



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Διπλωματική Εργασία

**Αναπαράσταση Γνώσης
με χρήση μη σχεσιακών ΒΔ
και αξιοποίησή της από
Ψηφιακούς Προσωπικούς Βοηθούς**

Νικόλαος Γιαρέλης
Α.Μ. 235740

Επιβλέπων
Καθηγητής Χρήστος Μπούρας

Συνεπιβλέπων
Καθηγητής Νίκος Καρακαπηλίδης

Μέλος Επιτροπής Αξιολόγησης
<Βαθμίδα Ονοματεπώνυμο>

Πάτρα, 2019

© Copyright συγγραφέας Νικόλαος Γιαρέλης, 2019

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών & Πληροφορικής του Πανεπιστημίου Πατρών δεν υποδηλώνει απαραίτητως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

*Αφιερωμένο στην μητέρα μου και στον πατέρα μου, οι οποίοι με έμαθαν από μικρή ηλικία ότι:
“Amat Victoria Curam.” / “Η νίκη αγαπά την προετοιμασία.”
Λατινική παροιμία, (Ανώνυμος)*

Περίληψη

Είναι σαφές ότι η αύξηση του μεγέθους των επιχειρήσεων και της πολυπλοκότητας των έργων τους δυσκολεύει τη συνολική διαχείρισή τους. Αξιοποιώντας τεχνολογίες αιχμής, η παρούσα εργασία προτείνει μια σειρά λύσεων σε κλασσικά προβλήματα Διαχείρισης Έργων. Για την αντιμετώπιση των προβλημάτων αυτών, τα σχετικά δεδομένα και οι μεταξύ τους συνδέσεις εισάγονται σε μια μη σχεσιακή βάση δεδομένων (graph database), πετυχαίνοντας έτσι μια ιδιαίτερα χρήσιμη και επεκτάσιμη οπτική τους αναπαράσταση. Για την αξιοποίηση των παραπάνω δεδομένων κατά την υποστήριξη καλά τεκμηριωμένων αποφάσεων, εισάγουμε στην προσέγγισή μας την τεχνολογία των ψηφιακών βοηθών (chatbots), οι οποίοι μπορούν να επικοινωνούν με τον Project Manager σε φυσική γλώσσα, να κατανοήσουν τις απαιτήσεις του και να απαντήσουν συνήθη ερωτήματα κατά τη διαχείριση ενός έργου. Μέσω της εφαρμογής των παραπάνω τεχνολογιών, η προτεινόμενη προσέγγιση συνεισφέρει στην ανακάλυψη πολύτιμης γνώσης και, γενικότερα, στη βελτίωση της ποιότητας της Διαχείρισης Έργων.

Abstract

It is clear that the growth of companies' size and the ever increasing complexity of their projects make their overall management more challenging. By utilizing cutting edge technologies, this thesis suggests an array of solutions to classic Project Management problems. To deal with these problems, the relevant data and connections between them are inserted in a non-relational (graph) database, which yields a highly useful and scalable visualization of them. Aiming to support well informed decisions in diverse project management issues, our approach also integrates the digital assistants (chatbots) technology, which enables communication with the stakeholders in natural language, understanding their requests and answering common questions that arise during the Project Management practice. Through the application of the aforementioned technologies, the recommended approach contributes to the discovery of valuable knowledge and, generally speaking, to the improvement of Project Management quality.

Περιεχόμενα

1	Εισαγωγή.....	1
1.1	Διαχείριση Έργου.....	1
1.2	Μεθοδολογίες Διαχείρισης Έργου	2
	Kanban	2
	Scrum	2
1.3	Σύγχρονες λύσεις διαχείρισης έργου.....	4
	Wrike.....	4
	Monday	7
	Scoro	9
	Jira	10
	Slack.....	12
1.4	Στόχος της εργασίας.....	13
1.5	Διάρθρωση της εργασίας.....	13
2	Graph Databases.....	15
2.1	Εξέλιξη των βάσεων δεδομένων	15
2.2	Η ανάγκη για τις Graph Databases.....	17
2.3	Neo4j Architecture	18
2.4	Graph databases vs. Relational databases vs. No-SQL databases.....	20
2.5	Cypher vs. SQL.....	21
2.6	Case Study: Jira Issues	23
3	Αξιοποίηση Μέτρων ομοιότητας σε ζητήματα Project Management	33
3.1	Εισαγωγή.....	33
	Καλός προσδιορισμός των στόχων του Project.....	33
	Διαχείριση Πόρων	33
	Διαχείριση Ρίσκου	33
	Κατανομή ανθρώπων σε υποέργα.....	34
3.2	Βασικά Μέτρα Ομοιότητας.....	34
3.2.0	Ορισμοί.....	36
3.2.1	Ευκλείδεια απόσταση	36
3.2.2	Ομοιότητα Cosine	36
3.2.3	Ομοιότητα Jaccard	36

3.2.4	Ομοιότητα Συσχέτισης Pearson	37
3.3	Ομοιότητα μεταξύ issues.....	37
3.3.1	Ομοιότητα Jaccard	38
3.3.2	Ομοιότητα Cosine	42
3.3.3	Ομοιότητα Pearson.....	47
3.3.4	Ευκλείδεια Απόσταση	53
3.4	Ομοιότητα μεταξύ προγραμματιστών	57
3.4.1	Ομοιότητα Jaccard	62
3.4.2	Ομοιότητα Cosine	66
3.5	Γράφος Εξάρτησης – PageRank.....	71
4	Αξιοποίηση Ψηφιακών Προσωπικών Βοηθών σε ζητήματα Project Management	73
4.1	Εισαγωγή.....	73
4.2	Σύγχρονες εφαρμογές των bots	75
4.3	Ανάπτυξη bots για επίλυση ζητημάτων Project Management	76
4.4	Εισαγωγή στο wit.ai	78
4.5	Ανάπτυξη ενός chatbot για επίλυση ζητημάτων Project Management	83
4.5.1	Απαιτήσεις εφαρμογής.....	83
4.5.2	Βήματα υλοποίησης εφαρμογής.....	94
4.5.3	Δομή λογισμικού εφαρμογής	95
4.6	Γραφική Διεπαφή εφαρμογής.....	97
4.7	Παραδείγματα χρήσης.....	101
5	Σύνοψη – Συμπεράσματα.....	109
	Βιβλιογραφία - Αναφορές	111

Λίστα Εικόνων

Εικόνα 1.1: Παράδειγμα Πίνακα Kanban (Πηγή: [62]).	3
Εικόνα 1.2: Παράδειγμα Πίνακα Scrum (Πηγή: [63]).	4
Εικόνα 1.3: Κύρια Οθόνη Wrike (Πηγή: [52]).	5
Εικόνα 1.4: Δευτερεύουσα Οθόνη Wrike – Οπτικοποίηση υποέργων του έργου (Πηγή: [53]).	6
Εικόνα 1.5: Δευτερεύουσα Οθόνη Wrike – Οπτικοποίηση χρόνου εκτέλεσης του έργου (Πηγή: [53]).	6
Εικόνα 1.6: Οθόνη Διαχείρισης project του Monday (Πηγή: [54]).	7
Εικόνα 1.7: Οθόνη Εργασιών Ομάδας του Monday (Πηγή: [54]).	8
Εικόνα 1.8: Οθόνη Παραγγελιών Πελάτη του Monday (Πηγή: [54]).	8
Εικόνα 1.9: Κύρια Οθόνη Scoro (Πηγή: [55]).	9
Εικόνα 1.10: Κύρια Οθόνη Jira με Πίνακα Kanban (Πηγή: [56]).	11
Εικόνα 1.11: Πίνακας Scrum (Πηγή: [57]).	11
Εικόνα 1.12: Κύρια Οθόνη Slack (Πηγή: [59]).	12
Εικόνα 2.1: Παράδειγμα γράφου.	17
Εικόνα 2.2: Σύνοψη αρχιτεκτονικής του neo4j (Πηγή: [10]).	18
Εικόνα 2.3: Βασικό μοντέλο της αρχιτεκτονικής του neo4j (Πηγή: [11]).	19
Εικόνα 2.4: Διαφορά μοντέλου σχεσιακής και graph database (Πηγή: [12]).	22
Εικόνα 2.5: Πίνακας αποτελεσμάτων – προγραμματιστές & αριθμός των issues.	24
Εικόνα 2.6: Πίνακας αποτελεσμάτων – προγραμματιστές & αριθμός των issues.	25
Εικόνα 2.7: Γράφος αποτελεσμάτων – προγραμματιστής (κόκκινος κόμβος) – issue (μπλε κόμβος).	25
Εικόνα 2.8: Γράφος μερικών αποτελεσμάτων – issue (μπλε κόμβος) – σχέσεις που τα συνδέουν (γκρι ετικέτες / ακμές).	26
Εικόνα 2.9: Γράφος μερικών αποτελεσμάτων – issue (μπλε κόμβος) – σχέσεις που τα συνδέουν (γκρι ετικέτες / ακμές).	27
Εικόνα 2.10: Πίνακας αποτελεσμάτων – μερικά ονόματα προγραμματιστών.	28
Εικόνα 2.11: Πίνακας αποτελεσμάτων – μερικά από τα ονόματα προγραμματιστών.	29
Εικόνα 2.12: Πίνακας αποτελεσμάτων – μερικά από τα ονόματα προγραμματιστών και το πλήθος των issues που έχει ο καθένας τους ανά κατηγορία.	30
Εικόνα 2.13: Πίνακας αποτελεσμάτων – όνομα issue και αριθμός issues που συνδέεται.	30
Εικόνα 2.14: Γράφος αποτελεσμάτων – issue (στο κέντρο κάθε συστάδας) που συνδέεται με πολλά issues.	31
Εικόνα 3.1: Μοντέλο διαδικασίας παραγωγής ετικετών.	35
Εικόνα 3.2: Ομοιότητα στο μοντέλο ετικετών.	38
Εικόνα 3.3: Πίνακας αποτελεσμάτων – HADOOP-15483, ομοιάζοντα issues και βαθμός ομοιότητας.	39
Εικόνα 3.4: Γράφος αποτελεσμάτων για τα παρόμοια issues και ετικέτες του issue HADOOP-15483.	40
Εικόνα 3.5: Πίνακας αποτελεσμάτων – ομοιάζοντα issues και βαθμός ομοιότητας.	41
Εικόνα 3.6: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι ετικέτες τους (πορτοκαλί κόμβοι).	42
Εικόνα 3.7: Πίνακας αποτελεσμάτων – HADOOP-14977, παρόμοια issues και ο βαθμός ομοιότητας τους.	43
Εικόνα 3.8: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι).	44

Εικόνα 3.9: Πίνακας μερικών αποτελεσμάτων – παρόμοια issues και ο βαθμός ομοιότητας τους.	45
Εικόνα 3.10: Γράφος μερικών αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι).....	47
Εικόνα 3.11: Πίνακας μερικών αποτελεσμάτων – HADOOP-15233, παρόμοια issues και ο βαθμός ομοιότητας τους.	48
Εικόνα 3.12: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι).....	49
Εικόνα 3.13: Πίνακας μερικών αποτελεσμάτων – παρόμοια issues και ο βαθμός ομοιότητας τους. ..	50
Εικόνα 3.14: Πίνακας μερικών αποτελεσμάτων – παρόμοια issues και ο βαθμός ομοιότητας τους. ..	52
Εικόνα 3.15: Πίνακας αποτελεσμάτων – HADOOP-15754, παρόμοια issues και ο βαθμός ομοιότητας τους.....	53
Εικόνα 3.16: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι).....	54
Εικόνα 3.17: Πίνακας μερικών αποτελεσμάτων – παρόμοια issues και ο βαθμός ομοιότητας τους. ..	55
Εικόνα 3.18: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι).....	57
Εικόνα 3.19: Μοντέλο οργάνωσης της βάσης, issues που συνδέονται μεταξύ τους (δεν οπτικοποιείται στο συγκεκριμένο παράδειγμα), issues που συνδέονται με ετικέτες (κοινές ή μη) και προγραμματιστές που συνδέονται με issues άμεσα και έμμεσα (μέσω μονοπατιού) με ετικέτες.....	58
Εικόνα 3.20: Πίνακας μερικών αποτελεσμάτων – όνομα προγραμματιστή, ετικέτες που «κληρονομεί» από τα δεδουλευμένα tasks και η συχνότητα κληρονομησής τους.	59
Εικόνα 3.21: Πίνακας μερικών αποτελεσμάτων – όνομα προγραμματιστή, ετικέτες που «κληρονομεί» από τα δεδουλευμένα tasks και η συχνότητα κληρονομησής τους.	60
Εικόνα 3.22: Πίνακας μερικών αποτελεσμάτων – όνομα προγραμματιστή, ετικέτα java που «κληρονομεί» από τα δεδουλευμένα tasks και η συχνότητα κληρονομησής της.	61
Εικόνα 3.23: Πίνακας αποτελεσμάτων – ο tom10271 και οι παρόμοιοι σε αυτόν προγραμματιστές..	62
Εικόνα 3.24: Γράφος αποτελεσμάτων – κόμβοι ετικέτες (πορτοκαλί), issues (μπλε) και προγραμματιστές (κόκκινο).	63
Εικόνα 3.25: Πίνακας αποτελεσμάτων – ζεύγη όμοιων προγραμματιστών και το ποσοστό ομοιότητας.	64
Εικόνα 3.26: Γράφος αποτελεσμάτων – κόμβους ετικέτες (πορτοκαλί), issues (μπλε) και προγραμματιστές (κόκκινο).	65
Εικόνα 3.27: Πίνακας αποτελεσμάτων – ο shn και οι παρόμοιοι σε αυτόν προγραμματιστές.	66
Εικόνα 3.28: Γράφος αποτελεσμάτων – κόμβοι ετικέτες (πορτοκαλί), issues (μπλε) και προγραμματιστές (κόκκινο).	67
Εικόνα 3.29: Πίνακας αποτελεσμάτων – ζεύγη όμοιων προγραμματιστών και το ποσοστό ομοιότητας.	68
Εικόνα 3.30: Γράφος μερικών αποτελεσμάτων – κόμβοι ετικέτες (πορτοκαλί), issues (μπλε) και προγραμματιστές (κόκκινο).	70
Εικόνα 3.31: Γράφος εξάρτησης (υπο-γράφος του συνολικού γραφήματος) – κόμβοι issues (μπλε) με ακμές τις σχέσεις εξάρτησης (depends_upon).	71
Εικόνα 3.32: Πίνακας μερικών αποτελεσμάτων – Issues με τα PageRank σκορ.....	72
Εικόνα 4.1: Στατιστική επί των ποσοστών επιτυχίας / αποτυχίας στο Project Management (Πηγή: [33]).....	76
Εικόνα 4.2: Στατιστική επί των μεγαλύτερων δυσκολιών στο Project Management (Πηγή: [34]).	77
Εικόνα 4.3: Γενικευμένο Μοντέλο εφαρμογής bot με χρήση του Wit.ai.....	78

Εικόνα 4.4: Κεντρική οθόνη προγραμματιστή bot στο wit.ai	79
Εικόνα 4.5: Εκμάθηση προθέσεων από προτάσεις στο wit.ai.....	80
Εικόνα 4.6: Γραφική παράσταση μεταξύ του επίπεδου ακρίβειας & εμπιστοσύνης (Πηγή: [39]).	82
Εικόνα 4.7: Αρχιτεκτονική του Project Management bot.	83
Εικόνα 4.8: Βήμα 1/5 της εγκατάστασης (Πηγή: [41])......	84
Εικόνα 4.9: Βήμα 2/5 της εγκατάστασης (Πηγή: [42])......	84
Εικόνα 4.10: Βήμα 3/5 της εγκατάστασης (Πηγή: [43])......	85
Εικόνα 4.11: Βήμα 4/5 της εγκατάστασης (Πηγή: [44])......	85
Εικόνα 4.12: Βήμα 5 / 5 της εγκατάστασης (Πηγή: [45])......	86
Εικόνα 4.13: Οθόνη εγγραφής / σύνδεσης χρήστη (Πηγή: [46])......	88
Εικόνα 4.14: Οθόνη διαχείρισης βάσεων (Πηγή: [47])......	89
Εικόνα 4.15: Οθόνη διαχείρισης βάσεων, δημιουργία βάσης 1 / 2 (Πηγή: [48])......	90
Εικόνα 4.16: Οθόνη διαχείρισης βάσεων, δημιουργία βάσης 2 / 2 (Πηγή: [49])......	91
Εικόνα 4.17: Οθόνη διαχείρισης βάσεων, κατόπιν δημιουργίας βάσης (Πηγή: [50])......	92
Εικόνα 4.18: Οθόνη διαχείρισης βάσης Jira, ενότητα Settings (Ρυθμίσεις).	93
Εικόνα 4.19: Ιεραρχία αρχείων του project.....	95
Εικόνα 4.20: Κύρια οθόνη γραφικής διεπαφής – Διάλογος με το chatbot.....	98
Εικόνα 4.21: Κύρια οθόνη γραφικής διεπαφής – Σφάλμα σύνδεσης με τον server.....	99
Εικόνα 4.22: Κύρια οθόνη γραφικής διεπαφής – Σφάλμα αποστολής μηνύματος στον server.	100
Εικόνα 4.23: Αποτελέσματα εύρεσης προγραμματιστών με την μεγαλύτερη δραστηριότητα.	101
Εικόνα 4.24: Αποτελέσματα εύρεσης των πιο εξαρτημένων έργων του συνολικού έργου.	101
Εικόνα 4.25: Μερική λίστα – παραγόμενων αποτελεσμάτων του bot.	102
Εικόνα 4.26: User story 1 – Διάλογος με το bot μέρος 1 / 3.....	103
Εικόνα 4.27: User story 1 – Διάλογος με το bot μέρος 2 / 3.....	104
Εικόνα 4.28: User story 1 – Μερική λίστα αποτελεσμάτων – Διάλογος με το bot μέρος 3 / 3.	105
Εικόνα 4.29: User story 2 – Διάλογος με το bot μέρος 1 / 3.....	106
Εικόνα 4.30: User story 2 – Διάλογος με το bot μέρος 2 / 3.....	107
Εικόνα 4.31: User story 2 – Πίνακας αποτελεσμάτων – Διάλογος με το bot μέρος 3 / 3.	108

Λίστα Πινάκων

Πίνακας 2.1: Σύγκριση διαφορετικών βάσεων δεδομένων.....	21
Πίνακας 4.1: Προαπαιτούμενες βιβλιοθήκες του project προς εγκατάσταση.....	87
Πίνακας 4.2: Περιγραφή αρχείων / φακέλων εφαρμογής.....	97

Συντομογραφίες

SaaS	Software as a Service
IoT	Internet of Things
DBMS	DataBase Management Systems
SQL	Structured Query Language
ISO	International Organization for Standardization
NLP	Natural Language Processing
TF	Term Frequency
TF-IDF	Term Frequency – Inverted Document Frequency
NLU	Natural Language Understanding
AI	Artificial Intelligence
ACID	Atomicity, Consistency, Isolation, Durability

Απόδοση Όρων

Project Management	Διαχείριση Έργου.
Project Manager	Διαχειριστής Έργου.
Software as a Service	Λογισμικό σαν υπηρεσία.
Graph Databases	Βάσεις Δεδομένων με οργάνωση γράφου.
Graph Algorithms	Αλγόριθμοι για γράφους.
Chatbots/bots /Virtual Assistants	Ψηφιακοί Βοηθοί.
Cloud (Computing)	(Υπολογιστική) νέφους.
Relational Databases	Σχεσιακές Βάσεις Δεδομένων.
Non-Relational	Μη Σχεσιακές Βάσεις Δεδομένων.
Big Data	Μεγάλα Δεδομένα.
Internet of Things	Διαδίκτυο των Πραγμάτων.
Labeled Property Graph Model	Μοντέλο γράφων ιδιοτήτων με ετικέτες.
Case Study	Μελέτη Περίπτωσης.
Tradeoff	Αντιστάθμιση.
Cluster	Ομάδα όμοιων αντικείμενων.
Clustering	Ομαδοποίηση αντικείμενων.
Natural Language Processing	Επεξεργασία φυσικής γλώσσας.
Term Frequency	Η συχνότητα με την οποία εμφανίζεται ένας όρος.
Term Frequency-Inverse Document Frequency	Η συχνότητα με την οποία εμφανίζεται ένας όρος στην πρόταση και η αντίστροφη συχνότητα με την οποία εμφανίζεται ένας όρος στο κείμενο.
Natural Language Understanding	Κατανόηση φυσικής γλώσσας.
Stemming	Διαδικασία περικοπής της κατάληξης ενός όρου με στόχο να επιστρέψουμε στο αρχικό λήμμα..
Lemmatization	Διαδικασία μετατροπής ενός όρου στο

	λεξιλογικό λήμμα από το οποίο προέρχεται.
Artificial Intelligence	Τεχνητή Νοημοσύνη.
Self-Learning	Αυτό μάθηση.
Machine Learning	Μηχανική Μάθηση.

