Ατομική Διπλωματική Εργασία

ΑΥΤΟΜΑΤΟΠΟΙΗΣΗ ΑΝΑΚΤΗΣΗΣ ΚΑΙ ΟΡΓΑΝΩΣΗΣ PULL REQUESTS ΑΠΟ ΤΟ GITHUB

Απόλλωνας Μιχαηλίδης

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αυτοματοποίηση Ανάκτησης και Οργάνωσης Pull Requests από το Dependabot στο GitHub

Απόλλωνας Μιχαηλίδης

Επιβλέπουσα Καθηγήτρια
Ελένη Κωνσταντίνου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2025

Ευχαριστίες

Η διπλωματική μου εργασία αποτελεί το τελευταίο βήμα των προπτυχιακών μου σπουδών στο Τμήμα της Πληροφορικής.

Με την ολοκλήρωση αυτού του σταδίου, θα ήθελα να εκφράσω τις ευχαριστίες μου, αρχικά στην κυρία Ελένη Κωνσταντίνου, καθηγήτρια μου και επιβλέπουσα της διπλωματικής μου εργασίας, για την καθοδήγηση που μου παρείχε και για την εμπιστοσύνη που μου έδειξε κατά την διάρκεια της διαδικασίας..

Επιπρόσθετα, θα ήθελα να ευχαριστήσω του συμφοιτητές μου για την υποστήριξη που μου παρείχαν για την ολοκλήρωση της διπλωματικής μου εργασίας.

Περίληψη

Σε αυτή την διπλωματική εργασία, παρουσιάζεται η ανάπτυξη ενός λογισμικού εργαλείου για την οργάνωση των pull requests [2] που δημιουργούνται αυτόματα από το Dependabot [1] σε διάφορα έργα, projects δηλαδή, στο GitHub. Το Dependabot είναι ένα bot, εργαλείο αυτοματοποίησης, που ελέγχει τις βιβλιοθήκες και τα πακέτα ενός έργου λογισμικού, και αν εντοπίσει διαθέσιμες νέες, πιο ασφαλείς εκδόσεις, δημιουργεί pull requests, δηλαδή αιτήματα συγχώνευσης.

Το εργαλείο που αναπτύχθηκε, συλλέγει τα pull requests που δημιουργούνται από το Dependabot με βάση των επιλογών/κριτηρίων, τα φίλτρα δηλαδή, του προγραμματιστή. Κάποια από αυτά τα κριτήρια, είναι το χρονικό διάστημα μεταξύ συγκεκριμένων ημερομηνιών, συγκεκριμένο repository [4], αν τα pull requests είναι merged [6] ή unmerged, ή ακόμη και με την κατάσταση τους (state).

Κύριος στόχος του εργαλείου, είναι η εξαγωγή των δεδομένων των pull requests, η επεξεργασία του πεδίου “body”, το οποίο βρίσκεται σε μορφή html [15] και περιέχει όλες τις αλλαγές και τις τροποποιήσεις (όπως για παράδειγμα commits, versions), καθώς και η αποθήκευση τους σε αρχείο JSON [3]. Η συγκεκριμένη μορφή αρχείου προσφέρει στον προγραμματιστή μια πιο κατανοητή και ευανάγνωστη αναπαράσταση δομημένων δεδομένων.

Τέλος, το εργαλείο επικεντρώνεται στην αυτοματοποίηση της διαδικασίας της εξαγωγής βασικών πληροφοριών από τα pull requests που προτείνει το Dependabot, όπως το όνομα του πακέτου, την συμβατότητα της αλλαγής καθώς και τις διάφορες αλλαγές και τροποποιήσεις που έχουν πραγματοποιηθεί.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή

Η ανάπτυξη λογισμικού βασίζεται ολοένα και περισσότερο στην χρήση εξωτερικών βιβλιοθηκών και πακέτων, καθώς μπορούν να προσφέρουν έτοιμες λύσεις και να επιταχύνουν με αυτόν τον τρόπο την διαδικασία της υλοποίησης.

Όμως, η συνεχής εξέλιξη των εξαρτήσεων, μπορεί να δημιουργήσεις κάποιες προκλήσεις, όπως για παράδειγμα η ασφάλεια και η σωστή λειτουργία των έργων λογισμικού, Έτσι, λόγω της συνεχούς εξέλιξης των εξαρτήσεων, δημιουργείται η ανάγκη για συστηματική παρακολούθηση και ενημέρωση.

Το Dependabot, είναι ένα bot στο GitHub, το οποίο αποτελεί βασικό εργαλείο αυτοματοποίησης της διαδικασίας της ενημέρωσης εξαρτήσεων. Δημιουργεί pull requests τα οποία προτείνουν ενημερώσεις, updates, αναβαθμίσεις δηλαδή, σε πακέτα και βιβλιοθήκες.

Παρόλο που με την βοήθεια του Depandabot η διαδικασία αυτοματοποιείται και απλοποιείται, σε μεγάλα projects, όπου ο αριθμός των pull requests είναι αρκετά μεγάλος, η διαχείριση των εξαρτήσεων γίνεται δύσκολη και χρονοβόρα, καθώς οι προγραμματιστές καλούνται να ελέγχουν κάθε pull request ξεχωριστά.

Η ανάγκη για μια καλύτερη οργάνωση και αυτοματοποίηση της ανάκτησης των pull requests του Dependabot, οδήγησε στην ανάπτυξη του παρόντος εργαλείου, το οποίο ανακτά και επεξεργάζεται δεδομένα και πληροφορίες από αυτά τα pull requests, και ακολούθως τα οργανώνει, βάση των κριτηρίων του προγραμματιστή, σε αρχεία τύπου JSON ώστε να είναι ευανάγνωστα, καθώς και άμεσα αξιοποιήσιμα, για περαιτέρω ανάλυση και επεξεργασία.

Η παρούσα διπλωματική εργασία εστιάζει στην υλοποίηση αυτής της λύσης, παρουσιάζοντας τον τρόπο συλλογής, φιλτραρίσματος και οργάνωσης των πιο σημαντικών δεδομένων και πληροφοριών, με σκοπό την βελτίωση της διαχείρισης ενημερώσεων εξαρτήσεων σε έργα λογισμικού.

1.2 Στόχοι Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία, έχει ως κύριο στόχο την ανάπτυξη ενός εργαλείου που θα αυτοματοποιεί την διαδικασία συλλογής, επεξεργασίας και οργάνωσης των pull requests, που δημιουργεί το Dependabot σε έργα λογισμικού που υπάρχουν στο GitHub.

Συγκεκριμένα, το εργαλείο έχει τους εξής στόχους:

1) Την συλλογή των pull requests που έχουν δημιουργηθεί από το Dependabot με βάση ορισμένα κριτήρια/φίλτρα, όπως συγκεκριμένο χρονικό διάστημα, συγκεκριμένα repositories, συγκεκριμένη κατάσταση -–state (open, closed, all), ή/και ακόμη κατά πόσο είναι merged ή unmerged.

2) Την εξαγωγή βασικών πληροφοριών και δεδομένων που περιέχονται στο πεδίο “body”, του κάθε Pull request, όπως τις αλλαγές, τα commits, τις εκδόσεις και άλλες σημαντικές πληροφορίες που παρέχει το κάθε pull request.

3) Την οργάνωση των πληροφοριών με βάση τα κριτήρια (φίλτρα) του προγραμματιστή, σε αρχεία με μορφή JSON, ώστε να μπορούν να είναι άμεσα αξιοποιήσιμες για περαιτέρω ανάλυση.

Με αυτόν τον τρόπο, το εργαλείο έχει ως στόχο να βοηθά στην πιο εύκολη και οργανωμένη παρακολούθηση των ενημερώσεων εξαρτήσεων, καθώς η καταγραφή των αλλαγών σε

αρχεία μορφής JSON, προσφέρει στον προγραμματιστή μια πιο κατανοητή και ευανάγνωστη αναπαράσταση δομημένων δεδομένων.

1.3 Δομή Διπλωματικής Εργασίας

Η διπλωματική εργασία αποτελείται από 6 κεφάλαια, όπως περιγράφονται παρακάτω:

- Κεφάλαιο 1: Εισαγωγή. Σύντομη εισαγωγή στο θέμα μαζί με τον κύριο στόχο της παρούσας διπλωματικής εργασίας.

- Κεφάλαιο 2: Θεωρητικό Υπόβαθρο και Βασικές Έννοιες. Σε αυτό το κεφάλαιο, γίνεται η περιγραφή βασικών εννοιών, εργαλείων και βιβλιοθηκών που χρησιμοποιήθηκαν για την ανάπτυξη του συστήματος.

- Κεφάλαιο 3: Καθορισμός Απαιτήσεων και Προδιαγραφών. Σε αυτό το κεφάλαιο, γίνεται αναφορά στις απαιτήσεις και τις προδιαγραφές του συστήματος. Περιγράφονται οι απαιτήσεις του συστήματος, καθώς, οι λειτουργικές και μη λειτουργικές απαιτήσεις, όπως και το μοντέλο ανάπτυξης που έχει ακολουθηθεί.

- Κεφάλαιο 4: Σχεδιασμός Συστήματος. Στο κεφάλαιο 4, αναλύεται η αρχιτεκτονική του συστήματος καθώς και οι λειτουργίες του εργαλείου.

- Κεφάλαιο 5: Αξιολόγηση Συστήματος. Σε αυτό το κεφάλαιο, γίνεται αναφορά στον έλεγχο του συστήματος. Επιπρόσθετα, παρουσιάζονται τα Feedbacks που παραθέτουν οι χρήστες και προτάσεις για βελτίωση του συστήματος.

- Κεφάλαιο 6: Συμπεράσματα. Στο τελευταίο κεφάλαιο, γίνεται μια επισκόπηση της διπλωματικής εργασίας και καταγράφονται τα συμπεράσματα. Επιπρόσθετα, περιγράφονται οι περιορισμοί του συστήματος, καθώς και οι διάφορες μελλοντικές επεκτάσεις και αναβαθμίσεις του εργαλείου ώστε να παρέχει περισσότερες λειτουργίες.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο – Βασικές Έννοιες

2.1 Το GitHub

Το GitHub είναι μια πλατφόρμα διαχείρισης και αποθήκευσης των έργων λογισμικού. Οι προγραμματιστές, μπορούν να συνεργάζονται, να πραγματοποιούν αλλαγές στον κώδικα, καθώς επίσης και να παρακολουθούν την εξέλιξη των projects.

Επιπρόσθετα, βασίζεται στο Git [5], το οποίο είναι ένα σύστημα ελέγχου έκδοσης, και καταγράφει τις αλλαγές στον πηγαίο κώδικα, επιτρέποντας επιστροφές σε προηγούμενες εκδόσεις.

Η πλατφόρμα, παρέχει επίσης την δυνατότητα στους προγραμματιστές να δημιουργούν και να δουλεύουν σε repositories, αποθετήρια δηλαδή. Μέσω των pull requests, οι χρήστες έχουν την δυνατότητα να συνεργάζονται μεταξύ τους και να εντοπίζουν πιθανά προβλήματα.

