

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



- Πτυχιακή εργασία -

Ολοκληρωμένη εφαρμογή για λήψη, αρχειοθέτηση και σύγκριση στιγμιότυπων
σελίδων διαδικτύου βασισμένη σε Docker containers

Σαούλος Παντελεήμων

Επιβλέπων: Μ. Στεφανιδάκης

Git repo: <https://github.com/Pantelhs/website-archiving-poc>

Επιβλέπων

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής
Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής
Ιόνιο Πανεπιστήμιο

Θεόδωρος Ανδρόνικος, Αναπληρωτής Καθηγητής
Ιόνιο Πανεπιστήμιο

Θεμιστοκλής Έξαρχος, Αναπληρωτής Καθηγητής
Ιόνιο Πανεπιστήμιο

Περιεχόμενα

Περιεχόμενα3

Περίληψη4

Summary5

Σκοπός6

Εφαρμογή λήψης στιγμιότυπων σελίδων διαδικτύου7

Τεχνολογίες8

Docker8

Flask9

MariaDB9

Flutter10

phpMyAdmin11

Εργαλεία12

Αρχιτεκτονική13

Εγκατάσταση14

Windows System14

Linux System19

Εικόνες εφαρμογής και επεξήγηση κώδικα25

Αρχική / Dashboard σελίδα25

Start Crawler Screen29

Stop Crawler Screen31

Σελίδα σύγκρισης στιγμιότυπων33

Crawler Backend logs39

Επόμενα βήματα41

Βιβλιογραφία – Δικτυογραφία42

Εικόνες43

Περίληψη

Τις τελευταίες δεκαετίες με την ταχεία άνοδο της χρήσης των ηλεκτρονικών υπολογιστών και τελευταία των κινητών παρατηρείται η τάση όλο και περισσότερων επιχειρήσεων να παρέχουν στους χρήστες τους ηλεκτρονικές σελίδες ώστε να τους δελεάσουν να επιλέξουν τις πλατφόρμες τους. Σαν αποτέλεσμα της ξαφνικής και ραγδαίας χρήσης του διαδικτύου παρατηρείται ταχεία ανάπτυξη τόσο των τεχνολογιών που χρησιμοποιούνται στη δημιουργία ηλεκτρονικών σελίδων αλλά και της μορφής και δομής αυτών των σελίδων. Με σκοπό την παρακολούθηση του παραπάνω φαινομένου υλοποιήθηκε ολοκληρωμένη εφαρμογή λήψης και αρχειοθέτησης σελίδων διαδικτύου με φιλικό προς τον χρήστη περιβάλλον χρήσης, με δυνατότητα κανονισμού λήψης πλήθους στιγμιότυπων ανά καθορισμένα χρονικά διαστήματα βάση τιμών που έχει δώσει ο χρήστης και έπειτα σύγκριση αυτών και τυχόν παλαιότερων στιγμιότυπων με σκοπό την εύρεση διαφορών μεταξύ τους.

Summary

In recent decades with the rapid rise in the use of computers and more recently mobile phones, there is a greater tendency for businesses to provide their clients/users with web pages in order to entice them choosing their platforms instead of others. As a result of the sudden and rapid use of the Internet, there is a lightning development of both the technologies used in the creation of internet pages and the form and structure of said pages. In order to monitor the above phenomenon, a comprehensive application has been developed that downloads a snapshot and subsequently archives it. This web page was implemented with a user-friendly interface which also provides the possibility to configure the number of snapshots taken at specified time intervals based on values given by the user. Finally, the snapshots can be compared with older ones enabling the user to find differences between them.

Σκοπός

Η εφαρμογή έχει δημιουργηθεί με σκοπό τη διευκόλυνση της διαδικασίας λήψης στιγμιότυπων του HTML κώδικα της επιλεγμένης από τον χρήστη ηλεκτρονικής σελίδας, και παροχή εύκολης προς αυτόν σελίδας για διαχείριση της εφαρμογής αλλά και προβολής των αποτελεσμάτων της. Ακόμα σαν στόχος της σελίδας είναι η διευκόλυνση του χρήστη στην αναγνώριση των κομματιών που αλλάζουν συχνότερα σε κάθε σελίδα με σκοπό την αναγνώριση τυχών προτύπων ή και μοτίβων αλλαγών.

Ο σκοπός της εφαρμογής είναι η λήψη στιγμιότυπων ηλεκτρονικών σελίδων που εισάγει ο χρήστης στο περιβάλλον χρήσης που περιλαμβάνεται στην υλοποίηση και έπειτα σύγκριση τους ώστε να γίνουν ποιο εύκολα εμφανές οι αλλαγές που γίνονται σε αυτές τις σελίδες σε βάθος χρόνου. Από τη στιγμή που δοθεί εντολή από τον χρήστη για να ξεκινήσει το backend της εφαρμογής να καταγράφει μια σελίδα, ανοίγει ένα νέο thread και ταυτόχρονα γράφεται στη βάση αυτή η εντολή για να μπορεί να φανεί σε δεύτερο χρόνο στο περιβάλλον χρήσης. Το backend έπειτα από κάθε επανάληψη λήψης στιγμιότυπου ενημερώνει τη βάση δεδομένων ώστε να μπορεί να δείξει το frontend στον χρήστη ανά πάσα στιγμή την πρόοδο της προηγούμενης προγραμματισμένης ενέργειας. Τέλος μετά από κάθε βήμα και λήψη ενός στιγμιότυπου αποθηκεύεται μαζί με τα αρχεία της εφαρμογής σε μορφή html αρχείου.

Εφαρμογή λήψης στιγμιότυπων σελίδων διαδικτύου

Με σκοπό τη διατήρηση της λειτουργικότητας σε όσο το δυνατό μεγαλύτερο εύρος λογισμικών/μηχανημάτων έχει επιλεγθεί η εφαρμογή να κατακερματιστεί σε Docker Containers έτσι ώστε να επωφεληθεί από την τυποποίηση του περιβάλλοντος στο οποίο ο τελικός χρήστης θα εγκαταστήσει την εφαρμογή. Συγκεκριμένα η εφαρμογή αποτελείται από δύο (2) απαραίτητα προς την εφαρμογή container για την ομαλή λειτουργία της και δύο (2) ακόμα προαιρετικά container που παρέχονται προς διευκόλυνση της χρήσης της.

Απαραίτητα Container:**1. Crawler Backend (Flask Rest-API, Python crawler functions)**

Περιλαμβάνει το Flask RestAPI που λαμβάνει τα οποιαδήποτε αιτήματα για ενέργειες του crawl και όλες τις συναρτήσεις του crawler και είναι γραμμένο σε python.

2. Crawler Database (MariaDB)

Container που παρέχεται από τους δημιουργούς της MariaDB και το οποίο αρχικοποιείται από το Crawler Backend κατά την εκκίνηση του.

Προαιρετικά Container:**1. Crawler Frontend Web/Android/iOS (Flutter)**

Γραμμένο σε Dart και χρησιμοποιώντας το Flutter Framework για να παράγει με κυρίως ενιαίο κομμάτι κώδικα σελίδες για Web / Android / iOS.

2. phpMyAdmin Database web interface

Container που παρέχεται από τους δημιουργούς του phpMyAdmin το οποίο δίνει στον χρήστη πρόσβαση σε διαχειριστικό περιβάλλον της βάσης μέσω ηλεκτρονικής σελίδας.

Τεχνολογίες

Παρακάτω αναλύονται οι τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής και οι λόγοι για τους οποίους επιλέχθηκαν.

Docker

Το Docker είναι μια open-source πλατφόρμα για ανάπτυξη και εκτέλεση εφαρμογών. Δίνει τη δυνατότητα για διαχωρισμό της εφαρμογής από τις υποδομές που αυτή χρειάζεται, ώστε να είναι δυνατή η εγκατάσταση της εφαρμογής σε διαφορετικού είδους μηχανήματα με ελάχιστες αλλαγές. Αυτό επιτυγχάνεται με την ενθυλάκωση της εφαρμογής σε ένα container το οποίο για την εφαρμογή πλέον παρέχει τον ρόλο του Operating System.

Χρησιμοποιώντας Containers κατά την υλοποίηση αλλά και καθώς την φόρτωση στο παραγωγικό περιβάλλον της εφαρμογής αποφεύγονται πιθανές ασυμβατότητες που θα μπορούσαν να προκύψουν από διαφορετικά συστήματα κατά τη διαδικασία υλοποίησης και τελικά με το σύστημα το οποίο υπάρχει στα παραγωγικά μηχανήματα.

Επιπρόσθετα παρέχει ένα ακόμα επίπεδο ασφαλείας καθώς η εφαρμογή μπορούμε να πούμε ότι είναι στεγανά κλεισμένη από τις υπόλοιπες τυχόν εγκατεστημένες εφαρμογές στο μηχάνημα καθώς έχει πρόσβαση μόνο σε ότι περιλαμβάνει το περιβάλλον της και ενδεχομένως στο δίκτυο βάση κανόνων που έχουν γραφτεί κατά τη διαδικασία υλοποίησης της. Είναι χρήσιμο να αναφερθεί πως δεν είναι η λύση για όλα τα πιθανά προβλήματα ασφαλείας που μπορεί να προκύψουν σε μια εφαρμογή και είναι απαραίτητο να ακολουθηθούν οι αρχές σωστού προγραμματισμού όπως θα γινόταν αν δεν υλοποιούνταν η εφαρμογή μέσα σε containers.

Τέλος με τη χρήση container είναι αρκετά πιο εύκολη η διαδικασία εγκατάστασης σε μια νέα υποδομή καθώς καθίσταται εφικτό να γίνει scale up σε

περίπτωση που αυτό χρειάζεται αν χρησιμοποιηθεί παράλληλα με τεχνολογίες όπως το Kubernetes. Αλλά και η διαδικασία λήψης backup μπορεί να γίνει σε επίπεδο container με έτοιμα εργαλεία που παρέχονται από την πλατφόρμα.

Σημειώνεται πως για τη χρήση του docker απαραίτητη είναι η εγκατάσταση του στον Server τον οποίο επιθυμούμε να φιλοξενήσει τα απαραίτητα container της εφαρμογής, η διαδικασία είναι αρκετά απλή και περιγράφεται εκτενώς στην σελίδα της εφαρμογής.

Flask

Για το κομμάτι του backend επιλέχθηκε να χρησιμοποιηθεί το Flask, ένα από τα πιο γνωστά project ανοιχτού κώδικα python Frameworks για υλοποίηση web εφαρμογών. Είναι ένα πιο απλό framework σε σχέση με το Django αλλά εξίσου δυνατό και εύκολο στην κατανόηση χάρη στο πόσο “Pythonic” μπορεί να χαρακτηριστεί.

Με το συγκεκριμένο framework δημιουργήθηκε το Rest-API της εφαρμογής το οποίο διαχειρίζεται τα τυχόν crawl tasks και αντλεί στατιστικά στοιχεία από τη βάση της εφαρμογής. Ακόμα παράγει δυναμικά δημιουργούμενες σελίδες έπειτα από αίτημα του χρήστη, για παράδειγμα σελίδες όπως αυτή με τις διαφορές μεταξύ δύο διαφορετικών archive.

MariaDB

Για τα απαραίτητα στατιστικά αλλά και πληροφορίες που χρειάζεται να αποθηκεύονται κατά τη χρήση της εφαρμογής επιλέχθηκε να χρησιμοποιηθεί ένα ακόμα project ανοιχτού κώδικα και συγκεκριμένα το MariaDB. Fork της σχεσιακής βάσης MySQL και με άτομα της αρχικής ομάδας υλοποίησης της MySQL, θεωρείται από τις καλύτερα διατηρημένες solutions αλλά και με έξτρα βελτιώσεις πάνω στο αρχικό project τόσο στο κομμάτι του performance που

επιτεύχθηκε με την αντικατάσταση των παλαιών engine με νέα τα οποία είναι αρκετά πιο σταθερά στην λειτουργία τους αλλά και πιο αποτελεσματικά στις ενέργειες που μπορεί να ζητήσει ο χρήστης.

Flutter

Ένα από τα ανερχόμενα στον τομέα των υβριδικών Framework για ανάπτυξη εφαρμογών για πολλές πλατφόρμες γράφοντας κυρίως σε μια γλώσσα από την Google. Με τις τελευταίες stable εκδόσεις παρέχει την δυνατότητα για δημιουργεί εφαρμογών σε κινητές συσκευές (Android/iOS), Web αλλά και Desktop Applications (Windows/Linux).