1

Εισαγωγή

1.1 Διαχείριση Έργου

Η Διαχείριση Έργου έχει οριστεί ως το σύνολο ενεργειών που αφορούν την ανάλυση, τον σχεδιασμό, την εκτέλεση, τον έλεγχο και την περάτωση της δουλειάς μιας ομάδας ατόμων με αποτελεσματικό και αποδοτικό τρόπο, με σκοπό τη δημιουργία ενός συγκεκριμένου προϊόντος ή υπηρεσίας. Οι διαχειριστές εφαρμόζουν γνώσεις, δεξιότητες, εργαλεία και τεχνικές, με στόχο να ικανοποιήσουν όλες τις απαιτήσεις και τις προσδοκίες που έχουν θέσει οι συμμετέχοντες στο έργο. Για την ικανοποίηση αυτών, πρέπει να τηρηθεί μια λίστα περιορισμών που αφορούν παραμέτρους όπως ο χρόνος, το κόστος, και η ποιότητα ενός προϊόντος ή υπηρεσίας [51].

Σύμφωνα με τον συγγραφέα του βιβλίου «Μάνατζμεντ / Αρχές Διοίκησης Επιχειρήσεων», αρκετοί ειδήμονες της διαχείρισης έργου δίνουν έμφαση στον καθοριστικό ρόλο των εργαζομένων δηλ. των ανθρώπων που επιτυγχάνουν τους στόχους της επιχείρησης αλλά και στους ανθρώπους που έχουν τις απαιτούμενες ικανότητες να διαχειρισθούν το προσωπικό της επιχείρησης. Η διαχείριση έργου είναι η επίτευξη αποτελεσμάτων μέσω των (άλλων) ανθρώπων [51].

Πέρα από τους ανθρώπους και την πολυπλοκότητα του ίδιου του έργου, υπάρχουν και άλλοι παράγοντες που επηρεάζουν την διαχείριση έργου που βρίσκονται στο περιβάλλον της επιχείρησης, π.χ. οικονομία, κράτος, νόμοι, πελάτες, προμηθευτές, ανταγωνιστές. Επειδή, η διαχείριση έργου έχει εξελιχθεί σε μια αρκετά σύνθετη διαδικασία, έχουν πραγματοποιηθεί απόπειρες αυτοματοποίησης της με πληροφοριακά συστήματα διαχείρισης έργων.

Σύμφωνα με τους συγγραφείς του βιβλίου «A Guide to the Project Management Body of Knowledge» [59], ένα πληροφοριακό σύστημα διαχείρισης έργου ορίζεται ως: “Ένα πληροφοριακό σύστημα που αποτελείται από εργαλεία και τεχνικές χρησιμοποιούμενες ώστε να συλλέξουν, ενσωματώσουν και διανεμούν τα αποτελέσματα των διαδικασιών της διαχείρισης έργου. Το πληροφοριακό σύστημα χρησιμοποιείται για να υποστηρίξει όλους τους τομείς του project, από την αρχική μέχρι την τελική φάση του, και μπορεί να περιλαμβάνει χειροκίνητα και αυτοματοποιημένα συστήματα.”

1.2 Μεθοδολογίες Διαχείρισης Έργου

Kanban

Η μεθοδολογία Kanban, δημιουργήθηκε αρχικά σαν σύστημα διαχείρισης παραγωγής από τον Taiichi Ohno ένα κορυφαίο μηχανικό παραγωγής της Toyota [60]. Η μεθοδολογία Kanban στην διαχείριση έργου πρωτοεμφανίστηκε το 2004 όταν ο David J. Anderson βοήθησε μια μικρή ομάδα στην Microsoft η οποία είχε χαμηλή απόδοση. Η μέθοδος Kanban οδηγεί την ομάδα ανάπτυξης στην οπτικοποίηση της ροής του έργου, στον περιορισμό των ανοιχτών υποέργων του έργου σε κάθε φάση του και στην μέτρηση της κάθε φάσης του έργου. Ο πίνακας Kanban παρέχει την οπτικοποίηση του έργου, αναδεικνύει τα υποέργα που αποτελούν προτεραιότητα, και μπορεί να αναδείξει και σημεία του έργου που μπορούν να προκαλέσουν συμφόρηση. Σκοπός της μεθόδου είναι να ελαχιστοποιεί τα ανοιχτά υποέργα, αναπτύσσοντας μόνο εκείνα που χρειάζονται κάθε φορά. Η μεθοδολογία αυτή δίνει έμφαση στη σταθερή ροή μικρών παραδοτέων, έτσι ώστε να μην διαμερίζουν συνεχώς την προσοχή τους οι προγραμματιστές και να οπτικοποιείται η πρόοδος του έργου στον πελάτη, αλλά και να μειώνεται η συνολική πολυπλοκότητα του έργου χάριν στην κατάτμησή του σε μικρότερα και πιο διαχειρίσιμα υποέργα.

Οι βασικές αρχές της Kanban είναι:

- Οπτικοποίηση της ροής του έργου.
- Περιορισμός των ανοιχτών υποέργων.
- Μέτρηση & Διαχείριση της ροής.
- Σαφήνεια στις διαδικασίες του έργου.
- Συνεργατική βελτίωση (μέσω μοντέλων και επιστημονικής μεθόδου)

Scrum

Η Scrum είναι μια μεθοδολογία διαχείρισης έργου (λογισμικού) για μικρές ανεξάρτητες ομάδες. Η scrum ορίζει τις εξής φάσεις ανάπτυξης: την αρχική φάση σχεδιασμού, κατόπιν τα sprints, και τέλος την φάση του κλεισίματος στην οποία ολοκληρώνεται η ανάπτυξη του έργου [61].

Στην φάση του αρχικού σχεδιασμού, η ομάδα σχεδιάζει την αρχιτεκτονική του έργου ορίζοντας έναν επικεφαλής αρχιτέκτονα έργου. Κατά την ανάπτυξη, η ομάδα θα κάνει αλλαγές σε αυτή την αρχιτεκτονική, αλλά για την ώρα την χρειάζεται σαν σημείο αφετηρίας. Ο αρχιτέκτονας του έργου θα επιβλέπει αν η αρχιτεκτονική του θα τηρηθεί σε όλες τις φάσεις του έργου.

Κατόπιν, ορίζεται μια σειρά από σύντομες φάσεις ανάπτυξης που λέγονται sprints (δηλαδή γρήγορα τρεξίματα), με σκοπό την σταδιακή παράδοση του έργου. Η ομάδα δημιουργεί μια λίστα με όλα τα ανοιχτά υποέργα του έργου που λέγεται backlog. Πριν την αρχή του κάθε sprint, τα μέλη της ομάδας διαλέγουν τα υποέργα που πιστεύουν ότι είναι εφικτό να τελειώσουν μέχρι το τέλος του sprint και αλλάζουν την προτεραιότητα των υποέργων. Κατά την διάρκεια ενός sprint, δεν επιτρέπονται αλλαγές από τους ενδιαφερόμενους του έργου εκτός της ομάδας ανάπτυξης. Επίσης, σε ημερήσια βάση,

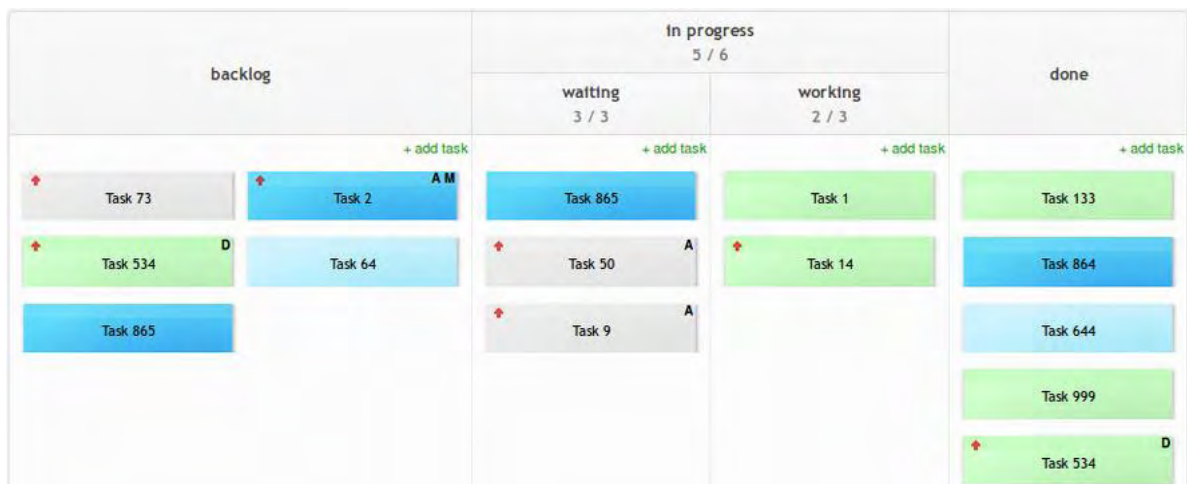
γίνονται Scrum meetings για να συζητηθεί η πρόοδος του έργου και να τεθούν νέοι στόχοι. Στο τέλος κάθε sprint παραδίδεται μέρος της λειτουργικότητας του συνολικού έργου.

Τα scrum meeting, το αρχικό backlog, η μέτρηση της προόδου του έργου, καθώς και η διαχείριση / ηγεσία της ομάδας γίνονται από τον Scrum Master. Μετά το τέλος του κάθε sprint, η ομάδα ανάπτυξης συναντιέται με τους υπόλοιπους ενδιαφερόμενους του έργου, μετόχους, υψηλόβαθμους managers, αντιπροσώπους του πελάτη κ.τ.λ. και συζητούν για την κατεύθυνση του project, τι θα αλλάξουν, προσθέσουν, επαναπροσδιορίσουν κ.τ.λ., και το πιο σημαντικό ερώτημα που τίθεται είναι αν αυτό το project πρέπει να συνεχιστεί.

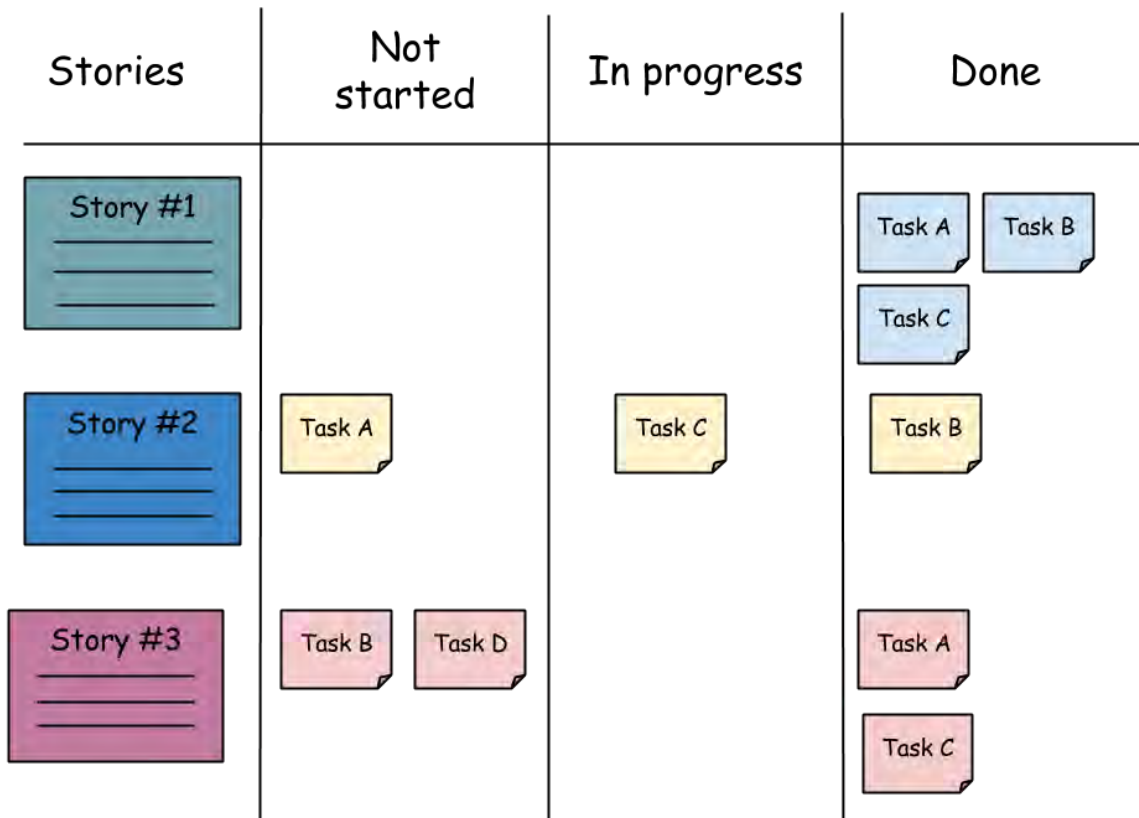
- Αν η απάντηση είναι ναι, τότε θα γίνει ακόμα ένα sprint και θα προστεθούν οι αλλαγές ή θα ενσωματωθούν οι νέοι στόχοι για το έργο.
- Αν η απάντηση είναι όχι, τότε το project σταματά ολοκληρωτικά γιατί πλέον δεν είναι βιώσιμο ή γίνεται ακόμα ένα τελευταίο sprint το οποίο αποτελεί την φάση του κλεισίματος του project στο οποίο το συνολικό έργο θα συγυριστεί πλήρως με σκοπό την τελική παράδοση του στον πελάτη.

Τα αποτελέσματα που προκύπτουν από την εφαρμογή της Scrum συνοψίζονται στα εξής:

- Το έργο μετατρέπεται σε μια σειρά διαχειρίσιμων υποέργων.
- Γίνεται πρόοδος, ακόμη και αν μεταβληθούν οι απαιτήσεις του έργου.
- Μεγάλη διαφάνεια στις διαδικασίες για όλους τους ενδιαφερόμενους του έργου.
- Οι πελάτες συμμετέχουν στην διαδικασία ανάπτυξης με συχνό feedback.
- Συνεργατική βελτίωση (μέσω μοντέλων και επιστημονικής μεθόδου).



Εικόνα 1.1: Παράδειγμα Πίνακα Kanban (Πηγή: [62]).



Εικόνα 1.2: Παράδειγμα Πίνακα Scrum (Πηγή: [63]).

Παρατηρούμε ότι οι μέθοδοι Kanban και Scrum έχουν παρόμοιες πρακτικές και αποτελέσματα για αυτό υπάρχουν προτάσεις υβριδικών λύσεων ανάμεσα τους [64], [65] με ένωση τους κάτω από τις Agile (ευέλικτες) μεθοδολογίες, οι οποίες στοχεύουν στην ίδια βασική αρχή με αυτές, δηλαδή την γρήγορη παράδοση μικρών και διαχειρίσιμων παραδοτέων του έργου. Αξίζει να αναφερθεί πως στη διαχείριση έργου υπάρχουν και άλλες πιο απλές μεθοδολογίες όπως η PERT και Gantt, οι οποίες εστιάζουν στον χρονικό προγραμματισμό των διάφορων υποέργων του έργου.

1.3 Σύγχρονες λύσεις διαχείρισης έργου

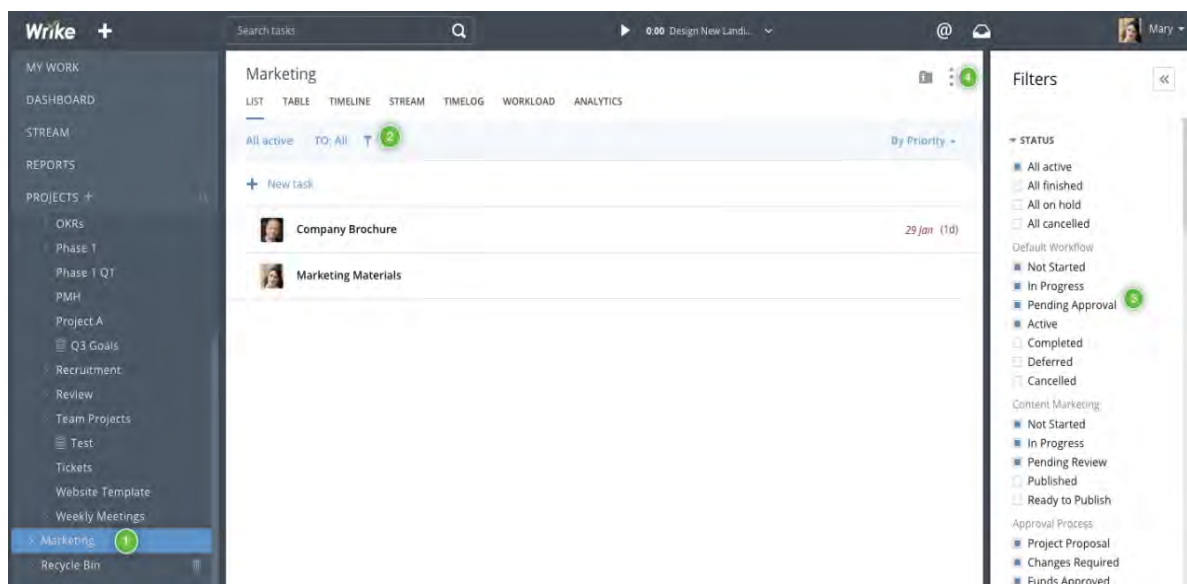
Wrike

Το Wrike είναι ένα πληροφοριακό σύστημα, κατασκευασμένο από την ομώνυμη εταιρία, που απευθύνεται σε project managers και σε εργαζόμενους που συμμετέχουν στο έργο. Ο σκοπός του είναι η μεγιστοποίηση του βαθμού οργάνωσης ενός έργου και η καλύτερη οπτικοποίηση των υποέργων του αλλά και η ευκολότερη εποπτεία του συνολικού έργου από τους project managers.

Πολλές εταιρίες χρησιμοποιούν το Wrike για το project management τους όπως οι: Google, Airbnb, TeamViewer, Sotheby's, De'Longhi Group και άλλες. Διατίθεται στο Cloud, ως SaaS (Software as a Service), σε τοπικό εγκατεστημένο λογισμικό για MacOSX, για Windows και σε mobile συστήματα όπως iOS & Android.

Προσφέρει μια πληθώρα δυνατοτήτων, όπως:

- Οργάνωση και αρχειοθέτηση έργων.
- Συνεργατική επικοινωνία εντός του συστήματος.
- Δημιουργία λεπτομερών αναφορών για τα έργα.
- Προσαρμογή της πλατφόρμας στις ανάγκες τις επιχείρησης.
- Εύκολη Διασύνδεση με άλλα πληροφοριακά συστήματα..
- Τεχνική υποστήριξη χρηστών.
- Ασφάλεια των δεδομένων των χρηστών.

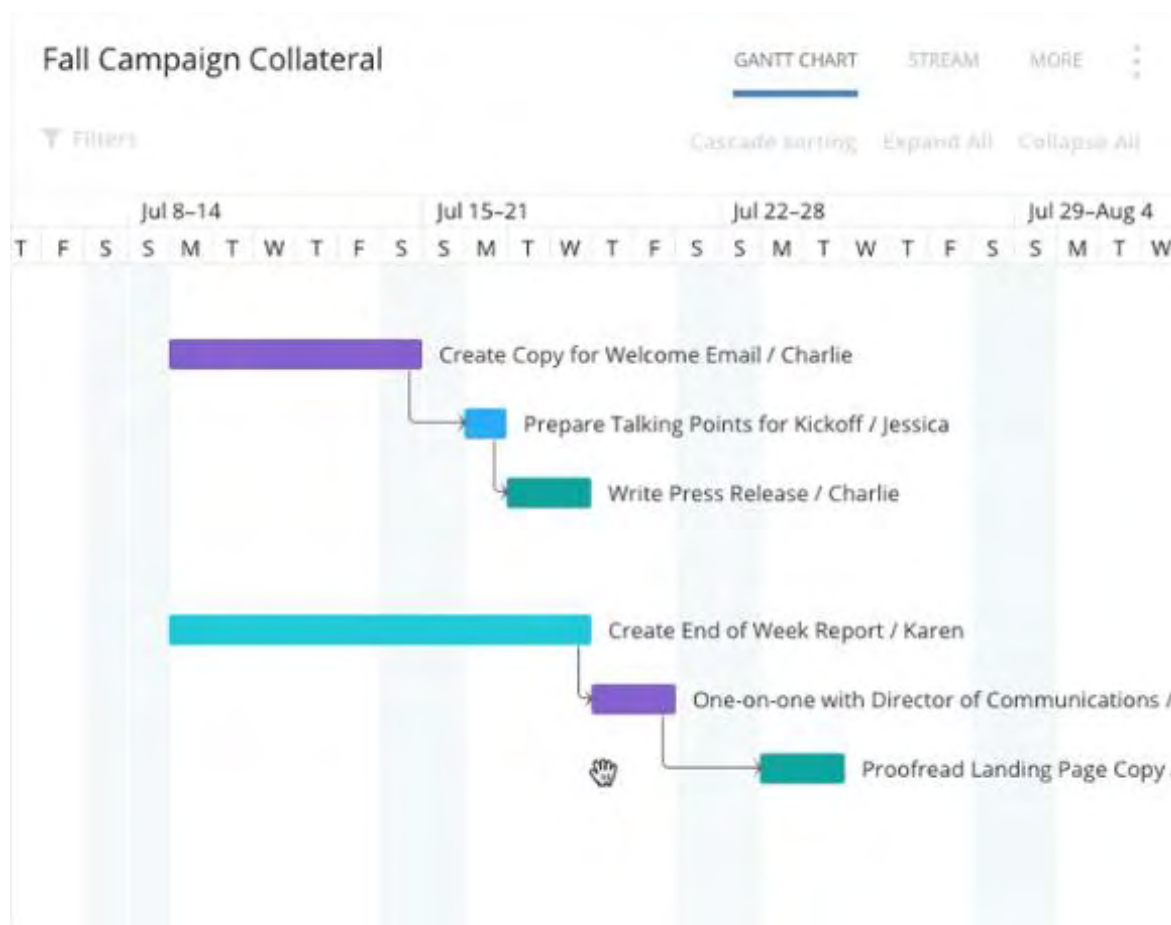


Εικόνα 1.3: Κύρια Οθόνη Wrike (Πηγή: [52]).