Επιπρόσθετα, στο GitHub, υπάρχουν bots, τα οποία είναι κάποια εργαλεία που βοηθούν στην αυτοματοποίηση διαδικασιών. Στην παρούσα διπλωματική εργασία, θα επικεντρωθούμε στο Dependabot, το οποίο είναι ένα εργαλείο αυτοματοποίησης, bot δηλαδή.

2.2 Τα Repositories

Ένα repository, αποθετήριο δηλαδή, είναι το πιο βασικό στοιχείο στο GitHub. Είναι υπεύθυνο για την αποθήκευση του πηγαίου κώδικα ενός project και σχετικά αρχεία, όπως επίσης και το ιστορικό των σχετικών αλλαγών και τροποποιήσεων, τις οποίες τροποποιήσεις οι προγραμματιστές έχουν την δυνατότητα να παρακολουθούν και να γνωρίζουν πότε και από ποιον έχουν εφαρμοστεί.

Επιπρόσθετα, τα αποθετήρια, μπορεί να είναι είτε δημόσια, είτε ιδιωτικά. Αν ο δημιουργός ενός συγκεκριμένου αποθετηρίου, επιθυμεί να συμμετέχουν εξωτερικοί προγραμματιστές, δηλαδή να μπορούν να δουλεύουν μέσα το ίδιο αποθετήριο με αυτόν, τότε θα δημιουργήσει ένα δημόσιο αποθετήριο, ενώ αντίθετα, αν οι εξωτερικοί προγραμματιστές είναι απλοί παρακολουθητές, τότε θα δημιουργήσει ένα ιδιωτικό αποθετήριο.

Γενικά, ένα repository, βοηθά στην οργάνωη και γενικότερα στην ανάπτυξη ενός project. Οι προγραμματιστές, συνεργάζονται μεταξύ τους, χρησιμοποιώντας pull requests. Με αυτόν τον τρόπο, ο κάθε προγραμματιστής μπορεί να προτείνει πιθανές τροποποιήσεις και βελτιστοποιήσεις.

2.3 Τα Pull Requests

Ένα pull request, είναι ένα αίτημα συγχώνευσης ενός συνόλου τροποποιήσεων από έναν κλάδο (branch [7]) σε άλλον. Επιπρόσθετα, ένα αίτημα συγχώνευσης προσφέρει την δυνατότητα στους προγραμματιστές που συνεργάζονται μεταξύ τους, να εξετάζουν και να συζητούν το προτεινόμενο σύνολο των αλλαγών, καθώς και να εγκρίνουν ή να απορρίπτουν ανάλογα, την συγχώνευση (merge [6]), πριν να ενσωματώσουν τις αλλαγές στον πηγαίο κώδικα.

Ακόμη, ένα pull request, περιέχει όλες τις διαφορές μεταξύ του πηγαίου κώδικα, και του target branch, την χρονική στιγμή υποβολής, καθώς και το μοναδικό αναγνωριστικό του προγραμματιστή.

Επιπρόσθετα, οι προγραμματιστές, μπορούν να δημιουργήσουν αιτήματα συγχώνευσης στα ακόλουθα:

1) GitHub Desktop
2) GitHub CodeSpaces
3) GitHub Mobile
4) GitHub CLI

2.4 Το Dependabot

Το Dependabot είναι ένα bot, στο GitHub, δηλαδή ένα από τα βασικότερα εργαλεία αυτοματοποίησης της διαδικασίας της διαχείρισης και ενημέρωσης εξαρτήσεων σε έργα λογισμικού. Με αυτόν τον τρόπο, βοηθά σημαντικά στην μείωση της χειροκίνητης παρακολούθησης από τους προγραμματιστές.

Οι εξαρτήσεις, περιέχουν εξωτερικές βιβλιοθήκες, όπως επίσης και πακέτα [11][12], και για αυτόν τον λόγο δημιουργείται η ανάγκη για την ασφάλεια και την σωστή διαχείριση των έργων λογισμικού.

Το Dependabot, ασκεί τακτικούς ελέγχους για νεότερες εκδόσεις εξαρτήσεων και ακολούθως δημιουργεί αυτόματα pull requests με προτάσεις για ενημερώσεις εξαρτήσεων.

Γενικότερα, το Dependabot εκτελεί 3 διαφορετικές λειτουργίες με σκοπό να παρέχει βοήθεια στους προγραμματιστές για την διαχείριση των εξαρτήσεων:

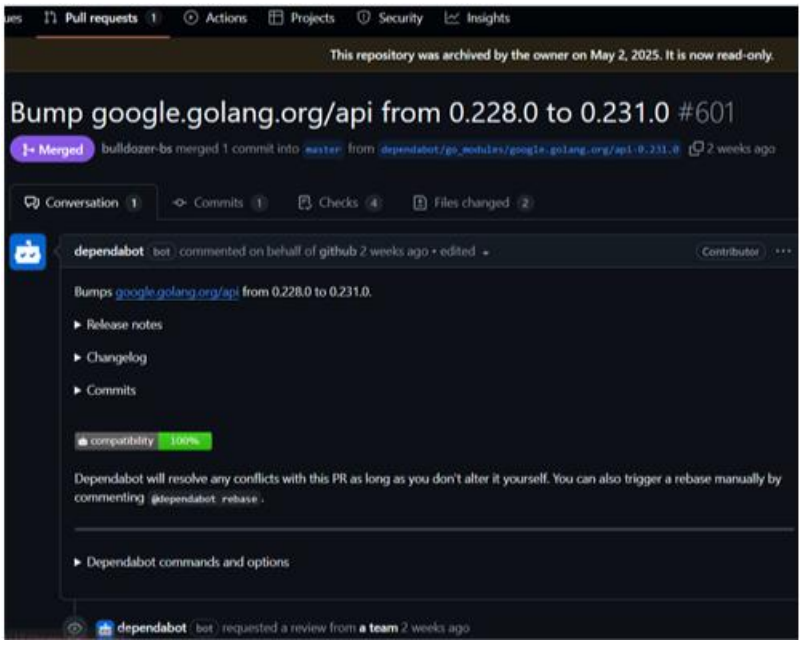
1) Dependabot Alerts: αυτόματη υποβολή pull requests, για ενημερώσεις ευπαθειών στις εξαρτήσεις που χρησιμοποιούνται στο αποθετήριο.

2) Dependabot Security Alerts: αυτόματη υποβολή pull requests για ενημερώσεις εξαρτήσεων που χρησιμοποιεί ο προγραμματιστής και έχουν ευπάθειες που αφορούν θέματα ασφάλειας.

3) Dependabot Version Updates: αυτόματη υποβολή pull requests για ενημερώσεις έτσι ώστε να βρίσκονται οι βιβλιοθήκες και τα πακέτα που χρησιμοποιούνται, στην πιο πρόσφατη και τελευταία έκδοση τους.

Επιπρόσθετα, το εργαλείο υποστηρίζει πληθώρα γλωσσών προγραμματισμού και package managers, όπως για παράδειγμα nmp (JavaScript [8]), pip (Python [9]), Maven (Java [10]). Αυτό, κάνει το Dependabot να είναι αρκετά εύχρηστο σε διαφορετικά περιβάλλοντα ανάπτυξης.

Εικόνα 2.4.1: Σε αυτήν την εικόνα, απεικονίζεται ένα pull request που προτείνει το Dependabot. Η ενημέρωση που προτείνει, αφορά την αναβάθμιση της βιβλιοθήκης “google.golang.org/api”, από την έκδοση 0.228.0 στην έκδοση 0.231.0, η οποία φαίνεται να έχει αριθμό 601. Το Compatibility Score, δηλαδή η συμβατότητα της αλλαγής, φαίνεται να είναι 100%, υπάρχει πλήρης συμβατότητα δηλαδή, ενώ περιέχει επίσης περισσότερες πληροφορίες σχετικά με τις αλλαγές που έχουν πραγματοποιηθεί σε κάθε έκδοση, στα πεδία Changelog, το Release Notes, καθώς και τα Commits.



Εικόνα 2.4.1 Dependabot Pull Request

Κεφάλαιο 3

Καθορισμός Απαιτήσεων και Προδιαγραφών

3.1 Απαιτήσεις Συστήματος

Οι απαιτήσεις συστήματος περιγράφουν τις τεχνικές και τις προϋποθέσεις κάτω από τις οποίες το σύστημα θα λειτουργεί. Το εργαλείο που υλοποιήθηκε, για να λειτουργεί σωστά, πρέπει να τρέχει σε ένα συγκεκριμένο περιβάλλον:

- Operating System (Λειτουργικό Σύστημα): το εργαλείο μπορεί να δουλέψει σε macOS, Windows (Ubuntu), καθώς και σε Linux.

- Εξαρτήσεις Λογισμικού: για την σωστή λειτουργεία του εργαλείου, απαιτείται να σχεδιαστεί το σύστημα σε γλώσσα προγραμματισμού Python [16], καθώς και ένας Web Browser, όπως πχ Chrome ή Firefox.

- Δικαιώματα και Πρόσβαση: ο προγραμματιστής, ούτως ώστε να του παρέχεται το δικαίωμα για να ανακτήσει pull requests από το GitHub, απαιτείται να κατέχει ένα έγκυρο GitHub API token.

3.2 Λειτουργικές Απαιτήσεις

Οι Λειτουργικές Απαιτήσεις περιγράφουν τις υπηρεσίες και τις λειτουργίες που πρέπει να υλοποιεί το σύστημα. Θα αναλυθούν δηλαδή, οι δυνατότητες που καλείται να προσφέρει το σύστημα:

1) Ανάκτηση pull requests: Το εργαλείο καλείται να χρησιμοποιεί το GitHub REST API [14], ούτως ώστε να πραγματοποιεί HTTP κλήσεις για την ανάκτηση των pull requests του Dependabot από το GitHub.

2) Φίλτρα: Το εργαλείο πρέπει να υλοποιεί διάφορα φίλτρα ανάκτησης pull request, δηλαδή να ανακτώνται συγκεκριμένα pull requests, ανάλογα με τα φίλτρα που παίρνει σαν είσοδο από τους προγραμματιστές (για παράδειγμα συγκεκριμένο repository). (Τα φίλτρα/κριτήρια που υλοποιήθηκαν στο σύστημα θα αναλυθούν στην υλοποίηση συστήματος).

3) Αποθήκευση δεδομένων και πεδίων των pull requests: Στο τελικό αρχείο αποθήκευσης, πρέπει να αποθηκεύονται οι πιο σημαντικές πληροφορίες και πεδία που περιέχει το κάθε Pull request ξεχωριστά. (Τα πεδία περιγράφονται στο 4.1.1, συγκεκριμένα στο REST API).

