16:00, ends at 00:00.

This idea seems good, but a better modified version is, instead of putting specific milestones (time stamps) at 04:00, 08:00, 12:00 … etc, let’s start when the first notification comes, so the distribution can be something like this (in reality):



Here you start counting the four hours limit (for each of the first four periods) just when you receive a notification, the same with the last period too.

Even if you received less than 5 notifications:



You still can take 4 hours (or less) starting from the first new notification.

1. **The Implemented Solution:**

Here I used a day based simulator, simulator that assumes that I cannot see the future notification (which is the actual scenario in real life), but I assumed that I can send the notification at any time just by specifying it (even after the time elapsed), but with no more than half a day maybe.

1. **Idea Strength:**

The idea’s strength is that, it’s dynamic, but once you received a notification, it will stay in the stack maximum 4 hours.

It can be better if we made some analytics over all the user experience, so we can know which period is better to send push notifications.

1. **Idea Weaknesses:**

At worst case, it will be five push notifications per day, and four hours difference between releasing a notification and receiving it – eight hours for the fifth notification; on the other side, this is not so bad and will happen very rarely.

**3. Toolkit, Setup:**

1. **Toolkit:**

Here I’m using:

- Python 3.5

- Pandas data frames library

Python; because it’s easy to develop prototype with it.

Pandas; because it’s so effective in processing data frames.

In reality, this problem may be better to be solved in Node.js because of the runtime streaming/processing, or C++ because it’s faster in processing.

1. **Setup the environment:**

- You can find the code repo here:  
<https://github.com/AbdelrahmanRadwan/Push-Notifications-compressor>   
  
- Or download it as zip file from dropbox here:  
<https://www.dropbox.com/sh/6flgdg3secjxyob/AAAgewGZTA-mQsfnRq8bwBMxa?dl=0>   
  
- Install the Python requirements:

$ pip3 install -r requirements.txt

1. **Running the code:**

- Run the algorithm (it will ask you to choose the notifications csv file and will save the result to push\_notifications\_results folder):

$ python3 algorithms/push\_notification\_compressor.py

- Run the unit test (it will ask you to choose the push notification csv file and will save the result to tester/testing\_results folder):

$ python3 tester/test.py

**4. Code and Functionalities:**

1. **The push notifications generator:**

In “\algorithms\push\_notification\_compressor.py”, you can find the algorithm, let’s discuss it simply:

A class user is created here, to hold and handle the user state regarding the bundled notifications, the received number of push notifications, friends who sent notifications and the earliest notification in the stack right now.

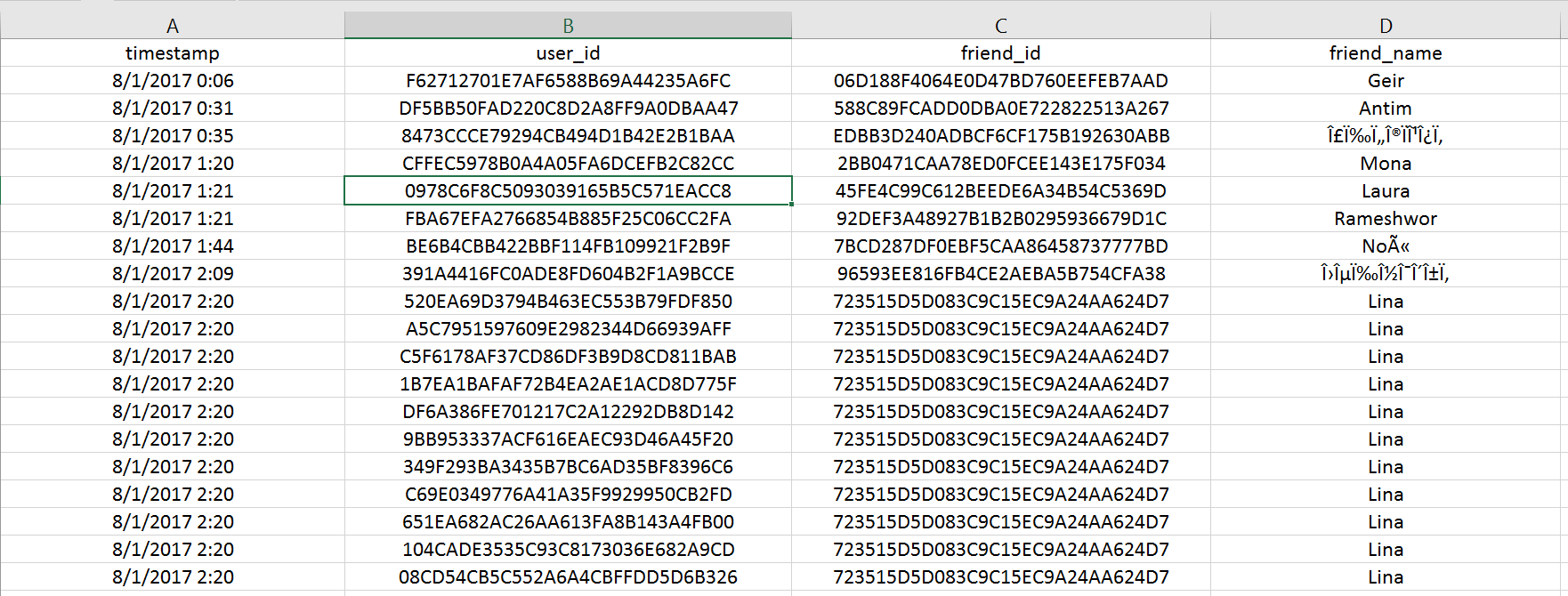
The user class contains these functions:

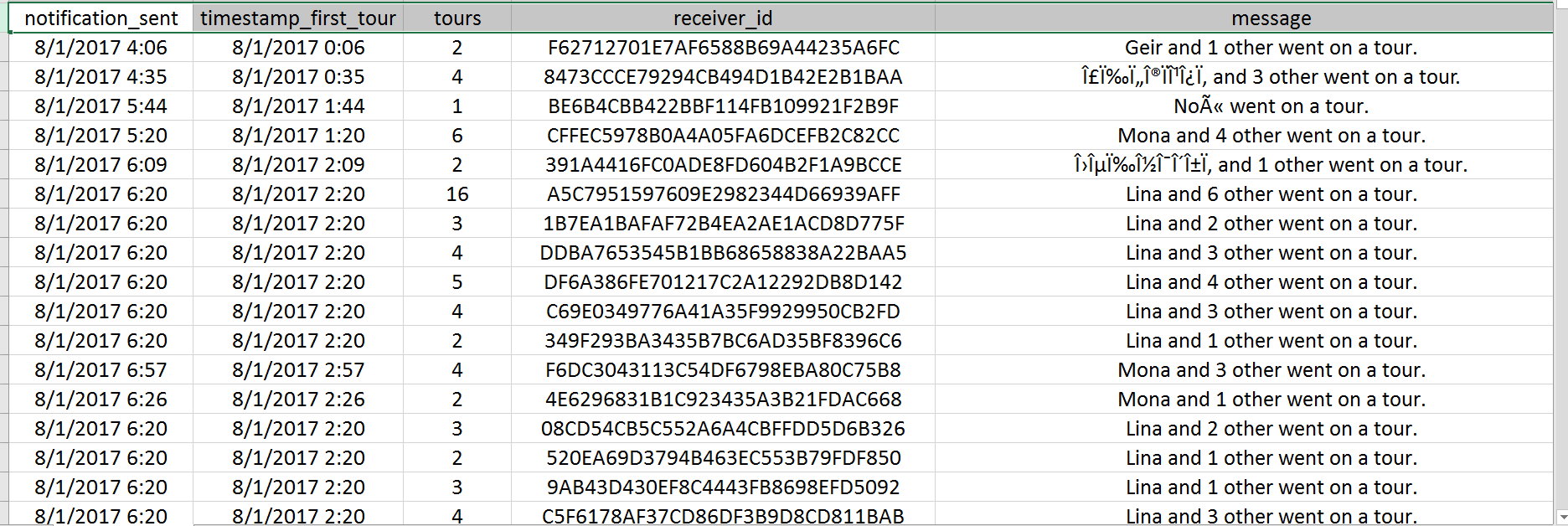
- Fire (): the responsible for sending the push notification.