Η γραφική διεπαφή της εφαρμογής όπως φαίνεται στην Εικόνα 1.3 έχει φτιαχτεί όμοια με άλλες γνωστές εφαρμογές π.χ. ηλεκτρονικού ταχυδρομείου κ.τ.λ. για ευκολία μάθησης από τον χρήστη. Ένα από τα μειονεκτήματα του Wrike είναι ότι δεν διαθέτει έτοιμες δομές συγκρότησης του έργου για να διευκολύνει τον Project Manager. Ένα άλλο είναι, ότι παρά την εξαιρετική υποστήριξη που παρέχει για την διαχείριση του ανθρώπινου δυναμικού, δεν υπάρχει υποστήριξη για διαχείριση οικονομικών πόρων.



Εικόνα 1.4: Δευτερεύουσα Οθόνη Wrike – Οπτικοποίηση υποέργων του έργου (Πηγή: [53]).



Εικόνα 1.5: Δευτερεύουσα Οθόνη Wrike – Οπτικοποίηση χρόνου εκτέλεσης του έργου (Πηγή: [53]).

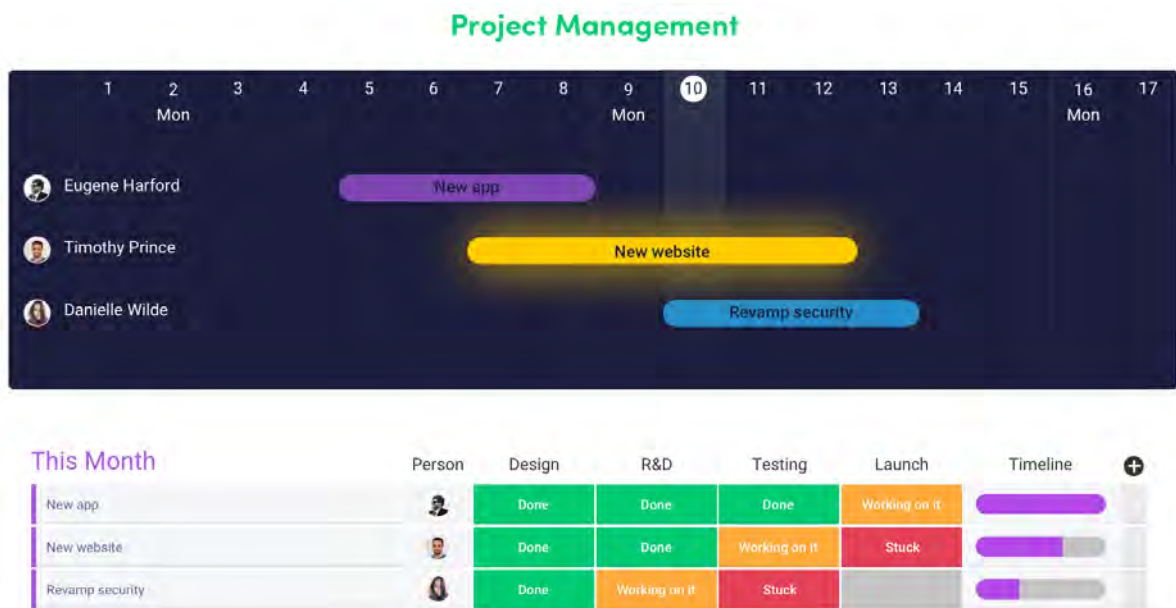
Monday

Το Monday είναι ένα πληροφοριακό σύστημα, κατασκευασμένο για το ίδιο δημογραφικό με το Wrike. Διακρίνεται για την απλότητα χρήσης του και την πολύ καλή τιμή του έναντι παρόμοιων λογισμικών.

Πολλές εταιρίες χρησιμοποιούν το Monday για το project management τους όπως οι: Carlsberg, Wix.com, Discovery Channel, TeamViewer, MIT Federal Credit Union, και άλλες. Διατίθεται στο Cloud, ως SaaS (Software as a Service), σε τοπικό εγκατεστημένο λογισμικό για MacOSX, για Windows και σε mobile συστήματα όπως iOS & Android.

Προσφέρει μια πληθώρα δυνατοτήτων, όπως:

- Οργάνωση, αρχειοθέτηση και χρονοπρογραμματισμός έργων.
- Συνεργατική επικοινωνία εντός του συστήματος.
- Συνεργατική διαμοίραση υποέργων εντός του συστήματος.
- Εύκολη Διασύνδεση με άλλα πληροφοριακά συστήματα.
- Τεχνική υποστήριξη χρηστών.
- Ασφάλεια των δεδομένων των χρηστών.
- Ο Project Manager μπορεί να οπτικοποιήσει τα δεδομένα του έργου με φιλικό προς τον χρήστη τρόπο (π.χ. για να παρουσιάσει την ως τώρα πορεία του έργου σε κάποιον πελάτη).



Εικόνα 1.6: Οθόνη Διαχείρισης project του Monday (Πηγή: [54]).

Η γραφική διεπαφή της εφαρμογής έχει φτιαχτεί για να είναι αρκετά φιλική προς το χρήστη σε σχέση με άλλα λογισμικά διαχείρισης έργων. Με την είσοδο του σε αυτό, ο Project Manager έχει πάρα πολλές προκατασκευασμένες επιλογές συγκρότησης του έργου αλλά μπορεί να συγκροτήσει την δική του δομή οργάνωσης του έργου κατά το δοκούν. Το Monday παρέχει επίσης, δυνατότητες συνδεσιμότητας με άλλα λογισμικά (π.χ. Dropbox, Excel, Google Calendar / Drive) και άλλα πληροφοριακά συστήματα όπως (π.χ. Slack, Trello, Integromat, Zapier). Τα κύρια μειονεκτήματα του Monday είναι ότι υποστηρίζει την διαχείριση οικονομικών πόρων αλλά μόνο του ανθρώπινου δυναμικού όπως και το Wrike, αλλά σε αντίθεση με το Wrike δεν επιτρέπει την εξαγωγή λεπτομερών αναφορών για τα δεδομένα του έργου.

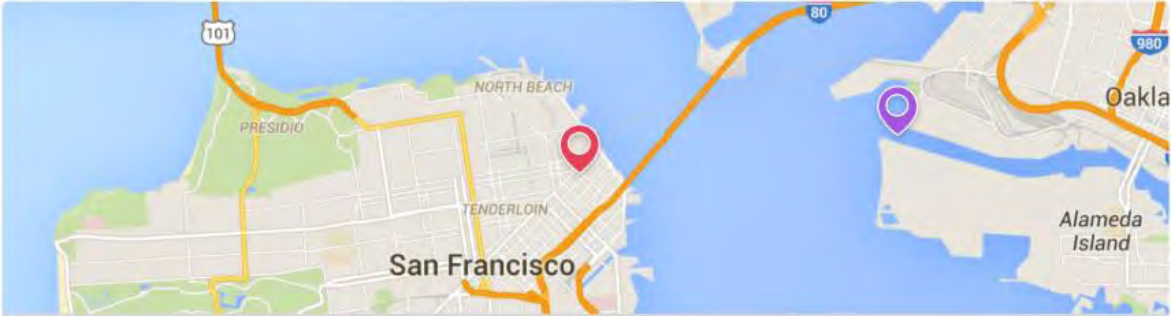
Team Tasks

Sprint 6 (This Week)	Owner	Helper	Status	Priority	Date
PRD for new task manager			Working on it	Medium	Jul 5
Define UX of creating a task			Working on it	Urgent	Jul 4
Finalize wording for SLA			Stuck	Urgent	Jun 29
New query button specs			Waiting for review	Medium	

Sprint 7 (Next week)	Sales Rep	Helper	Status	Priority	Date
PRD for template feature			Brainstorming	Urgent	Jul 9
Code UX for creating task				Medium	Jul 13

Εικόνα 1.7: Οθόνη Εργασιών Ομάδας του Monday (Πηγή: [54]).

Orders



This Week	Person	Payment	Status	Country	Adress
New app		Paid	Shipped	United States	929 Sycamore Street, SF, CA, 94104
New website		Paid	Shipped	United States	2932 Gladwell Street, Dallas, TX, 75247
Revamp security		Cancelled	Cancelled	United States	3510 Saint Clair Street, Memphis, MS, 38...

Εικόνα 1.8: Οθόνη Παραγγελιών Πελάτη του Monday (Πηγή: [54]).

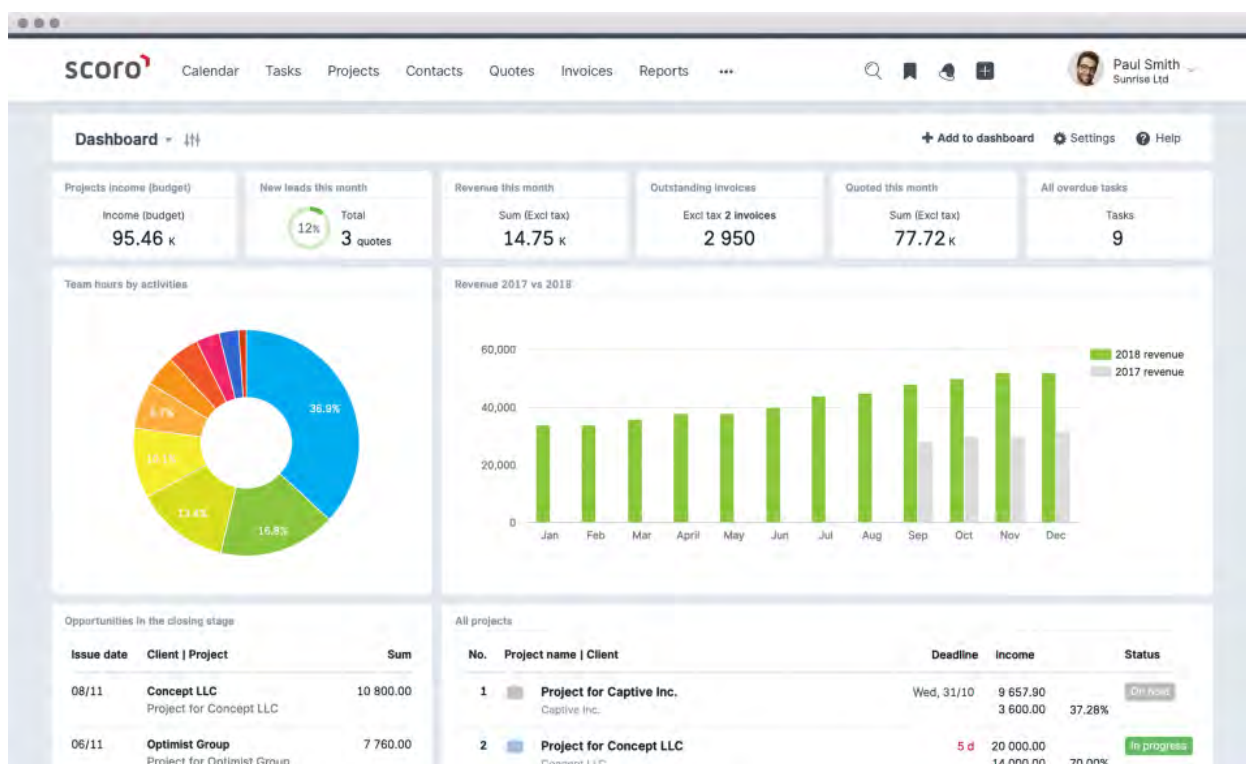
Scoro

Το Scoro είναι κατασκευασμένο για το ίδιο δημογραφικό με το Wrike. Σκοπός του είναι η μεγιστοποίηση της οργάνωσης καθώς και αναλυτική παρακολούθηση όλων των πόρων του έργου. Η φιλοσοφία σχεδίασης του είναι all-in-one (όλα-σε-ένα) δηλαδή οι περισσότερες δυνατότητες που απαιτούνται για την διαχείριση έργου παρέχονται από το σύστημα έτσι ώστε να μην απαιτούνται πολλές διασυνδέσεις με εξωτερικές εφαρμογές. Η σχεδίαση αυτή αποσκοπεί στη δημιουργία ενός κεντρικού περιβάλλοντος ελέγχου για τον Project Manager, οπότε η σχεδίαση αυτή έρχεται σε αντίθεση με τα προηγούμενα πληροφοριακά συστήματα που εξετάσαμε.

Πολλές εταιρίες χρησιμοποιούν το Monday για το project management τους όπως οι: Newton, Grant Thornton, ZAVODBBDO, WSI, Sotheby's, και άλλες. Διατίθεται στο Cloud, ως SaaS (Software as a Service) και σε mobile συστήματα όπως iOS & Android.

Προσφέρει μια πληθώρα δυνατοτήτων, όπως:

- Οργάνωση, αρχειοθέτηση και χρονοπρογραμματισμός έργων.
- Συνεργατική επικοινωνία εντός του συστήματος.
- Δημιουργία λεπτομερών αναφορών για τα έργα.
- Εύκολη Διασύνδεση με άλλα πληροφοριακά συστήματα.
- Τεχνική υποστήριξη χρηστών.
- Ασφάλεια των δεδομένων των χρηστών.
- Ο Project Manager μπορεί να οπτικοποιήσει τα δεδομένα του έργου με φιλικό προς τον χρήστη τρόπο (π.χ. για να παρουσιάσει την ως τώρα πορεία του έργου σε κάποιον πελάτη).



Εικόνα 1.9: Κύρια Οθόνη Scoro (Πηγή: [55]).

Η γραφική διεπαφή της εφαρμογής έχει φτιαχτεί έτσι ώστε να είναι ευπροσάρμοστη στις ανάγκες του Project Manager. Με την είσοδο του σε αυτό, ο Project Manager έχει πάρα πολλές προκατασκευασμένες επιλογές συγκρότησης του έργου, αλλά μπορεί να συγκροτήσει την δική του δομή οργάνωσης του έργου κατά το δοκούν. Το Scoro διαθέτει επίσης δυνατότητες συνδεσιμότητας με άλλα λογισμικά (π.χ. Dropbox, Excel, Google Calendar / Drive) και άλλα πληροφοριακά συστήματα όπως (π.χ. Xero, Zapier). Το κύριο πλεονέκτημα του Scoro είναι ότι καλύπτει πλήρως την διαχείριση πόρων, από το ανθρώπινο δυναμικό μέχρι την οργάνωση και αρχειοθέτηση των έργων, την διαχείριση των οικονομικών πόρων του έργου αλλά και την δημιουργία λεπτομερών αναφορών για την πορεία του. Το κύριο μειονέκτημα του είναι δεν υποστηρίζει την αυτόματη παραγωγή γραφημάτων Gantt, τα οποία είναι κρίσιμα για την διαχείριση έργου.

Jira

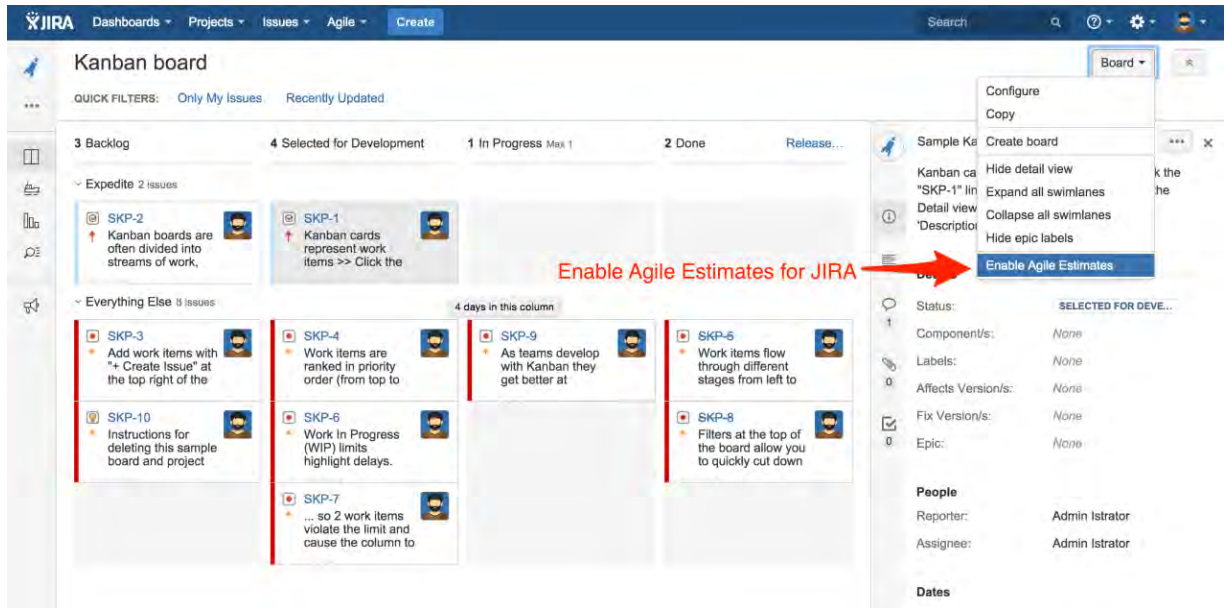
Το Jira είναι ένα πληροφοριακό σύστημα, κατασκευασμένο από την εταιρία Atlassian, που απευθύνεται σε project managers και σε εργαζόμενους που συμμετέχουν στο έργο. Σκοπός του είναι η μεγιστοποίηση της οργάνωσης καθώς και αναλυτική παρακολούθηση του έργου, ώστε οι project managers να ασκούν διαρκή έλεγχο από τους διαχειριστές. Το Jira είναι σχεδιασμένο βάσει των Agile μεθοδολογιών λογισμικού.

Πολλές εταιρίες χρησιμοποιούν το Jira για το project management τους όπως οι: Square, Ebay, Spotify, Cisco, Airbnb, και άλλες. Διατίθεται στο Cloud, ως SaaS (Software as a Service), σε τοπικό λογισμικό για MacOSX & Windows και σε mobile συστήματα όπως iOS & Android.

Προσφέρει μια πληθώρα δυνατοτήτων, όπως:

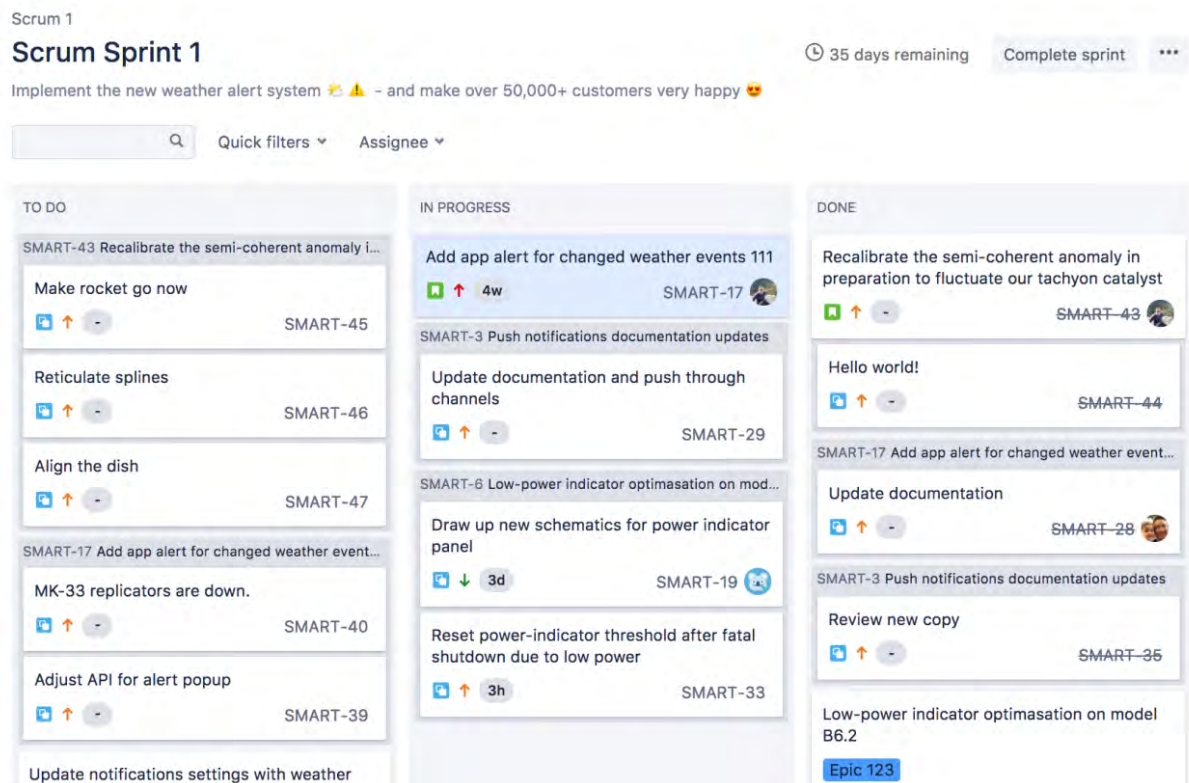
- Οργάνωση, αρχειοθέτηση έργων.
- Διαρκής έλεγχος προόδου στα υποέργα ενός έργου.
- Συνεχής παραδοτέα προς τους πελάτες.
- Δημιουργία λεπτομερών αναφορών για τα έργα.
- Διασύνδεση με άλλα πληροφοριακά συστήματα.
- Τεχνική υποστήριξη χρηστών.
- Ασφάλεια των δεδομένων των χρηστών.
- Ο Project Manager μπορεί να οπτικοποιήσει τα δεδομένα του έργου με φιλικό προς τον χρήστη τρόπο (π.χ. για να παρουσιάσει την ως τώρα πορεία του έργου σε κάποιον πελάτη).