4) Εξαγωγή δεδομένων και πληροφοριών από το σώμα, (πεδίο “body”) του pull request: το συγκεκριμένο πεδίο του pull request, περιέχει πληροφορίες σχετικά με τις αλλαγές, τις εκδόσεις, την συμβατότητα, καθώς και τα commits που προτείνει το Dependabot. Αυτά τα δεδομένα, βρίσκονται σε μορφή HTML, και το εργαλείο καλείται να μετατρέψει αυτές τις πληροφορίες σε δομημένη μορφή JSON, έτσι ώστε να είναι πιο ευανάγνωστες, ξεκάθαρες και δομημένες οι τροποποιήσεις και γενικά όλες οι πληροφορίες που παρέχονται.

5) Ανάκτηση και αποθήκευση δεδομένων και πληροφοριών: τα δεδομένα και οι πληροφορίες που ανακτώνται για κάθε pull request ξεχωριστά από το GitHub με author το Dependabot, πρέπει να αποθηκεύονται σε αρχεία τύπου JSON.

6) Ονομασία αρχείων αποθήκευσης των δεδομένων και των πληροφοριών των pull requests: το σύστημα πρέπει να αποθηκεύει τις πληροφορίες που ανακτώνται σε αρχεία που να τροποποιείται το όνομα τους με βάση την κατηγορία των pull requests,

τα φίλτρα δηλαδή, που παίρνει σαν είσοδο το πρόγραμμα από την γραμμή εντολών και να μην δημιουργούνται hardcoded files.

7) Διεπαφή Χρήστη – Command Line Interface (CLI): Ο προγραμματιστής πρέπει να έχει την δυνατότητα να τρέχει το εργαλείο εισάγοντας εντολές μέσω του πληκτρολογίου, δηλαδή από την γραμμή εντολών, ενώ από εκεί θα πρέπει να καθορίζει την κατηγορία των Pull requests που επιθυμεί να ανακτήσει από το GitHub, με τα διάφορα φίλτρα που υλοποιούνται, καθώς και το προσωπικό του έγκυρο GitHub API Token, ώστε να έχει δικαίωμα πρόσβασης και ανάκτησης δεδομένων από το GitHub.

3.3 Μη Λειτουργικές Απαιτήσεις

Οι Μη Λειτουργικές Απαιτήσεις, αφορούν ποιοτικά χαρακτηριστικά του συστήματος, όπως για παράδειγμα η χρηστικότητα, η απόδοση, η επεκτασιμότητα, η φορητότητα και η ασφάλεια.

1) Ασφάλεια: Το προσωπικό GitHub API Token του προγραμματιστή, έχει δικαίωμα μόνο ο ίδιος ο προγραμματιστής να το γνωρίζει, καθώς δεν αποθηκεύεται σε κάποιο αρχείο και ταυτόχρονα δεν είναι προσβάσιμο από κανένα άλλο προγραμματιστή και γενικότερα, χρήστη του εργαλείου.

2) Φορητότητα: Το εργαλείο μπορεί να εκτελεστεί και να λειτουργήσει σε όλα τα λειτουργικά συστήματα, όπως για παράδειγμα Windows, macOS και Linux, χωρίς να απαιτούνται επιπλέον τροποποιήσεις στον κώδικα ή ενσωμάτωση επιπρόσθετων βιβλιοθηκών.

3) Επεκτασιμότητα: Το σύστημα πρέπει να υλοποιηθεί με τέτοιο τρόπο, έτσι ώστε μελλοντικά, να μπορούν να ενσωματώνονται με ευκολία αλλαγές και βελτιώσεις στον τρόπο λειτουργίας του εργαλείου και στα pull requests, όπως για παράδειγμα καινούρια templates.

4) Χρηστικότητα: Το σύστημα πρέπει να παρέχει στον προγραμματιστή αρκετές επεξηγήσεις και περιγραφή του τρόπου λειτουργίας του, καθώς και εγχειρίδιο για τον τρόπο χρήσης του εργαλείου, όπως για παράδειγμα ο τρόπος εισαγωγής και

εκτέλεσης από την γραμμή εντολών (φίλτρα, GitHub API token), πρέπει να επεξηγείται από το σύστημα.

Επιπρόσθετα, το σύστημα πρέπει να εμφανίζει τα κατάλληλα μηνύματα για την πρόοδο της ανάκτησης και επεξεργασίας των pull requests.

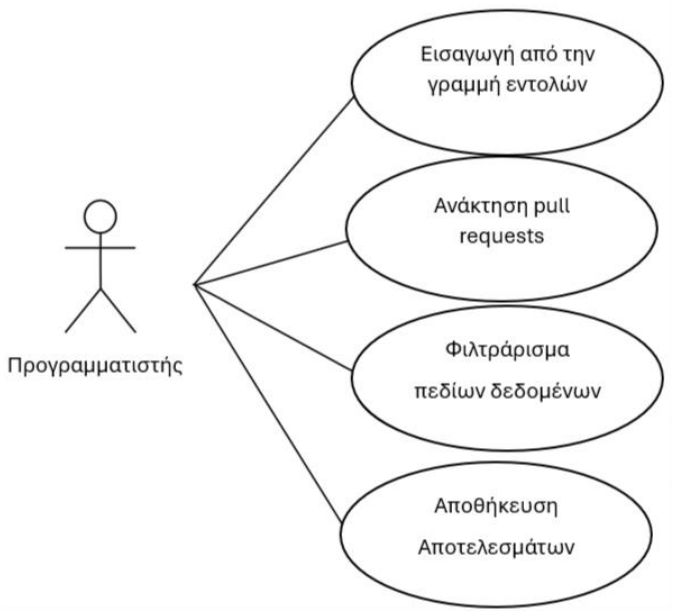
5) Απόδοση: Το σύστημα καλείται να είναι αρκετά αποδοτικό και γρήγορο, δηλαδή να ανακτά τα pull requests (μέχρι 5000 ανά ώρα που είναι το επιτρεπτό όριο), να τα επεξεργάζεται και να τα αποθηκεύει σε ένα αρκετά σύντομο και επιτρεπτό χρονικό διάστημα.

3.4 Use – Case Diagram

Ο πίνακας 3.4.1, επεξηγεί και περιγράφει το κάθε Use – Case που απεικονίζεται στην εικόνα 3.4.2.

Use Case	**Περιγραφή**
Εισαγωγή από την γραμμή εντολών (Command Line Interface)	Ο προγραμματιστής εισάγει από την γραμμή εντολών, το έγκυρο GitHub API Token του, καθώς και τα κριτήρια – φίλτρα της επιλογής του, για τα ανάλογα pull requests που επιθυμεί να ανακτήσει
Ανάκτηση Pull Requests	Το σύστημα καλεί GitHub API, για την ανάκτηση των επιλεγμένων pull requests
Φιλτράρισμα πεδίων δεδομένων και parsing	Το εργαλείο εφαρμόζει regular expressions και “φιλτράρει” τα επιλεγμένα πεδία δεδομένων
Αποθήκευση Αποτελεσμάτων σε Αρχείο	Το σύστημα αποθηκεύει τα δεδομένα που έχουν εξαχθεί στο ανάλογο αρχείο μορφής JSON

Πίνακας 3.4.1



Εικόνα 3.4.2 Use – Case Diagram

3.5 Μοντέλο Ανάπτυξης

Τα πληροφοριακά συστήματα, για την ανάπτυξη τους και την παραγωγή περιεχομένου και λογισμικού, χρησιμοποιούν μοντέλα κύκλου ζωής. Το κάθε μοντέλο περιγράφει την διαδικασία – μεθοδολογία που ακολούθησε ο προγραμματιστής ή η ομάδα προγραμματιστών, μέχρι και την ολοκλήρωση του συστήματος.

Στην παρούσα διπλωματική εργασία, για την ανάπτυξη του συστήματος, ακολουθήθηκε το μοντέλο του καταρράκτη. Η διαδικασία ανάπτυξης του συστήματος, ακολουθεί συγκεκριμένα διαδοχικά στάδια:

1) Ανάλυση απαιτήσεων και προδιαγραφών
2) Σχεδιασμός συστήματος
3) Ανάπτυξη κώδικα και υλοποίηση συστήματος
4) Δοκιμή συστήματος
5) Λειτουργία και συντήρηση

Κεφάλαιο 4

Σχεδιασμός Συστήματος

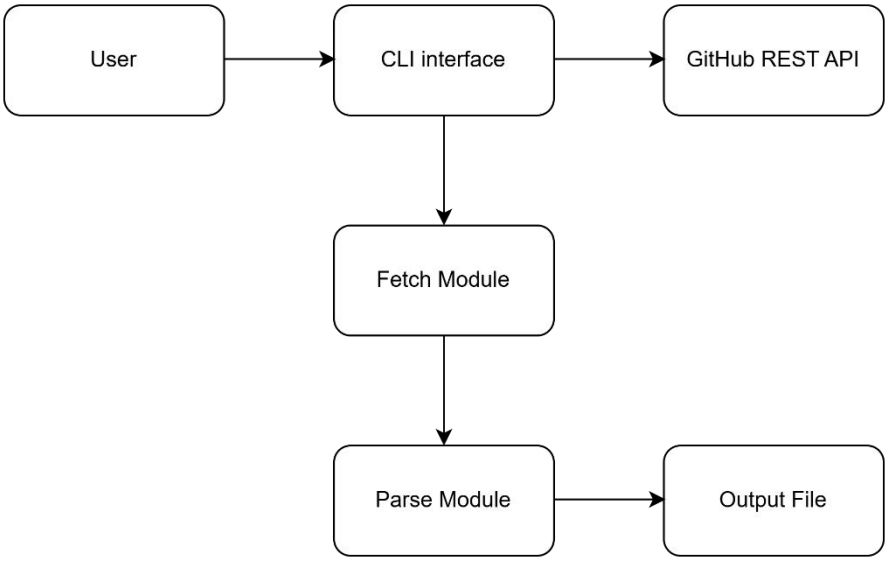
4.1 Αρχιτεκτονική Συστήματος

Η Αρχιτεκτονική, όπως απεικονίζεται στην εικόνα 4.1.1, του συστήματος που υλοποιήθηκε για την διαδικασία της ανάκτησης, επεξεργασίας και αποθήκευσης των Pull requests που προτείνει το Dependabot, από το GitHub, βασίζεται σε ένα απλό interface γραμμής εντολών, Command Line Interface (CLI), η οποία αλληλοεπιδρά με το GitHub REST API, μέσω των HTTP requests, έτσι ώστε να έχει την δυνατότητα να ανακτήσει δεδομένα από τα pull requests του Dependabot.

Στο επίπεδο του χρήστη, ο προγραμματιστής, αλληλοεπιδρά με το σύστημα μέσω της γραμμής εντολών (CLI), εισάγοντας τις παραμέτρους που ζητά το εργαλείο, δηλαδή το GitHub API Token, καθώς και τα διάφορα φίλτρα της επιλογής του (repository, start date, end date, merged/unmerged, state – open/closed), ανάλογα με τα pull requests που ο προγραμματιστής επιθυμεί να ανακτήσει.