- notify (): the responsible for managing the push notifications numbers and notification range (the 4-hour rule that we discussed before).

- new\_notification (): the responsible for adding one notification insert the current range stack.

**The flow of the algorithm:**

* We go through all the records in the notification sheet (please make sure that the headers format like the following).
* Every user of these users has unique ID, we map this unique ID to a user object (user defined class), to represent the user’s state now(regarding receiving and aggregating the notifications).
* Firstly we check if this is a new day or not (in reality we will have a watch or clock running by default, so we don’t need to do such check, I did it to avoid doing a long simulator), if it’s a new day, then we have to fire all the push notifications hold from the previous day (this is mostly the last notification in the previous day), we fire it with a time period equals to the last minute in the preivious day or the time of the first notification in the stack now + 4 hours (in case that this is one of the first four notifications in the day and this is less than the end of the day).
* Now we notify the user, by calling the function notify in the user object, we check the time frame in which the user is experiencing now (based on the number of received notifications and push notifications in their state now), and decide if we have to fire the held notifications or to push the coming one to the stack (based on the time difference, is it more than four hours or not).
* If we reached the end of the file (this presents reaching the last few minutes in the day, in reality), we have to fire all the notifications held so far in users’ stacks.

After running the script, it will ask you to provide the csv file which includes the notifications, and it will log to the terminal what is going on (the pushed push notifications), it also will save it to a csv file (/push\_notifications\_results/push\_notifications.csv):

1. **The Unit Test**

I created a unit test to check if the results mentioned above (the push notification csv file) are meeting the algorithm expectations or not.

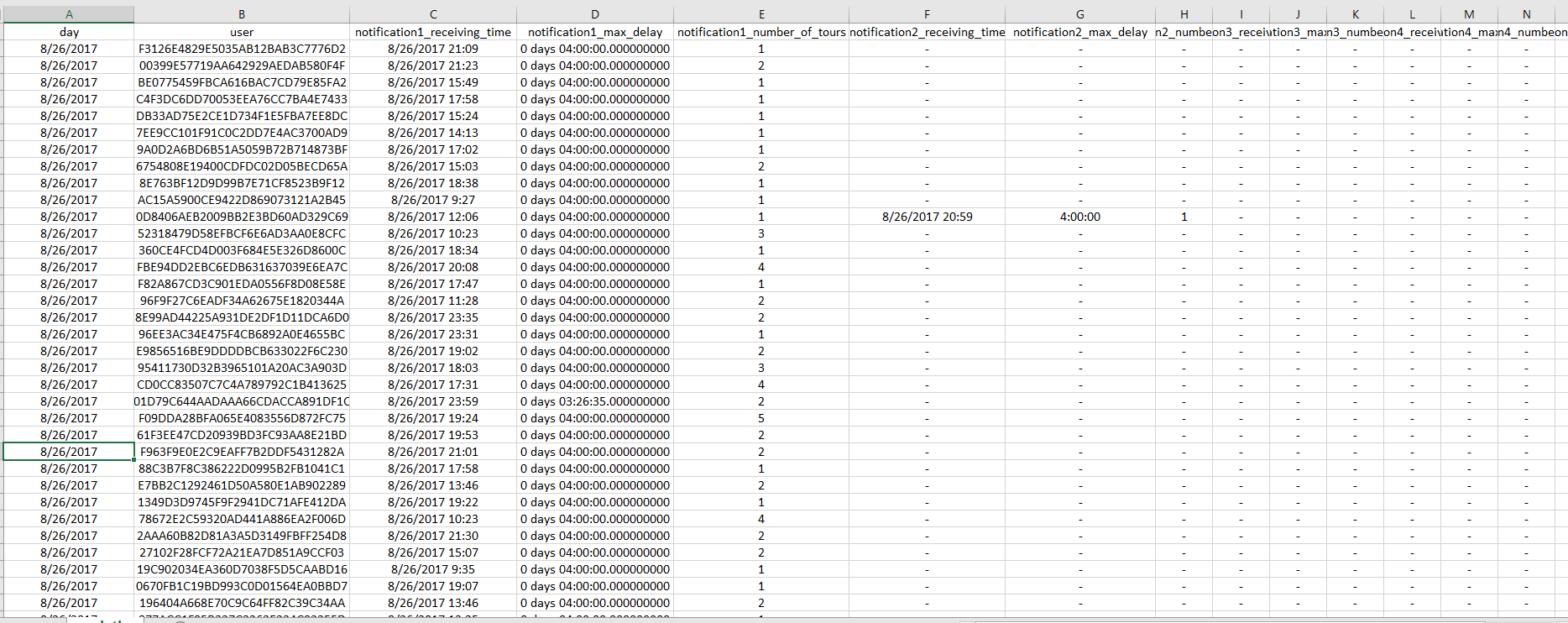
You can find the unit test at (/tester/test.py).