Η γραφική διεπαφή της εφαρμογής έχει φτιαχτεί βάσει δύο μεθοδολογιών της διαχείρισης έργου την Kanban και την Scrum. Επομένως η αρχική οθόνη δομείται αναλόγως της μεθοδολογίας που θα διαλέξει ο project manager. Το Scoro διαθέτει επίσης, δυνατότητες συνδεσιμότητας με άλλα λογισμικά (π.χ. Dropbox, Excel, Google Calendar / Drive) και άλλα πληροφοριακά συστήματα όπως (π.χ. Xero, Zapier). Το κύριο μειονέκτημα του Jira είναι δεν υποστηρίζει επικοινωνία σε πραγματικό χρόνο ή έστω ειδοποιήσεις όταν γίνεται κάποια αλλαγή σε υποέργο που αφορά συγκεκριμένους εργαζόμενους.



Εικόνα 1.10: Κύρια Οθόνη Jira με Πίνακα Kanban (Πηγή: [56]).

Εναλλακτικά, στην κύρια οθόνη ενσωματώνεται ο πίνακας Scrum της Εικόνας 1.11



Εικόνα 1.11: Πίνακας Scrum (Πηγή: [57]).

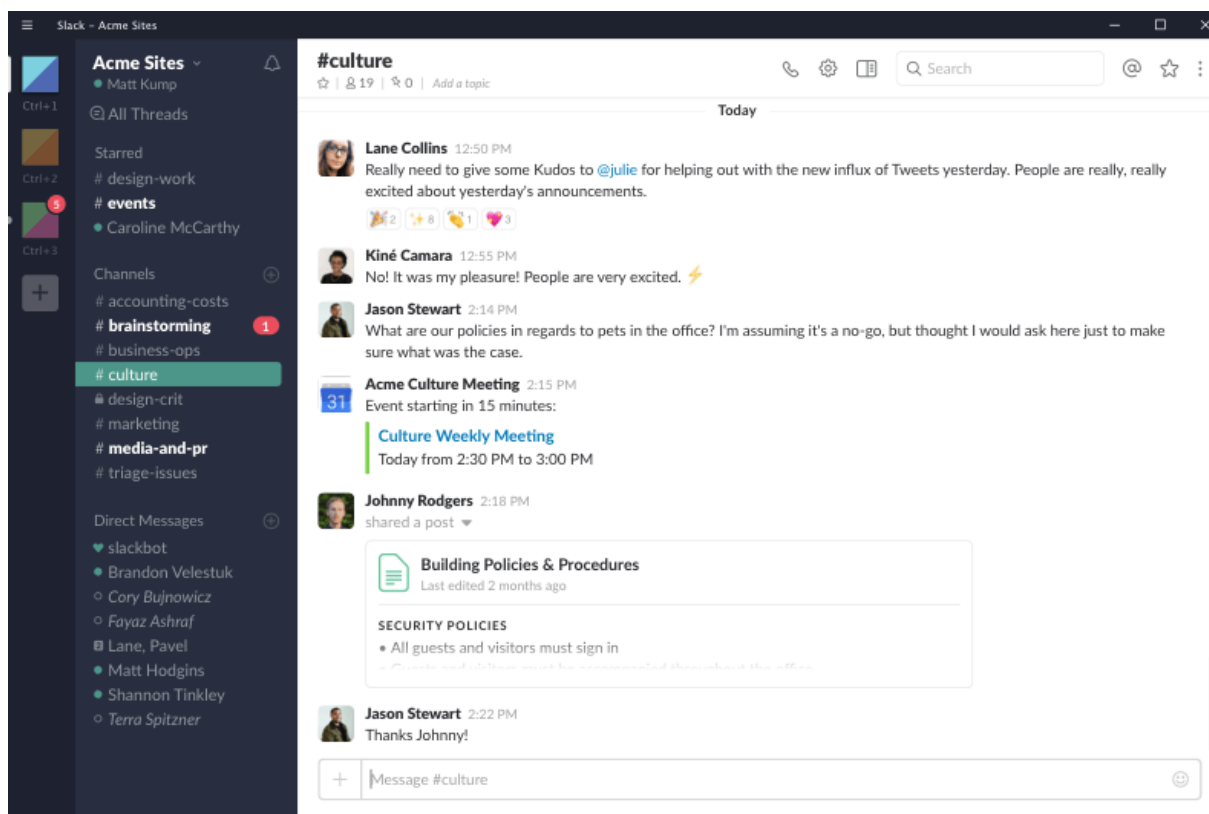
Slack

Το Slack είναι μια πλατφόρμα επικοινωνίας, και όχι ένα πληροφοριακό σύστημα διαχείρισης έργου όπως αυτά που είδαμε στις παραπάνω ενότητες, κατασκευασμένο από την εταιρία Slack Technologies. Σκοπός του είναι η βελτίωση της επικοινωνίας και του συντονισμού ανάμεσα στους εργαζομένους κάτι στο οποίο τα παραπάνω λογισμικά υστερούν καθώς και αναλυτική παρακολούθηση του έργου, ώστε οι project managers να ασκούν διαρκή έλεγχο από τους διαχειριστές. Το Jira είναι σχεδιασμένο βάσει των Agile μεθοδολογιών λογισμικού όπως φαίνεται και από τις δυνατότητες του παρακάτω.

Πολλές εταιρίες χρησιμοποιούν το Jira για το project management τους όπως οι: Airbnb, Target, 21 Century Fox, Oracle, Autodesk, και άλλες. Διατίθεται στο Cloud, ως SaaS (Software as a Service), σε τοπικό λογισμικό για MacOSX & Windows και σε mobile συστήματα όπως iOS & Android.

Προσφέρει μια πληθώρα δυνατοτήτων, όπως:

- Επικοινωνία και ειδοποιήσεις εργαζομένων σε πραγματικό χρόνο.
- Άμεση επικοινωνία μέσω ομαδικών κλήσεων / βιντεοκλήσεων.
- Υποστήριξη ατομικών και ομαδικών μηνυμάτων.
- Ευκολία στην χρήση.
- Εύκολη διασύνδεση με άλλα πληροφοριακά συστήματα.
- Τεχνική υποστήριξη χρηστών.
- Ασφάλεια των δεδομένων των χρηστών.



Εικόνα 1.12: Κύρια Οθόνη Slack (Πηγή: [59]).

Η γραφική διεπαφή της εφαρμογής έχει φτιαχτεί ώστε να είναι εύκολο στην εκμάθηση και τη χρήση. Το κύριο πλεονέκτημα του Slack είναι ότι διευκολύνει την επικοινωνία ακόμα και όταν ένα έργο γίνεται μεγάλο, σε αντίθεση με τα κλασσικά λογισμικά διαχείρισης έργων.

1.4 Στόχος της εργασίας

Η παρούσα εργασία έχει ως στόχο την ανάπτυξη σύγχρονων λύσεων για τυπικά προβλήματα Διαχείρισης Έργων, αξιοποιώντας state-of-the-art τεχνολογίες, όπως οι Graph Databases και τα chatbots (ψηφιακοί βοηθοί). Είναι σαφές ότι όσο αυξάνεται το μέγεθος των επιχειρήσεων καθώς και η πολυπλοκότητα των έργων (λογισμικού), τόσο περισσότερο δυσχεραίνεται η διαχείρισή τους. Η συνολική εποπτεία της δουλειάς διαφορετικών ανθρώπων σε πολλά και διαφορετικά ζητήματα, (συχνά συνδεδεμένα μεταξύ τους), είναι δύσκολο να πραγματοποιηθεί χειροκίνητα. Για την αντιμετώπιση του προβλήματος αυτού, τα σχετικά δεδομένα και οι μεταξύ τους συνδέσεις εισάγονται σε μια graph database, πετυχαίνοντας έτσι μια ιδιαίτερα χρήσιμη και επεκτάσιμη οπτικοποίησή τους. Στη συνέχεια, για ακόμη καλύτερη ανάλυση πιθανών προβλημάτων διαχείρισης έργων, αλλά και για την υποστήριξη καλά τεκμηριωμένων αποφάσεων, εισάγουμε στην προσέγγισή μας την τεχνολογία των chatbots, τα οποία «συνομιλώντας» με τον Project Manager σε φυσική γλώσσα μπορούν να «καταλάβουν» τις απαιτήσεις του και «απαντήσουν» συνήθη ερωτήματα που προκύπτουν κατά την διαχείριση ενός έργου. Μέσω της εφαρμογής των δύο παραπάνω τεχνολογιών, μπορούμε να ανακαλύψουμε (ευκολότερα και πιο στοχευμένα) γνώση που προκύπτει από τα δεδομένα, τις σχέσεις και την επεξεργασία τους, και γενικότερα να βελτιώσουμε την ποιότητα της Διαχείρισης Έργων.

1.5 Διάρθρωση της εργασίας

Στα πλαίσια του παραπάνω στόχου, η εργασία διαρθρώνεται ως εξής:

Στο Κεφάλαιο 2 εισάγουμε και αναλύουμε θέματα graph databases. Περιγράφουμε την ανάγκη για την χρήση τους, την αρχιτεκτονική μιας πολύ γνωστής graph database (Neo4j), καθώς και την μοντελοποίηση που μπορούμε να κάνουμε με αυτήν σε σύγκριση με τις σχεσιακές βάσεις δεδομένων. Στη συνέχεια, συγκρίνουμε την εκφραστικότητα της Cypher έναντι της SQL. Τέλος παραθέτουμε μερικά χρήσιμα ερωτήματα σε Cypher, την γλώσσα της graph database του Neo4j.

Στο κεφάλαιο 3 περιγράφουμε εναλλακτικά μέτρα ομοιότητας που μπορούμε να χρησιμοποιήσουμε για να λύσουμε κλασσικά ζητήματα του Project Management. Κάνουμε μια εισαγωγή για το ποια είναι αυτά τα ζητήματα. Περιγράφουμε γνωστά μέτρα ομοιότητας και την μέθοδο εφαρμογής τους σε κείμενα, αναπτύσσουμε ερωτήματα σε Cypher για να εξετάσουμε την ομοιότητα ανάμεσα σε issues (υποέργα του Project) και ανάμεσα σε προγραμματιστές. Εφαρμόζουμε τον αλγόριθμο PageRank πάνω στον γράφο εξάρτησης για να βρούμε τα πιο εξαρτημένα μεταξύ τους υποέργα.

Στο Κεφάλαιο 4 αναλύουμε τους ψηφιακούς βοηθούς (chatbots / Virtual assistants). Κάνουμε μια σύντομη εισαγωγή στην ως τώρα πορεία τους. Εξετάζουμε τις σύγχρονες εφαρμογές των chatbots. Προτείνουμε 2 τρόπους που τα chatbots θα μπορούσαν να

διευκολύνουν το Project Management. Κάνουμε μια σύντομη εισαγωγή στο wit.ai μια πλατφόρμα κατασκευής chatbot που θα χρησιμοποιήσουμε για να κατασκευάσουμε ένα chatbot το οποίο εξειδικεύεται στην αντιμετώπιση συχνών προβλημάτων του Project management. Παρουσιάζουμε και εξηγούμε την γραφική διεπαφή της εφαρμογής. Τέλος παρέχουμε παραδείγματα χρήσης της εφαρμογής βασισμένα πάνω σε πραγματικά προβλήματα Project Management.

Στο Κεφάλαιο 5 συνοψίζουμε την εργασία με τα συμπεράσματα που πρόεκυψαν και σκιαγραφούμε περιοχές για επέκταση της δουλειάς που έγινε στην παρούσα διπλωματική εργασία

2

Graph Databases

2.1 Εξέλιξη των βάσεων δεδομένων

Μια βάση δεδομένων είναι μια οργανωμένη συλλογή σχετιζόμενων δεδομένων, η οποία αποθηκεύεται και προσπελάσσεται ηλεκτρονικά από ένα υπολογιστικό σύστημα [66].

Με την πάροδο του χρόνου, οι εφαρμογές που αναπτύσσονταν γίνονταν ολοένα και πιο απαιτητικές σε αποθηκευτικό χώρο, αποθηκεύοντας ολοένα και περισσότερα δεδομένα. Για αυτό το λόγο, υπήρξαν προσπάθειες να καταστρωθεί μια συστηματική και αποτελεσματική προσέγγιση για να μπορούμε να διαχειριστούμε αυτό τον τεράστιο όγκο δεδομένων. Παρόλο, που τα πρώτα μη-σχεσιακά συστήματα βάσεων δεδομένων (non-Relational DBMS) είχαν αναπτυχθεί από τα μέσα της δεκαετίας του 1960, θα εστιάσουμε την προσοχή μας στις αρχές της δεκαετίας του 1980, μια περίοδος που σηματοδοτείται από την έλευση των σχεσιακών βάσεων δεδομένων (Relational Database systems) με χρήση της SQL, μια γλώσσα δομημένων ερωτημάτων.

Οι βάσεις δεδομένων αρχικά είχαν σχεδιαστεί για αποθήκευση και ανάκτηση δεδομένων, και από κατασκευής τους είναι αρκετά αποτελεσματικές για τον εν λόγω σκοπό. Εξαιρέση αποτελεί η περίπτωση ισχυρά διασυνδεδεμένων δεδομένων, όπου οι κλασσικές σχεσιακές βάσεις δεδομένων μειώνουν την απόδοσή τους. Κάτι το οποίο συμβαίνει διότι στις τελευταίες τα δεδομένα οργανώνονται σε πίνακες που συνδυάζονται μεταξύ τους με χρήση της εντολής JOIN. Όσο μεγαλύτεροι γίνονται οι πίνακες δεδομένων, τα ερωτήματα ανάκτησης στην βάση που περιέχουν JOIN επιβαρύνουν αισθητά το χρόνο εκτέλεσης (και κατ'επέκταση απόκρισης) των ερωτημάτων αυτών

Με την έλευση του πρωτοκόλλου Web 2.0 καθώς και των Cloud Computing, Big Data και Internet-of-Things(IoT), τα δεδομένα συσχετιζόνταν και συνδέονταν μεταξύ τους ολοένα και περισσότερο, οπότε τα ερωτήματα JOIN έγιναν ακόμα πιο πολύπλοκα, οδηγώντας στο να φτάσουμε σε σημείο συμφόρησης όσον αφορά την απόδοση των κλασσικών σχεσιακών βάσεων δεδομένων.

Μηχανικοί υπολογιστών και επιστήμονες πληροφορικής ανά τον κόσμο προσπάθησαν να βελτιστοποιήσουν εναλλακτικές σχεσιακές βάσεις με διαφορετικούς βαθμούς επιτυχίας. Ωστόσο στο τέλος της δεκαετίας του 90, ήταν πλέον προφανές ότι τα δεδομένα προς αποθήκευση και ανάκτηση ήταν εξίσου σημαντικά όσο και οι συνδέσεις ανάμεσα τους, Οπότε ακολουθώντας αυτήν την νοοτροπία σχεδιάστηκαν οι μη-σχεσιακές βάσεις δεδομένων.

Οι μη-σχεσιακές βάσεις δεδομένων (NoSQL databases) σε αντίθεση με τις σχεσιακές (SQL databases) δεν απαιτούν πλέον οργάνωση των δεδομένων σε μεγάλους πίνακες, μπορούν να αποθηκεύσουν τα δεδομένα με ανομοιομορφο ή μη κανονικοποιημένο τρόπο στην μνήμη σεβόμενες τις ανάγκες της εφαρμογής, αποφεύγοντας την χρήση της εντολής JOIN σε μεγάλους πίνακες, επιτυγχάνοντας έτσι μια γραμμική κλιμάκωση της χρονικής πολυπλοκότητας εκτέλεσης τους (δηλαδή είτε για μικρό είτε για μεγάλο πλήθος εγγράφων, έχουν ένα σχετικά σταθερό χρόνο εκτέλεσης)

Στην αρχή της δεκαετίας του 2000, εμφανίστηκε μια νέα κατηγορία των No-SQL βάσεων δεδομένων, οι λεγόμενες Graph Databases (δηλαδή βάσεις δεδομένων με οργάνωση γράφου), με στόχο να λύσει τα θέματα μοντελοποίησης επιστημονικών προβλημάτων καλύτερα από την μοντελοποίηση με χρήση πινάκων από κλασσικές βάσεις δεδομένων.

Παρακάτω, θα συζητήσουμε την φύση των graph databases, την αρχιτεκτονική και απόδοση τους σε αντιδιαστολή με τις κλασσικές σχεσιακές βάσεις δεδομένων. Πριν προχωρήσουμε, επισημαίνουμε το γεγονός ότι οι σχεσιακές βάσεις δεδομένων σε μεγάλους οργανισμούς χρησιμοποιούνται ευρέως ακόμα. Αυτό οφείλεται σε διάφορος λόγους, όπως στον φόβο και τη δυσκολία αντικατάστασης συστημάτων τέτοιας κλίμακας, στο συχνά υψηλό κόστος αναβάθμισης σε ένα πολύ μεγάλο οργανισμό με τεράστιο όγκο δεδομένων, αλλά και στο ότι οι βάσεις αυτές είναι αξιόπιστες και καλά δοκιμασμένες. Επιπλέον τα δεδομένα των οργανισμών αυτών ταιριάζουν στο tabular μοντέλο που παρέχουν οι SQL βάσεις. Σε κάθε περίπτωση, οι ειδικοί θεωρούν τις No-SQL βάσεις δεδομένων ως την τελευταία λέξη της τεχνολογίας.

Σύμφωνα με την γνώμη μας, η τελευταία λέξη της τεχνολογίας είναι οι Graph Databases, όσον αφορά την διαχείριση δεδομένων. Ήδη πολλά πανεπιστήμια και επιχειρήσεις αναγνωρίζουν την πραγματική αξία της γνώσης που παράγεται από συνδέσεις ανάμεσα στα υπάρχοντα δεδομένα που έχουν συλλέξει με το πέρασμα του χρόνου.

2.2 Η ανάγκη για τις Graph Databases

Στο ερώτημα “γιατί να χρησιμοποιήσουμε μια Graph Database?”, μια πιθανή απάντηση (πέρα από την απόδοσή τους που θα εξετάσουμε παρακάτω) θα μπορούσε να είναι ότι πολλά προβλήματα της επιστήμης των υπολογιστών μοντελοποιούνται με τη χρήση γράφων και οι αλγόριθμοι για να λυθούν τα προβλήματα είναι βελτιστοποιημένοι για χρήση γράφων.

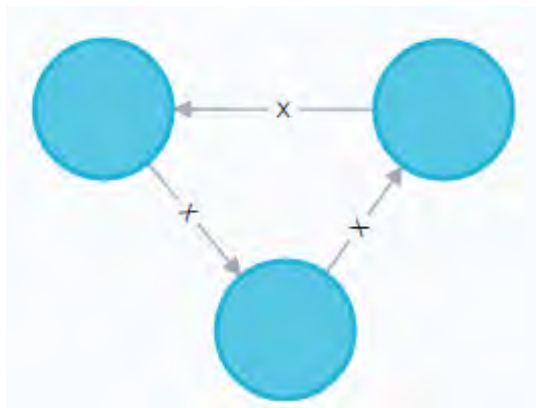
Μέχρι τώρα έχουμε χρησιμοποιήσει τον όρο γράφο εκτενώς, χωρίς όμως να ξεκαθαρίσουμε σε τι αναφερόμαστε. Ο όρος γράφος που είναι ομώνυμος με τον αντίστοιχο όρο από την θεωρία γράφων στα μαθηματικά, αναφέρεται ως:

“Ένας γράφος είναι ένας αφηρημένος τύπος δεδομένων που αποτελείται από ένα πεπερασμένο σύνολο κόμβων και ακμών τα οποία γραφικά απεικονίζονται ως κύκλοι και βέλη αντίστοιχα [1].”

Η ιδέα πίσω από τους κόμβους είναι ότι μπορούν να απεικονίσουν οντότητες του φυσικού κόσμου όπως, για παράδειγμα, ανθρώπους, αντικείμενα, τοποθεσίες, κατηγορίες κλπ.