Στο επίπεδο λειτουργίας, αρχικά, εκτελούνται authenticated HTTP requests, καθώς και API calls, ούτως ώστε να γίνει ταυτοποίηση του προγραμματιστή και στην συνέχεια να έχει το δικαίωμα της ανάκτησης των pull requests.

Στην συνέχεια, στο επίπεδο λειτουργίας, πραγματοποιείται η εξαγωγή των απαραίτητων πληροφοριών, ενώ σε συνδυασμό με τα Regular Expressions της Python, από το αρχείο patterns.json, εξάγονται και επεξεργάζονται οι πληροφορίες και τα δεδομένα και τελικά αποθηκεύονται σε αρχείο μορφής JSON.



Εικόνα 4.1.1 Διάγραμμα Αρχιτεκτονικής Συστήματος

4.1.1 Βιβλιοθήκες και Πακέτα Python

Σε αυτό το κεφάλαιο, θα δούμε τις βιβλιοθήκες και τα πακέτα που χρησιμοποιήθηκαν στην ανάπτυξη του συστήματος.

1) requests: χρησιμοποιείται για να εκτελούνται HTTP κλήσεις (calls) στο GitHub REST API

2) json: χρησιμοποιείται για την εκτέλεση σειριοποίησης και αποσειριοποίησης των δεδομένων JSON

3) re: χρησιμοποιείται για την μετατροπή (parsing) HTML/SVG περιεχομένου μέσω κανονικών εκφράσεων (regular expressions)

4) time: χρησιμοποιείται για τις καθυστερήσεις ανάμεα στα API calls σε περιπτώσεις όπου ξεπερνιέται το όριο (rate limit)

5) datetime, timedelta: χρησιμοποιείται για την διαχείριση και το φιλτράρισμα των δεδομένων και των πληροφοριών με βάση τις ημερομηνίες

6) os: χρησιμοποιείται για την αλληλεπίδραση με το λειτουργικό σύστημα

7) glob: χρησιμοποιείται για να εντοπίζονται τα αρχεία με χρήση patterns, (για παράδειγμα: *.json)

4.1.2 Τεχνολογίες και Εργαλεία

Σε αυτό το κεφάλαιο, θα δούμε τις τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν στην παρούσα διπλωματική εργασία.

JSON

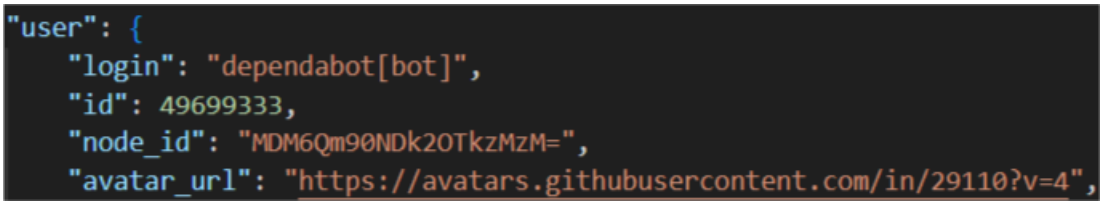
Το JSON (JavaScript Object Notation), είναι μια τυπική μορφή αρχείου που χρησιμοποιεί κείμενο για την μεταφορά αντικειμένων δεδομένων σε τύπους δεδομένων πίνακα.

Επιπρόσθετα, η μορφή αρχείου JSON, αποτελεί μια αρκετά ευανάγνωστη και ξεκάθαρη μορφή αναπαράστασης δομημένων δεδομένων, η οποία βοηθά τις εφαρμογές στο να αναλύσουν και να δημιουργούν αρχεία.

Το εργαλείο που αναπτύχθηκε, για την παρούσα διπλωματική εργασία, χρησιμοποιεί την μορφή τύπου JSON, για την διαδικασία της εξαγωγής και της αποθήκευσης των δεδομένων και των πληροφοριών των pull requests, που ανακτώνται και επεξεργάζονται, δίνοντας μια δομημένη και ευανάγνωστη μορφή, για περαιτέρω επεξεργασία, ανάγνωση και χρήση των δεδομένων.

Στην εικόνα 4.2.2, παρουσιάζεται ένα κομμάτι δομημένης μορφής JSON. Το πεδίο “user”, αντιστοιχεί σε ένα αντικείμενο με τύπο dictionary, με πεδία τα οποία αποτελούνται από

ζεύγη δεδομένων, “key – value”, για παράδειγμα το πεδίο “login”, αντιστοιχεί στο “key”, ενώ το “dependabot[bot]”, αντιστοιχεί στο “value”.



Εικόνα 4.1.2 Κομμάτι Δομημένης Μορφής JSON

Python

Η Python [23], είναι μια γλώσσα προγραμματισμού η οποία δημιουργήθηκε από τον Guido van Rossum και κυκλοφόρησε για πρώτη φορά το 1991.

Η Python, είναι μια αρκετά ευέλικτη, ευανάγνωστη, user – friendly γλώσσα προγραμματισμού, η οποία περιλαμβάνει πληθώρα έτοιμων βιβλιοθηκών, συναρτήσεων και ταυτόχρονα δομών δεδομένων, και για αυτόν τον λόγο επιλέχθηκε για την ανάπτυξη του εργαλείου της παρούσας διπλωματικής εργασίας.

Επιπρόσθετα, θεωρήθηκε η κατάλληλη γλώσσα προγραμματισμού, για το παρόν εργαλείο, καθώς υποστηρίζει κλήσεις REST API, μορφή JSON, ενώ ταυτόχρονα περιέχει έτοιμες βιβλιοθήκες για την ευκολία χειρισμού των δεδομένων και των πληροφοριών των pull requests.

VS Code

Το εργαλείο ανάπτυξης κώδικα που χρησιμοποιήθηκε για την παρούσα διπλωματική εργασία, είναι το Microsoft Visual Studio Code (VS Code [17]), το οποίο δημιουργήθηκε από την Microsoft και κυκλοφόρησε για πρώτη φορά το 2015.

REST API

Το REST API [19] [22] (Representational State Transfer Application Programming Interface), είναι ένα μέσο αλληλεπίδρασης μεταξύ υπολογιστικών συστημάτων, έσω του διαδικτύου.

Το API λειτουργεί μέσω HTTP calls (κλήσεων), χρησιμοποιώντας τις μεθόδους GET, POST, PUT, DELETE, οι οποίες μέθοδοι ευθυγραμμίζονται με τις λειτουργίες CRUD (Create, Read, Update, Delete), οι οποίες χρησιμοποιούνται για τον χειρισμό των πόρων στο διαδίκτυο.

Επιπρόσθετα, οι απαντήσεις που επιστρέφονται από τις μεθόδους που έχουν αναφερθεί, έχουν τις μορφές, κυρίως JSON, XML, HTML κλπ.

Στην παρούσα διπλωματική εργασία, έχουμε χρησιμοποιήσει το REST API του GitHub, με κύριο σκοπό να ανακτήσουμε δεδομένα και πληροφορίες από τα pull requests που δημιουργεί το Dependabot στο GitHub.

Παρακάτω οι πληροφορίες και τα πεδία που ανακτούμε από κάθε pull request, τα οποία θεωρήθηκαν ως τα σημαντικότερα δεδομένα:

“id”: το μοναδικό αναγνωριστικό του pull request, είναι απαραίτητο για την αναγνώριση του pull request και την διαχείριση τους από τα APIs
“number”: ο αριθμός του pull request, χρήσιμος για αναφορά και παραπομπές
“title”: ο τίτλος του pull request, ο οποίος περιγράφει συνοπτικά τον σκοπό του κάθε pull request, για να καταλάβει ο χρήστης περί τίνος πρόκειται
“state”: δείχνει την κατάσταση του pull request, αν είναι open ή closed, και είναι χρήσιμο για την παρακολούθηση της ροής της εργασίας

“draft”: δείχνει αν το pull request είναι έτοιμο για να συγχωνευτεί, ή να αξιολογηθεί και περιέχει τις τιμές true ή false

“user”: ο δημιουργός του pull request, και είναι χρήσιμο για επικοινωνία μαζί του ή ακόμη και συνεργασία

“created_at”: δείχνει το χρονικό σημείο στο οποίο έχει δημιουργηθεί το pull request

“updated_at”: δείχνει το χρονικό σημείο στο οποίο έχει ενημερωθεί το pull request

“closed_at”: δείχνει το χρονικό σημείο στο οποίο έχει κλείσει το pull request

“merged_at”: δείχνει το χρονικό σημείο στο οποίο έχει συγχωνευτεί το pull request

“merge_commit_sha”: το sha number του commit το οποίο έχει πραγματοποιηθεί κατά την συγχώνευση, χρησιμεύει στην ιχνηλάτηση των αλλαγών στον κώδικα

“html_url”: ο σύνδεσμος του pull request στο GitHub, ο οποίος μας δείχνει όλες τις πληροφορίες σχετικά με το pull request

“labels”: περιέχει ετικέτες που βοηθούν στην κατηγοριοποίηση και στο φιλτράρισμα των pull requests

“body”: περιέχει πληροφορίες και δεδομένα, τα οποία περιγράφονται τα πεδία τους πιο κάτω, ανάμεσα στις αγκύλες (“{}”)

{

“title”: ο τίτλος του pull request, ο οποίος περιγράφει συνοπτικά τον σκοπό του κάθε pull request, για να καταλάβει ο χρήστης περί τίνος πρόκειται

“url”: ο σύνδεσμος με τις πληροφορίες για τις αλλαγές, που περιέχει συνολικά όλες τις αλλαγές ανά έκδοση

“latest_version”: δείχνει την πιο πρόσφατη έκδοση του Pull request

“dependabot_compatibility_score_link”: το link που περιέχει μέσα το ποσοστό (%) συμβατότητας της ενημέρωσης, σε τι ποσοστό είναι δηλαδή ασφαλής και ταυτόχρονα συμβατή

“dependabot_compatibility_score”: περιέχει το ποσοστό (%) συμβατότητας της ενημέρωσης, σε τι ποσοστό είναι δηλαδή ασφαλής και ταυτόχρονα συμβατή

“release_notes”: καταγράφονται οι αλλαγές και οι διαφορές ανά έκδοση, και είναι σημαντικό για να καταλάβουμε τις επιπτώσεις του pull request στον κώδικα

“changelog”: καταγράφονται οι αλλαγές και οι διαφορές ανά έκδοση, και είναι σημαντικό για να καταλάβουμε τις επιπτώσεις του pull request στον κώδικα

“commits”: περιέχει τις επιμέρους αλλαγές του pull request, χρήσιμο στους προγραμματιστές ώστε να ελέγξουν την ποιότητα και το περιεχόμενο αυτών των αλλαγών.

}

4.2 Λειτουργίες Συστήματος

Σε αυτό το κεφάλαιο, περιγράφονται οι λειτουργίες του συστήματος που έχουν υλοποιηθεί. Αρχικά, το εργαλείο αναπτύχθηκε σε γλώσσα προγραμματισμού Python και χρησιμοποιεί regular expressions από αρχείο τύπου JSON.