Το συγκεκριμένο Framework έχει το πλέον μεγαλύτερο εύρος συσκευών για το οποίο μπορεί να παράξει εκτελέσιμη εφαρμογή χωρίς να χρειάζεται διαφορετικές υλοποιήσεις για την κάθε μια από αυτές. Ακόμα έχει δοθεί μεγάλη βάση στο engine το οποίο βασίζεται, με αποτέλεσμα να είναι αρκετά αποτελεσματικό και να παρέχει λειτουργία συγκρίσιμη με κώδικα που έχει γραφτεί σε native γλώσσα της κάθε πλατφόρμας, τόσο σε θέματα ταχύτητας αλλά και ανταπόκρισης σε input του χρήστη. Το engine που χρησιμοποιείται έχει επιλεγεί να μεταφράζει τον κώδικα που του δίνεται κατευθείαν σε “pixel” χωρίς να τον “μεταφράζει” πρώτα σε native component της κάθε πλατφόρμας, με αυτόν τον τρόπο αν και χάνεται το native feel, σε μερικές περιπτώσεις επιτυγχάνεται να είναι ακριβώς το ίδιο αποτέλεσμα σε όλες τις πλατφόρμες. Επιπρόσθετα με τη δυνατότητα για hot reload του κώδικα που γράφει ο προγραμματιστής, δηλαδή τη δυνατότητα για εμφάνιση των αλλαγών στην εφαρμογή έπειτα από νέα υλοποίηση χωρίς να χρειάζεται compile και build της εφαρμογής ξανά, επιτρέπει για πολύ πιο γρήγορους χρόνους υλοποίησης σε σχέση με χρήση native γλωσσών.

Τέλος, από τα μεγαλύτερα προτερήματα της υλοποίησης του frontend μιας εφαρμογής χρησιμοποιώντας το Flutter είναι τόσο η παροχή έτοιμων κομματιών

κώδικα για διάφορα UI στοιχεία με αποτέλεσμα να επιταχύνει κατά πολύ τον ρυθμό ανάπτυξης αλλά και η ταχύτητα με την οποία γίνεται η υλοποίηση ταυτόχρονα για πολλές πλατφόρμες, χωρίς να χρειάζεται γνώση των διαφορετικών γλωσσών προγραμματισμού που χρησιμοποιούν οι πλατφόρμες αυτές σε βάθος.

phpMyAdmin

Για την πιο εύκολη διαχείριση και παρακολούθηση των αλλαγών που γίνονται στη βάση της εφαρμογής μπορεί να χρησιμοποιηθεί το προαιρετικό container που περιέχεται στο docker-compose phpMyAdmin, το συγκεκριμένο εργαλείο ανεβάζει μια ιστοσελίδα από την οποία μπορεί να γίνει διαχείριση βάσεων SQL ή MariaDB χωρίς ανάγκη για χρήση περαιτέρω εργαλείων. Είναι γραμμένη σε PHP και δίνει στους διαχειριστές της βάσης τη δυνατότητα για να καλούν οποιαδήποτε SQL statement χρειάζονται ή και να ολοκληρώνουν μέσα από το γραφικό αντίστοιχες ενέργειες που μπορεί να χρειάζονται.

Από τις πιο χρήσιμες λειτουργίες του είναι η δυνατότητα για λήψη στιγμιότυπων της βάσης με σκοπό την πιθανή επαναφορά αν αυτό είναι αναγκαίο μελλοντικά αλλά και η δυνατότητα για δημιουργία διαφόρων διαγραμμάτων από τα δεδομένα που είναι αποθηκευμένα στην βάση.

Εργαλεία

Παρακάτω αναγράφονται μερικά από τα σημαντικότερα εργαλεία που χρησιμοποιήθηκαν κατά την υλοποίηση της συγκεκριμένης εφαρμογής και θεωρούνται χρήσιμα ακόμα και κατά τη χρήση της χωρίς βέβαια αυτά να είναι αναγκαία για την λειτουργία της.

Postman

Από τα πιο πολύ χρησιμοποιημένα εργαλεία τόσο για τη δημιουργία αλλά και για τη χρήση οποιοδήποτε API. Ακόμα περιέχει χρήσιμα εργαλεία με σκοπό να εξομαλύνει και επιταχύνει την διαδικασία ανάπτυξης των πιθανών API από ομάδες προγραμματιστών αλλά και να δώσει πρόσβαση σε αυτά για όλα τα στάδια του API lifecycle.

DBeaver

Πολύ γνωστό έργο ανοιχτού κώδικα για διαχείριση SQL βάσεων δεδομένων. Παρέχει τη δυνατότητα για σύνδεση στα περισσότερα είδη SQL βάσεων από το ίδιο περιβάλλον χρήσης. Ακόμα έχει ενσωματωμένο SQL editor και δυνατότητα για αυτοματοποίηση πιθανών query για γρήγορη χρήση τους.

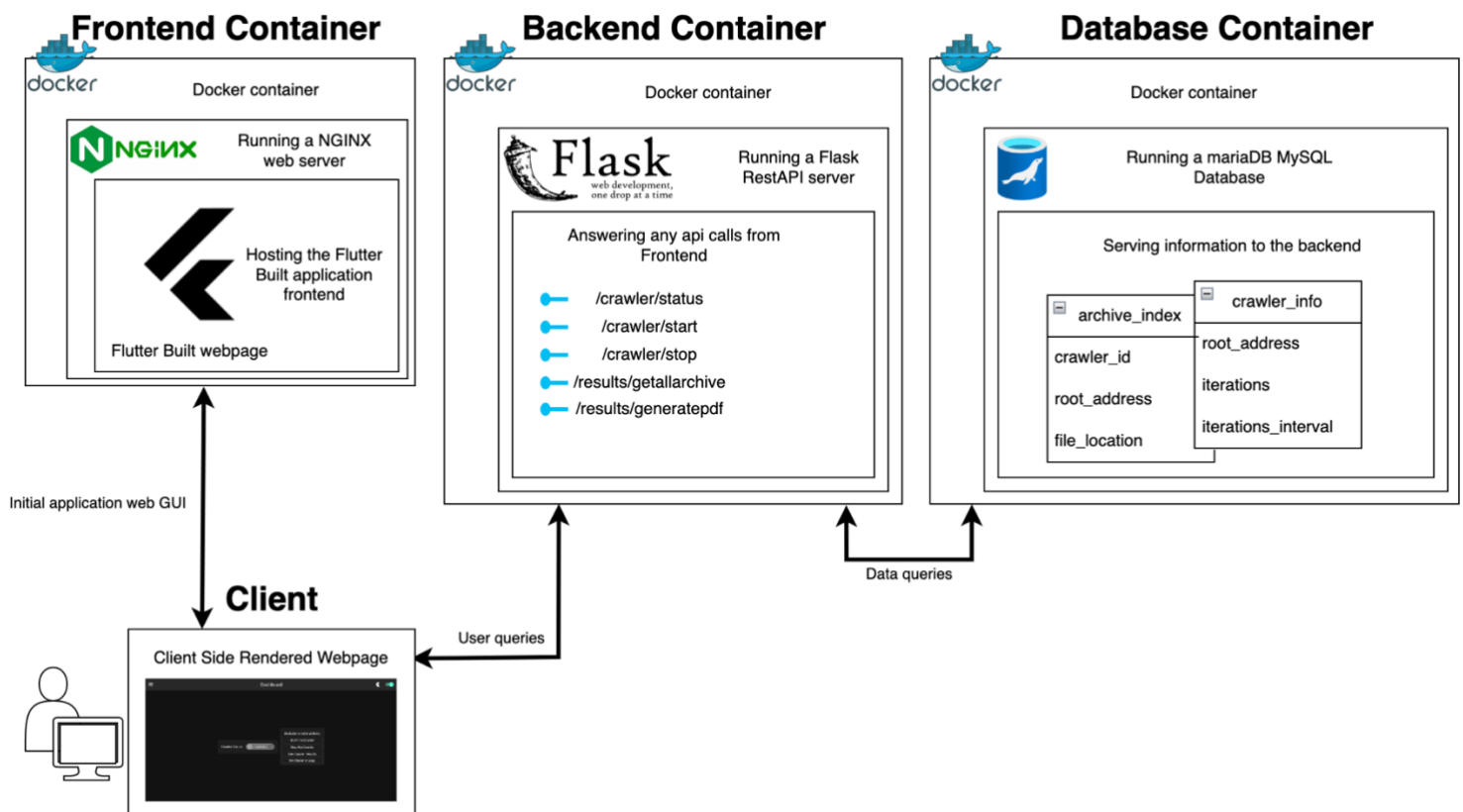
Visual Studio Code

Έργο ανοιχτού κώδικα για επεξεργασία πηγαίου κώδικα εφαρμογών. Αρκετά ελαφρύ και γρήγορο στην εκτέλεση του και με τη δυνατότητα για παραμετροποίηση του ώστε να πληροί τις προϋποθέσεις ώστε να χρησιμοποιηθεί για γραφή εφαρμογών στις περισσότερες αν όχι όλες τις σύγχρονες γλώσσες προγραμματισμού.

Αρχιτεκτονική

Παρακάτω φαίνεται η αρχιτεκτονική που έχει επιλεχθεί να χρησιμοποιηθεί για την εφαρμογή έτσι ώστε να είναι δυνατό στον χρήστη να την εγκαταστήσει σε οποιοδήποτε σύστημα επιλέξει και χάρη του docker να έχει τα ίδια αποτελέσματα με ελάχιστες παραμετροποιήσεις όπου αυτές είναι απαραίτητες.

Είναι χρήσιμο να σημειωθεί ότι η σελίδα της εφαρμογής γίνεται render στη συσκευή του χρήστη και οποιαδήποτε αιτήματα γίνονται από αυτόν στέλνονται κατευθείαν στο Backend server και όχι πρώτα στο server όπου ‘ζει’ το Frontend. Σαν αποτέλεσμα το Frontend container δίνει μόνο την αρχική σελίδα χρήστη η οποία μετά με αιτήματα προς το backend ενημερώνεται και δίνει στον χρήστη πλέον την κατάσταση της εφαρμογής.



Εγκατάσταση

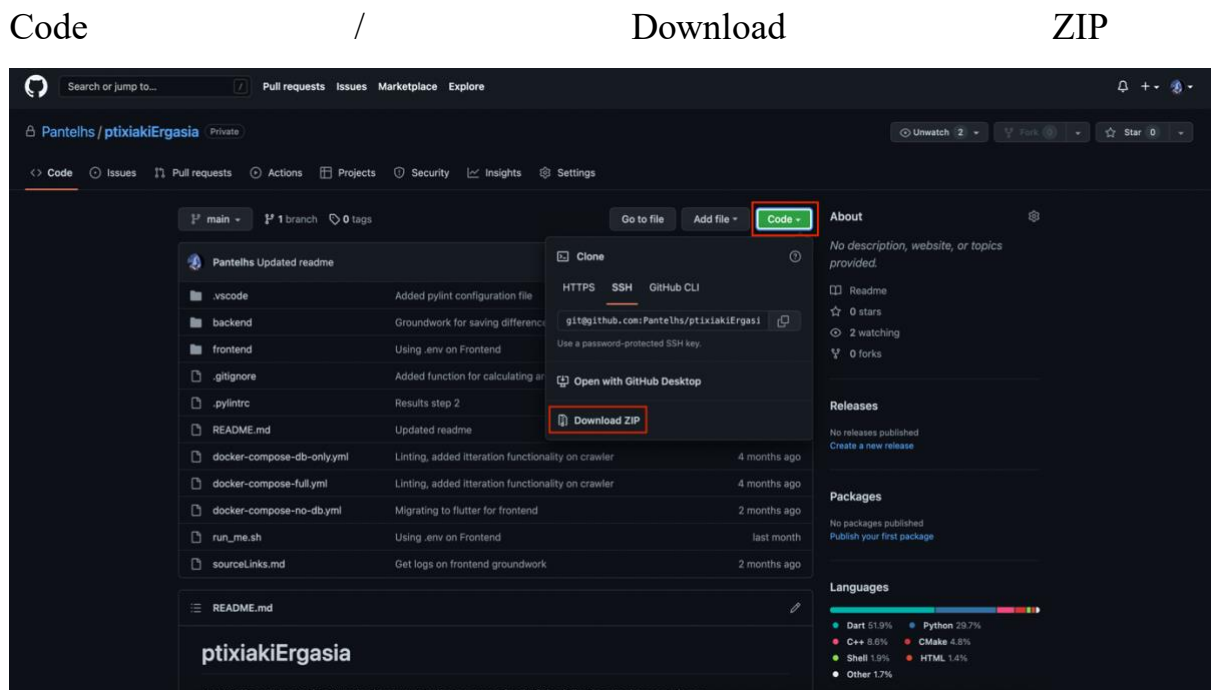
Η εφαρμογή έχει γραφεί με σκοπό να βασιστεί σε Docker containers αν και αυτό δεν είναι άκρως αναγκαίο αν υπάρχει η απαραίτητη βάση για την εφαρμογή εγκατεστημένη. Παρακάτω αναγράφονται τα βήματα για την εγκατάσταση της εφαρμογής τόσο χρησιμοποιώντας το συμπεριλαμβανόμενο docker-compose άλλα και σαν stand alone εγκατάσταση σε Windows και Linux OS.