Η ιδέα πίσω από τις ακμές είναι ότι μπορούν να απεικονίσουν σχέσεις ανάμεσα σε δύο ή περισσότερες οντότητες του φυσικού κόσμου. Οι γράφοι μπορούν να χρησιμοποιηθούν για να μοντελοποιήσουν μια ευρεία γκάμα οντοτήτων, όπως ένα υπολογιστικό δίκτυο, ένα οδικό δίκτυο, ένα δίκτυο συσχετιζόμενων ανθρώπων, ή οποιαδήποτε άλλη συλλογή αντικειμένων που μπορούν να συνδεθεί μεταξύ τους.

Ένα παράδειγμα (κατευθυνόμενου) γράφου 3 κόμβων που συνδέεται με τις ακμές με την ετικέτα ‘x’ φαίνεται στην Εικόνα 2.1.



Εικόνα 2.1: Παράδειγμα γράφου.

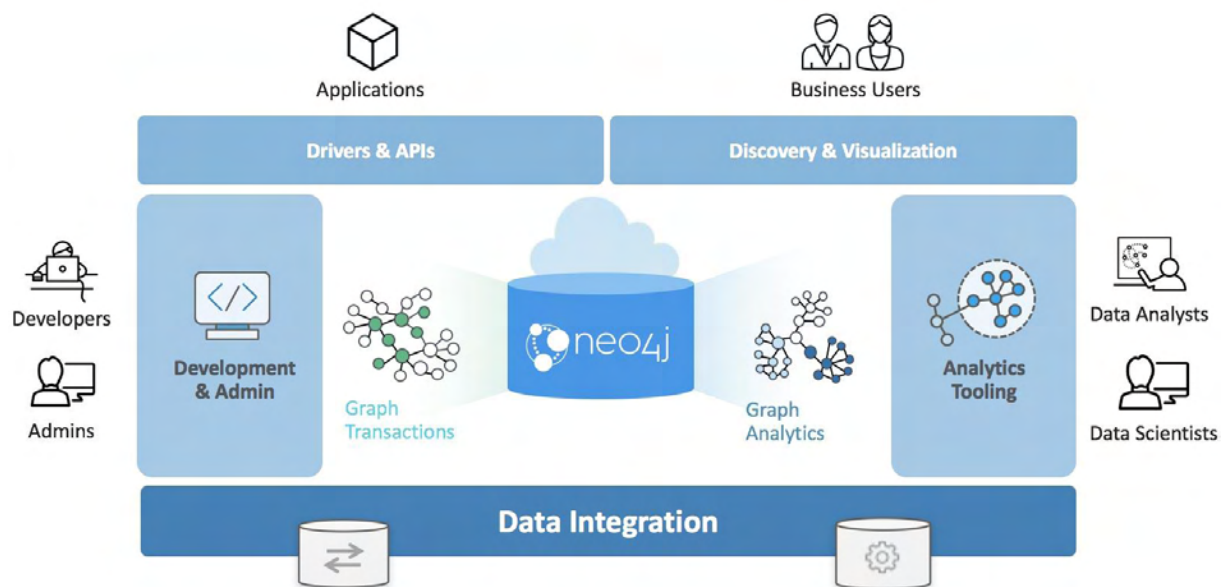
Επιπροσθέτως ορίζουμε τις graph databases ως εξής:

“Μια graph database είναι μια μη-σχεσιακή βάση δεδομένων η οποία υποστηρίζει όλες τις λειτουργίες μιας παραδοσιακής βάσης δεδομένων CREATE, INSERT, ALTER, DELETE χρησιμοποιώντας το μοντέλο του γράφου για την οργάνωση των δεδομένων [68].”

2.3 Neo4j Architecture

Σύμφωνα με την ιστοσελίδα του, το Neo4j είναι μια graph database η οποία σε αντίθεση με άλλες graph / No-SQL databases ακολουθεί την αρχή του ACID (Atomicity, Consistency, Isolation, Durability), χρησιμοποιώντας ένα σύστημα που χτίζεται με το μοντέλο του γράφου για την αποθήκευση και επεξεργασία δεδομένων. Είναι διαθέσιμη με άδεια ανοιχτού λογισμικού και υλοποιείται σε Java. Όπως και άλλες graph databases, μπορεί να προσπελαστεί με διάφορες γλώσσες ερωτημάτων όπως η Cypher, SPARQL, GraphQL και Gremlin. Στο πλαίσιο της εργασίας αυτής θα χρησιμοποιήσουμε την Cypher η οποία είναι η ενδεδειγμένη γλώσσα ερωτημάτων τόσο από την κοινότητα της Neo4j όσο και από την εταιρία που την ανέπτυξε [9].

Η αρχή του ACID περιγράφει τον τρόπο με τον οποίο οι σχεσιακές βάσεις δεδομένων εγγυώνται την εγκυρότητα των δεδομένων σε περίπτωση σφαλμάτων, διακοπών ρεύματος κ.τ.λ. Οι No-SQL βάσεις δεδομένων δεν ακολουθούν την αρχή του ACID, ανταλλάσσοντας τη σταθερότητα σε ποικίλους βαθμούς (που μπορούν να επιλέξουν οι μηχανικοί) με τη μέγιστη δυνατή απόδοση. Είναι αξιοσημείωτο το γεγονός ότι η Neo4j, θυσιάζοντας λίγη απόδοση σε σχέση με τις No-SQL βάσεις δεδομένων, διατηρεί την ασφάλεια των δεδομένων όπως οι σχεσιακές βάσεις, καθιστώντας την μία άξιο διάδοχο τους για όλες τις μοντέρνες εφαρμογές που θα περιγράψουμε σε επόμενη ενότητα [7].

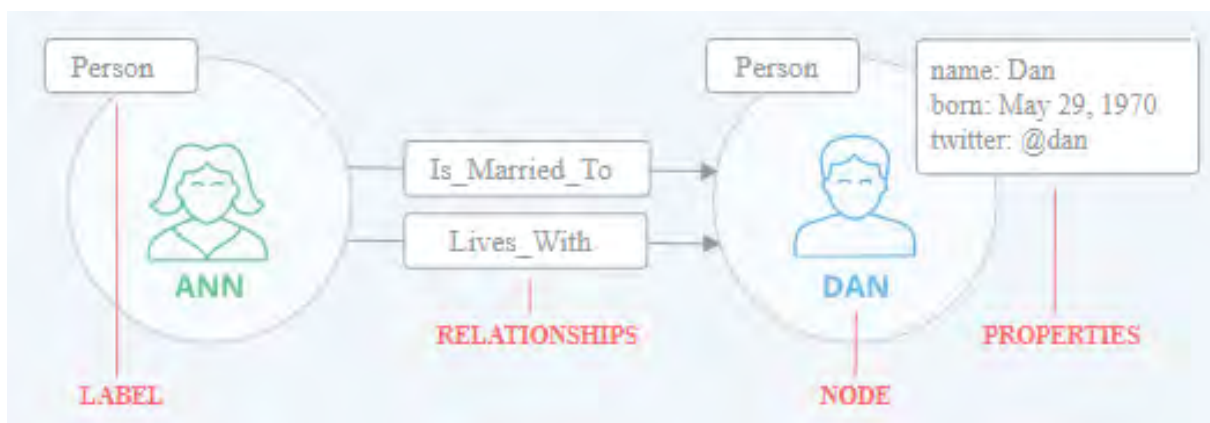


Εικόνα 2.2: Σύνοψη αρχιτεκτονικής του neo4j (Πηγή: [10]).

Όπως φαίνεται και στην Εικόνα 2.2, πολλά διαφορετικά είδη χρηστών μπορούν να έχουν πρόσβαση στη βάση δεδομένων για διάφορους σκοπούς.

Το γραφικό περιβάλλον της neo4j ακολουθεί μια ολιστική προσέγγιση, οπτικοποιώντας εύκολα τα δεδομένα της, για απλούς χρήστες της επιχείρησης και δίνοντας την δυνατότητα στους αναλυτές δεδομένων να τρέξουν τα ερωτήματα τους στην βάση από την γραμμή εντολών εντός της ίδιας εφαρμογής. Παράλληλα οι προγραμματιστές μπορούν να κάνουν αλλαγές στη βάση παράγοντας περισσότερες συνδέσεις και δεδομένα. Τα δικαιώματα της κάθε κατηγορίας χρηστών μπορούν εύκολα να ρυθμιστούν από τους διαχειριστές της.

Το μοντέλο που υλοποιείται εσωτερικά σε μια βάση δεδομένων της neo4j, το οποίο ονομάζεται Labeled Property Graph Model (μοντέλο γράφων ιδιοτήτων με ετικέτες), περιγράφεται στην Εικόνα 2.3.



Εικόνα 2.3: Βασικό μοντέλο της αρχιτεκτονικής του neo4j (Πηγή: [11]).

Οι κόμβοι (συμβολίζονται με κύκλο), που είναι η βασική μορφή δεδομένων, συνδέονται με άλλους κόμβους μέσω σχέσεων. Μπορούν επίσης να έχουν μια ή περισσότερες ιδιότητες καθώς και ετικέτες που χαρακτηρίζουν τον κόμβο ως προς τον ρόλο του στο γράφο.

Οι σχέσεις (συμβολίζονται με βέλος και ορθογώνιο), που συνδέουν δύο κόμβους, είναι κατευθυνόμενες και μπορούν να έχουν μια ή περισσότερες ιδιότητες. Οι κόμβοι του γράφου μπορούν να έχουν πολλαπλές ή, ακόμη και αμφίδρομες σχέσεις.

Οι ιδιότητες είναι τιμές που έχουν όνομα και τιμή, (συνήθως είναι αλφαριθμητικά). Επίσης μπορούν να χρησιμοποιηθούν σαν ευρετήριο τιμών, και να τους επιβάλλουμε λογικούς περιορισμούς.

Οι ετικέτες είναι τιμές με τις οποίες οργανώνουμε σύνολα κόμβων (βάσει μιας κοινής ετικέτας). Ένας κόμβος μπορεί να έχει πολλές ετικέτες. Γενικά, οι ετικέτες αποθηκεύονται σε ευρετήριο στη βάση δεδομένων ώστε να επιταχύνεται η εύρεση των κόμβων αλλά και οι ίδιες οι ετικέτες μπορούν να αποθηκεύουν δεδομένα σε μορφή ευρετηρίου (σύνθετες ετικέτες).

2.4 Graph databases vs. Relational databases vs. No-SQL databases

Παραπάνω συζητήσαμε τις ιδιότητες των τριών διαφορετικών κατηγοριών βάσεων δεδομένων, οι οποίες είναι οι σχεσιακές (Relational) οι απλές μη σχεσιακές (No-SQL) και οι μη σχεσιακές με οργάνωση γράφου βάσεις δεδομένων (Graph Databases). Στην παρούσα ενότητα θα τις συγκεντρώσουμε σε έναν αναλυτικό πίνακα.

	Relational	Neo4j	No-SQL
Αποθήκευση Δεδομένων	Αποθήκευση δεδομένων σε προκαθορισμένους πίνακες με διασυνδεδεμένα δεδομένα σε ξεχωριστούς πίνακες. Μείωση της απόδοσης των ερωτημάτων.	Εγγενής υποστήριξη για συνδέσεις μεταξύ δεδομένων. Η αποθήκευση σε γράφους καταργεί την ανάγκη ευρετηρίου και τα δεδομένα με τις συνδέσεις τους προσπελούνται γρηγορότερα.	Μη εγγενής υποστήριξη για συνδέσεις μεταξύ δεδομένων. Μείωση απόδοσης λόγω αύξησης της πολυπλοκότητας με την προσθήκη περισσότερων συνδέσεων.
Μοντελοποίηση Δεδομένων	Το μοντέλο που πρέπει να αναπτυχθεί διαφέρει από το αρχικό λογικό μοντέλο. Η μορφή των δεδομένων πρέπει να είναι γνωστή από την αρχή, Οι αλλαγές απαιτούν επανασχεδιασμό της βάσης.	Κατάλληλο για μεγάλες αρχιτεκτονικές επιχειρήσεων. Επιτρέπει τον πλήρη έλεγχο δεδομένων με την σύσταση ad-hoc λύσεων. Εύκολη οπτικοποίηση κατανοητή από προγραμματιστές, αρχιτέκτονες λογισμικού και μέλη της επιχείρησης.	Ακατάλληλη για μεγάλες αρχιτεκτονικές επιχειρήσεων που απαιτούν πολλές στήλες γνωρισμάτων και highly interconnected data, όπου οι σχέσεις μεταξύ των δεδομένων είναι εξίσου σημαντικές με τα ίδια τα δεδομένα. Το πρόβλημα μοντελοποίησης μεταφέρεται στο επίπεδο λογισμικού.
Γλώσσα Ερωτημάτων	SQL: Μια γλώσσα ερωτημάτων με αυξανόμενη πολυπλοκότητα εκτέλεσης όσο αυξάνεται ο αριθμός των JOIN για την	Cypher: Μια γλώσσα ερωτημάτων φτιαγμένη για προσπέλαση διασυνδεδεμένων δεδομένων σε	Πολλές διαφορετικές γλώσσες υπάρχουν αλλά καμία δεν παρέχει εγγενή τρόπο προσπέλασης

	προσπέλαση διασυνδεδεμένων δεδομένων.	γράφο.	των διασυνδεδεμένων δεδομένων.
Υποστήριξη ασφαλούς συναλλαγής	Πλήρης υποστήριξη της αρχής του ACID.	Πλήρης υποστήριξη της αρχής του ACID.	Μερικής υποστήριξη της αρχής του ACID. Πιθανή διαφθορά δεδομένων.
Επεξεργασία σε μεγάλη κλίμακα	Πολυπλοκότητα διασύνδεσης δεδομένων => Αυξημένο κόστος.	Το μοντέλο γράφων σχεδιάστηκε για διασύνδεση δεδομένων ακόμη και σε μεγάλη κλίμακα.	Βελτιστοποίηση για κατανάλωση δεδομένων.

Πίνακας 2.1: Σύγκριση διαφορετικών βάσεων δεδομένων

2.5 Cypher vs. SQL

Μέχρι αυτό το σημείο, συγκρίναμε τις σχεσιακές και μη-σχεσιακές databases βάσει των τεχνικών χαρακτηριστικών τους. Στη συνέχεια θα δείξουμε την απλότητα και την εκφραστικότητα της Cypher, μιας γλώσσας ερωτημάτων που σχεδιάστηκε για γράφους, έναντι της SQL (μιας γλώσσας ερωτημάτων που σχεδιάστηκε για πίνακες). Για να δείξουμε τα εν λόγω χαρακτηριστικά θα χρησιμοποιήσουμε ένα συγκεκριμένο πρόβλημα διασυνδεδεμένων δεδομένων, το οποίο εναρμονίζεται πλήρως με σύγχρονες εφαρμογές. Συγκεκριμένα, εξετάζουμε το παράδειγμα μιας απλής μηχανής προτάσεων προϊόντων. Σε αυτό το παράδειγμα έχουμε δεδομένα που είναι ισχυρά συνεκτικά. Οι πελάτες συσχετίζονται με προϊόντα και εταιρίες, τα προϊόντα σχετίζονται με άλλα προϊόντα και οι πελάτες συσχετίζονται με άλλους πελάτες [5].

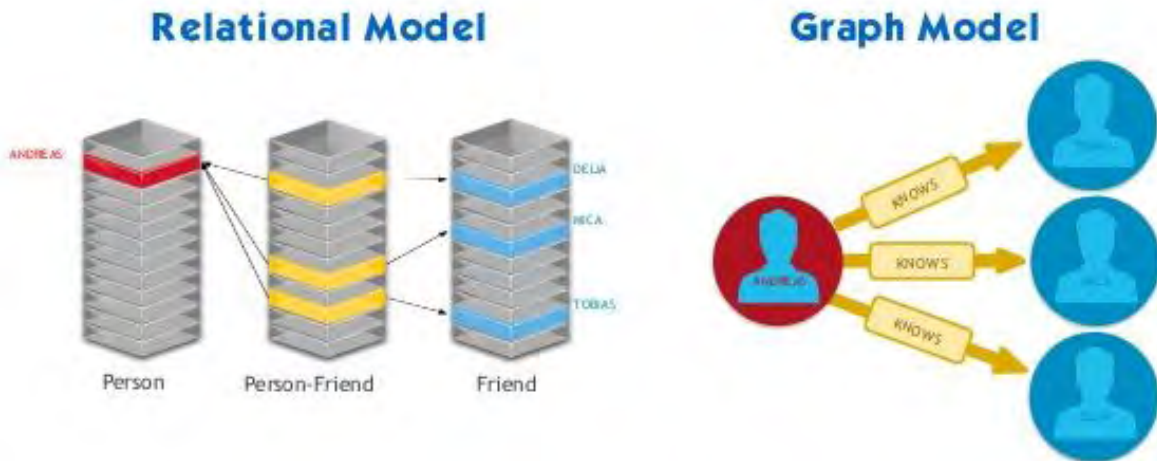
Για να είναι η μηχανή προτάσεων προϊόντων αποτελεσματική, πρέπει να ισχύουν οι τρεις παρακάτω προϋποθέσεις:

- Να μοντελοποιούνται τα δεδομένα καθώς και οι σχέσεις ανάμεσα σε αυτά.
- Να πραγματοποιούνται προτάσεις σε πραγματικό χρόνο με τη χρήση ερωτημάτων στη βάση
- Να εμπλουτίζεται το μοντέλο προσθέτοντας εύκολα (με μικρό κόστος) περισσότερα δεδομένα και συνδέσεις

Εξετάζουμε αρχικά την μοντελοποίηση δεδομένων σε μια σχεσιακή βάση δεδομένων. Χρειαζόμαστε ξεχωριστούς πίνακες για να αναπαραστήσουμε τους πελάτες, τις παραγγελίες και τα προϊόντα, καθώς και ενδιάμεσους πίνακες που αντιπροσωπεύουν τη σειρά με την οποία αγοραστήκαν. Στη συνέχεια θα προκύψουν και άλλοι πίνακες για να εκφραστούν και άλλες σχέσεις ανάμεσα στους υπάρχοντες. Οπότε το τελικό μοντέλο απέχει αρκετά από το αυτό που σχεδιάσαμε αρχικά, θα είναι δυσανάγνωστο για όλους πλην τους προγραμματιστές

που δουλεύουν σε αυτό, και εν τέλει θα είναι δύσκολο στη λειτουργία, συντήρηση και επέκτασή του ακόμη και για τους τελευταίους.

Εξετάζουμε στη συνέχεια την μοντελοποίηση δεδομένων σε μια graph database. Το μοντέλο που χρησιμοποιεί αξιοποιεί κόμβους και ακμές, καθώς και κάποιες ιδιότητες επί αυτών, όπως μια ετικέτα στις ακμές, που δείχνει τη σημασία ή “βαρύτητα” της συγκεκριμένης ακμής. Συνεπώς το μοντέλο αυτό θα μοιάζει περισσότερο με το αρχικό λογικό μοντέλο που σχεδιάσαμε. Η διαφορά μοντελοποίησης φαίνεται στην Εικόνα 2.4.



Εικόνα 2.4: Διαφορά μοντέλου σχεσιακής και graph database (Πηγή: [12]).

Πέρα από την μοντελοποίηση δεδομένων, παρουσιάζουμε παρακάτω το ερώτημα που θα χρησιμοποιηθεί για τις προτάσεις προϊόντων στις δύο γλώσσες. Σε Cypher θα είναι:

```
MATCH (u:Customer {customer_id:'customer-one'})-[:BOUGHT]->(p:Product)<-[:BOUGHT]-(peer:Customer)-[:BOUGHT]->(reco:Product)
WHERE not (u)-[:BOUGHT]->(reco)
RETURN reco as Recommendation, count(*) as Frequency
ORDER BY Frequency DESC LIMIT 5;
```

Το παραπάνω ερώτημα, αρχικά ταιριάζει κάθε πελάτη που αγόρασε ένα προϊόν με πελάτες που αγόρασαν και εκείνοι το προϊόν, και κατόπιν βρίσκει τα προϊόντα που αγόρασαν οι τελευταίοι, τα ταξινομεί ως προς το πόσες φορές αγοράστηκαν, και τέλος επιστρέφει τα 5 κορυφαία εξ αυτών. Η δήλωση WHERE αφαιρεί τα προτεινόμενα προϊόντα που έχει αγοράσει ήδη ο κάθε πελάτης. Κάθε βελάκι (-, ->) εκφράζει μια σχέση που χρειάζεται πολλαπλά ερωτήματα JOIN μεταξύ πινάκων, όπως φαίνεται και στο αντίστοιχο ερώτημα SQL σε μια σχεσιακή βάση δεδομένων:

```
SELECT product.product_name as Recommendation, count(1) as Frequency
FROM product, customer_product_mapping, (SELECT cpm3.product_id, cpm3.customer_id
FROM Customer_product_mapping cpm, Customer_product_mapping cpm2,
Customer_product_mapping cpm3
WHERE cpm.customer_id = 'customer-one'
and cpm.product_id = cpm2.product_id
```

```

and cpm2.customer_id != 'customer-one'
and cpm3.customer_id = cpm2.customer_id
and cpm3.product_id not in (select distinct product_id
FROM Customer_product_mapping cpm
WHERE cpm.customer_id = 'customer-one')
) recommended_products
WHERE customer_product_mapping.product_id = product.product_id
and customer_product_mapping.product_id in recommended_products.product_id
and customer_product_mapping.customer_id = recommended_products.customer_id
GROUP BY product.product_name
ORDER BY Frequency desc

```

Αυτό το SQL statement δεν υποφέρει μόνο από θέματα απόδοσης λόγω της πολυπλοκότητας των JOIN εντολών του. Η απόδοσή του θα πέφτει όσο μεγαλώνουν τα δεδομένα. Για να αντιμετωπιστεί αυτό το πρόβλημα, πολλές εταιρίες χρησιμοποιούν ερωτήματα τα οποία επιστρέφουν τιμές που έχουν εκ των προτέρων υπολογίσει (pre – computed results), έναντι του να επιστρέφουν επίκαιρα δεδομένα σε πραγματικό χρόνο.