4.2.1 Ανάκτηση Pull Requests του Dependabot από το GitHub

Στο αρχικό στάδιο, το εργαλείο ζητά από τον προγραμματιστή, να εισάγει από την γραμμή εντολών, το GitHub API token του, έτσι ώστε να έχει δικαίωμα να ανακτήσει pull requests από το GitHub. Ο χρήστης, αφού εισάγει το token του, έχει την επιλογή να διαλέξει ανάμεσα σε διάφορα φίλτρα, δηλαδή να επιλέξει ποια pull requests επιθυμεί να ανακτήσει από το GitHub. Η διαδικασία αυτή γίνεται στο αρχείο final.py.

Τα φίλτρα που έχει στην διάθεση του να επιλέξει είναι:

i) repository/owner: αν ο χρήστης εισάγει το όνομα του αποθετηρίου και τον ιδιοκτήτη του, μπορεί να ανακτήσει pull requests μόνο από ένα συγκεκριμένο repository

ii) specific dates: ο προγραμματιστής έχει την επιλογή να ανακτήσει pull requests μεταξύ συγκεκριμένων ημερομηνιών, εισάγοντας start date και end date από την γραμμή εντολών

iii) merged/unmerged: εισάγοντας αυτό το φίλτρο, παρέχεται η δυνατότητα από το σύστημα να ανακτηθούν τα pull requests που έχουν συγχωνευτεί (merged) ή όχι (unmerged), ανάλογα με την είσοδο του χρήστη

iv) state: υπάρχει η δυνατότητα, επιπλέον, να ανακτηθούν pull requests, ανάλογα με την κατάσταση (state) στην οποία βρίσκονται, αν είναι δηλαδή open ή closed

Επιπρόσθετα, κάθε φορά που αποστέλλονται HTTP requests, και ξεκινά η διαδικασία ανάκτησης pull requests, γίνεται διαχείριση από το εργαλείο, των status codes, δηλαδή την

κατάσταση που βρίσκεται η διαδικασία, καθώς και η διαχείριση του rate limit. Ακόμη, το σύστημα χρησιμοποιεί pagination (100 prs/page), έτσι ώστε να μειωθούν τα API calls.

Τα pull requests που ανακτώνται αποθηκεύονται σε αρχείο JSON.

4.2.2 Επεξεργασία Δεδομένων και Πληροφοριών

Αφού ανακτηθούν τα δεδομένα των επιλεγμένων pull requests από το GitHub, προχωράμε στο αρχείο parsing.py. Σε αυτό το αρχείο, αναπτύχθηκε ο κώδικας, στον οποίο γίνεται η επεξεργασία των δεδομένων.

Αρχικά, διαβάζονται, τα δεδομένα και οι πληροφορίες των pull requests που έχουν ανακτηθεί και έχουν αποθηκευτεί στην συνέχεια στο αρχείο JSON. Στην συνέχεια, ο κώδικας καλεί το αρχείο patterns.json, το οποίο περιλαμβάνει τα regular expressions τα οποία θα βοηθήσουν στην επεξεργασία των δεδομένων που έχουν την μορφή HTML.

Ακολούθως, ο κώδικας στο αρχείο parsing.py, εξάγει το πεδίο “body” που περιλαμβάνεται σε κάθε pull request, το οποίο περιέχει δεδομένα για τις αλλαγές ανά έκδοση, commits, συμβατότητα των αλλαγών, καθώς και άλλες επιπρόσθετες πληροφορίες που έχουν αναφερθεί πιο πάνω (η παράγραφος με την επεξήγηση του JSON). Στην συνέχεια, το εργαλείο αφού εξάγει την πληροφορία που είναι σε μορφή HTML, χρησιμοποιεί τα regular expressions από το αρχείο patterns.json, έτσι ώστε να μετατρέψει τα δεδομένα σε μορφή JSON και να τα αποθηκεύσει σε αρχεία τύπου JSON.

4.2.3 Αποθήκευση Δεδομένων σε Αρχείο

Στην συνέχεια, αφού γίνει η επεξεργασία των δεδομένων στο αρχείο parsing.py, στο ίδιο αρχείο, γίνεται η επιλογή πεδίων για την εξαγωγή της σημαντικότερης πληροφορίας από το κάθε pull request, όπως αναλύθηκε πιο πάνω (η παράγραφος με την επεξήγηση του JSON).

Τέλος, αφού έχουν επιλεχθεί τα σημαντικότερα πεδία, ούτως ώστε να μην χαθεί κάποια σημαντική πληροφορία, αποθηκεύονται σε αρχεία τύπου JSON, με ονομασία ανάλογα με τα φίλτρα που έχουν επιλεχθεί για την ανάκτηση των pull requests.

Στην εικόνα 4.2.3.1 παρουσιάζεται ένα παράδειγμα αποθήκευσης των πιο σημαντικών πεδίων των pull requests.

Εικόνα 4.2.3.1 Σημαντικά Πεδία Pull Request

Κεφάλαιο 5

Αξιολόγηση Συστήματος

5.1 Έλεγχος Συστήματος

Μετά το τέλος της ανάπτυξης του κώδικα, αλλά και κατά την διάρκεια της συγγραφής, για την παρούσα διπλωματική εργασία, ακολούθησε η διαδικασία ελέγχου του συστήματος.

Η διαδικασία ελέγχου, από την αρχή της συγγραφής του κώδικα μέχρι και το τέλος, περιλαμβάνει τα εξής βήματα:

1) Ανάκτηση δοκιμαστικού pull request: Αρχικά, πραγματοποιήθηκε ένα API call, με σκοπό την ανάκτηση ενός δοκιμαστικού pull request, έτσι ώστε να μπορέσω να ξεκινήσω την διαδικασία της επεξεργασίας.

2) Υλοποίηση επεξεργασίας Pull request: Στην συνέχεια, ξεκίνησε η ανάπτυξη του κώδικα ο οποίος θα βοηθούσε στην επεξεργασία των pull requests, με στόχο την εξαγωγή της πληροφορίας από το πεδίο “body” του pull request.

3) Δοκιμές parsing: Αφού ξεκίνησε η διαδικασία της ανάπτυξης του κώδικα, δηλαδή τα regular expressions σε αρχείο τύπου JSON, και του κώδικα σε Python, γινόταν συνεχής εκτέλεση και έλεγχος με είσοδο το “body” του pull request, που είχε ανακτηθεί, με στόχο κάθε φορά η επίτευξη της σωστής εξαγωγής των πεδίων που περιείχε το “body” (σε HTML περιεχόμενο), καθώς και η αποθήκευση τους σε αρχείο με δομημένη μορφή JSON. Με αυτό τον τρόπο, καθοριζόταν κάθε φορά η

ακρίβεια των regular expressions. (αυτό ήταν το πιο απαιτητικό και σύνθετο κομμάτι της διπλωματικής μου εργασίας).

4) Δοκιμές με Διάφορα Φίλτρα: Με την ολοκλήρωση της βασικής λειτουργικότητας, όταν δηλαδή επιτεύχθηκε ο σκοπός, δηλαδή το parsing, η εξαγωγή όλων των πεδίων από το συγκεκριμένο pull request καθώς και η αποθήκευση τους σε αρχείο JSON, ξεκίνησε η δοκιμή του εργαλείου με διαφορετικές παραμέτρους από την γραμμή εντολών, έτσι ώστε να ανακτηθούν pull requests με διαφορετικά κριτήρια, ανάλογα με τα φίλτρα εισαγωγής, για να υπάρξει η δυνατότητα επεξεργασίας και εξαγωγής δεδομένων σε ένα ευρύτερο πεδίο από pull requests.

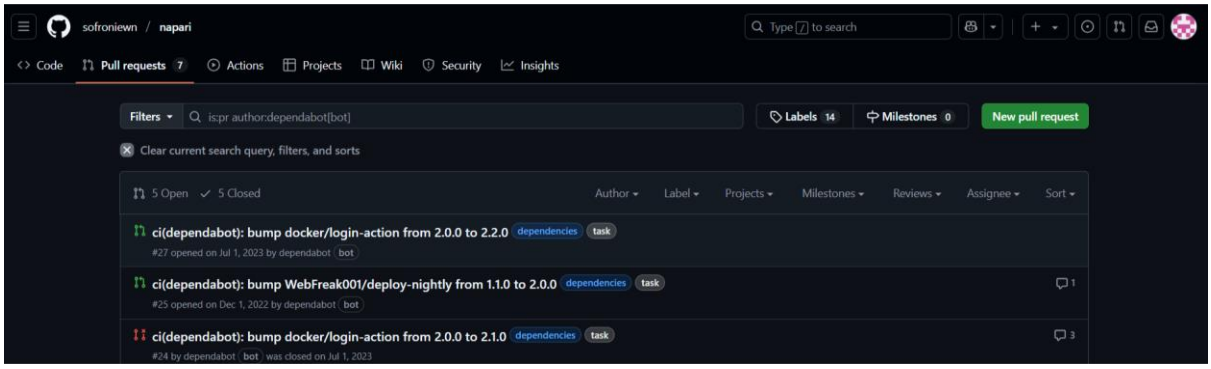
5) Τελικός Έλεγχος: Τελικά, με την ανάκτηση των pull requests από διάφορα αποθετήρια, εκτελέστηκε έλεγχος, σε διάφορα pull requests. Συγκεκριμένα, πραγματοποιήθηκε σύγκριση, ένα προς ένα τα πεδία που περιέχονταν στα “body” του pull requests.

Παρακάτω παρατίθενται παραδείγματα εκτέλεσης και επεξεργασίας:

Παράδειγμα 1:

```
michaelides79@APOLLONAS:~/ptixiaki2$ python3 final.py --token                    --repo sofroniewn/napari
Fetched 10 PRs (page 1)
Stored into prs_sofroniewn_napari.json
Read prs_sofroniewn_napari.json
Wrote prs_sofroniewn_napari.json_parsed.json
```

Εικόνα 5.1.1 Εισαγωγή Παραμέτρων από Γραμμή Εντολών



Εικόνα 5.1.2 sofroniewn/napari Repository

Στην εικόνα 5.1.1, απεικονίζεται η είσοδος από τον χρήστη στην γραμμή εντολών. Αρχικά, ο χρήστης εισάγει το GitHub API token του, το οποίο είναι η συμβολοσειρά μετά το “--token”, δηλαδή “ghp…Cc1”.

Ακολούθως, η επόμενη παράμετρος εκτέλεσης, είναι το αποθετήριο και ο ιδιοκτήτης του αποθετηρίου, δηλαδή μετά το “--repo”, η συμβολοσειρά “ sofroniewn/napari ”. Αυτό σημαίνει πως το αποθετήριο από το οποίο θα ανακτηθούν τα pull requests ονομάζεται “ sofroniewn” και το όνομα του ιδιοκτήτη “sofroniewn/napari”.

Από το παράδειγμα εκτέλεσης, φαίνεται ότι ανακτήθηκαν συνολικά 10 pull requests από το συγκεκριμένο αποθετήριο.