Windows System

1. Απαραίτητο για την εγκατάσταση της εφαρμογής είναι η λήψη των αρχείων της, αυτό είναι δυνατό είτε χρησιμοποιώντας το git ή κατεβάζοντάς τα από τη σελίδα του GitHub

1.1. Κατέβασμα των αρχείων της εφαρμογής από το GitHub σαν ZIP

1.2. Για να γίνει λήψη από την σελίδα του GitHub από οποιοδήποτε σύγχρονο browser ανοίγουμε την σελίδα του repository του project και επιλέγουμε



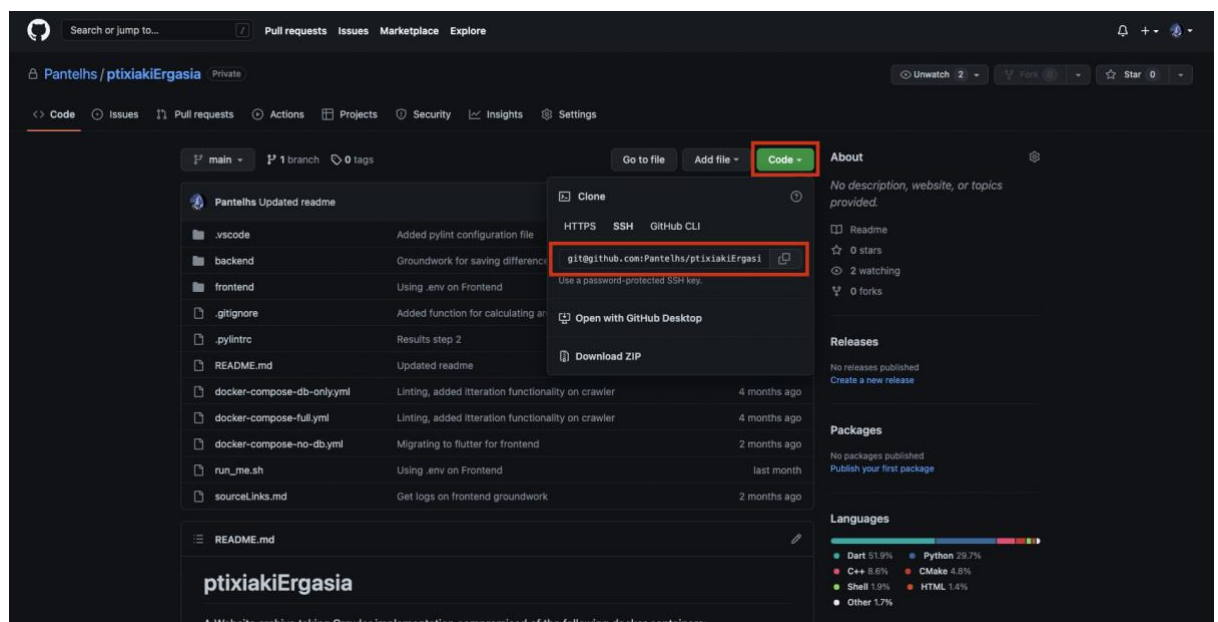
1.2.1. Μεταφέρουμε το αρχείο Zip που κατέβηκε στο προηγούμενο βήμα σε οποιοδήποτε σημείο θέλουμε να ζουν τα αρχεία της εφαρμογής και πατώντας δεξί κλικ επιλέγουμε αποσυμπίεση.

1.3. Κατέβασμα των αρχείων της εφαρμογής χρησιμοποιώντας το git από το GitHub

1.3.1. Για να είναι δυνατό το κατέβασμα των αρχείων της εφαρμογής χρησιμοποιώντας το git είναι πρώτα απαραίτητη η εγκατάσταση του. Αρχικά κατεβάζουμε το εκτελέσιμο από το [Git](#) και το εγκαθιστούμε.

1.3.2. Έπειτα ανοίγουμε ένα παράθυρο PowerShell και κάνουμε navigate στον φάκελο στον οποίο θέλουμε να αποθηκευτούν τα αρχεία της εφαρμογής.

1.4. Ανοίγουμε το repository του project στο GitHub και αντιγράφουμε το link του project που βρίσκεται κάτω από το κουμπί Code.



1.4.1. Αφού έχουμε αλλάξει στον φάκελο που θέλουμε, κάνουμε επικόλληση την εντολή που αντιγράφηκε στο προηγούμενο βήμα.

1.4.2. Αφού εκτελεστεί η εντολή τα αρχεία της εφαρμογής πλέον έχουν κατέβει στον φάκελο από όπου εκτελέστηκε.

2. Το Frontend της εφαρμογής είναι γραμμένο σε Flutter και για να δημιουργηθεί το εκτελέσιμο είναι απαραίτητη η εγκατάσταση του Flutter Development Package.

2.1. Κατεβάζουμε από [Flutter Download](#) το installation bundle και το κάνουμε αποσυμπίεση σε όποιο σημείο θέλουμε να γίνει τελικά η εγκατάσταση τους.

2.2. Ενημερώνουμε το Path των Windows προσθέτοντας το πλήρες path για το flutter/bin που κατέβηκε στο προηγούμενο βήμα.

2.3. Στην περίπτωση που θέλουμε να δημιουργηθούν εκτελέσιμα για κινητά είναι απαραίτητο να ακολουθηθούν και τα υπόλοιπα βήματα στη σελίδα από όπου κατέβηκαν τα αρχεία έτσι ώστε να τηρούνται οι προϋποθέσεις για Android applications development.

3. Για να λειτουργήσει η εφαρμογή χρειάζεται απαραίτητα μια βάση δεδομένων και το backend της ώστε να μπορεί να ολοκληρώσει αιτήματα. Ανεβάζοντας την εφαρμογή χρησιμοποιώντας το docker-compose είναι ο πιο εύκολος τρόπος για να πληρούνται τα παραπάνω, στην περίπτωση που δεν θέλουμε να τρέξει η εφαρμογή με Docker container, η διαδικασία αναλύεται παρακάτω.

3.1. Εγκατάσταση χρησιμοποιώντας Docker

3.1.1. Στην περίπτωση που θέλουμε η εφαρμογή να τρέχει χρησιμοποιώντας Docker container αρχικά πρέπει να εγκαταστήσουμε το [Docker Desktop](#) κατεβάζοντας το executable και ακολουθώντας τα βήματα που αναγράφονται στη σελίδα για την σωστή εγκατάσταση του.

3.1.2. Στον φάκελο με τα αρχεία της εφαρμογής δημιουργούμε ένα αρχείο με όνομα .env , έπειτα ανοίγουμε το αρχείο αυτό με οποιοδήποτε text editor και εισάγουμε τα παρακάτω:

```
WEBPAGE_URL="https://www.apple.com" # The site to crawl over
DIFF_THRESHOLD="0.90" # The difference threshold between 0 and 1
MARIADB_IP="mariadb" # The IP of mariaDB, 'mariadb' if using docker-compose
MARIADB_USER="DB_user" # User to be used by crawler
MARIADB_PASSWORD="AbcD34FER$%1" # Password for MARIADB_USER
MARIADB_PORT="3306" # The mariaDB listening port
MARIADB_DATABASE="crawlerDB" # The Database name to be used by crawler
MARIADB_ROOT_PASSWORD="changeMe" # The root user's password
```

3.1.3. Ανοίγουμε PowerShell και κάνουμε navigate στον φάκελο όπου προηγουμένως κατεβάσαμε τα αρχεία της εφαρμογής και τρέχουμε το περιεχόμενο `run_me.sh` script το οποίο είναι υπεύθυνο για την αρχικοποίηση όλων των container αλλά και για την παραγωγή των εκτελέσιμων για το frontend της εφαρμογής. Για την εκτέλεση του εισάγουμε στο PowerShell `./run_me.sh`

3.2.Εγκατάσταση Χρησιμοποιώντας υπάρχουσα Βάση Δεδομένων

3.2.1. Απαραίτητο είναι να υπάρχει εγκατεστημένη Python 3 στον υπολογιστή, η εγκατάσταση της μπορεί να γίνει εύκολα κατεβάζοντας τον installer από την σελίδα της [Python](#).

3.2.2. Στον φάκελο με τα αρχεία της εφαρμογής δημιουργούμε ένα αρχείο με όνομα `.env` , έπειτα ανοίγουμε το αρχείο αυτό με οποιοδήποτε text editor και εισάγουμε τα παρακάτω και συγκεκριμένα με προσοχή τα στοιχεία της MariaDB :

```
WEBPAGE_URL="https://www.apple.com" # The site to crawl over
DIFF_THRESHOLD="0.90" # The difference threshold between 0 and 1
MARIADB_IP="mariadb" # The IP of mariaDB, 'mariadb' if using docker-compose
MARIADB_USER="DB_user" # User to be used by crawler
MARIADB_PASSWORD="AbcD34FER$%1" # Password for MARIADB_USER
MARIADB_PORT="3306" # The mariaDB listening port
MARIADB_DATABASE="crawlerDB" # The Database name to be used by crawler
MARIADB_ROOT_PASSWORD="changeMe" # The root user's password
```

3.2.3. Για να ξεκινήσει το Backend της εφαρμογής ανοίγουμε PowerShell αλλάζουμε φάκελο στον φάκελο με τα αρχεία της εφαρμογής που κατεβάσαμε στο προηγούμενο βήμα.

3.2.4. Αλλάζουμε φάκελο στο backend και εκτελούμε
`python3 ./backend.py`

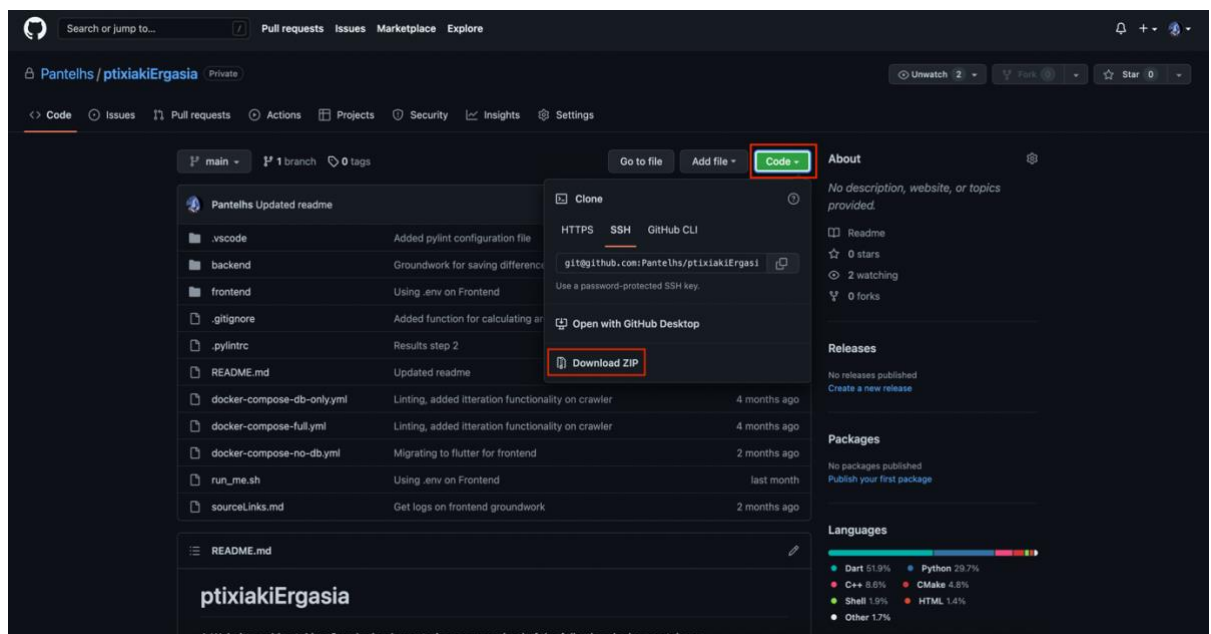
4. Αφού γίνουν τα παραπάνω η εφαρμογή είναι προσβάσιμη πλέον από <http://<server's ip>:8080>

Linux System

1. Απαραίτητο για την εγκατάσταση της εφαρμογής είναι η λήψη των αρχείων της, αυτό είναι δυνατό είτε χρησιμοποιώντας το git ή κατεβάζοντας τα από τη σελίδα του GitHub.