Γίνεται σαφές ότι μια σχεσιακή βάση δεδομένων έχει δύσκολη επεκτασιμότητα, σε αντίθεση με μια graph database η οποία είναι επεκτάσιμη, δηλαδή είναι εύκολο να προσθέσουμε νέα δεδομένα και συνδέσεις.

Η ειδοποιός διαφορά έγκειται στο γεγονός ότι ένα ερώτημα σε graph databases δεν υπολογίζει τις συνδέσεις τη στιγμή που καλείται, αλλά τις αποθηκεύει μια φορά κατά την εισαγωγή τους στη βάση και μετά τις ανακτά σε ελάχιστο χρόνο (σε σχέση με το να τις υπολόγιζε κάθε φορά).

2.6 Case Study: Jira Issues

Το Jira είναι μια εμπορική εφαρμογή για την διαχείριση issues, δηλαδή σφαλμάτων λογισμικού καθώς και την διαχείριση του ίδιου του project (ανάπτυξης λογισμικού) όσον αφορά τα λάθη του και το ιστορικό τους. Αρχικά, εμείς θα εισάγουμε τα δεδομένα από τα issues, τους προγραμματιστές, τις παραγόμενες ετικέτες από τα issues -και τις σχέσεις ανάμεσα τους- από ένα μεγάλο σετ δεδομένων του Hadoop project στην βάση της neo4j. Από αυτό το σετ δεδομένων παράγονται 6,445 κόμβοι και 22,745 σχέσεις ανάμεσά τους. Κατόπιν, θα αναπτύξουμε ερωτήματα σε Cypher τα οποία θα παράγουν γνώση από τα υπάρχοντα δεδομένα. Τα αποτελέσματα των ερωτημάτων μπορούν εύκολα να οπτικοποιηθούν σε κόμβους (δεδομένα) και συνδεόμενες ακμές (σχέσεις ανάμεσα στα δεδομένα). Τα στιγμιότυπα των ερωτημάτων θα επισυνάπτονται κάτω από το αντίστοιχο ερώτημα σε Cypher.

Αρχικά περιγράφουμε μερικά γενικά βοηθητικά ερωτήματα:

Απεικόνιση όλων των κόμβων και ακμών:

```
MATCH (n) OPTIONAL MATCH (n)-[r]-() RETURN n,r;
```

Διαγραφή όλων των κόμβων:

```
MATCH (n) DETACH DELETE n;
```

Στη συνέχεια περιγράφουμε ορισμένα εξειδικευμένα ερωτήματα που μας δίνουν καλύτερη εικόνα των δεδομένων.

Q1. Ποιοί είναι οι πρώτοι 5 προγραμματιστές που δημιούργησαν τα περισσότερα issues;

Αναζητούμε τους πρώτους 5 προγραμματιστές βάσει του πόσα issues έχουν δημιουργήσει. Στην κορυφή του πίνακα βρίσκονται οι προγραμματιστές που δημιούργησαν τα περισσότερα.

```
MATCH (p:Person)-[r:creates]->(i:Issue)
RETURN p.key as Programmer, count(i) AS Issue_count
ORDER BY Issue_count DESC LIMIT 5;
```

Programmer	Issue_count
"stevel@apache.org"	198
"ajisakaa"	47
"mackrorysd"	30
"jojochuang"	23
"danielzhou"	21

Εικόνα 2.5: Πίνακας αποτελεσμάτων – προγραμματιστές & αριθμός των issues.

Q2. Ποιοί είναι οι πρώτοι 5 προγραμματιστές που τους έχουν ανατεθεί τα λιγότερα issues;

Αναζητούμε τους πρώτους 5 προγραμματιστές βάσει του πόσα λιγότερα issues έχουν δημιουργήσει. Στην κορυφή του πίνακα βρίσκονται οι προγραμματιστές που τους έχουν ανατεθεί τα λιγότερα.

```
MATCH (p:Person)-[r:is_assigned_to]->(i:Issue)
RETURN p.key as Programmer, count(i) AS Issue_count
ORDER BY Issue_count ASC LIMIT 5;
```

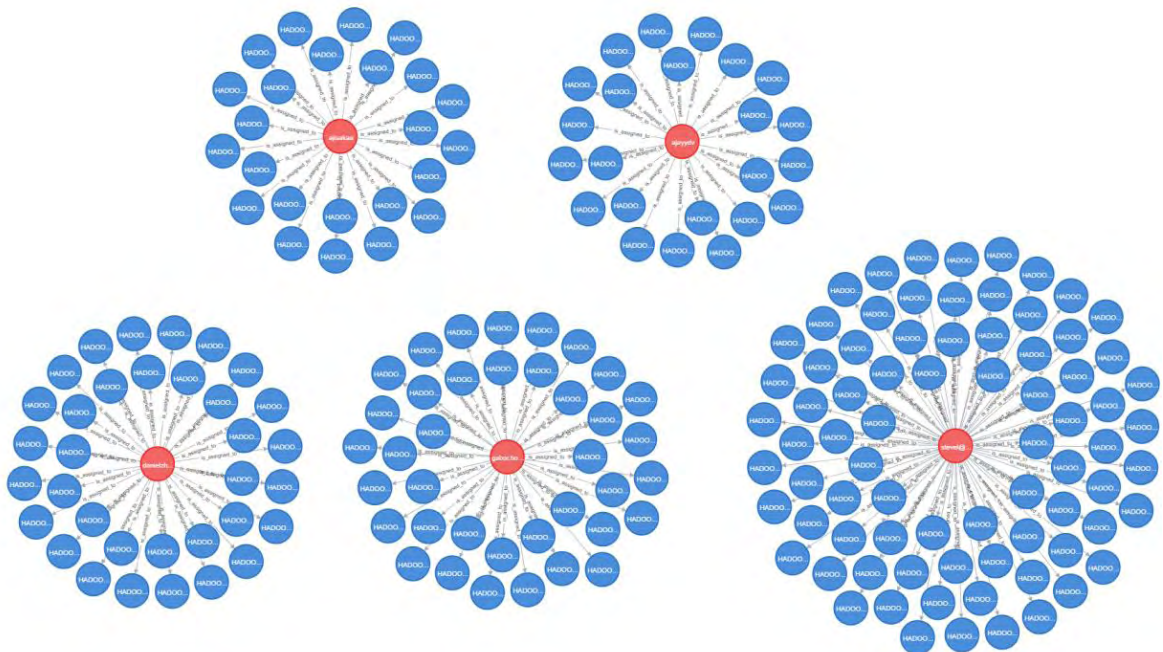
Programmer	Issue_count
"andrioni"	1
"cmukka20"	1
"rajesh.balamohan"	1
"jiwq"	1
"iapicker"	1

Εικόνα 2.6: Πίνακας αποτελεσμάτων – προγραμματιστές & αριθμός των issues.

Αν θέλαμε να οπτικοποιήσουμε γραφικά την πληροφορία ενός εκ των παραπάνω ερωτημάτων π.χ. του Q1, θα αλλάζαμε λίγο το ερώτημα ως εξής:

```

MATCH (i1:Issue)--(i2:Issue)
MATCH (p:Person)-[r:is_assigned_to]->(i:Issue)
WITH p, count(i) AS issue_count
ORDER BY issue_count DESC LIMIT 5
WITH collect(p.key) AS Programmers
UNWIND Programmers AS Programmer
MATCH (p:Person {key: Programmer})-[r:is_assigned_to]->(i:Issue)
RETURN p,r,i
    
```



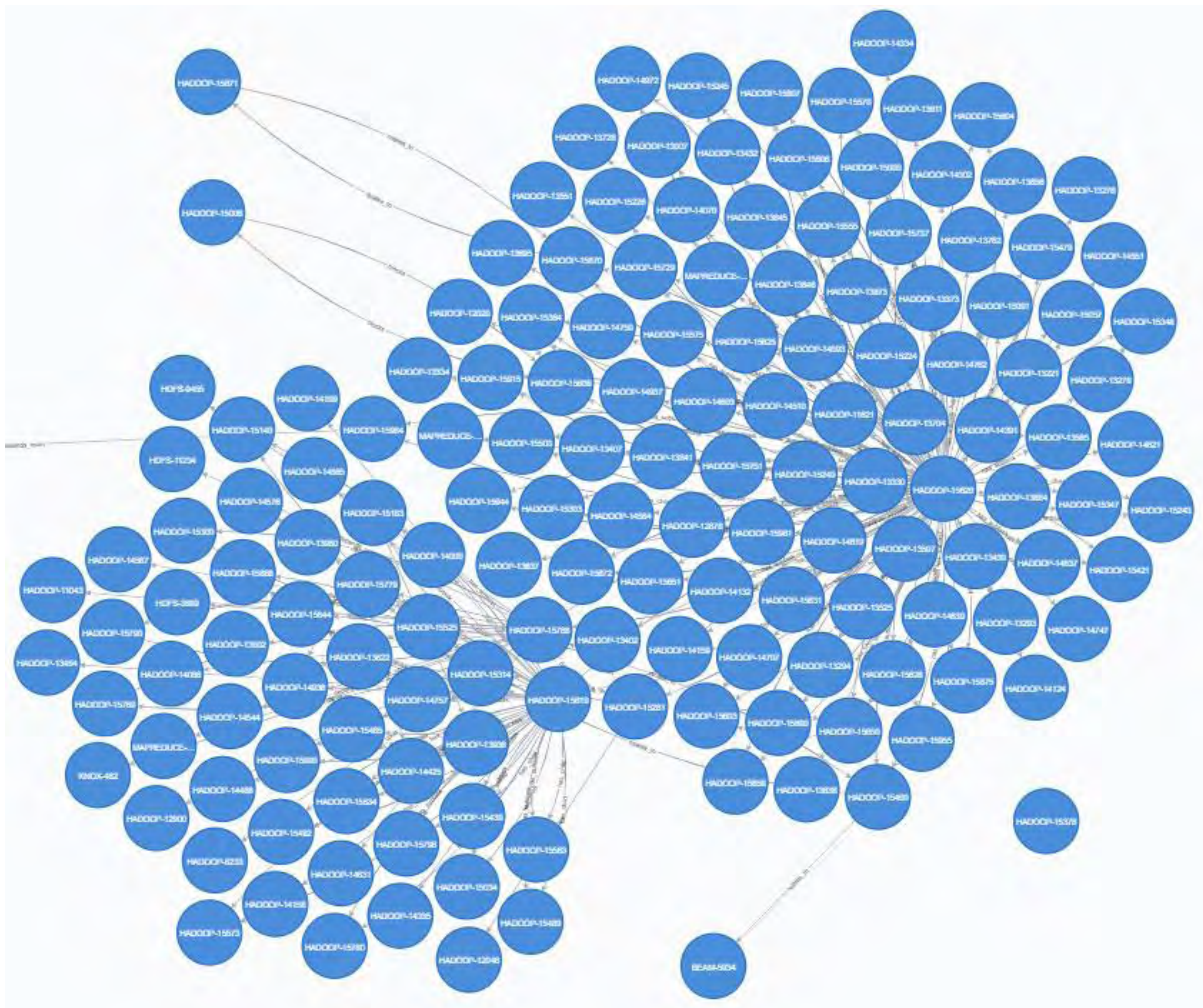
Εικόνα 2.7: Γράφος αποτελεσμάτων – προγραμματιστής (κόκκινος κόμβος) – issue (μπλε κόμβος).

Q3. Ποιά issues είναι ανοιχτά;

Αναζητούμε τα ανοιχτά, δηλαδή τα μη επιλυθέντα issues.

Γενικά τα issues μπορεί να έχουν ως status τις εξής τιμές: open (ανοιχτά), closed (κλειστά), resolved (επιλυθέντα), patch available (διαθέσιμη αναβάθμιση), in progress (σε εξέλιξη), reopened (ανοιχτά ξανά).

```
MATCH (i:Issue {status: "Open"})-[r]-() RETURN i, r;
```

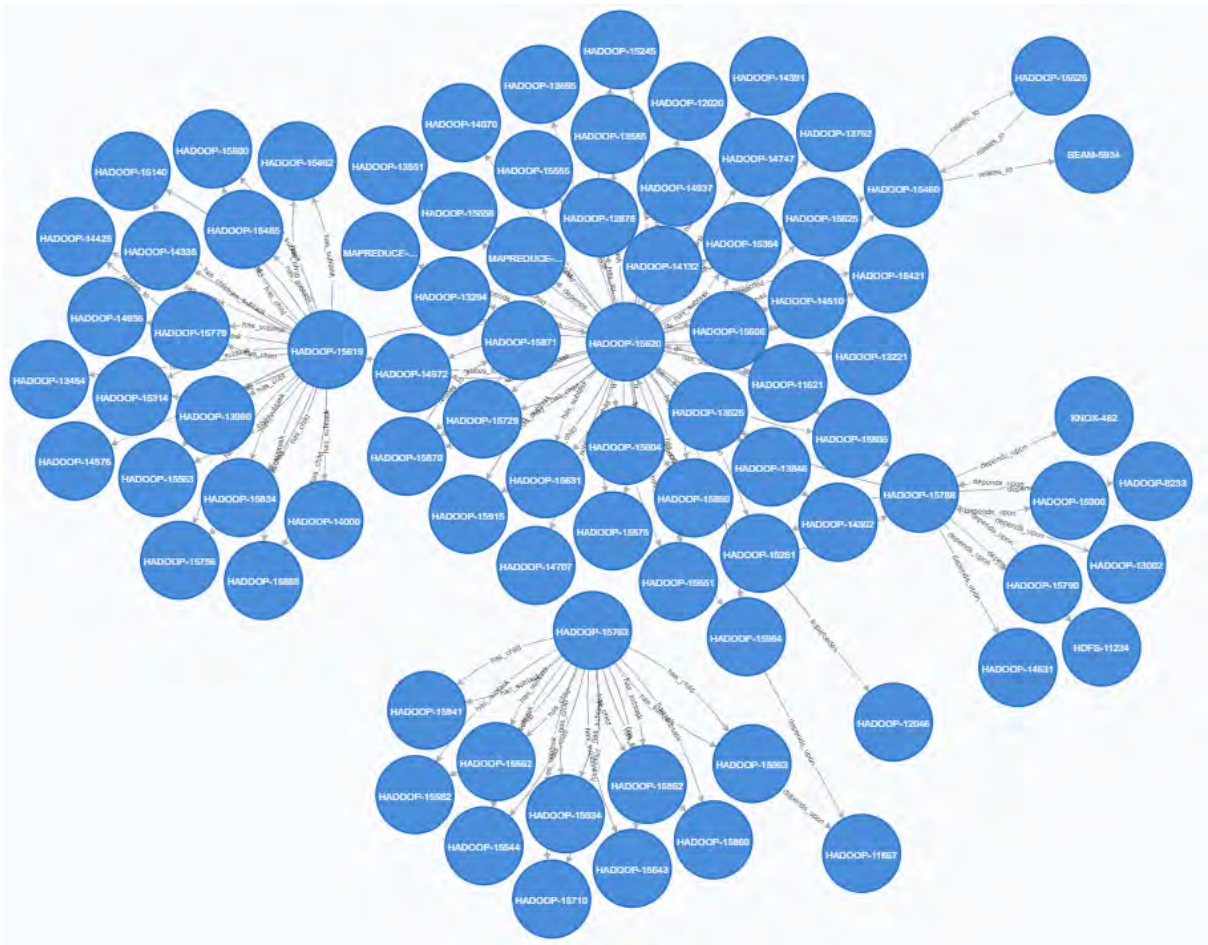


Εικόνα 2.8: Γράφος μερικών αποτελεσμάτων – issue (μπλε κόμβος) – σχέσεις που τα συνδέουν (γκρι ετικέτες / ακμές).

Q4. Ποιά issues είναι ανοιχτά και έχουν μέγιστη προτεραιότητα;

Αναζητούμε τα ανοιχτά, δηλαδή τα μη επιλυθέντα issues που έχουν μέγιστη προτεραιότητα. Γενικά τα issues μπορεί να έχουν ως προτεραιότητα τις τιμές: Major (μέγιστη) ή Minor (ελάχιστη).

```
MATCH (i:Issue {status: "Open", priority: "Major"})-[r]-()
RETURN i, r;
```



Εικόνα 2.9: Γράφος μερικών αποτελεσμάτων – issue (μπλε κόμβος) – σχέσεις που τα συνδέουν (γκρι ετικέτες / ακμές).

Q5. Ποιοί προγραμματιστές βρίσκονται σε μη ανοιχτά issues (NOT open);

Αναζητούμε προγραμματιστές που δουλεύουν σε issues τα οποία δεν είναι ανοιχτά (open) αλλά έχουν οποιαδήποτε άλλη τιμή στο status τους.

```
MATCH (p:Person)-[]-(i:Issue)
WITH p.key AS Programmer, collect(i.status) AS Status
MATCH (p:Person {key: Programmer})
WHERE NOT "Open" IN Status
RETURN Programmer;
```

Programmer
"wujinhu"
"logesh0589"
"ayushtkn"
"vivek.kumar"
"dhirensk@gmail.com"
"tsammut"
"pgolash"
"zvenczel"
"ssulav"

Εικόνα 2.10: Πίνακας αποτελεσμάτων – μερικά ονόματα προγραμματιστών

Q6. Ποιοί προγραμματιστές βρίσκονται μόνο σε ανοιχτά issues;

Αναζητούμε προγραμματιστές που δουλεύουν σε issues τα οποία είναι μόνο ανοιχτά και δεν έχουν καμία άλλη τιμή στο status τους.

```
MATCH (p:Person)-[]-(i:Issue)
WITH p.key AS Programmer, collect(i.status) AS Status
MATCH (p:Person {key: Programmer})
WHERE NONE(s IN Status WHERE (s = "Resolved" OR s = "Patch
Available" OR s = "Closed"))
RETURN Programmer;
```

Programmer
"bjh"
"oneisall"
"umamaheswararao"
"elaineang"
"moist"
"kgyrtkirk"
"cyl"
"vancobuca"
"sanxiago"

Εικόνα 2.11: Πίνακας αποτελεσμάτων – μερικά από τα ονόματα προγραμματιστών

Q7. Πόσα και ποιού τύπου issues έχουν αναλάβει οι προγραμματιστές;

Αναζητούμε προγραμματιστές που δουλεύουν σε issues και για καθέναν από αυτούς τα ομαδοποιούμε σε λίστες όπου το πρώτο στοιχείο είναι το status του issue και το δεύτερο στοιχείο είναι το πλήθος τέτοιων τύπων issues.

```
MATCH (p:Person)-[]-(i:Issue)
WITH p, i.status AS Status, count(*) AS StatusCount
RETURN p.key AS Programmer,
       collect([Status, StatusCount]) AS IssueType;
```

Programmer	IssueType
"wujinhu"	[["Resolved", 36], ["Patch Available", 6]]
"bjh"	[["Open", 2]]
"lukmajercak"	[["Resolved", 9], ["Open", 2]]
"logesh0589"	[["Resolved", 2]]
"ayushtkn"	[["Resolved", 7]]

Εικόνα 2.12: Πίνακας αποτελεσμάτων – μερικά από τα ονόματα προγραμματιστών και το πλήθος των issues που έχει ο καθένας τους ανά κατηγορία

Q8. Ποια είναι τα πρώτα 5 issues που συνδέονται με τα περισσότερα issues;

Αναζητούμε τα 5 πρώτα Issues που έχουν τις περισσότερες συνδέσεις με άλλα.

```
MATCH (i1:Issue)--(i2:Issue)
WITH i1 AS Issue, count(i2) AS Issue_count
RETURN Issue.key AS Issue_name, Issue_count
ORDER BY Issue_count DESC LIMIT 5;
```