Επιπρόσθετα, τα pull requests που ανακτήθηκαν, αποθηκεύτηκαν στο αρχείο “prs_sofroniewn_napari.json”, το οποίο δημιουργήθηκε δυναμικά, σύμφωνα με το όνομα του αποθετηρίου και του ιδιοκτήτη του.

Τέλος, τα pull requests ανακτήθηκαν και έχουν αποθηκευτεί στο αρχείο “prs_sofroniewn_napari.json”, επεξεργάστηκαν και έχουν εξαχθεί από αυτά, οι πιο σημαντικές πληροφορίες οι οποίες αποθηκεύτηκαν στο αρχείο “prs_sofroniewn_napari_parsed.json”.

Εικόνα 5.1.3 Release Notes JSON File

▼ Release notes

Sourced from docker/login-action's releases.

v2.2.0

- Switch to actions-toolkit implementation by `@crazy-max` in docker/login-action#409 docker/login-action#470 docker/login-action#476
- Bump `@aws-sdk/client-ecr` and `@aws-sdk/client-ecr-public` to 3.347.1 in docker/login-action#524 docker/login-action#364 docker/login-action#363
- Bump minimatch from 3.0.4 to 3.1.2 in docker/login-action#354
- Bump json5 from 2.2.0 to 2.2.3 in docker/login-action#378
- Bump http-proxy-agent from 5.0.0 to 7.0.0 in docker/login-action#509
- Bump https-proxy-agent from 5.0.1 to 7.0.0 in docker/login-action#508

Εικόνα 5.1.4 Release Notes Section Changes


```
"commits": [
    {
        "info": "Merge pull request",
        "sha": "465a07811f14bebb1938fbed4728c6a1ff8901fc",
        "link": "https://github.com/docker/login-action/commit/465a07811f14bebb1938fbed4728c6a1ff8901fc",
        "issue_link": null,
        "number": null
    },
    {
        "info": "Merge pull request",
        "sha": "360b4b5fefbd590a50c2ff0c234dbd9fa6b08759",
        "link": "https://github.com/docker/login-action/commit/360b4b5fefbd590a50c2ff0c234dbd9fa6b08759",
        "issue_link": null,
        "number": null
    },
    {
        "info": "update generated content",
        "sha": "c156700b2346aa481b427d93907904c3d5f50f72",
        "link": "https://github.com/docker/login-action/commit/c156700b2346aa481b427d93907904c3d5f50f72",
        "issue_link": null,
        "number": null
    },
    {
        "info": "bump",
        "sha": "f605cf145efa455c90bb0b4759f5907b4279df40",
        "link": "https://github.com/docker/login-action/commit/f605cf145efa455c90bb0b4759f5907b4279df40",
```


Εικόνα 5.1.5 Commits JSON File

▼ Commits

- `465a078` Merge pull request #524 from crazy-max/bump-aws
- `360b4b5` Merge pull request #512 from jhihruei/change/update-gitlab-readme
- `c156700` update generated content
- `f605cf1` bump `@aws-sdk/client-ecr` and `@aws-sdk/client-ecr-public` to 3.347.1
- `2a93a3e` Merge pull request #508 from docker/dependabot/npm_and_yarn/https-proxy-agent...
- `422e90f` update generated content
- `bc8c4d0` build(deps): bump https-proxy-agent from 5.0.1 to 7.0.0
- `052c2c4` Merge pull request #509 from docker/dependabot/npm_and_yarn/http-proxy-agent-...
- `beabccd` update generated content
- `b56ed1c` build(deps): bump http-proxy-agent from 5.0.0 to 7.0.0

Εικόνα 5.1.6 Commits Section Changes

Εικόνα 5.1.7 Πεδίο Compatibility Score

Στην εικόνα 5.1.3, παρουσιάζεται κομμάτι από την τελική μορφή αποθήκευσης του επεξεργασμένου pull request, στο αρχείο “prs_sofroniewn_napari_parsed.json”.

Αρχικά, πραγματοποιήθηκε έλεγχος για το compatibility score, το οποίο είναι άγνωστο όπως φαίνεται στην εικόνα 5.1.7. Έτσι, στην εικόνα 5.1.3, ορθά το πεδίο του compatibility score είναι null, ενώ είναι συμπληρωμένο μόνο το πεδίο με το compatibility score link, που σημαίνει ότι γίνεται σωστή η εξαγωγή της πληροφορίας.

Ακολούθως, το πεδίο latest version στην εικόνα 5.1.3 εξάγεται σωστά, καθώς το τελευταίο version είναι το 2.2.0, όπως φαίνεται στην εικόνα 5.1.4.

Επιπρόσθετα, στην εικόνα 5.1.3, στον πίνακα με τα “changes” που ανήκει στο template release notes, κάτω από την έκδοση 2.2.0, συγκριτικά με την εικόνα 5.1.4, εξάγονται ορθά οι πληροφορίες με τα πεδία “description” το οποίο αποτελεί την περιγραφή της αλλαγής, “link”, “by” το οποίο αναπαριστά τον συγγραφέα (author), και “number” ο οποίος είναι ο αριθμός της συγκεκριμένης αλλαγής. Οποιοδήποτε πεδίο είναι null, συνεπάγεται ότι δεν υπάρχει κάποια πληροφορία για εξαγωγή.

Στο ίδιο μοτίβο, η ίδια διαδικασία ελέγχου που πραγματοποιήθηκε στο template release notes, διαπράχθηκε και στο κομμάτι με τα commits, όπου ξαναέγινε η ίδια διαδικασία για τα αντίστοιχα πεδία, όπως αναπαρίστανται στις εικόνες 5.1.5 και 5.1.6.

Παράδειγμα 2:

Εικόνα 5.1.8 Εισαγωγή Παραμέτρων από Γραμμή Εντολών

Στην εικόνα 5.1.8, απεικονίζεται η είσοδος από τον χρήστη στην γραμμή εντολών. Αρχικά, ο χρήστης εισάγει το GitHub API token του, το οποίο είναι η συμβολοσειρά μετά το “--token”, δηλαδή “ghp…Cc1”.

Ακολούθως, η επόμενη παράμετρος εκτέλεσης, είναι το αποθετήριο και ο ιδιοκτήτης του αποθετηρίου, δηλαδή μετά το “--repo”, η συμβολοσειρά “ odoo/odoo ”. Αυτό σημαίνει πως το αποθετήριο από το οποίο θα ανακτηθούν τα pull requests ονομάζεται “ odoo” και το όνομα του ιδιοκτήτη “ odoo/odoo ”.

Στην συνέχεια, η επόμενη παράμετρος εκτέλεσης, είναι η κατάσταση state, δηλαδή θα ανακτηθούν όλα τα pull requests που έχουν state closed.

Από το παράδειγμα εκτέλεσης, φαίνεται ότι ανακτήθηκαν συνολικά 4 pull requests από το συγκεκριμένο αποθετήριο.

Επιπρόσθετα, τα pull requests που ανακτήθηκαν, αποθηκεύτηκαν στο αρχείο “prs_odoo_state_closed.json”, το οποίο δημιουργήθηκε δυναμικά, σύμφωνα με το όνομα του αποθετηρίου και του ιδιοκτήτη του.

Τέλος, τα pull requests ανακτήθηκαν και έχουν αποθηκευτεί στο αρχείο “prs_odoo_state_closed.json”, επεξεργάστηκαν και έχουν εξαχθεί από αυτά, οι πιο σημαντικές πληροφορίες οι οποίες αποθηκεύτηκαν στο αρχείο “prs_odoo_state_closed_parsed.json”.

Εικόνα 5.1.9 Release Notes JSON File

▼ Release notes

Sourced from pillow's releases.

8.1.1

https://pillow.readthedocs.io/en/stable/releasenotes/8.1.1.html

8.1.0

https://pillow.readthedocs.io/en/stable/releasenotes/8.1.0.html

Changes

- Fix TIFF OOB Write error #5175 [@radarhere]
- Fix for Buffer Read Overrun in PCX Decoding #5174 [@radarhere]
- Fix for SGI Decode buffer overrun #5173 [@radarhere]
- Fix OOB Read when saving GIF of xsize=1 #5149 [@wiredfool]
- Add support for PySide6 #5161 [@hugovk]
- Moved QApplication into one test #5167 [@radarhere]
- Use disposal settings from previous frame in APNG #5126 [@radarhere]
- Revert "skip wheels on 3.10-dev due to wheel#354" #5163 [@radarhere]
- Better _binary module use #5156 [@radarhere]

Εικόνα 5.1.10 Release Notes Section Changes

```
"commits": [
    {
        "info": "8.1.1 version bump",
        "sha": "741d8744a54bedbc49f16922c61a06fcb3681f53",
        "link": "https://github.com/python-pillow/Pillow/commit/741d8744a54bedbc49f16922c61a06fcb3681f53",
        "issue_link": null,
        "number": null
    },
    {
        "info": "Added 8.1.1 release notes to index",
        "sha": "179cd1c8f94aabc47e9e522e01683ea9aadbd3a5",
        "link": "https://github.com/python-pillow/Pillow/commit/179cd1c8f94aabc47e9e522e01683ea9aadbd3a5",
        "issue_link": null,
        "number": null
    },
    {
        "info": "Update CHANGES.rst [ci skip]",
        "sha": "7d296653da045e18b379c991797f933e054a7476",
        "link": "https://github.com/python-pillow/Pillow/commit/7d296653da045e18b379c991797f933e054a7476",
        "issue_link": null,
        "number": null
    },
    {
        "info": "Credits",
        "sha": "d25036fca7c8658b698492088361453bb20073e2",
        "link": "https://github.com/python-pillow/Pillow/commit/d25036fca7c8658b698492088361453bb20073e2",
        "issue_link": null,
        "number": null
    },
```


Εικόνα 5.1.11 Commits JSON File

▼ Commits

- 741d874 8.1.1 version bump
- 179cd1c Added 8.1.1 release notes to index
- 7d29665 Update CHANGES.rst [ci skip]
- d25036f Credits
- 973a4c3 Release notes for 8.1.1
- 521dab9 Use more specific regex chars to prevent ReDoS
- 8b8076b Fix for CVE-2021-25291
- e25be1e Fix negative size read in TiffDecode.c
- f891baa Fix OOB read in SgiRleDecode.c
- cbfdde7 Incorrect error code checking in TiffDecode.c

Εικόνα 5.1.12 Commits Section Changes

Εικόνα 5.1.13 Πεδίο Compatibility Score

Στην εικόνα 5.1.9, παρουσιάζεται κομμάτι από την τελική μορφή αποθήκευσης του επεξεργασμένου pull request, στο αρχείο “prs_odoo_parsed.json”.