1.1. Κατέβασμα των αρχείων της εφαρμογής από το GitHub σαν ZIP

- 1.1.1. Για να γίνει λήψη από την σελίδα του GitHub από οποιοδήποτε σύγχρονο browser ανοίγουμε την σελίδα του repository του project και επιλέγουμε Code / Download ZIP



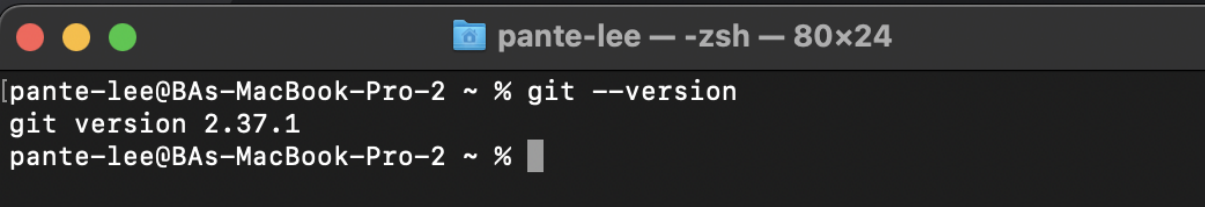
- 1.1.2. Μεταφέρουμε το αρχείο Zip που κατέβηκε στο προηγούμενο βήμα σε οποιοδήποτε σημείο θέλουμε να ζουν τα αρχεία της εφαρμογής.

- 1.1.3. Για να μπορέσουμε να κάνουμε unzip το κατεβασμένο αρχείο είναι απαραίτητο να εγκατασταθεί η αντίστοιχη λειτουργία γράφοντας την παρακάτω εντολή στο terminal `sudo apt-get install zip unzip`

- 1.1.4. Έπειτα ανοίγοντας ένα terminal στον φάκελο όπου έχουμε αποθηκεύσει το zip αρχείο που κατεβάσαμε σε προηγούμενο βήμα τρέχουμε `unzip ptixiakiErgasia-main.zip`

1.2. Κατέβασμα των αρχείων της εφαρμογής χρησιμοποιώντας το git από το GitHub

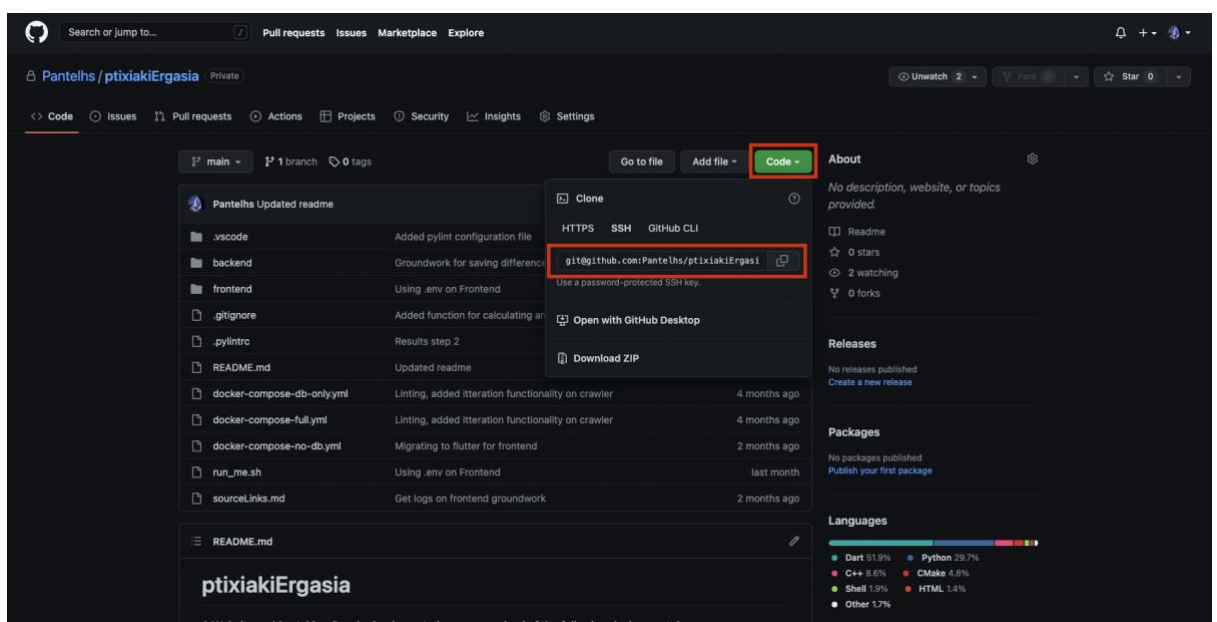
1.2.1. Για να είναι δυνατό το κατέβασμα των αρχείων της εφαρμογής χρησιμοποιώντας το git είναι πρώτα απαραίτητος ο έλεγχος αν αυτό είναι προ εγκατεστημένο καθώς στα περισσότερα Linux distros είναι μέσα στα βασικά πακέτα που είναι προεπιλεγμένα. Αρχικά γράφουμε σε ένα terminal `git --version`



```
pante-lee@BAs-MacBook-Pro-2 ~ % git --version
git version 2.37.1
pante-lee@BAs-MacBook-Pro-2 ~ %
```

Στην περίπτωση που μας επιστρέψει έκδοση πάει να πει ότι το git είναι εγκατεστημένο και μπορούμε να συνεχίσουμε στο επόμενο βήμα, διαφορετικά χρειάζεται να γίνει εγκατάσταση του git γράφοντας στο terminal `sudo apt install git-all`

1.3. Ανοίγουμε το repository του project στο GitHub και αντιγράφουμε το link του project που βρίσκεται κάτω από το κουμπί Code.



- 1.3.1. Έπειτα ανοίγουμε ένα νέο terminal στον φάκελο όπου θέλουμε να υπάρχουν τα αρχεία της εφαρμογής και κάνουμε επικόλληση την εντολή που έγινε αντιγραφή στο προηγούμενο βήμα.
 - 1.3.2. Αφού εκτελεστεί η εντολή τα αρχεία της εφαρμογής πλέον έχουν κατέβει στον φάκελο από όπου εκτελέστηκε.
2. Το Frontend της εφαρμογής είναι γραμμένο σε Flutter και για να δημιουργηθεί το εκτελέσιμο είναι απαραίτητη η εγκατάσταση του Flutter Development Package.
 - 2.1.Ο ευκολότερος τρόπος για την εγκατάσταση του είναι χρησιμοποιώντας το snapd ή το brew στην περίπτωση που είμαστε από mac μηχανήμα γράφοντας σε ένα terminal τα εξής:
Linux: `sudo snap install flutter --classic`
Mac: `brew install --cask flutter`
3. Για να λειτουργήσει η εφαρμογή χρειάζεται απαραίτητα μια βάση δεδομένων και το backend της ενεργό ώστε να μπορεί να ολοκληρώσει αιτήματα. Ανεβάζοντας την εφαρμογή χρησιμοποιώντας το docker-compose είναι ο πιο εύκολος τρόπος για να πληρούνται τα παραπάνω, στην περίπτωση που δεν θέλουμε να τρέξει η εφαρμογή με Docker container η διαδικασία αναλύεται στη συνέχεια.
 - 3.1.Εγκατάσταση χρησιμοποιώντας Docker
 - 3.1.1. Στην περίπτωση που θέλουμε η εφαρμογή να τρέχει χρησιμοποιώντας Docker container αρχικά πρέπει να εγκαταστήσουμε το Docker Desktop ακολουθώντας τα βήματα που αναγράφονται στην σελίδα τους.
 - 3.1.2. Στον φάκελο με τα αρχεία της εφαρμογής δημιουργούμε ένα αρχείο με όνομα .env , έπειτα ανοίγουμε το αρχείο αυτό με οποιοδήποτε text editor και εισάγουμε τα παρακάτω:

```
WEBPAGE_URL="https://www.apple.com" # The site to crawl over

DIFF_THRESHOLD="0.90" # The difference threshold between 0 and 1

MARIADB_IP="mariadb" # The IP of mariaDB, 'mariadb' if using docker-compose

MARIADB_USER="DB_user" # User to be used by crawler

MARIADB_PASSWORD="AbcD34FER$%1" # Password for MARIADB_USER

MARIADB_PORT="3306" # The mariaDB listening port

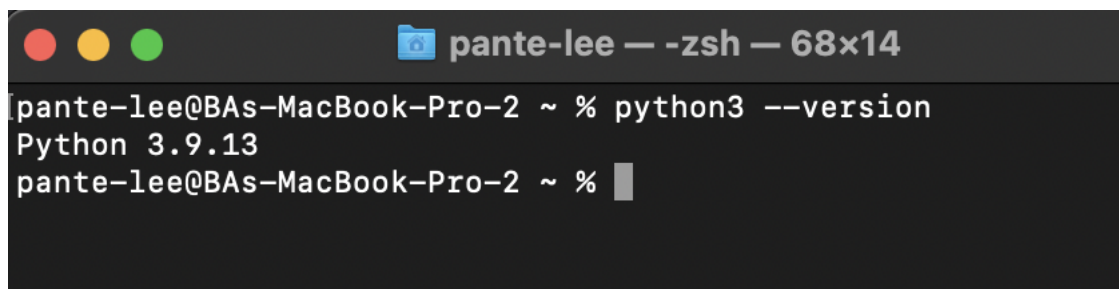
MARIADB_DATABASE="crawlerDB" # The Database name to be used by crawler

MARIADB_ROOT_PASSWORD="changeMe" # The root user's password
```

4.1.1. Ανοίγουμε ένα νέο terminal και κάνουμε navigate στον φάκελο όπου προηγουμένως κατεβάσαμε τα αρχεία της εφαρμογής και τρέχουμε το περιεχόμενο `run_me.sh` script το οποίο είναι υπεύθυνο για την αρχικοποίηση όλων των container αλλά και για την παραγωγή των εκτελέσιμων για το frontend της εφαρμογής. Για την εκτέλεση του εισάγουμε στο terminal `./run_me.sh`

3.2.Εγκατάσταση Χρησιμοποιώντας υπάρχουσα Βάση Δεδομένων

3.2.1. Απαραίτητο είναι να γίνει έλεγχος αν υπάρχει εγκατεστημένη Python 3 στον υπολογιστή, στα περισσότερα Linux distros είναι μέσα στα βασικά πακέτα που είναι προεπιλεγμένα. Για να ελέγξουμε αν είναι εγκατεστημένη γράφουμε σε ένα terminal `python3 --version`



```
pante-lee@BAs-MacBook-Pro-2 ~ % python3 --version
Python 3.9.13
pante-lee@BAs-MacBook-Pro-2 ~ %
```

Στην περίπτωση που μας επιστρέψει έκδοση πάει να πει ότι η python3 είναι εγκατεστημένη και μπορούμε να συνεχίσουμε στο επόμενο βήμα, διαφορετικά χρειάζεται να γίνει εγκατάσταση της γράφοντας στο terminal

Linux:

```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt-get update
sudo apt-get install python3.8
```

Mac:

```
Brew install python
```

3.2.2. Στον φάκελο με τα αρχεία της εφαρμογής δημιουργούμε ένα αρχείο με όνομα .env , έπειτα ανοίγουμε το αρχείο αυτό με οποιοδήποτε text editor και εισάγουμε τα παρακάτω και συγκεκριμένα με προσοχή τα στοιχεία της MariaDB :

```
WEBPAGE_URL="https://www.apple.com" # The site to crawl over
DIFF_THRESHOLD="0.90" # The difference threshold between 0 and 1
MARIADB_IP="mariadb" # The IP of mariaDB, 'mariadb' if using docker-compose
MARIADB_USER="DB_user" # User to be used by crawler
MARIADB_PASSWORD="AbcD34FER$%1" # Password for MARIADB_USER
MARIADB_PORT="3306" # The mariaDB listening port
MARIADB_DATABASE="crawlerDB" # The Database name to be used by crawler
MARIADB_ROOT_PASSWORD="changeMe" # The root user's password
```

3.2.3. Για να ξεκινήσει το Backend της εφαρμογής ανοίγουμε ξανά ένα terminal μέσα στον φάκελο με τα αρχεία της εφαρμογής στον φάκελο με όνομα backend και έπειτα εκτελούμε `python3 ./backend.py`

4. Αφού γίνουν τα παραπάνω η εφαρμογή είναι προσβάσιμη πλέον από <http://<server's ip>:8080>

