Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

ΠΑΠΑΝΔΡΕΟΥ ΓΕΩΡΓΙΟΣ ΑΕΜ: 10372

Ο σκοπός της εργασίας είναι η ασύγχρονη λήψη πληροφοριών συναλλαγών για ένα σύνολο συμβόλων από το API του Finnhub μέσω του Raspberry Pi. Για την υλοποίηση της χρησιμοποιήθηκε η μεθοδολογία producer-consumer, σε συνδιασμό με τις απαραίτητες συναρτήσεις για την επικοινωνία με το Finnhub. Για την επεξήγηση του προγράμματος, θα εστιάσουμε με μεγαλύτερη προσοχή στις συναρτήσεις των νημάτων και στη βασική συνάρτηση του κώδικα.

Ο κώδικας της εργασίας και τα αποτελέσματα βρίσκονται στο παρακάτω το link:
https://github.com/NotGorgy/GitHashingExercise/tree/main

(α) Η συνάρτηση `main` αρχικοποιεί τα δεδομένα και τα αρχεία καταγραφής, δημιουργεί την κοινή ουρά και συνδέεται με τον πάροχο δεδομένων μέσω WebSocket *(συναρτήσεις 2 & 3)*. Στη συνέχεια, εκκινεί τα νήματα που επεξεργάζονται τα δεδομένα. Αναμένει τον τερματισμό τους, όταν δηλαδή ληφθεί το σήμα `SIGINT` (Ctrl+C), έπειτα του οποίου κλείνει τα αρχεία και απελευθερώνει τους πόρους της ουράς για ομαλό τερματισμό.

(β) Η συνάρτηση `producer` είναι υπεύθυνη για τη διατήρηση της σύνδεσης WebSocket με τον **server**, την διαχείριση των προσπαθειών επανασύνδεσης και την λήψη δεδομένων.

Εντός του κύριου βρόχου, ελέγχεται συνεχώς η σύνδεση, και όσο η σημαία `termination` είναι `false`, η συνάρτηση καλεί την `lws_service(context, 1000)` για να διατηρήσει την σύνδεση ενεργή.

Λογική Διαχείρισης Σύνδεσης:

- Αν το `connection_flag` είναι 0 ή -1, αυτό σημαίνει πρόβλημα:
  - `0`: Λόγω αποσύνδεσης ή απουσίας δεδομένων => προσπάθεια επανεκκίνησης της σύνδεσης.
  - `-1`: Υπάρχει σφάλμα => πρέπει να γίνει επανασύνδεση.

  Σε κάθε περίπτωση, η σύνδεση επανεκκινείται με τις `lws_cancel_service()` και `lws_context_destroy()`, που εκτελούν την αποσύνδεση, ενώ η συνάρτηση `create_client()` φτιάχνει πάλι το WebSocket.
- Αν παρατηρηθούν αρκετές παρατεταμένες αποτυχίες επανασύνδεσης => 10sec timeout
- Όταν το `connection_flag` γίνει `1`, η σύνδεση έχει αποκατασταθεί και συνεχίζεται η λήψη των δεδομένων.

Ύστερα με τη βοήθεια της συνάρτησης `callback_ws` που χειρίζεται τα διάφορα γεγονότα WebSocket γίνεται η λήψη και ανάλυση των δεδομένων από τις συναλλαγές.

- **Καθιέρωση της σύνδεσης** => ο πελάτης προχωρά στην προετοιμασία για την εγγραφή και αποστολή δεδομένων (`i`)
- **Δυνατότητα εγγραφής δεδομένων από τον πελάτη** => εκτελείται η εγγραφή στα καθορισμένα σύμβολα με την αποστολή μηνυμάτων (`ii`)
- **Λήψη μηνύματος** => αναλύονται των δεδομένων JSON (`iii`)
- **Κλείσιμο σύνδεσης** => προσπάθεια επανασύνδεσης
- **Σφάλμα σύνδεσης** => προσπάθεια επανασύνδεσης

`(i)` Η συνάρτηση `lws_callback_on_writable` σηματοδοτεί ότι η σύνδεση wsi μπορεί να γράψει δεδομένα. Αυτό προκαλεί την ενεργοποίηση του callback για την αποστολή δεδομένων όταν η σύνδεση είναι έτοιμη.

(ii) Η συνάρτηση **`send_message`** μορφοποιεί και στέλνει μηνύματα μέσω της σύνδεσης WebSocket, για την εγγραφή στα καθορισμένα σύμβολα.