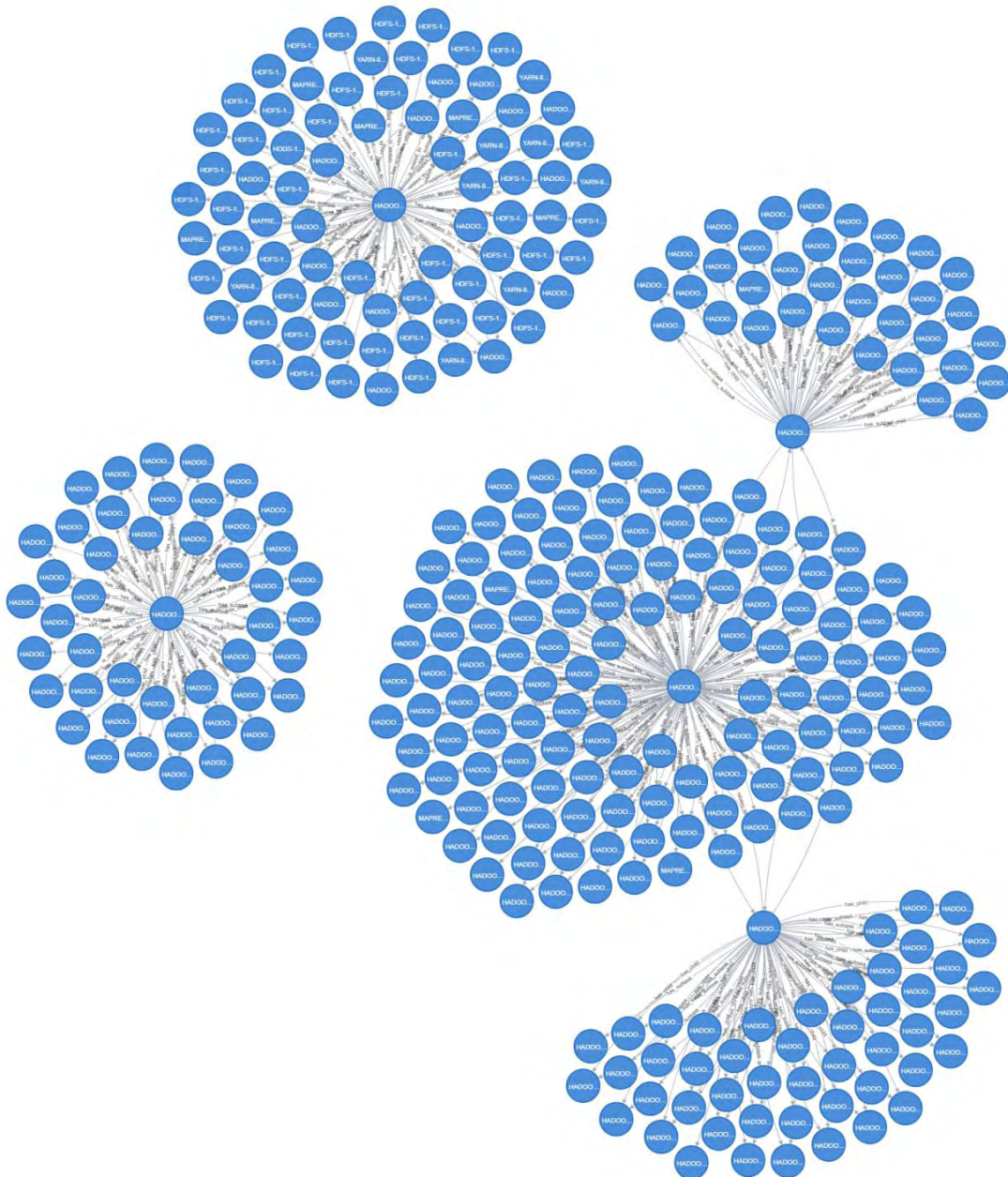
Issue_name	Issue_count
"HADOOP-15620"	188
"HADOOP-15475"	106
"HADOOP-15407"	90
"HADOOP-15619"	85
"HADOOP-15220"	63

Εικόνα 2.13: Πίνακας αποτελεσμάτων – όνομα issue και αριθμός issues που συνδέεται

Αν θέλαμε να οπτικοποιήσουμε γραφικά το παραπάνω ερώτημα θα το αλλάζαμε ως εξής (τα αποτελέσματα εμφανίζονται στην Εικόνα 2.14):

```

MATCH (i1:Issue)--(i2:Issue)
WITH i1 AS Issue, count(i2) AS Issue_count
ORDER BY Issue_count DESC LIMIT 5
WITH collect(Issue.key) AS Collection
UNWIND Collection AS issue_key
MATCH (i1:Issue {key: issue_key })-[r]-(i2:Issue)
RETURN i1,r,i2;
    
```



Εικόνα 2.14: Γράφος αποτελεσμάτων – issue (στο κέντρο κάθε συστάδας) που συνδέεται με πολλά issues.

3

Αξιοποίηση Μέτρων ομοιότητας σε ζητήματα Project Management

3.1 Εισαγωγή

Το Project Management είναι η πρακτική της ανάλυσης, σχεδίασης, εκτέλεσης, ελέγχου και περάτωσης της δουλειάς ενός έργου μιας ομάδας ανθρώπων με σκοπό την επίτευξη ορισμένων στόχων σχετικών με το κόστος, το χρόνο κ.τ.λ. Η κύρια δυσκολία του Project Management έγκειται στην ταυτόχρονη ικανοποίηση των ακόλουθων ζητημάτων [14]:

Καλός προσδιορισμός των στόχων του Project

Οι στόχοι του project, το συνολικό έργο και όλα τα επιμέρους έργα (υποέργα που απαιτούνται για την περάτωση του) θα πρέπει να έχουν οριστεί με πλήρη σαφήνεια, έτσι ώστε να αποφευχθούν οποιοδήποτε είδους καθυστερήσεις.

Διαχείριση Πόρων

Για την επιτυχή ολοκλήρωση του project, αξιοποιούνται χρόνος, χρήμα και φυσικά άνθρωποι. Συχνά, αυτοί οι πόροι ανταγωνίζονται μεταξύ τους, π.χ. όταν μικραίνουμε τον χρόνο εκτέλεσης του project, αυξάνεται το κόστος και οι άνθρωποι που συμμετέχουν σε αυτό).

Διαχείριση Ρίσκου

Κατά τη διάρκεια του σχεδιασμού ενός project, θα πρέπει να φτιαχτούν σενάρια τα οποία προβλέπουν διάφορα ρίσκα (όπως αυτά ορίζονται στο ISO 31000 [15]). Τα ρίσκα αυτά έχουν συνήθως αρνητικές επιπτώσεις (καθυστέρηση στην παράδοση του project, project εκτός προϋπολογισμού κλπ).

Κατανομή ανθρώπων σε υποέργα

Καθ' όλη την διάρκεια ενός project, ο Project Manager καλείται να κατανείμει ανθρώπους σε υποέργα του project. Η διαδικασία αυτή δυσχεραίνεται όσο περισσότερη ανεπαρκής επικοινωνία υπάρχει μεταξύ των ανθρώπων που δουλεύουν σε αυτό. Πρώτον, η ανεπαρκής επικοινωνία ανάμεσα στον Project Manager και στους προγραμματιστές, καθιστά τον πρώτο να μην γνωρίζει πλήρως τις δεξιότητες του καθενός προγραμματιστή, γεγονός το οποίο οδηγεί τον Project Manager να αποφασίσει με ελλιπή δεδομένα. Δεύτερον, η ανεπαρκής επικοινωνία ανάμεσα στους ίδιους τους προγραμματιστές, λόγω ανομοιότητας στον τρόπο εργασίας, οδηγεί σε δυσκολία διαχείρισης του εργασιακού φόρτου και την επικείμενη καθυστέρηση που αυτή συνεπάγεται.

Στην παρακάτω ενότητα θα ορίσουμε θέματα ομοιότητας στις δουλειές που αναλαμβάνουν οι προγραμματιστές, θέματα ομοιότητας στον τρόπο εργασίας τους, καθώς και κριτήρια που αναδεικνύουν τις δεξιότητες τους.

3.2 Βασικά Μέτρα Ομοιότητας

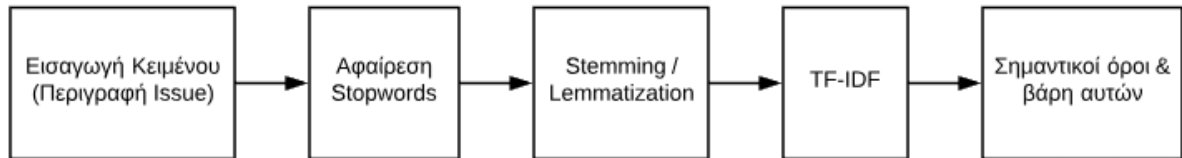
Ως τώρα, στην προσπάθειά μας, να παράγουμε το γράφο που περιλαμβάνει τα Jira Issues, τους προγραμματιστές τους και τις μεταξύ τους σχέσεις, χρησιμοποιήσαμε τα δεδομένα. Για να μπορέσουμε να παράγουμε περισσότερη γνώση από τον εν λόγω γράφο θα αξιοποιήσουμε ένα πεδίο που βρίσκεται σε κάθε issue, το οποίο είναι το description, δηλαδή η περιγραφή του κάθε issue. Το πεδίο αυτό, αποτελείται από μη-δομημένα δεδομένα (κείμενο σε φυσική γλώσσα που έχει γραφτεί από τους προγραμματιστές που δούλεψαν στο συγκεκριμένο issue).

Στόχος μας είναι να οργανώσουμε όλα τα issues (βάσει της περιγραφής τους) σε συστάδες (clusters) χρησιμοποιώντας βασικά μέτρα ομοιότητας τα οποία θα δράσουν σε θεματικές ετικέτες που παράχθηκαν κατόπιν εφαρμογής NLP μεθοδολογιών στις περιγραφές των issues.

Πρώτον, πρέπει να βρούμε τους πιο σημαντικούς όρους στο κείμενο. Για να το κάνουμε αυτό σπάμε το κείμενο σε προτάσεις, και κατόπιν λέξεις, αφαιρώντας συνήθεις λέξεις που δεν προσδίδουν πρόσθετη σημασία στην ανάλυση που κάνουμε όπως, πολλά οριστικά ή αόριστα άρθρα (a, an, the) ή πολλές διαζευκτικές εκφράσεις (and, or, not). Στη συνέχεια για κάθε λέξη εφαρμόζουμε την διαδικασία του Stemming (περικοπή καταλήξεων για να καταλήξουμε στο ουσιαστικό που προκύπτει από την λέξη) ή την διαδικασία του Lemmatization (μετατροπή σύνθετου όρου στον βασικό όρο/λήμμα- όπως θα τον βρίσκαμε σε ένα λεξικό).

Αφού οι όροι έρθουν στην κατάλληλη μορφή μετράμε τη συχνότητα εμφάνισης τους στο κείμενο TF (term-frequency), θεωρώντας πως οι πιο εμφανιζόμενοι όροι χαρακτηρίζουν το κείμενο. Αυτή η μετρική, από μόνη της μπορεί να είναι παραπλανητική καθώς το κείμενο μπορεί να περιλαμβάνει πολλά οριστικά ή αόριστα άρθρα (a, an, the) ή πολλές διαζευκτικές εκφράσεις (and, or, not) ή χρονικές / συγκριτικές εκφράσεις (then, than). Για αυτό το λόγο, θα χρησιμοποιήσουμε την πιο σύνθετη μετρική του TF-IDF (term frequency – inverse document

frequency / συχνότητα όρου – ανάστροφη συχνότητα κειμένου). Η φυσική σημασία της μετρικής είναι να βρει τους πιο σημαντικούς όρους στο κείμενο και τα βάρη τους, δηλαδή αριθμητικές τιμές που δείχνουν την σημασία τους, δίνοντας μεγαλύτερη σημασία στους όρους που εμφανίζονται συχνά αλλά όχι αρκετά συχνά για να αποτελούν περιττή πληροφορία στο κείμενο. Η διαδικασία που ακολουθήσαμε συνοψίζεται στην Εικόνα 3.1.



Εικόνα 3.1: Μοντέλο διαδικασίας παραγωγής ετικετών

Το επόμενο βήμα, μετά την παραγωγή των θεματικών ετικετών βάσει του κειμένου, είναι να εξετάσουμε την ομοιότητα ανάμεσα στα issues (ή / και στους προγραμματιστές) χρησιμοποιώντας μέτρα απόστασης / ομοιότητας.

Ένα μέτρο απόστασης / ομοιότητας εξετάζει το βαθμό εγγύτητας ή μη εγγύτητας ανάμεσα στα χαρακτηριστικά (όροι) των αντικείμενων (κειμένων) που διαχωρίζουν τα δεδομένα (issues / προγραμματιστές) σε clusters. Τα χαρακτηριστικά αυτά ποικίλουν αναλόγως του προβλήματος, οπότε δεν υπάρχει μια γενική καλύτερη μετρική για όλα τα προβλήματα clustering.

Γενικά, τα μέτρα απόστασης / ομοιότητας μετρούν την απόσταση ή ομοιότητα ανάμεσα σε δύο αντικείμενα βάσει ενός συνόλου χαρακτηριστικών και παράγουν έναν πραγματικό αριθμό, συνήθως στο διάστημα $[0, 1]$, ο οποίος αν προσεγγίζει στο 0 δείχνει ότι τα δεδομένα δεν είναι κοντινά ενώ αν προσεγγίζει στο 1 ότι είναι πολύ κοντινά. Παρακάτω περιγράψουμε ορισμένα μέτρα απόστασης / ομοιότητας [13]:

3.2.0 Ορισμοί

Έστω ένα σύνολο κειμένων $d = \{d_1, d_2, \dots, d_n\}$ και $t = \{t_1, t_2, \dots, t_n\}$ το σύνολο διακριτών όρων κειμένου και το βάρος του κάθε όρου $w_{i,x}$, δηλαδή πόσες φορές εμφανίζεται. Θεωρούμε επίσης ότι TF_{di} είναι η συχνότητα του κάθε όρου ενός κειμένου.

3.2.1 Ευκλείδεια απόσταση

Αποτελεί ένα από τα πιο βασικά μέτρα για γεωμετρικά προβλήματα και μη. Η φυσική της σημασία είναι ότι εκφράζει την απόσταση ανάμεσα σε δύο σημεία. Χρησιμοποιείται ευρέως σε προβλήματα clustering και ορίζεται ως:

$$D_{\text{EUCL}}(t_a, t_b) = \sqrt{\sum_{i=0}^n (w_{i,a} - w_{i,b})^2}$$

όπου a, b τα δύο προς σύγκριση κείμενα και $w_{i,x}$ το βάρος του κάθε όρου του κειμένου. Όταν είναι 0, τα κείμενα ταυτίζονται.

3.2.2 Ομοιότητα Cosine

Όταν δύο οποιαδήποτε κείμενα d_a και d_b αναπαρίστανται ως διανύσματα t_a και t_b αντίστοιχα τα οποία περιέχουν τους όρους και τα βάρη τους, η ομοιότητα τους αντιστοιχεί στη συσχέτισή τους (δηλαδή στην γωνία που σχηματίζουν τα διανύσματα όπως αυτή εκφράζεται στο εσωτερικό τους γινόμενο που βρίσκεται στον παρανομαστή του κλάσματος).

$$S_{\text{COS}}(t_a, t_b) = \frac{\tau_a \cdot \tau_b}{\|\tau_a\| \times \|\tau_b\|}$$

όπου t_1 και t_2 είναι n -διάστατα διανύσματα που περιέχουν τους όρους και τα βάρη τους. Κυμαίνεται ανάμεσα στο διάστημα $[0, 1]$. Γίνεται 1 όταν τα κείμενα ταυτίζονται και 0 όταν είναι τελείως διαφορετικά.

3.2.3 Ομοιότητα Jaccard

Η ομοιότητα (ή συντελεστής) Jaccard, μετρά την ομοιότητα στο κλάσμα ανάμεσα στην τομή και στην ένωση των κειμένων. Αυτό, επιτυγχάνεται συγκρίνοντας τα κοινά βάρη των διανυσμάτων όρων των κειμένων έναντι των βαρών που ανήκουν σε ένα ή και στα 2 διανύσματα όρων των κειμένων t_a και t_b .

$$S_{\text{JACC}}(t_a, t_b) = \frac{t_a \cdot t_b}{\|t_a\|^2 + \|t_b\|^2 - t_a \cdot t_b}$$

Κυμαίνεται ανάμεσα στο διάστημα $[0, 1]$. Γίνεται 1 όταν τα κείμενα ταυτίζονται και 0 όταν είναι τελείως διαφορετικά.

3.2.4 Ομοιότητα Συσχέτισης Pearson

Η ομοιότητα ή συντελεστής συσχέτισης Pearson, μετρά την ομοιότητα ανάμεσα σε δύο συσχετιζόμενα διανύσματα όρων κείμενου t_a και t_b .

$$S_{\text{PEAR}}(t_a, t_b) = \frac{n \sum_{i=1}^n w_{i,a} \times w_{i,b} - TF_a \times TF_b}{\sqrt{[n \sum_{i=1}^n w_{i,a}^2 - TF_a^2][n \sum_{i=1}^n w_{i,b}^2 - TF_b^2]}}$$

Σε αντίθεση με τα άλλα μέτρα απόστασης / ομοιότητας κυμαίνεται στο διάστημα $[-1, 1]$. Γίνεται 1 όταν τα κείμενα ταυτίζονται, 0 όταν είναι τελείως διαφορετικά και -1 όταν τα κείμενα συνδέονται με γραμμική σχέση στην οποία το ένα αυξάνεται και το άλλο μειώνεται ως προς τα διανύσματα τους.

3.3 Ομοιότητα μεταξύ *issues*

Σε αυτή την ενότητα, θα ενσωματώσουμε σε ερωτήματα Cypher, τα παραπάνω μέτρα ομοιότητας, αφού πρώτα ορίσουμε το μοντέλο ομοιότητας ανάμεσα στους κόμβους.

Εσωτερικά στην βάση υφίστανται κόμβοι τύπου Issue (θέμα) ή Person (άνθρωπος, αναφέρεται στους προγραμματιστές), οι οποίοι έχουν κοινές ετικέτες μεταξύ τους. π.χ. Issues με κοινές ετικέτες, ή προγραμματιστές με κοινές ετικέτες. Ένα ακραίο παράδειγμα κόμβων με ομοιότητα 1.0 (100%) είναι το ακόλουθο:

Αυτές οι ετικέτες (κόμβοι τύπου Tag) παρήχθησαν με την χρήση NLP μεθόδων στην περιγραφή του κάθε issue. Κάθε ακμή περιέχει ένα βάρος (weight) που εκφράζει το TF-IDF score της συγκεκριμένης ετικέτας.

Κάθε ένας από τα παρατιθέμενα μέτρα ομοιότητας εξετάζει τους κοινούς κόμβους (ετικέτες) ανάμεσα στους συγκρινόμενους κόμβους, καθώς και το βάρος των ακμών με μοναδική εξαίρεση τον Jaccard (στο τελευταίο).



Εικόνα 3.2: Ομοιότητα στο μοντέλο ετικετών

3.3.1 Ομοιότητα Jaccard

Q9. Ποιά issues είναι όμοια με ένα συγκεκριμένο issue; (π.χ. HADOOP-15483)

Αναζητούμε τα issues που είναι παρόμοια με το HADOOP-15483 σύμφωνα με το μέτρο ομοιότητας Jaccard με ποσοστό ομοιότητας μεγαλύτερο του 50% και εμφανίζουμε τα ζεύγη όμοιων issues καθώς και το βαθμό ομοιότητας τους.

```

MATCH (i:Issue)-[:has_tag]->(t:Tag)
WITH {item:id(i), categories: collect(id(t))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.jaccard.stream(data,
  {topK:1, similarityCutoff: 0.5})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
WHERE from.key = 'HADOOP-15483' OR to.key = 'HADOOP-15483'
//Only show records that contain the issue's name
RETURN from, to, round(factor * value)/factor AS similarity;
    
```

from	to	similarity
"HADOOP-15484"	"HADOOP-15483"	0.57
"HADOOP-15483"	"HADOOP-15482"	0.67

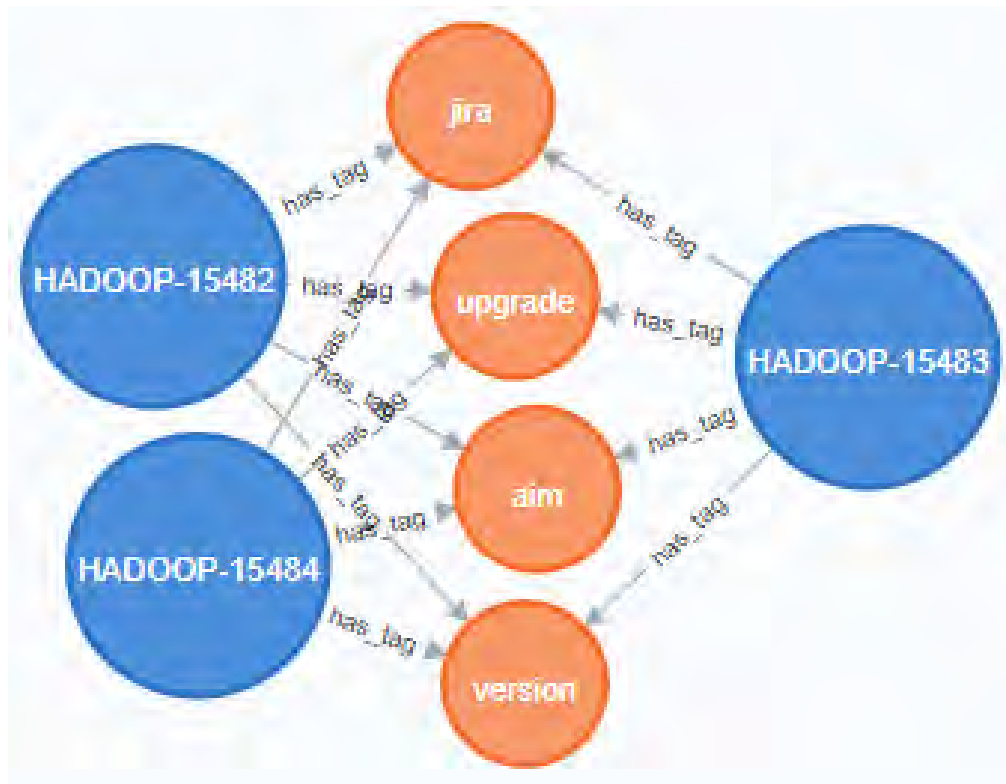
Εικόνα 3.3: Πίνακας αποτελεσμάτων – HADOOP-15483, ομοιάζοντα issues και βαθμός ομοιότητας.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής:

```

MATCH (i:Issue)-[:has_tag]->(t:Tag)
WITH {item:id(i), categories: collect(id(t))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.jaccard.stream(data,
    {topK:1, similarityCutoff: 0.5})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to
WHERE from.key = 'HADOOP-15483' OR to.key = 'HADOOP-15483'
//Only show records that contain the issue's name
WITH collect(from) AS from_list, collect(to) AS to_list
MATCH (i1:Issue)-[r1:has_tag]->(t:Tag)-[r2:has_tag]-
(i2:Issue)
WHERE (i1.key IN from_list) AND (i2.key IN to_list)
RETURN i1, i2, r1, r2, t;

```



Εικόνα 3.4: Γράφος αποτελεσμάτων για τα παρόμοια issues και ετικέτες του issue HADOOP-15483.