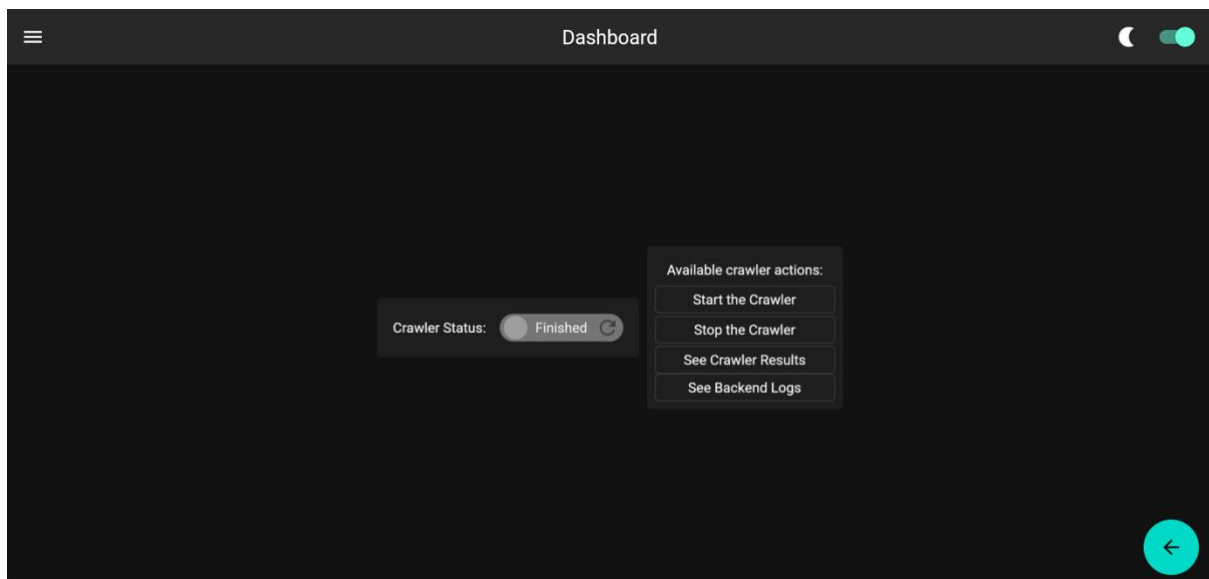
Αφού ολοκληρωθούν τα παραπάνω βήματα η εφαρμογή πλέον είναι έτοιμη για να ολοκληρώσει αιτήματα από τον χρήστη δεδομένου ότι βρίσκεται στο ίδιο δίκτυο ή διαφορετικά, αν έχουν γίνει οι απαραίτητες ρυθμίσεις στο δίκτυο του χρήστη, και από το εξωτερικό δίκτυο αν και αυτό δεν προτείνεται, καθώς δεν έχει προστεθεί στην εφαρμογή λογική αυθεντικοποίησης.

Εικόνες εφαρμογής και επεξήγηση κώδικα

Παρακάτω παρατίθενται εικόνες από το περιβάλλον χρήσης της εφαρμογής αλλά και επεξήγηση μερικών κομματιών κώδικα του backend που χρησιμοποιούνται σε κάθε μια από αυτές.

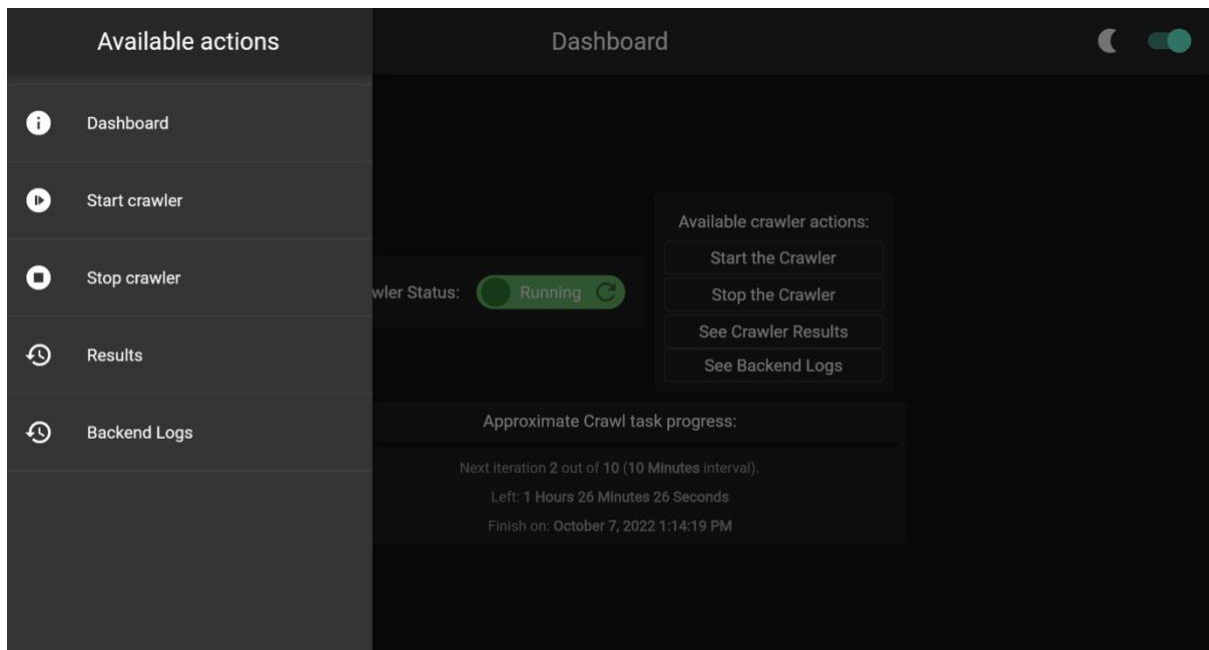
Αρχική / Dashboard σελίδα

Σαν αρχική σελίδα του περιβάλλοντος χρήσης της εφαρμογής έχει επιλεγθεί να εμφανίζεται η κατάσταση του crawler και κουμπιά με τα οποία ο χρήσης μπορεί να συνεχίσει σε άλλες σελίδες από τις οποίες γίνονται οι υπόλοιπες δυνατές ενέργειες. Ακόμα σε όλες τις σελίδες δίνεται η δυνατότητα για εναλλαγή μεταξύ light mode και dark mode.



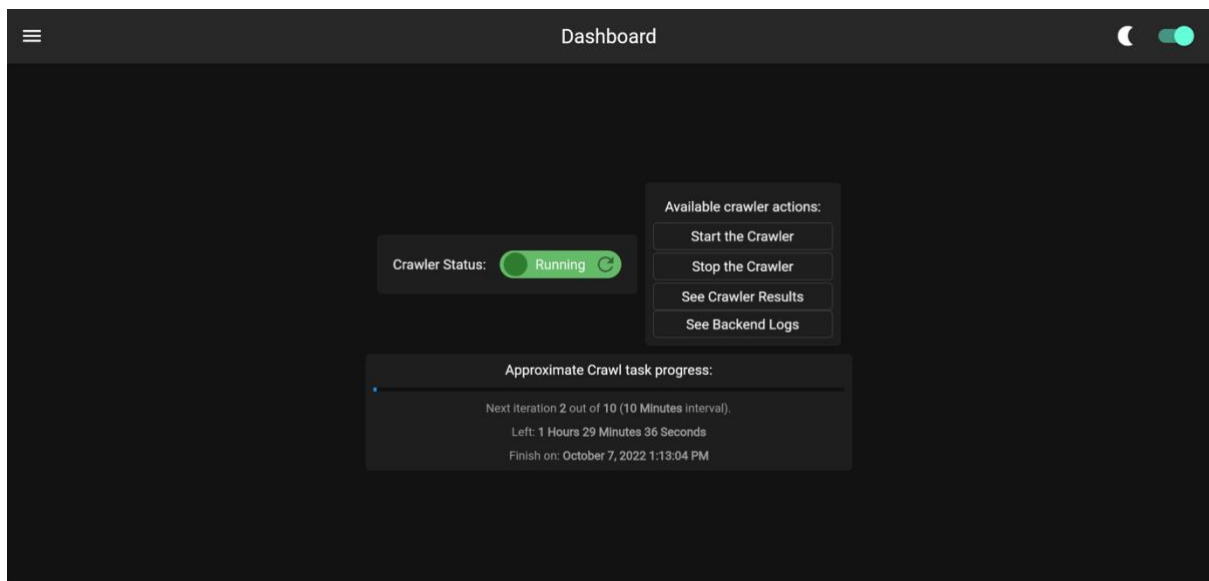
1 Dashboard

Σε όλες τις σελίδες της εφαρμογής υπάρχει ένα συρόμενο από τα αριστερά μενού με συντομεύσεις για όλες τις σελίδες που περιέχουν τις ενέργειες της εφαρμογής. Και αντίστοιχα στο κάτω δεξιά μέρος της σελίδας κουμπί με το οποίο πηγαίνει τον χρήστη στην αμέσως προηγούμενη σελίδα όποτε αυτό είναι δυνατό.

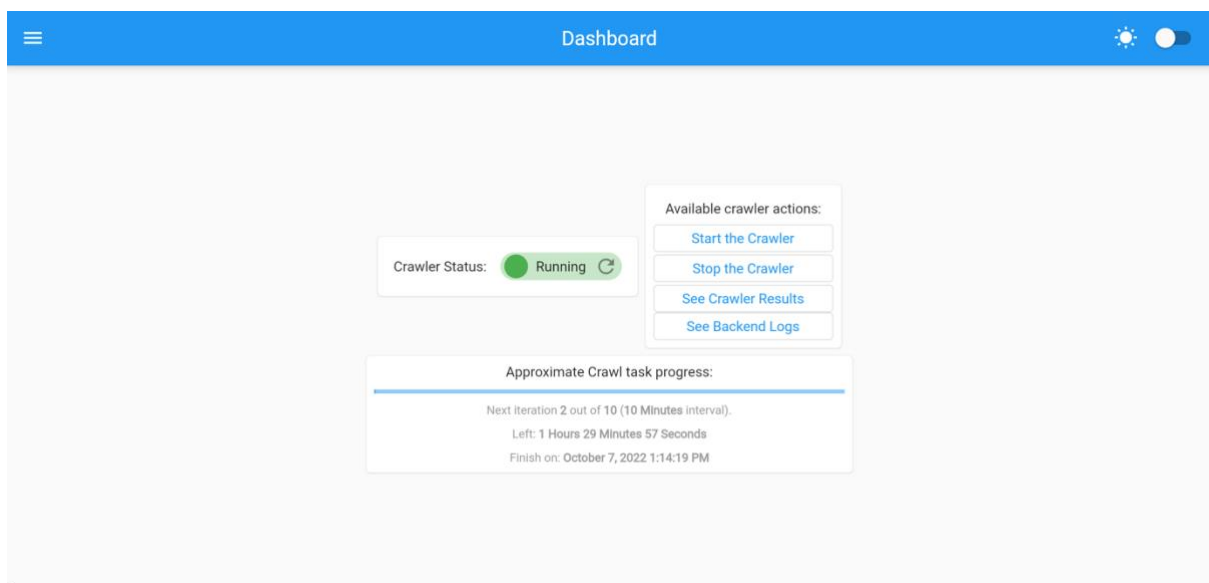


2 Side panel available actions

Στην περίπτωση που υπάρχει μια ενεργή διαδικασία για λήψη στιγμιότυπων μια σελίδας αυτό είναι εμφανές στην αρχική σελίδα από το Crawler Status το οποίο είναι πράσινο αλλά και από το progress bar το οποίο αναγράφει και κάποια στατιστικά όπως σε ποιο βήμα βρίσκεται αυτή την στιγμή αλλά και τον αναμενόμενο χρόνο ολοκλήρωσης της διαδικασίας. Είναι χρήσιμο να σημειωθεί εδώ πως ο χρόνος που αναγράφεται είναι εκτίμηση με παραδοχή ότι κάθε βήμα κάνει συγκεκριμένο χρόνο για να ολοκληρωθεί με αποτέλεσμα να είναι πιθανό η εκτίμηση να μην είναι πάντα σωστή.