(iii) Η συνάρτηση **`parse_json_data`** εξάγει δεδομένα μετοχών από τα μηνύματα JSON που λαμβάνονται. Ελέγχει αν η δομή είναι έγκυρη και εξάγει τα πεδία σύμβολο, τιμή, χρόνος και όγκο, προσθέτοντας τις έγκυρες συναλλαγές στην κοινή ουρά μαζί με τη χρονική στιγμή εισαγωγής. Για ασφαλή εισαγωγή δεδομένων χρησιμοποιείται mutex. Επίσης παρακολουθεί αν λαμβάνονται συνεχόμενα μηνύματα χωρίς δεδομένα με τον μετρητή `continues_pings` ώστε να ειδοποιήσει για επανεκκίνηση την σύνδεσης αν χρειαστεί.

(γ) Η συνάρτηση **`consumer_read_data`** διαβάζει τα δεδομένα μέσω της κοινής ουράς, διασφαλίζοντας αποκλειστική πρόσβαση με τη χρήση mutex. Τα δεδομένα αποθηκεύονται στα αντίστοιχα .txt αρχεία και ενημερώνεται ο πίνακας των candlesticks για κάθε σύμβολο *(συναρτηση 6)*. Επίσης, υπολογίζει τις καθυστερήσεις μεταξύ της λήψης δεδομένων από το Finnhub και της επεξεργασίας τους από τον παραγωγού και καταναλωτή, τα οποία αποθηκεύει στα αρχεία `finnhub_producer_delay.txt` και `producer_consumer_delay.txt`.

(δ) Η συνάρτηση **`sleepyhead`** εκτελεί έναν βρόχο που περιμένει 1 λεπτό ή μέχρι να ληφθεί το σήμα τερματισμού (`SIGINT`). Αν η αναμονή λήξει λόγω του χρονικού ορίου, αποθηκεύει τις δεδομένα candlestick για κάθε σύμβολο με συλλεγμένα δεδομένα. Επίσης, υπολογίζει και καταγράφει τον συνολικό όγκο συναλλαγών και τον SMA των τελευταίων 15 λεπτών. Σε περίπτωση που δεν υπάρχουν δεδομένα για κάποιο σύμβολο, ενημερώνεται το `connection_flag` για να προσπαθήσει επανασύνδεση. Τέλος υπολογίζει το χρονικό διάστημα μεταξύ της δημιουργίας των candlestick για κάθε σύμβολο.

Για τον υπολογισμό του SMA των 15 λεπτών χρησιμοποιήθηκε ο ορισμός από την ιστοσελίδα

$$SMA = \frac{A1 + A2 + ... + An}{n} \text{ where } An = \text{the price of an asset at period } n, n = \text{the number of periods}$$

Σύντομη περιγραφή υπόλοιπων συναρτήσεων

1. Η συνάρτηση **`handle_sigint`** χειρίζεται το σήμα τερματισμού SIGINT (Ctrl+C). Όταν λαμβάνει το σήμα, ενεργοποιεί τη σημαία `termination` και ξυπνάει τυχόν νήματα που βρίσκονται σε αναμονή. Με την αλλαγή της σημαίας, τα νήματα τερματίζουν τη λειτουργία τους, και τέλος τερματίζει και η `main`.
2. Η συνάρτηση **`create_client`** δημιουργεί τη σύνδεση WebSocket ρυθμίζοντας το απαραίτητο περιβάλλον (όπως ρυθμίσεις πρωτοκόλλου, παραμέτρους σύνδεσης, διαχείριση SSL και ρυθμίσεις του context) για την εκτέλεση της σύνδεσης. Επιπλέον, καθορίζει τις απαιτούμενες παραμέτρους, όπως η διεύθυνση του host, το API Key, η θύρα και οι ρυθμίσεις SSL.
3. Η συνάρτηση **`create_txt_files`** δημιουργεί τα εξής αρχεία:

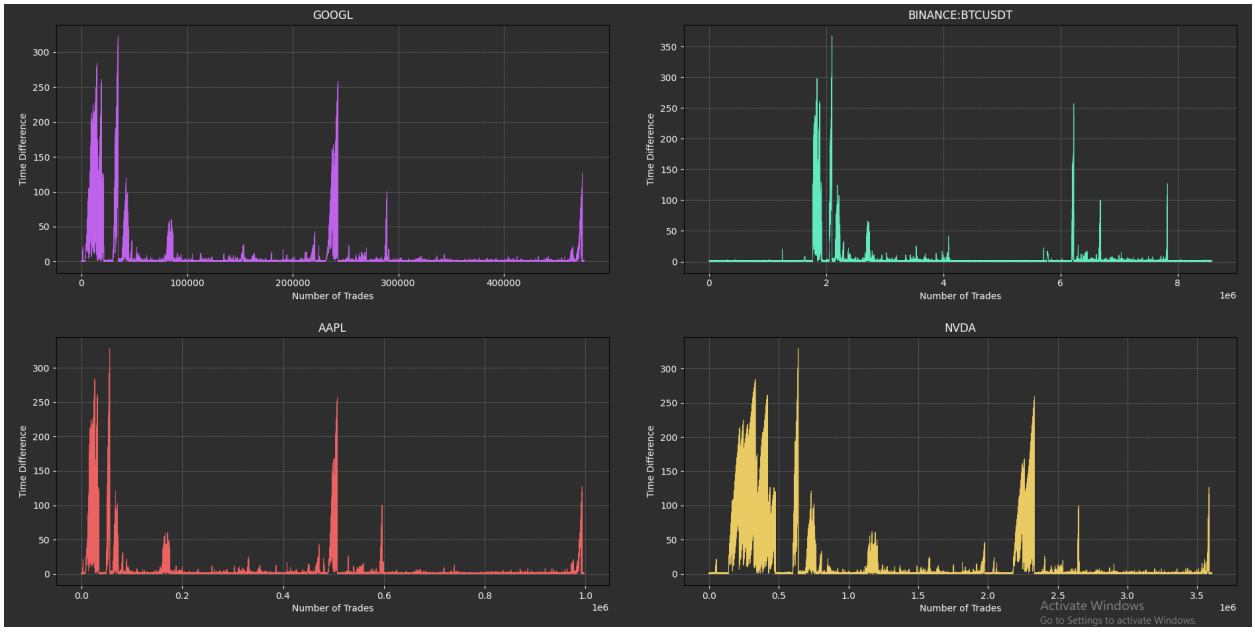
- Για κάθε σύμβολο μετοχής:
    - Βασικές πληροφορίες (τιμή, όγκος, χρόνος).
    - Candlestick δεδομένα (αρχική, τελική, μέγιστη, ελάχιστη τιμή, συνολικός όγκος).
    - Δεδομένα για τον κινούμενο μέσο όρο (τιμές συναλλαγών και συνολικός όγκος των τελευταίων 15 λεπτών).
- Επιπλέον:
    - Δύο αρχεία για τις καθυστερήσεις μεταξύ της λήψης δεδομένων από το Finnhub και της επεξεργασίας τους από τον παραγωγού και καταναλωτή.

- Ένα αρχείο για το χρονικό διάστημα μεταξύ της δημιουργίας των candlestick για κάθε σύμβολο.