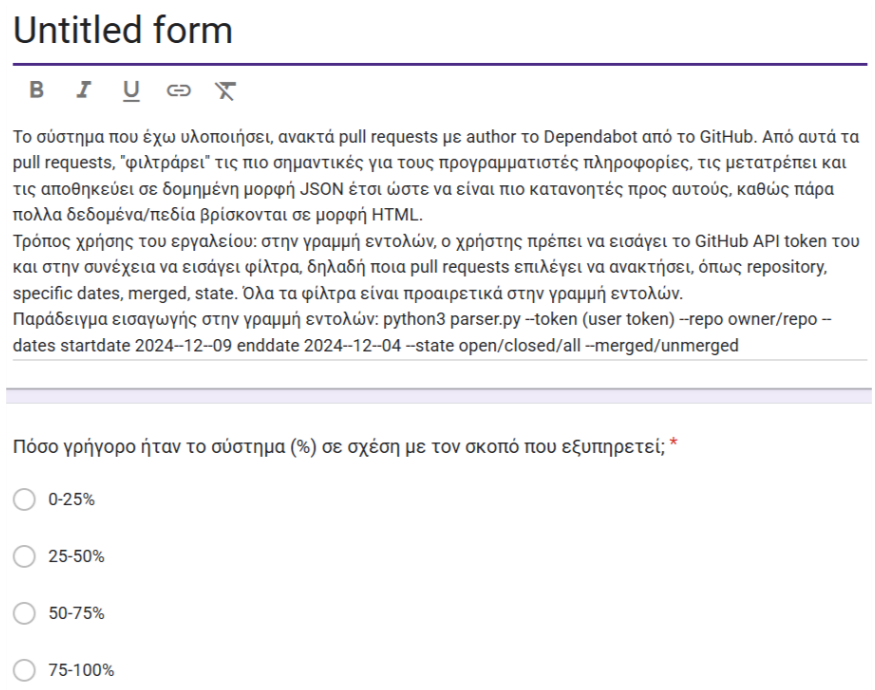
Αρχικά, πραγματοποιήθηκε έλεγχος για το compatibility score, το οποίο είναι άγνωστο όπως φαίνεται στην εικόνα 5.1.13. Έτσι, στην εικόνα 5.1.9, ορθά το πεδίο του compatibility score είναι 92%, ενώ είναι συμπληρωμένο μόνο το πεδίο με το compatibility score link, που σημαίνει ότι γίνεται σωστή η εξαγωγή της πληροφορίας.

Ακολούθως, το πεδίο latest version στην εικόνα 5.1.9 εξάγεται σωστά, καθώς το τελευταίο version είναι το 8.1.1, όπως φαίνεται στην εικόνα 5.1.10.

Επιπρόσθετα, στην εικόνα 5.1.9, στον πίνακα με τα “changes” που ανήκει στο template release notes, κάτω από την έκδοση 8.1.0, συγκριτικά με την εικόνα 5.1.10, εξάγονται ορθά οι πληροφορίες με τα πεδία “description” το οποίο αποτελεί την περιγραφή της αλλαγής, “link”, “by” το οποίο αναπαριστά τον συγγραφέα (author), και “number” ο οποίος είναι ο αριθμός της συγκεκριμένης αλλαγής. Οποιοδήποτε πεδίο είναι null, συνεπάγεται ότι δεν υπάρχει κάποια πληροφορία για εξαγωγή.

Στο ίδιο μοτίβο, η ίδια διαδικασία ελέγχου που πραγματοποιήθηκε στο template release notes, διαπράχθηκε και στο κομμάτι με τα commits, όπου ξαναέγινε η ίδια διαδικασία για τα αντίστοιχα πεδία, όπως αναπαρίστανται στις εικόνες 5.1.11 και 5.1.12.

5.2 Feedback από χρήστες


Untitled form

Το σύστημα που έχω υλοποιήσει, ανακτά pull requests με author το Dependabot από το GitHub. Από αυτά τα pull requests, "φιλτράρει" τις πιο σημαντικές για τους προγραμματιστές πληροφορίες, τις μετατρέπει και τις αποθηκεύει σε δομημένη μορφή JSON έτσι ώστε να είναι πιο κατανοητές προς αυτούς, καθώς πάρα πολλα δεδομένα/πεδία βρίσκονται σε μορφή HTML.
Τρόπος χρήσης του εργαλείου: στην γραμμή εντολών, ο χρήστης πρέπει να εισάγει το GitHub API token του και στην συνέχεια να εισάγει φίλτρα, δηλαδή ποια pull requests επιλέγει να ανακτήσει, όπως repository, specific dates, merged, state. Όλα τα φίλτρα είναι προαιρετικά στην γραμμή εντολών.
Παράδειγμα εισαγωγής στην γραμμή εντολών: python3 parser.py --token (user token) --repo owner/repo --dates startdate 2024--12--09 enddate 2024--12--04 --state open/closed/all --merged/unmerged

Πόσο γρήγορο ήταν το σύστημα (%) σε σχέση με τον σκοπό που εξυπηρετεί; *

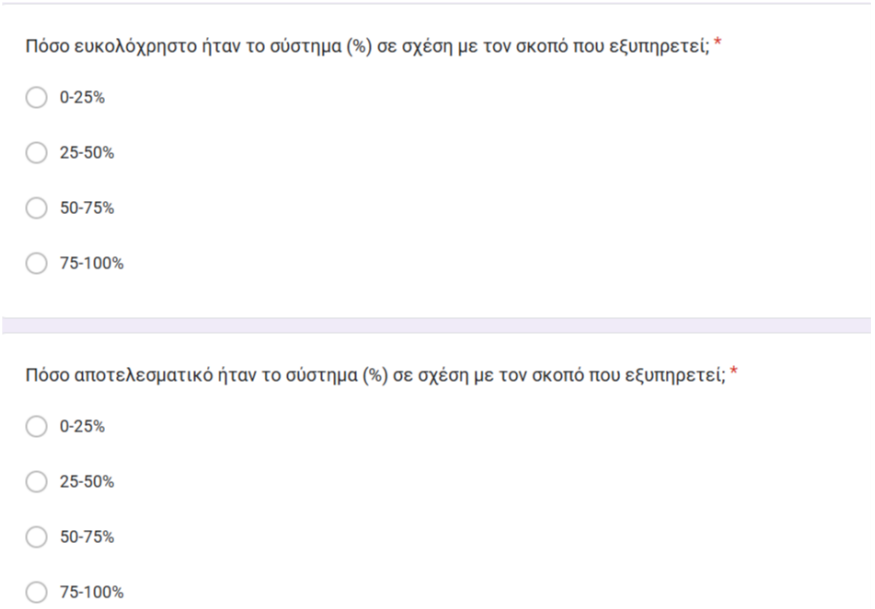
0-25%

25-50%

50-75%

75-100%

Εικόνα 5.2.1 Ερωτηματολόγιο για feedback


Πόσο ευκολόχρηστο ήταν το σύστημα (%) σε σχέση με τον σκοπό που εξυπηρετεί; *

0-25%

25-50%

50-75%

75-100%

Πόσο αποτελεσματικό ήταν το σύστημα (%) σε σχέση με τον σκοπό που εξυπηρετεί; *

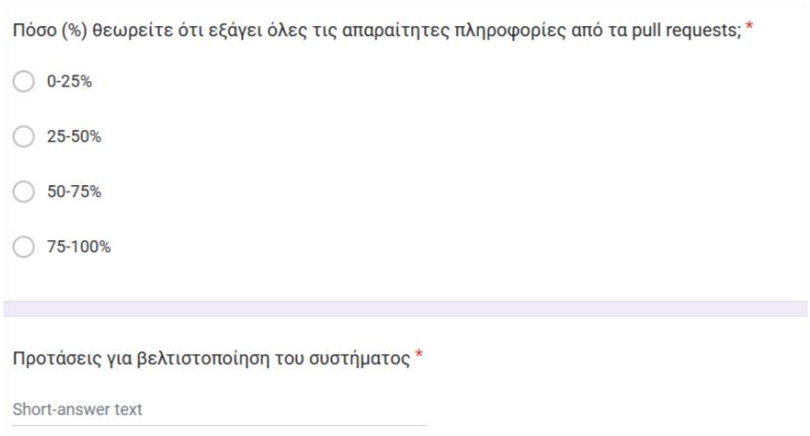
0-25%

25-50%

50-75%

75-100%

Εικόνα 5.2.2 Ερωτηματολόγιο για feedback



Εικόνα 5.2.3 Ερωτηματολόγιο για feedback

Κατά την προσπάθεια ολοκλήρωσης και βελτιστοποίησης του παρόντος συστήματος, πραγματοποιήθηκε μια μικρή έρευνα με σκοπό να πάρουμε και τις απόψεις άλλων προγραμματιστών, κατά πόσο είναι εύχρηστο, γρήγορο και αποτελεσματικό το εργαλείο που έχει αναπτυχθεί. Οι χρήστες που έλαβαν το ερωτηματολόγιο για την έρευνα ήταν απόφοιτοι του τμήματος Πληροφορικής.

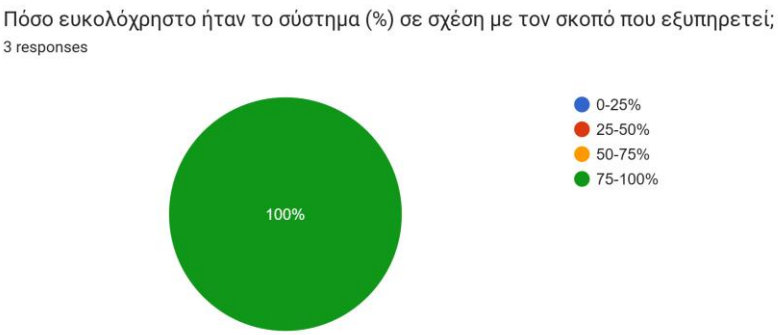
Αρχικά, δόθηκε μια σύντομη επεξήγηση του εργαλείου μαζί με τον σκοπό του, ενώ ταυτόχρονα περιεγράφηκε ο τρόπος εκτέλεσης του κώδικα, από την γραμμή εντολών.

Οι ερωτήσεις που έχουν γίνει στους προγραμματιστές είναι οι εξής (όπως φαίνεται στις εικόνες 5.2.1, 5.2.2, 5.2.3):

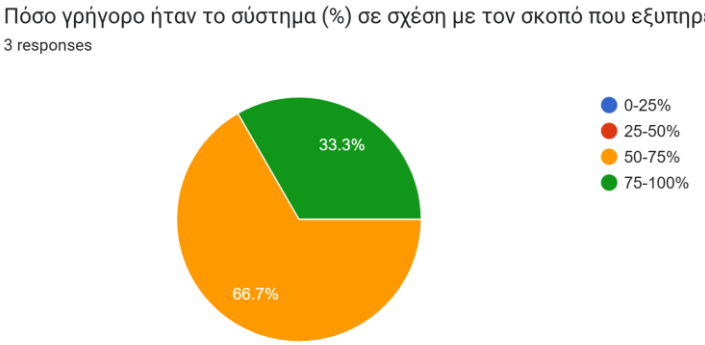
1) Πόσο γρήγορο ήταν το σύστημα (%) σε σχέση με τον σκοπό που εξυπηρετεί;
2) Πόσο ευκολόχρηστο ήταν το σύστημα (%) σε σχέση με τον σκοπό που εξυπηρετεί;
3) Πόσο αποτελεσματικό ήταν το σύστημα (%) σε σχέση με τον σκοπό που εξυπηρετεί;
4) Πόσο (%) θεωρείτε ότι εξάγει όλες τις απαραίτητες πληροφορίες από τα pull requests;
5) Προτάσεις για βελτιστοποίηση του συστήματος.