3 Dashboard dark mode με ενεργό crawl task



4 Dashboard light mode με ενεργό crawl task

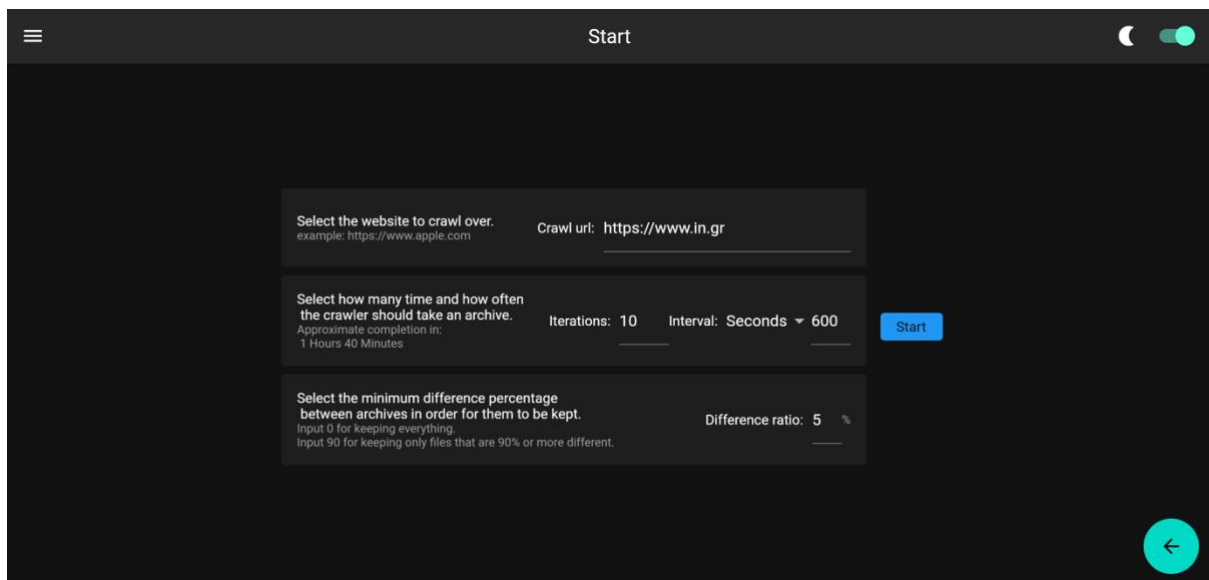
Για να είναι δυνατό η σελίδα να δείχνει στον χρήστη την κατάσταση του crawler χρησιμοποιεί το API που προσφέρει το backend της εφαρμογής και συγκεκριμένα το endpoint ‘/crawler/status’

```
@app.route('/crawler/status', methods=['GET'])
@cross_origin()
def crawler_status():
    response = None
    crawler_status = []
    try:
        if SUB_PROCESS is None:
            response = jsonify(success=True, running=False,
                               crawlers=crawler_status)
        else:
            process_running = SUB_PROCESS.poll()
            if process_running is None:
                crawler_status = Database.get_active_crawlers()
                response = jsonify(
                    success=True, running=True, crawlers=crawler_status)
            else:
                response = jsonify(
                    success=True, running=False, crawlers=crawler_status)
    except Exception as exc:
        current_app.logger.error(exc, exc_info=True)
        response = jsonify(success=False)
    return response
```

Το backend χρησιμοποιεί την global variable SUB_PROCESS για να κρατάει το identifier του process που τρέχει το προηγούμενο αίτημα του χρήστη, έτσι με το αίτημα για λήψη της κατάστασης της εφαρμογής αρχικά ελέγχεται αν η SUB_PROCESS έχει τιμή και αν ναι απαντάει στο frontend ότι υπάρχει ενεργό αίτημα, καθώς και το πότε ξεκίνησε και πόσες φορές θα γίνει λήψη στιγμιότυπου για να γνωρίζει ο χρήστης την πλήρη κατάσταση του αιτήματος του.

Start Crawler Screen

Για να ξεκινήσει μια νέα εργασία για λήψη στιγμιότυπων παρέχεται σελίδα έτσι ώστε να είναι εύκολη η εισαγωγή των απαραίτητων στοιχείων όπως η αρχική σελίδα, πόσα στιγμιότυπα αλλά και με πόσο χρόνο διαφορά μεταξύ του θα γίνονται και τέλος τι διαφορά θα πρέπει να έχει το τελευταίο στιγμιότυπο από το προηγούμενο για να αποθηκευτεί έτσι ώστε να μειωθεί ο αριθμός άχρηστων ίδιων στιγμιότυπων.



1 Crawler Start screen

Την έναρξη κάθε αιτήματος χειρίζεται το endpoint ‘/crawler/start’

```
@app.route('/crawler/start', methods=['POST'])
@cross_origin()
def crawler_start():
    global SUB_PROCESS
    global SUB_PROCESS_CRAWLER_ID
    response = None
    force_start = False
    repeat_times = 1
    interval_seconds = 600
    diff_threshold = 95
```

```
crawl_url = env_variables.get_env_var("WEBPAGE_URL")

try:
    if 'force_start' in request.json:
        force_start = request.json.get('force_start')

    if SUB_PROCESS is not None:
        process_running = SUB_PROCESS.poll()
        if process_running is None:
            if force_start is True:
                current_app.logger.info(
                    "Got force_start True, killing old crawler.")
                SUB_PROCESS.kill()
                Database.update_finished_crawler(
                    crawler_id=SUB_PROCESS_CRAWLER_ID, status="Stopped")
                SUB_PROCESS = None
                SUB_PROCESS_CRAWLER_ID = None
            else:
                return jsonify(success=True, started=False)

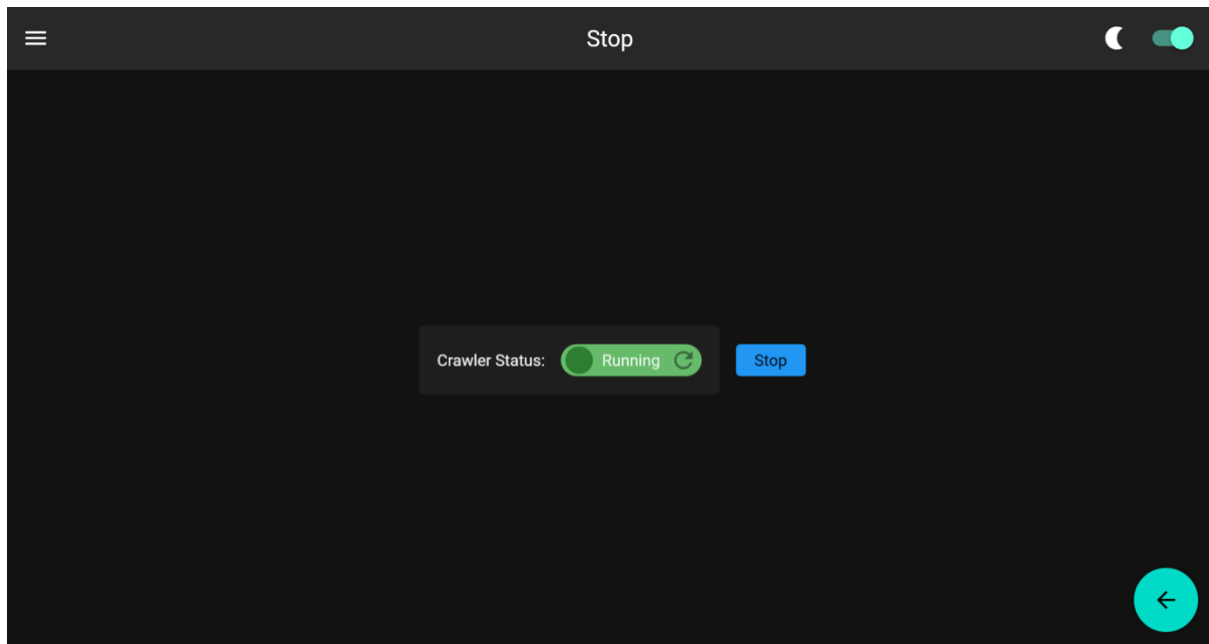
    if 'repeat_times' in request.json:
        repeat_times = request.json.get('repeat_times')
    if 'interval_seconds' in request.json:
        interval_seconds = request.json.get('interval_seconds')
    if 'diff_threshold' in request.json:
        diff_threshold = request.json.get('diff_threshold')
    if 'crawl_url' in request.json:
        crawl_url = request.json.get('crawl_url')
    current_app.logger.debug(
        f"Going to repeat {repeat_times} times by {interval_seconds} seconds interval on {crawl_url} "
        f"using a {diff_threshold}% threshold. "
    )
    crawler_id = Database.insert_new_crawl_task(
        crawl_url, repeat_times, interval_seconds)
    SUB_PROCESS_CRAWLER_ID = crawler_id
    SUB_PROCESS = subprocess.Popen(["python3", "crawler.py", str(
        repeat_times), str(interval_seconds), str(diff_threshold), str(crawl_url)])
    Database.update_new_crawl_task_pid(
        crawler_id, int(SUB_PROCESS.pid))
    response = jsonify(success=True, started=True)
except Exception as exc:
```

```
current_app.logger.error(exc, exc_info=True)
response = jsonify(success=False, started=False)
return response
```

Με σκοπό την όσο το δυνατό πιο εύκολη αυτοματοποίησης λήψης στιγμιότυπων δίνεται η δυνατότητα ο χρήστης να ζητήσει να λαμβάνονται αυτά ανά χρονικό διάστημα που θέτει στο διαχειριστικό καθώς και πόσες φορές θα επαναληφθεί για μια συγκεκριμένη σελίδα, καθώς και το ελάχιστο ποσοστό διαφορών για να αποθηκευτεί ένα στιγμιότυπο. Αυτή η πληροφορία στέλνεται στο backend στην μορφή post κλήσης ώστε να ξεκινήσει το αίτημα, στην περίπτωση που τρέχει κάποιο αίτημα επιστρέφεται αντίστοιχο μήνυμα στον χρήστη ώστε να του δοθεί η δυνατότητα για να σταματήσει το συγκεκριμένο αμέσως και να ξεκινήσει το επόμενο. Στην περίπτωση που δεν τρέχει κάποιο αίτημα ή δοθεί από τον χρήστη εντολή για να σταματήσει το προηγούμενο, αρχικά ενημερώνεται η βάση με τις λεπτομέρειες του αιτήματος και έπειτα ξεκινάει νέο sub process του οποίου identifier καταγράφεται στην global variable SUB_PROCESS

Stop Crawler Screen

Αντίστοιχα όταν πλέον υπάρχει ένας ενεργός crawler και για οποιοδήποτε λόγο πρέπει να σταματήσει μπορεί να δοθεί εντολή χρησιμοποιώντας την παρακάτω σελίδα.



2Crawler Stop screen

Στην περίπτωση που ο χρήστης θέλει να σταματήσει ένα αίτημα, αυτό εκπληρώνεται από το endpoint `/crawler/stop`

```
@app.route('/crawler/stop', methods=['GET'])
@cross_origin()
def crawler_stop():
    global SUB_PROCESS
    global SUB_PROCESS_CRAWLER_ID
    response = None
    try:
        if SUB_PROCESS is None:
            current_app.logger.info(
                "No subprocess to kill.")
            response = jsonify(success=True, stopped=False)
        else:
            process_running = SUB_PROCESS.poll()
            if process_running is None:
                # SUB_PROCESS.subprocess is alive
```

```
current_app.logger.info("Killing subprocess.")
SUB_PROCESS.kill()
Database.update_finished_crawler(
    crawler_id=SUB_PROCESS_CRAWLER_ID, status="Stopped")
SUB_PROCESS = None
SUB_PROCESS_CRAWLER_ID = None
response = jsonify(success=True, stopped=True)
else:
    current_app.logger.info(
        "No subprocess to kill.")
    response = jsonify(success=True, stopped=False)
except Exception as exc:
    current_app.logger.error(exc, exc_info=True)
    response = jsonify(success=False)
return response
```

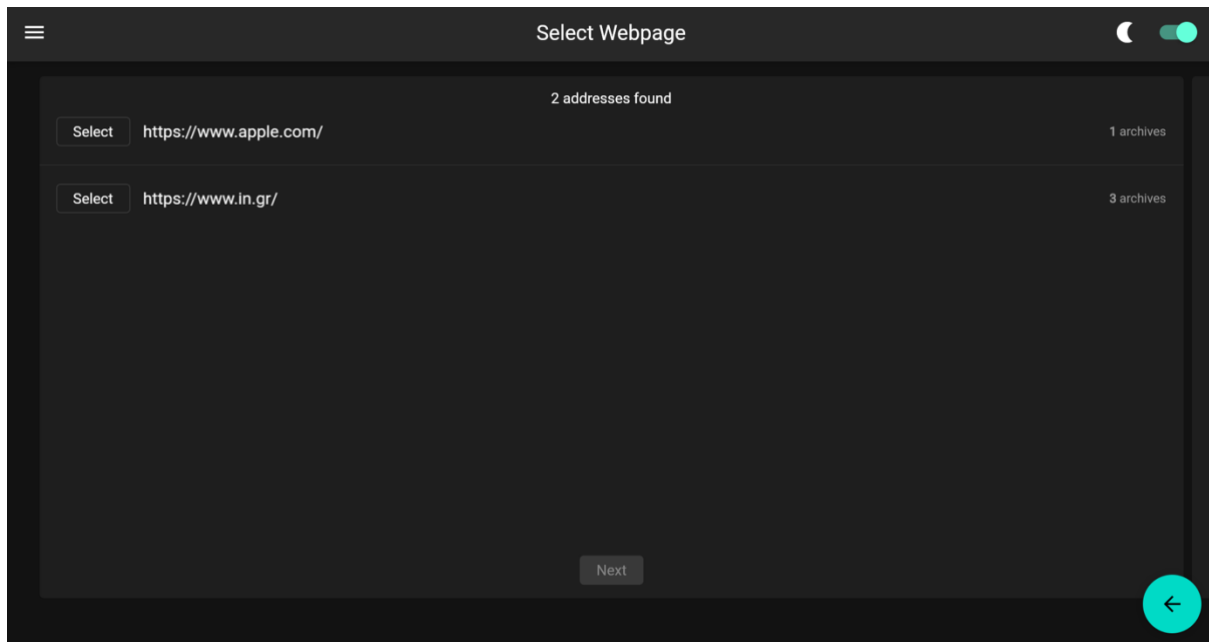
Δουλεύει με την ίδια λογική όπως και το endpoint για την κατάσταση της εφαρμογής, δηλαδή ελέγχει αν έχει τιμή η global variable SUB_PROCESS, αν δεν έχει πάει να πει ότι δεν υπάρχει κάποιο ενεργό αίτημα και επιστρέφει αντίστοιχο αίτημα στον χρήστη. Στην περίπτωση που έχει value τότε δίνεται εντολή για να γίνει ‘kill’ η ρουτίνα αυτή και έπειτα ενημερώνεται αντίστοιχα και η βάση με την νέα κατάσταση του αιτήματος.