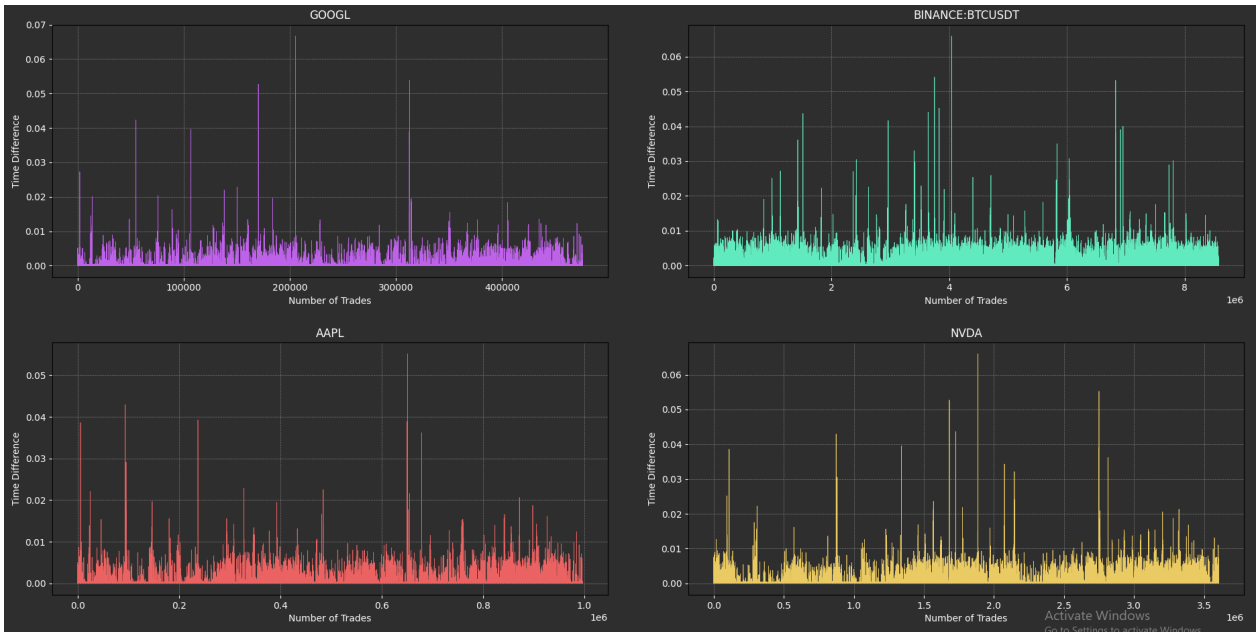
6. Η συνάρτηση `process_trade` ενημερώνει τα δεδομένα των candlestick με βάση τις νέες συναλλαγές. Επιπλέον αρχικοποιεί τα δεδομένα τους κατά την πρώτη συναλλαγή ανά λεπτό.
7. Η `static struct lws_protocols protocols[]` ορίζει τα πρωτόκολλα WebSocket που θα χρησιμοποιηθούν στην εφαρμογή. Αυτός ο πίνακας περιλαμβάνει πληροφορίες για το κάθε πρωτόκολλο, όπως το όνομά του και τη συνάρτηση callback που θα κληθεί όταν συμβεί κάποιο γεγονός σχετικό με το συγκεκριμένο πρωτόκολλο.

ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΡΟΓΡΑΜΜΑΤΟΣ

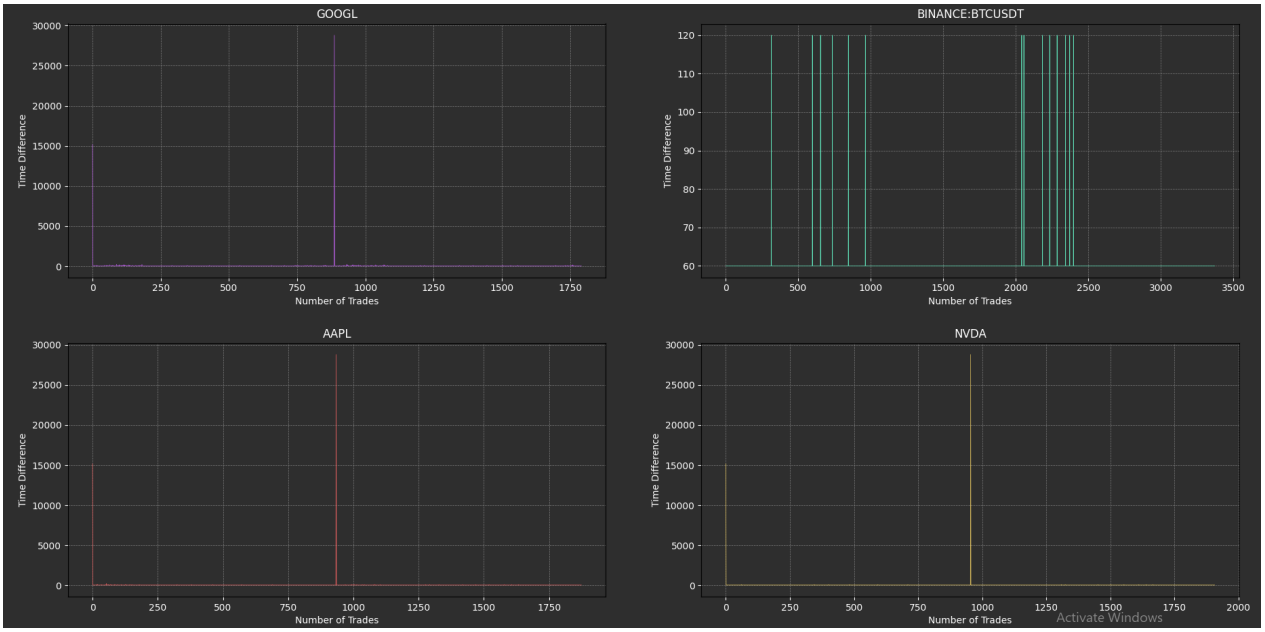
Διάγραμμα 1 καθυστερήσεων μεταξύ Finnhub και εισαγωγής των δεδομένων στην κοινή ουρά από τον producer