Οι χρήστες, απάντησαν με ποσοστό επί τοις εκατό (%), με 0 το ελάχιστο και 100 το μέγιστο, ενώ έχουν καταθέσει επίσης μερικές προτάσεις για το πώς μπορεί να βελτιωθεί το εργαλείο.

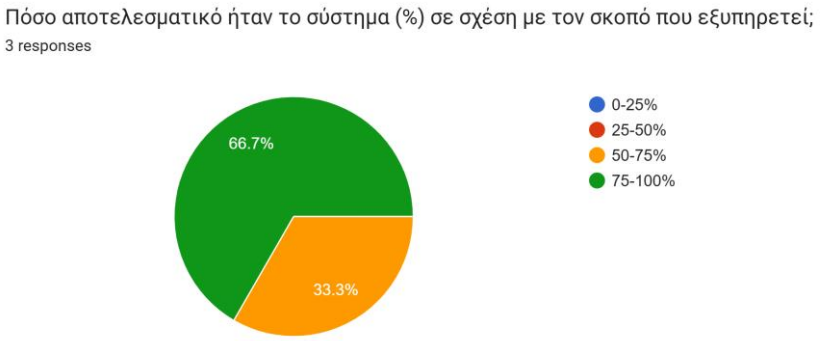
Τα αποτελέσματα της έρευνας απεικονίζονται στις γραφικές παραστάσεις 5.2.4, 5.2.5, 5.2.6, 5.2.7, ενώ οι προτάσεις για βελτίωση από τους προγραμματιστές φαίνονται στην εικόνα 5.2.8.



Εικόνα 5.2.4 Αποτελέσματα feedback

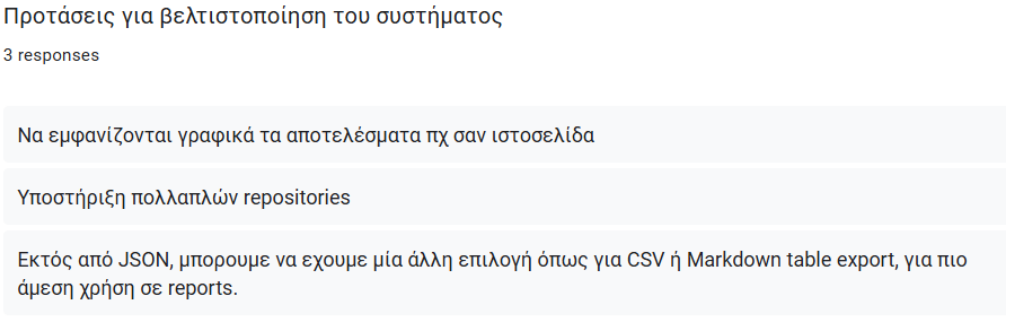


Εικόνα 5.2.5 Αποτελέσματα feedback



Εικόνα 5.2.6 Αποτελέσματα feedback

Εικόνα 5.2.8 Αποτελέσματα feedback



Εικόνα 5.2.9 Προτάσεις χρηστών

5.3 Περιορισμοί

1) Όριο ανάκτησης pull requests από το GitHub: το GitHub περιορίζει την ποσότητα των pull requests που ο προγραμματιστής μπορεί να ανακτήσει ανά ώρα, υπάρχει rate limit 5000 API calls ανά ώρα. Ο προγραμματιστής οφείλει να σέβεται αυτά τα όρια.

2) Απαιτείται κάποια εξοικείωση του χρήστη με το CLI

3) Περιορισμός template: ο κώδικας που αναπτύχθηκε, λειτουργεί μόνο για τα templates που συναντήθηκαν μέχρι στιγμής. Αν όμως υπάρξει νέο template, για παράδειγμα Security, δεν θα γίνει σωστά η επεξεργασία και η εξαγωγή των δεδομένων

4) Δεν μπορεί να υποστηρίξει πολλαπλά repositories

5) Όταν γίνεται η μετατροπή του πεδίου body από HTML σε JSON, σε μια αλλαγή σε κάποιο version, μόνο ο πρώτος αριθμός της αλλαγής γίνεται parse

6) Δεν υλοποιήθηκε η εμφάνιση της βοήθειας, δηλαδή δεν εμφανίζει το πως πρέπει να χρησιμοποιούνται ακριβώς τα φίλτρα, μόνο μέσα στον κώδικα

Κεφάλαιο 6

Συμπεράσματα

6.1 Σύνοψη

Συνοψίζοντας, στην παρούσα διπλωματική εργασία, πραγματοποιήθηκε η ανάπτυξη ενός συστήματος, σε γλώσσα προγραμματισμού Python. Συγκεκριμένα, ο ρόλος του εργαλείου είναι να ανακτά, να επεξεργάζεται, να μετατρέπει και να αποθηκεύει τις σημαντικότερες πληροφορίες και τα δεδομένα από τα Pull Requests, με δημιουργό το Dependabot στο GitHub.

Επιπρόσθετα, το εργαλείο επιτρέπει στον προγραμματιστή να επιλέγει συγκεκριμένα pull requests που επιθυμεί να ανακτήσει, εισάγοντας καθορισμένα κριτήρια (repo/owner, merged/unmerged, date, state) από την γραμμή εντολών.

Στο ίδιο μοτίβο , το εργαλείο πραγματοποιεί μετατροπή (parsing) χρησιμοποιώντας τα regular expressions/patterns και τελικά εκτελεί αποθήκευση σε αρχεία με δομημένη μορφή JSON.

Η διαδικασία που εκτελέστηκε, ακολούθησε το μοντέλο του καταρράκτη, ενώ τελικά πραγματοποιήθηκε έλεγχος για την αποτελεσματικότητα και την ευχρηστία του συστήματος μέσω χειροκίνητων διαδικασιών.

Ακόμη, δημιουργήθηκε ένα σύντομο, απλό, αλλά ταυτόχρονα επεξηγηματικό ερωτηματολόγιο, το οποίο στάλθηκε σε προγραμματιστές, οι οποίοι προγραμματιστές ήταν απόφοιτοι του τμήματος της Πληροφορικής του Πανεπιστημίου Κύπρου, οι οποίοι με την

σειρά τους κατέθεσαν το δικό τους Feedback, καθώς και τις προτάσεις τους, με στόχο τις μελλοντικές βελτιστοποιήσεις του συστήματος.

Σύμφωνα με τα αποτελέσματα που έχουν καταθέσει οι χρήστες, το εργαλείο αποδείχτηκε πως είναι αρκετά γρήγορο, αποτελεσματικό και εύχρηστο, ενώ υπάρχουν αρκετά περιθώρια για την μελλοντική του βελτίωση.

6.2 Μελλοντικές Επεκτάσεις

Για ένα σύστημα σαν αυτό, που έχει υλοποιηθεί στην παρούσα διπλωματική εργασία, το οποίο ανακτά pull requests, τα επεξεργάζεται, μετατρέπει τα δεδομένα τους από την μορφή HTML σε δομημένη μορφή JSON, μπορούν να υπάρξουν πολλές μελλοντικές βελτιστοποιήσεις.

Μελλοντικά, το εργαλείο μπορεί να εξελιχθεί σε διάφορους τομείς, όπως:

1) Το εργαλείο θα μπορούσε να υποστηρίζει την ταυτόχρονη αποστολή πολλαπλών GitHub API Tokens από ομάδες προγραμματιστών, έτσι ώστε να μπορεί να παρακαμφτεί και να ξεπεραστεί το όριο (rate – limit δηλαδή), των 5000 pull requests ανά ώρα, και έτσι με αυτόν τον τρόπο να υπάρχει η πιθανότητα ανάκτησης 5000 Pull requests ανά token.

2) Ανάπτυξη Graphical User Interface (GUI), έτσι ώστε να μπορεί να είναι το σύστημα πιο ευκολόχρηστο, και ταυτόχρονα να είναι πιο ευχάριστη η εμπειρία του προγραμματιστή, χρησιμοποιώντας το εργαλείο.

3) Ακόμη, μια μελλοντική βελτίωση που θα μπορούσε να πραγματοποιηθεί στο εργαλείο, είναι η υποστήριξη περισσότερων μορφών εξόδου, εκτός από την μορφή JSON, όπως για παράδειγμα CSV, ή ακόμη και Markdown Table Export, για πιο άμεση χρήση σε reports και σε βάσεις δεδομένων.

4) Επιπλέον, μια άλλη σημαντική βελτιστοποίηση, θα ήταν η υποστήριξη πολλαπλών repositories, δηλαδή να παρέχεται η δυνατότητα στον χρήστη, να ανακτά pull

requests από διάφορα repositories, και όχι μόνο από ένα συγκεκριμένο, όπως γίνεται στο παρόν στάδιο.

5) Τέλος, μια ακόμη μελλοντική προσθήκη που θα ήταν αρκετά χρήσιμη, είναι η υποστήριξη περισσότερων bots, εκτός του Dependabot, το οποίο είναι το εργαλείο αυτοματοποίησης που δουλέψαμε σε αυτήν την διπλωματική εργασία.

Βιβλιογραφία

[1] About Dependabot - GitHub Docs https://docs.github.com/en/code-security/getting-started/dependabot-quickstart-guide

[2] About pull requests – GitHub Docs https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

[3] About JSON (JavaScript Object Notation) - GitHub Docs https://github.com/topics/json

[4] About repositories - GitHub Docs https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories

[5] Git - About Version Control - Git SCM Documentation https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[6] "Merging a pull request" - GitHub Docs https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/merging-a-pull-request

[7] About branches - GitHub Docs https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches

[8] About npm - npm Docs https://docs.npmjs.com/about-npm

[9] pip documentation - pip User Guide https://pip.pypa.io/en/stable/

[10] Introduction to Maven - Apache Maven Project https://maven.apache.org/what-is-maven.html

[11] Software Library - ScienceDirect https://www.freecodecamp.org/news/what-is-a-library-in-programming/

[12] What Is a Package? – Oracle Docs What is a Package?

[13] Vulnerabilities in Python Packages - Mahmoud Alfadel, Diego Elias Costa, Emad Shihab https://das.encs.concordia.ca/pdf/alfadel_saner2021.pdf

[14] Rest API - https://www.geeksforgeeks.org/rest-api-introduction/

[15] HTML: HyperText Markup Language - https://developer.mozilla.org/en-US/docs/Web/HTML

[16] About Python - https://www.w3schools.com/python/python_intro.asp

[17] vs code - https://en.wikipedia.org/wiki/Visual_Studio_Code#:~:text=Visual%20Studio%20Code%2C%20commonly%20referred,embedded%20version%20control%20with%20Git