Σελίδα σύγκρισης στιγμιότυπων

Έπειτα από ολοκληρωμένες ενέργειες για λήψη στιγμιότυπων είναι δυνατό να δημιουργηθεί σελίδα η οποία περιέχει τις διαφορές μεταξύ δύο από τα προηγούμενα στιγμιότυπα μια ηλεκτρονική σελίδας. Για να γίνει αυτό έχει γραφτεί η παρακάτω σελίδα η οποία περιέχει μια σειρά από υπό-σελίδες σε μορφή καρουζέλ στις οποίες είναι τα απαραίτητα βήματα:

- αρχικά επιλογή της ηλεκτρονικής σελίδας της οποίας θέλουμε να συγκρίνουμε τα στιγμιότυπα. Για να μπορεί να επιλεγθεί μια από τις σελίδες

είναι απαραίτητο να έχουν ληφθεί από δύο και πάνω στιγμιότυπα ώστε να είναι δυνατό να παραχθεί η διαφορά μεταξύ τους.

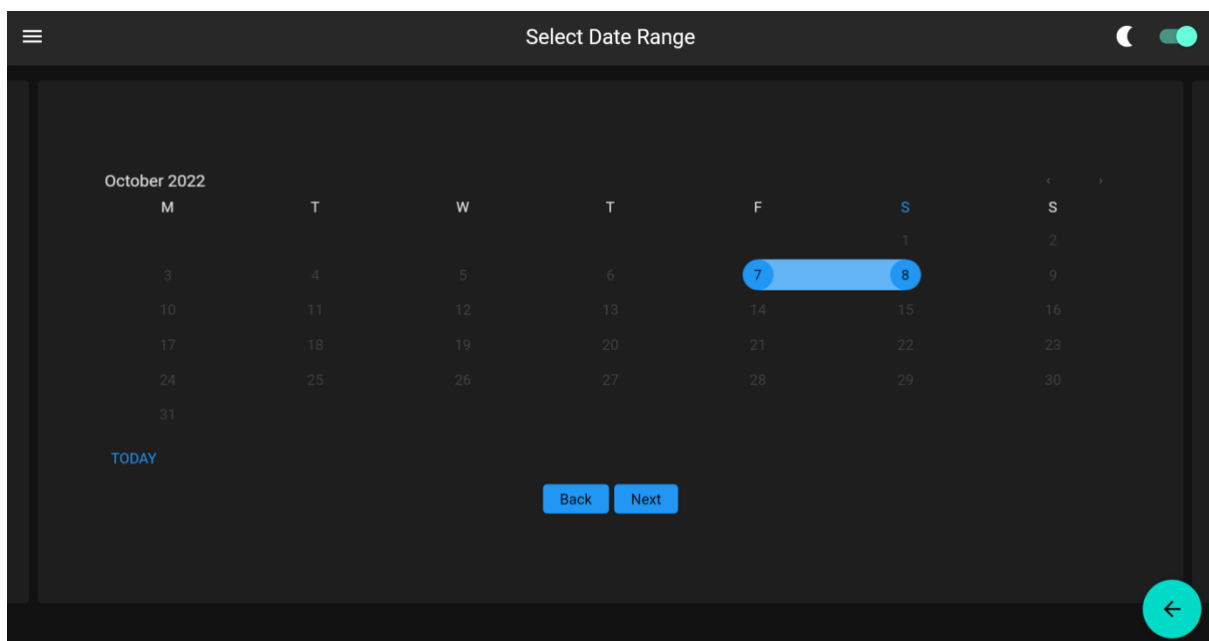


Αρχικά για να δοθεί μια λίστα με τις διαθέσιμες σελίδες από τις οποίες έχουν ληφθεί στιγμιότυπα χρησιμοποιείται το endpoint `‘/results/getalladdresses’`

```
@app.route('/results/getalladdresses', methods=['GET'])
@cross_origin()
def crawler_results_getalladdresses():
    response = None
    try:
        result = Database.get_addresses_and_archive_sum()
        response = jsonify(success=True, addresses=result)
    except Exception as exc:
        current_app.logger.error(exc)
        response = jsonify(success=False)
    return response
```

Το οποίο αρχικά διαβάζει από την βάση όλα τα διαθέσιμα στιγμιότυπα και αφού τα ομαδοποιήσει τα επιστρέφει στο Frontend ώστε να δοθούν σαν επιλογές στον χρήστη

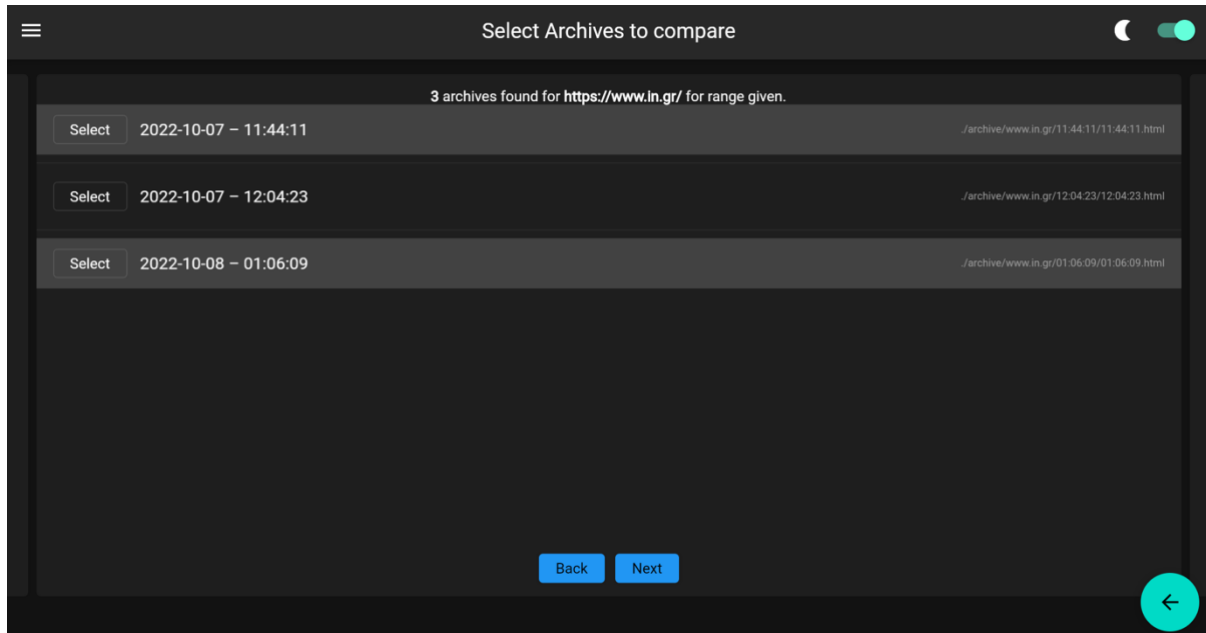
- Επιλογή εύρους ημερομηνιών που έχουν ληφθεί στιγμιότυπα για την σελίδα που επιλέχθηκε στο προηγούμενο βήμα. Στην περίπτωση που έχουν ληφθεί περισσότερα από ένα στιγμιότυπα αλλά μονάχα μέσα σε μια ημέρα, δεν εμφανίζεται η παρακάτω σελίδα αλλά γίνεται αυτόματη επιλογή της και η διαδικασία συνεχίζει στο επόμενο βήμα.



Αφού ο χρήστης έχει επιλέξει για ποια σελίδα θέλει να δει όλα τα διαθέσιμα στιγμιότυπα, είναι απαραίτητο να του δοθεί η δυνατότητα να διαλέξει και ημερομηνίες από τις οποίες επιθυμεί να συγκρίνει, αυτό επιτυγχάνεται καλώντας τρία endpoint από το backend ώστε να ληφθούν για την συγκεκριμένη σελίδα η

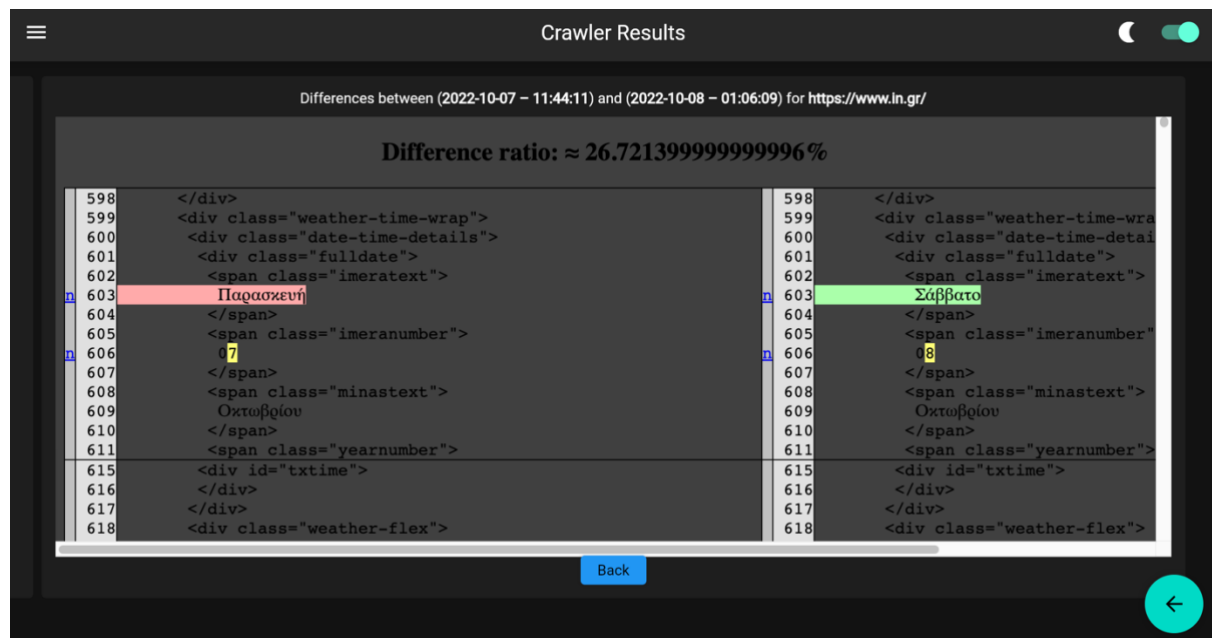
πιο παλιά ημέρα που έχει ληφθεί στιγμιότυπο, η πιο πρόσφατη καθώς και όλες οι υπόλοιπες μεταξύ τους.