***Διάγραμμα 2* καθυστερήσεων μεταξύ εισαγωγής των δεδομένων στην κοινή ουρά από τον producer και αποθήκευση τους από τον consumer**



***Διάγραμμα 3* χρονικών διαφόρων μεταξύ της αποθήκευσης των candlesticks για κάθε σύμβολο μετοχής**



- Οι υψηλές καθυστερήσεις του Finnhub οφείλονται πιθανώς στις στιγμές που επιχειρείται επανασύνδεση καταστρέφοντας και ξαναδημιουργώντας το **Web Socket**.
- Η καθυστέρηση αποθήκευσης των δεδομένον από τη στιγμή που εισάγονται στην κοινή ουρά οφείλεται κυρίως στην αναμονή για την απελευθέρωση του **mutex** από τον producer**.** Οι καθυστερήσεις αυτές μπορούν να αυξηθούν σε περιόδους **έντονου φόρτου δεδομένων,** καθώς η χρηση του mutex από τον producer αυξάνεται. Παραδείγματος χάριν, κατά τη διάρκεια των δοκιμών, διαπιστώθηκε ότι η μετοχή του 'BTCUSDT' έχει πολύ μεγαλύτερο όγκο δεδομένων σε σύγκριση με τα υπόλοιπα σύμβολα, γεγονός που εξηγεί την παρατηρούμενη μεγαλύτερη καθυστέρηση στο διάγραμμα.
- Λόγω των **ωρών λειτουργίας της αγοράς** υπάρχουν διαστήματα απουσίας δεδομένων για ορισμένες μετοχές, γεγονος που καταγράφει μια μεγάλη καθυστέρηση στην αποθήκευση του επόμενου candlestick. Ωστόσο, παρατηρούμε ότι η χρονική διαφορά είναι **σχεδόν μηδενική** κατά τις περιόδους που λαμβάνονται δεδομένα, επιβεβαιώνοντας ότι δεν υπάρχει 'drift' στον χρόνο δημιουργίας τους. Οι καθυστερήσεις 120 δευτερολέπτων του διαγράμματος 'BTCUSDT' δηλώνουν ότι δεν στάλθηκαν δεδομένα κατά τη διάρκεια του συγκεκριμένου λεπτού για το σύμβολο, επομένως εκτελέστηκε επανασύνδεση, η οποία φαίνεται να διόρθωσε το πρόβλημα.

Ποσοστό χρόνου αδρανείς της CPU:

Ο υπολογισμός του γίνεται με βάση τα δεδομένα:

- **real**: Αυτή η τιμή δείχνει τον συνολικό χρόνο που το πρόγραμμα λειτούργησε.
- **user**: Αυτή η τιμή δείχνει το χρόνο CPU που ξοδεύτηκε στην εκτέλεση του προγράμματος στον χρήστη (user space).
- **sys**: Αυτή η τιμή δείχνει το χρόνο CPU που ξοδεύτηκε στην εκτέλεση του προγράμματος στο σύστημα (kernel space).

Επομένως:

$$\pi = 1 - \frac{user + sys}{real} \cdot 100\,\% = 1 - \frac{(26 \cdot 60 + 57.760) + (2 \cdot 60 + 7.536)}{3386 \cdot 60 + 46.599} \cdot 100\,\% \simeq (1 - 0.0086) \cdot 100\,\% = 99.14\,\%$$