**How it works:**

* It will ask you to select a push notification csv file like the above mentioned one, in the same format.
* It makes some analysis on this csv file and saves the results to another csv file (/tester/test\_results/analytics.csv).

**The testing flow:**

* It goes through all the push notifications.
* Make data transformation to another data structure (dictionary of date => dictionary of users => tuple of analytics), the analytics are, the first notification time, the maximum delay that this user experienced in this day for each notification during the whole day and the number of tours sent per notification.
* We walk through this analytics dictionary of dictionary of analytics, and save it to csv file.
* In the csv analytics file, you can see these columns:  
  - day: the day of receiving the notification.  
  - User: the id of the user (it’s unique per day).  
  - notification1\_receiving\_time: the time in which this user received a push notification in this day.  
  - notification1\_max\_delay: the maximum delay happened before receiving this push notification (the delay is the time elapsed between action happens and getting it as push notification).  
  - notification1\_number\_of\_tours: number of tours included in the above mentioned push notification.  
    
  and the same with all the notifications from 2 to 5 (we agreed that the user gets five notifications maximum per day).



**How the system is expected to be used as a product:**

In real life, you would have clock class, which is a simulator for the time now, and you would need to add a flag (notification late fire date/time) which is an indicator to the maximum latency by which you have to fire the push notification, if you have some flag like this, you would be able to say if you have to fire this notification stack for this user at this moment or not.

Maybe we can parallize it too.

**5. Results**

1. **Unit test**

Based on the unit test results (I upladed it [**here**](https://docs.google.com/spreadsheets/d/19qmTGjal4czqpm74ZLKOBM7jr1s_lOSVoCIZfAHqK14/edit#gid=578445613) and gave you edit permission so you can make your own analytics/evaluations online easily), we can see that:

* Average number of notifications per day for each user is 2.5
* Average delay in notifications is 3 hours.

**Future work and how can we enhance the algorithm:**

Enhancing the algorithm can be done by adapting the ranges more to be user specified, I mean let’s assume that the user X used to use the app in the morning, and he/she usually doesn’t use it in the evening, so it will be more logical to move the push notifications range to evening, and don’t send in the morning at all based on the fact that the user use the app a lot in the morning.

**My experience with** **komoot challenge:**

* I like this kind of problems, because they are real life and so interesting, challenging and useful.
* I feel happy to develop such algorithm, this idea jumped to my mind suddenly when I read the challenge document, and I was exited towards it.
* Although the idea is not complex, implementing it and testing it took me time, specially testing it and creating the unit test, because I discovered a lot of bugs and fixed them.
* Overall, the challenge is not hard, it’s cool and exciting.
* It took me less than half a day to document my work, and around a day to implement the idea and maybe half a day or less to plan the solution architecture and the flow.
* I learnt that planning is so important, and testing is incredibally important, I solved the challenge around 3 days ago, and didn’t create automated testing script, but yesterday I decieded to extract some analytics to see if this solution is really good or not, and found that the results are wrong, because of some logical bugs.
* It’s always hard to simulat something, specially the clock, so I would like to know from you how do you usually test such features, all the apps that I worked on so far include direct request/response, but this is the first time-dependent task I work on it, and it’s really interesting.