- Βάση του εύρους που επιλέχθηκε ή, αν είναι αυτόματα επιλεγμένη, η μοναδική ημέρα για την οποία υπάρχουν στιγμιότυπα, εμφανίζεται η λίστα με όλα τα στιγμιότυπα που έχουν γίνει και η ώρα κατά την οποία πάρθηκαν. Σε αυτό το βήμα απαραίτητο είναι η επιλογή δύο από αυτών ώστε να καταστεί δυνατή η δημιουργία της σελίδας με τις διαφορές τους από το backend της εφαρμογής. Χρήσιμο είναι να αναφερθεί ότι στο δεξιά μέρος της οθόνης για κάθε ένα από τα στιγμιότυπα αναγράφεται το relative path του αρχείου που έχει αποθηκευτεί το κάθε στιγμιότυπο σε περίπτωση που είναι χρήσιμο για τον χρήστη να το ανοίξει και αν δει τον κώδικα HTML που είχε ηλεκτρονική σελίδα για την οποία πάρθηκε τη συγκεκριμένη χρονική στιγμή.



Αφού γίνει επιλογή ημερομηνιών αυτές στέλνονται πίσω στο backend και βάσει αυτών διαβάζεται από τη βάση όλα τα στιγμιότυπα που έχουν ληφθεί για τη συγκεκριμένη ιστοσελίδα μέσα σε αυτές τις ημέρες.

- Τέλος, αφού ολοκληρωθούν τα προηγούμενα βήματα, γίνεται αίτημα στο backend της εφαρμογής για να παραχθεί η σελίδα με τις διαφορές τους η οποία τελικά με χρήση iFrame εισάγεται στην σελίδα που έχει μπροστά του ο χρήστης. Στο πάνω μέρος της σελίδας αναγράφεται και το ποσοστό των διαφορών μεταξύ των δύο στιγμιότυπων όπου όσο μεγαλύτερο είναι αυτό το ποσοστό τόσο περισσότερες είναι οι διαφορές μεταξύ τους. Για την δημιουργία της σελίδας με τις διαφορές μεταξύ των στιγμιότυπων χρησιμοποιείται η βιβλιοθήκη Difflib η οποία με τις συναρτήσεις που παρέχει επιτρέπει τον υπολογισμό ποσοστού διαφοράς μεταξύ των αρχείων αλλά και δυναμική παραγωγή του αρχείου που αναγράφει όλες τις διαφορές μεταξύ τους. Συγκεκριμένα με τη χρήση της παραπάνω βιβλιοθήκης δίνεται αρχικά δυνατότητα στον χρήστη να διαλέξει από τι ποσοστό διαφορών και πάνω επιθυμεί να κρατάει στιγμιότυπα και τέλος παράγεται σε HTML η σελίδα που δείχνει τις διαφορές μεταξύ τους.



Σαν τελευταίο βήμα αφού επιλεγθούν τα συγκεκριμένα στιγμιότυπα που επιθυμεί ο χρήστης αποστέλλονται πίσω στο backend τα id τους στο endpoint `‘/results/generatehtml/<first_archive_id>/<second_archive_id>’` ώστε να δημιουργηθεί σε HTML η σελίδα που αναγράφει τις διαφορές μεταξύ τους χρησιμοποιώντας την συνάρτηση HtmlDiff της βιβλιοθήκης Difflib.

```
@app.route('/results/generatehtml/<first_archive_id>/<second_archive_id>', methods=['GET'])
@cross_origin()
def crawler_results_html(first_archive_id=None, second_archive_id=None):
    response = None
    if first_archive_id is None or second_archive_id is None:
        response = jsonify(success=False)
    else:
        try:
            locations = Database.get_archives_location_from_id(
                first_archive_id, second_archive_id)
            if len(locations) == 2:
                with open(locations[0][0], 'r', encoding='utf-8') as file_a:
                    with open(locations[1][0], 'r', encoding='utf-8') as file_b:
                        diff = difflib.HtmlDiff(wrapcolumn=60)
                        diff._styles = diff._styles + """
                            table.diff {
                                font-family:Courier;
                                border:medium;
                                margin-left: auto;
                                margin-right: auto;
                            }
                            """
                        result = diff.make_file(
                            file_a, file_b, context=True)
                        float_ratio = FileSystem.calculate_file_difference(
                            locations[0][0], locations[1][0])
                        percentage = (1 - round(float_ratio, 6)) * 100
                        return render_template('archive_diffs.html', generated=result, ratio=percentage)
            else:
                current_app.logger.error(
                    f"Database returned {len(locations)} locations, need exactly 2 to run.")
                response = jsonify(success=False)
```

```
except Exception as exc:
    current_app.logger.error(exc)
    response = jsonify(success=False)
return response
```

Για να είναι ποιο ευανάγνωστο στον χρήστη χρησιμοποιείται ένα template στο οποίο βάση της ρύθμισης του χρώματος που έχει επιλεγθεί (dark/light mode) παραμετροποιείται αντίστοιχα, καθώς προστίθεται και το ποσοστό διαφοράς μεταξύ των στιγμιότυπων που έχουν επιλεγθεί στο πάνω μέρος της σελίδας.

Crawler Backend logs

Για τη διευκόλυνση της διαχείρισης της εφαρμογής έχει προστεθεί δυνατότητα για προβολή των logs του backend της εφαρμογής χρησιμοποιώντας την παρακάτω σελίδα. Κατά την πλοήγηση του χρήστη στην σελίδα με τα logs ανοίγεται socket με το backend της εφαρμογής μέσω του οποίου ενημερώνεται η σελίδα με τα logs που έχει γράψει το backend. Η συγκεκριμένη ενημέρωση γίνεται από την μεριά του backend κάθε 15 δευτερόλεπτα και αποστέλνεται προς όλα τα ενεργά socket connection. Έχει προστεθεί δυνατότητα για παύση των λήψεων ενημερώσεων από το πάνω δεξιά μέρος της οθόνης και συνέχεια αντίστοιχα όταν θελήσει ο χρήστης να αρχίσει πάλι από το ίδιο σημείο.



```
[2022-10-07 11:42:55][ DEBUG][ check_page_protocol:WebCrawler.py:123] -- Original url redirects to https://
[2022-10-07 11:42:55][ DEBUG][ create_page_folder:FileSystem.py: 43] -- Directory www.apple.com Created.
[2022-10-07 11:42:55][ DEBUG][ get_root_page_links:WebCrawler.py: 68] -- First crawl for requested address,
[2022-10-07 11:42:55][ DEBUG][ get_encoding:WebCrawler.py:173] -- Page's preferred encoding is utf-8
[2022-10-07 11:42:55][ DEBUG][ save_page:FileSystem.py: 68] -- Saving archive on: ./archive/www.ap
[2022-10-07 11:42:55][ DEBUG][ save_page:FileSystem.py: 74] -- 11:42:55 snapshot saved.
[2022-10-07 11:42:55][ DEBUG][ get_links:WebCrawler.py: 96] -- Found 132 links under https://www.
[2022-10-07 11:42:55][ DEBUG][ get_links:WebCrawler.py:106] -- Going to iterate over 98. Here goes
[2022-10-07 11:42:55][ DEBUG][ insert_links_found:Database.py:311] -- Inserted 76 links to DB.
[2022-10-07 11:42:55][ DEBUG][ crawler_main:crawler.py: 15] -- [pid:7] Ended a crawl after 1.08 se
[2022-10-07 11:42:55][ DEBUG][ _main:crawler.py: 55] -- [pid:7] Iteration 1 / 10 finished
[2022-10-07 11:43:37][ INFO][ crawler_stop:backend.py:187] -- Killing subprocess.
[2022-10-07 11:44:09][ DEBUG][ crawler_start:backend.py:155] -- Going to repeat 10 times by 600 sec
[2022-10-07 11:44:09][ DEBUG][ set_crawler_id:WebCrawler.py: 28] -- Setting crawler id: 2
[2022-10-07 11:44:09][ DEBUG][ set_root_page_url:WebCrawler.py: 39] -- Setting root page to crawl: https://
[2022-10-07 11:44:09][ DEBUG][ set_diff_threshold:WebCrawler.py: 56] -- Setting diff threshold: 0.95
[2022-10-07 11:44:09][ INFO][ _main:crawler.py: 47] -- [pid:8][crawler_id:2] Crawler start
[2022-10-07 11:44:09][ INFO][ _main:crawler.py: 52] -- [pid:8] Iteration 1 / 10 started
[2022-10-07 11:44:10][ DEBUG][ check_page_protocol:WebCrawler.py:123] -- Original url redirects to https://
[2022-10-07 11:44:10][ DEBUG][ create_page_folder:FileSystem.py: 43] -- Directory www.in.gr Created.
[2022-10-07 11:44:11][ DEBUG][ get_root_page_links:WebCrawler.py: 68] -- First crawl for requested address,
[2022-10-07 11:44:11][ DEBUG][ get_encoding:WebCrawler.py:173] -- Page's preferred encoding is utf-8
[2022-10-07 11:44:11][ DEBUG][ save_page:FileSystem.py: 68] -- Saving archive on: ./archive/www.in
[2022-10-07 11:44:11][ DEBUG][ save_page:FileSystem.py: 74] -- 11:44:11 snapshot saved.
[2022-10-07 11:44:11][ DEBUG][ get_links:WebCrawler.py: 96] -- Found 420 links under https://www.
[2022-10-07 11:44:11][ DEBUG][ get_links:WebCrawler.py:106] -- Going to iterate over 350. Here goes
[2022-10-07 11:44:11][ DEBUG][ insert_links_found:Database.py:311] -- Inserted 264 links to DB.
[2022-10-07 11:44:11][ DEBUG][ crawler_main:crawler.py: 15] -- [pid:8] Ended a crawl after 1.99 se
[2022-10-07 11:44:11][ DEBUG][ _main:crawler.py: 55] -- [pid:8] Iteration 1 / 10 finished
[2022-10-07 11:44:26][ DEBUG][ connected:backend.py: 50] -- New websocket client connected, to
```

Επόμενα βήματα

Ο σκοπός την παρούσας εργασίας είναι να δοθεί μια εφαρμογή η οποία να παρέχει όλες τις απαραίτητες λειτουργίες για τη συλλογή στιγμιότυπων ηλεκτρονικών σελίδων αλλά και των διαφορών μεταξύ τους. Κατά τη διάρκεια της υλοποίησης της αναγνωρίστηκαν σημεία όπου θα μπορούσαν να προστεθούν παραπάνω λειτουργίες ή και βελτιώσεις στις υπάρχουσες, συγκεκριμένα σαν επόμενα βήματα αναγνωρίζονται τα εξής:

- Προσθήκη δυνατότητας διαγραφής παλαιότερων στιγμιότυπων για αποφυγή κατανάλωσης χώρου όταν αυτό δεν είναι απαραίτητο.
- Προσθήκη δυνατότητας για κατέβασμα των διαφορών μεταξύ στιγμιότυπων σε μορφή pdf από το περιβάλλον χρήσης στον υπολογιστή του χρήστη.
- Προσθήκη δυνατότητας για προβολή ενός στιγμιότυπου από το περιβάλλον χρήσης της εφαρμογής χωρίς να χρειάζεται να ανοιχτούν τα αρχεία της εφαρμογής στον server από τον οποίο τρέχει.
- Μετασχηματισμός του περιβάλλοντος χρήστη ώστε να καθορίζεται δυναμικά η διάταξη της εφαρμογής βάση του μεγέθους της οθόνης του χρήστη.
- Προσθήκη δυνατότητας για προγραμματισμό λήψης στιγμιότυπων σε μεταγενέστερη χρονική περίοδο χωρίς να είναι απαραίτητο να ξεκινήσει αμέσως η ενέργεια.
- Δυνατότητα για εκτέλεση περισσότερων από μιας ενεργειών λήψης στιγμιότυπων ταυτόχρονα.
- Δυνατότητα για πλήρη λήψη στιγμιότυπου μιας ηλεκτρονικής σελίδας περιλαμβάνοντας όλες τις συνδεδεμένες σελίδες της, βάσει των link που υπάρχουν στην αρχική σελίδα που δόθηκε από τον χρήστη.

Βιβλιογραφία – Δικτυογραφία

Docker [<https://www.docker.com/>]

Kubernetes [<https://kubernetes.io/>]

Flask [<https://flask.palletsprojects.com/en/2.2.x/>]

MariaDB [<https://mariadb.com/docs/>]

Flutter [<https://flutter.dev/>]

phpMyAdmin [<https://www.phpmyadmin.net/>]

DBeaver [<https://dbeaver.io/>] Used during development for DB monitoring

Postman [<https://www.postman.com/>] Used during development for RestAPI calls

DiffLib [<https://docs.python.org/3/library/difflib.html>]

HtmlDiff [<https://docs.python.org/3/library/difflib.html#difflib.HtmlDiff>]

Visual Studio Code [<https://code.visualstudio.com/>]

Project Repo [<https://github.com/Pantelhs/website-archiving-poc>]

Εικόνες

1 25

2 Side panel available 26

3 Dashboard dark mode με ενεργό27

27