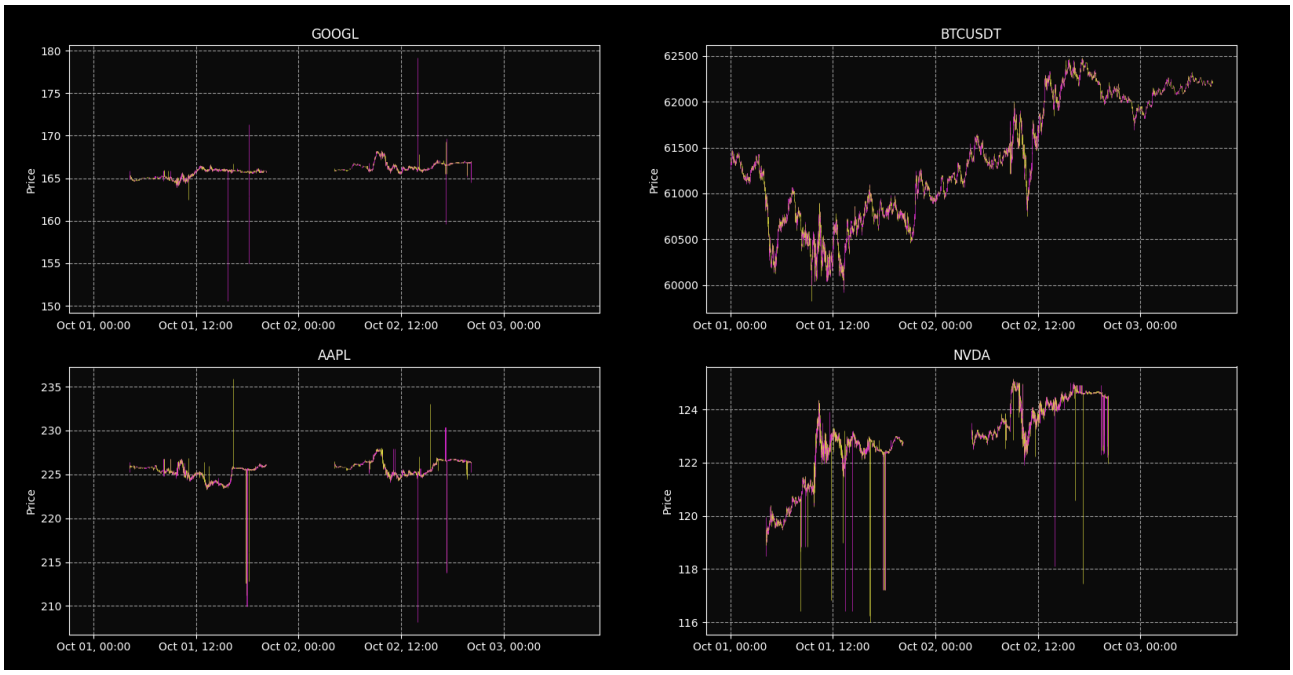
Το αποτέλεσμα είναι λογικό, καθώς για το μεγαλύτερο χρονικό διάστημα το πρόγραμμα βρίσκεται σε **αναμονή**, περιμένοντας να λάβει δεδομένα συναλλαγών από το Finnhub.

```
real    3386m46.599s
user    26m57.760s
sys     2m7.536s
george@pi:~/tests $
```

Αξιοπιστία προγράμματος:

Το πρόγραμμα είναι σχεδιασμένο να λειτουργεί για μεγάλα χρονικά διαστήματα (ημέρες) λόγω της διαχείρισης περιπτώσεων αποσύνδεσης, σφαλμάτων στη σύνδεση και απουσίας δεδομένων, όπως αναλύεται στην περιγραφή της υλοποίησης του κώδικα. Η ικανότητά του να ανταπεξέρχεται σε αυτές τις περιπτώσεις επιβεβαιώθηκε κατά τη διάρκεια των δοκιμών. Επιπλέον, δοκιμάστηκε και η αποσύνδεση του δικτύου, κατά την οποία το πρόγραμμα δεν διέκοψε τη λειτουργία του και συνέχισε κανονικά μετά την επανασύνδεσή του. Το πρόγραμμα χρειάστηκε να παραμείνει ενεργό πολλαπλές φορές για διαστήματα μεγαλύτερα από 24 ώρες προκειμένου να επαληθευτεί η λειτουργιά του. Επιπλέον, από τα τελικά αποτελέσματα φαίνεται ότι λειτούργησε χωρίς βλάβες για 56.4 συνεχείς ώρες. Τέλος, όπως αποδεικνύεται από τα διαγράμματα (3) και (4), δεν υπάρχουν απώλειες δεδομένων εκτός από τις χρονικές περιόδους κατά τις οποίες το ίδιο το Finnhub δεν μεταδίδει δεδομένα για ορισμένες μετοχές.

Διάγραμμα 4 candlesticks



Διάγραμμα 5 συνολικού όγκου συναλλαγών και SMA των τελευταίων 15 λεπτών