Q10. Ποιά issues είναι όμοια με ποιά άλλα issues;

Αναζητούμε τα issues που είναι παρόμοια μεταξύ τους σύμφωνα με το μέτρο ομοιότητας Jaccard με ποσοστό ομοιότητας μεγαλύτερο του 70% και εμφανίζουμε τα ζεύγη όμοιων issues καθώς και το βαθμό ομοιότητας τους.

```

MATCH (i:Issue)-[:has_tag]->(t:Tag)
WITH {item:id(i), categories: collect(id(t))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.jaccard.stream(data,
    {topK:1, similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
RETURN from, to, round(factor * value)/factor AS similarity;
    
```

from	to	similarity
"HADOOP-15978"	"HADOOP-15979"	1.0
"HADOOP-15990"	"HADOOP-15409"	0.76
"HADOOP-15964"	"HADOOP-15963"	0.84
"HADOOP-15633"	"HADOOP-15615"	0.7
"HADOOP-15754"	"HADOOP-15800"	1.0
"HADOOP-15764"	"HADOOP-15756"	0.71
"HADOOP-15644"	"HADOOP-15646"	0.92
"HADOOP-15727"	"HADOOP-15052"	0.8
"HADOOP-15555"	"HADOOP-15556"	1.0
"HADOOP-15465"	"HADOOP-15522"	0.77
"HADOOP-15521"	"HADOOP-15506"	1.0
"HADOOP-15266"	"HADOOP-15255"	1.0
"HADOOP-15238"	"HADOOP-15003"	0.84
"HADOOP-15111"	"HADOOP-14993"	1.0

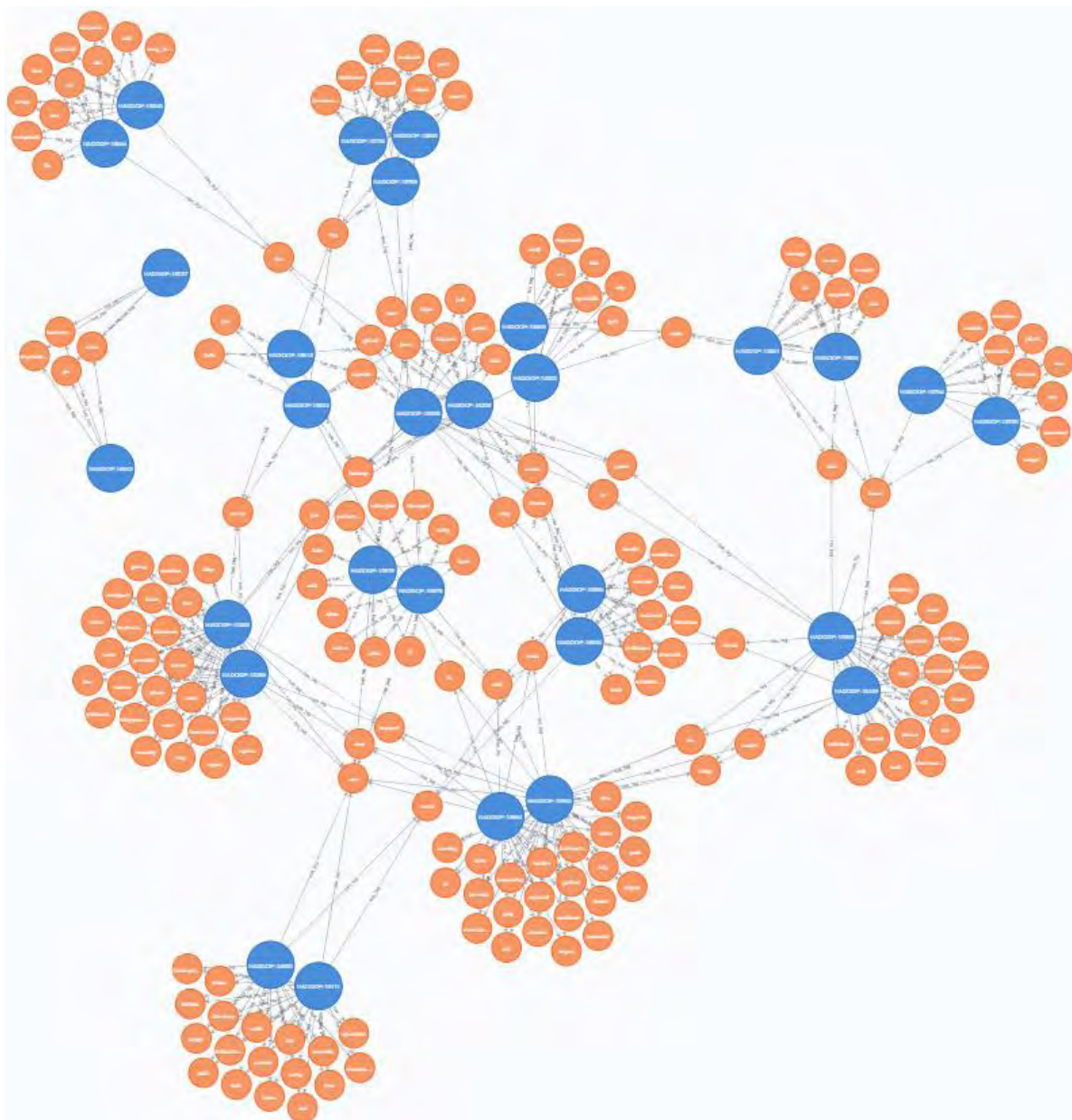
Εικόνα 3.5: Πίνακας αποτελεσμάτων – ομοιάζοντα issues και βαθμός ομοιότητας.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής:

```

MATCH (i:Issue)-[:has_tag]->(t:Tag)
WITH {item:id(i), categories: collect(id(t))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.jaccard.stream(data,
{topK: 1, similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1).key AS from, algo.asNode(item2).key AS to
WITH collect(from) as from_list, collect(to) AS to_list
MATCH (i1:Issue)-[r1:has_tag]
->(t:Tag)<-[r2:has_tag]-(i2:Issue)
WHERE i1.key IN from_list AND i2.key IN to_list
RETURN i1, r1, t, r2, i2;

```



Εικόνα 3.6: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι ετικέτες τους (πορτοκαλί κόμβοι)

3.3.2 Ομοιότητα Cosine

Q11. Ποιά issues είναι όμοια με ένα συγκεκριμένο issue; (π.χ HADOOP-14977)

Αναζητούμε τα issues που είναι παρόμοια με το HADOOP-14977 σύμφωνα με το μέτρο ομοιότητας Cosine με ποσοστό ομοιότητας μεγαλύτερο του 70% και εμφανίζουμε τα ζεύγη όμοιων issues καθώς και το βαθμό ομοιότητας τους.

```
MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
```

```

WITH collect(Data) AS data
CALL algo.similarity.cosine.stream(data,
  {topK: 1, similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
WHERE from = 'HADOOP-14977' OR to = 'HADOOP-14977'
//Only show records that contain the issue's name
RETURN from, to, round(factor * value)/factor AS similarity;

```

from	to	similarity
"HADOOP-15866"	"HADOOP-14977"	0.98
"HADOOP-15713"	"HADOOP-14977"	0.98
"HADOOP-15610"	"HADOOP-14977"	0.98
"HADOOP-15404"	"HADOOP-14977"	0.99
"HADOOP-15384"	"HADOOP-14977"	0.98
"HADOOP-15109"	"HADOOP-14977"	0.99

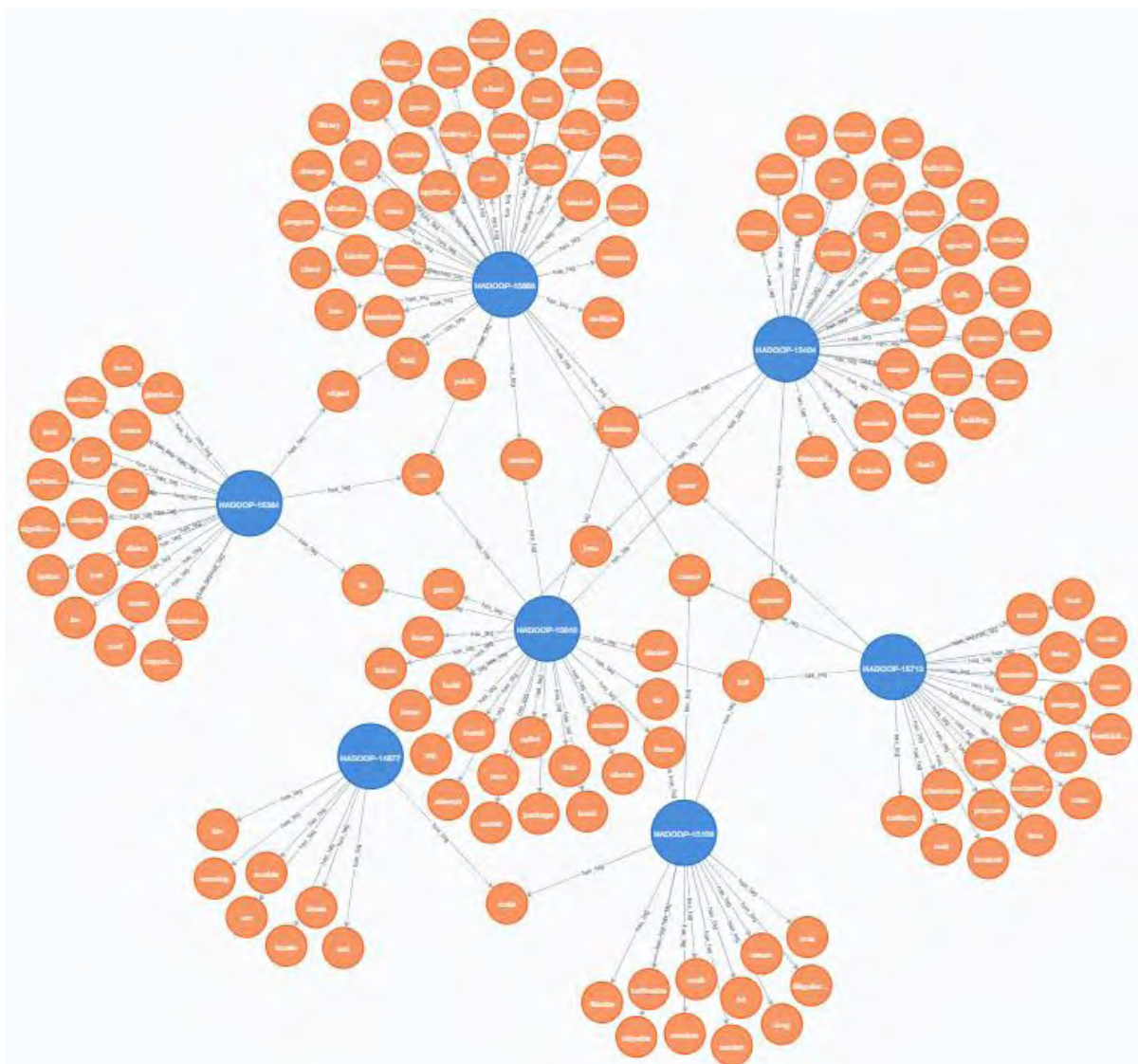
Εικόνα 3.7: Πίνακας αποτελεσμάτων – HADOOP-14977, παρόμοια issues και ο βαθμός ομοιότητας τους.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής:

```

MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.cosine.stream(data,
  {topK: 1, similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to
WHERE from.key = 'HADOOP-14977' OR to.key = 'HADOOP-14977'
//Only show records that contain the issue's name
WITH collect(from) AS from_list, collect(to) AS to_list
MATCH (i1:Issue)-[r1:has_tag]->(t:Tag)
WHERE i1.key IN (from_list + to_list)
RETURN i1, r1, t;

```



Εικόνα 3.8: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι)

Q12. Ποιά issues είναι όμοια με ποιά άλλα issues;

Αναζητούμε τα issues που είναι παρόμοια μεταξύ τους σύμφωνα με το μέτρο ομοιότητας Cosine με ποσοστό ομοιότητας μεγαλύτερο του 70% και εμφανίζουμε τα ζεύγη όμοιων issues καθώς και το βαθμό ομοιότητας τους.

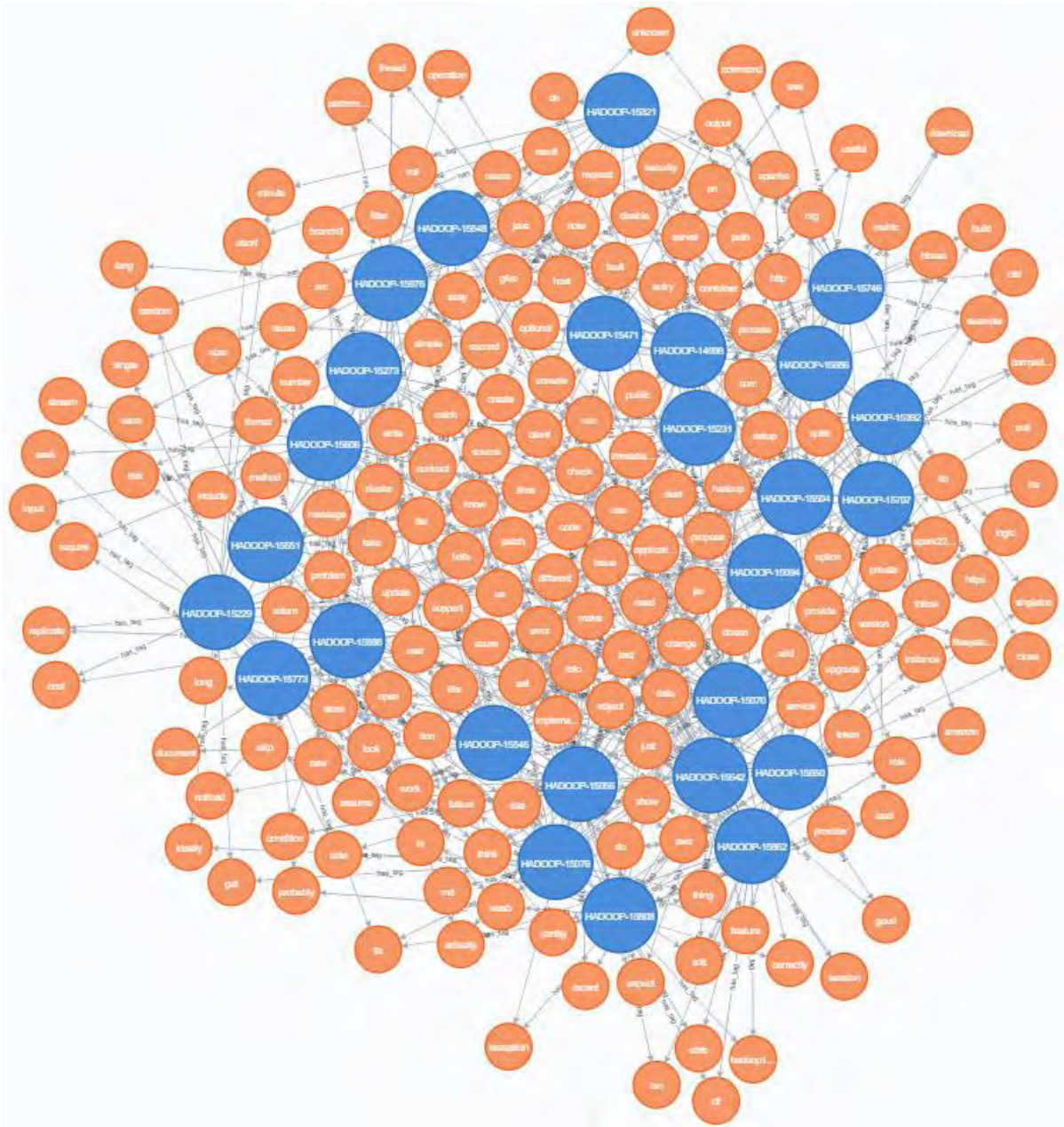
```
MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.cosine.stream(data,
{topK:1, similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
RETURN from, to, round(factor * value)/factor AS similarity;
```

from	to	similarity
"HADOOP-15978"	"HADOOP-15903"	1.0
"HADOOP-15992"	"HADOOP-15268"	0.99
"HADOOP-15759"	"HADOOP-15240"	0.99
"HADOOP-15991"	"HADOOP-15780"	0.97
"HADOOP-15990"	"HADOOP-15619"	1.0
"HADOOP-15982"	"HADOOP-15899"	0.99
"HADOOP-15409"	"HADOOP-15870"	1.0
"HADOOP-15989"	"HADOOP-15265"	0.98
"HADOOP-15988"	"HADOOP-15474"	1.0
"HADOOP-15619"	"HADOOP-15903"	1.0
"HADOOP-15987"	"HADOOP-15268"	1.0
"HADOOP-15986"	"HADOOP-15085"	0.98
"HADOOP-15985"	"HADOOP-15900"	1.0
"HADOOP-15966"	"HADOOP-15573"	1.0
"HADOOP-15994"	"HADOOP-15802"	1.0
"HADOOP-15993"	"HADOOP-15958"	1.0

Εικόνα 3.9: Πίνακας μερικών αποτελεσμάτων – παρόμοια issues και ο βαθμός ομοιότητας τους.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής:

```
MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i),
weights: collect(coalesce(r.weight, algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.cosine.stream(data,
{degreeCutoff: 45, topK: 1, similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1).key AS from,
algo.asNode(item2).key AS to
WITH collect(from) AS from_list,
collect(to) AS to_list
MATCH (i1:Issue)-[r1:has_tag]
->(t:Tag)<-[r2:has_tag]-(i2:Issue)
WHERE i1.key IN from_list AND i2.key IN to_list
RETURN i1, r1, t, r2, i2;
```



Εικόνα 3.10: Γράφος μερικών αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι)

3.3.3 Ομοιότητα Pearson

Q13. Ποιά issues είναι όμοια με ένα συγκεκριμένο issue; (π.χ. HADOOP-15233)

Αναζητούμε τα issues που είναι παρόμοια με το HADOOP-15233 σύμφωνα με το μέτρο ομοιότητας Pearson με ποσοστό ομοιότητας μεγαλύτερο του 70% και εμφανίζουμε τα ζεύγη όμοιων issues καθώς και το βαθμό ομοιότητας τους.

```
MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
```

```

algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.pearson.stream(data, {topK:1,
similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
WHERE from = 'HADOOP-15233' OR to = 'HADOOP-15233'
//Only show records that contain the issue's name
RETURN from, to, round(factor * value)/factor AS similarity

```

from	to	similarity
"HADOOP-15927"	"HADOOP-15233"	0.98
"HADOOP-15777"	"HADOOP-15233"	1.0
"HADOOP-15851"	"HADOOP-15233"	1.0
"HADOOP-15811"	"HADOOP-15233"	1.0
"HADOOP-15801"	"HADOOP-15233"	1.0
"HADOOP-15300"	"HADOOP-15233"	1.0
"HADOOP-15034"	"HADOOP-15233"	1.0
"HADOOP-15707"	"HADOOP-15233"	1.0
"HADOOP-15645"	"HADOOP-15233"	1.0
"HADOOP-15673"	"HADOOP-15233"	1.0
"HADOOP-15570"	"HADOOP-15233"	0.95
"HADOOP-15598"	"HADOOP-15233"	1.0
"HADOOP-15176"	"HADOOP-15233"	1.0
"HADOOP-15022"	"HADOOP-15233"	1.0
"HADOOP-15390"	"HADOOP-15233"	0.88

Εικόνα 3.11: Πίνακας μερικών αποτελεσμάτων – HADOOP-15233, παρόμοια issues και ο βαθμός ομοιότητας τους.

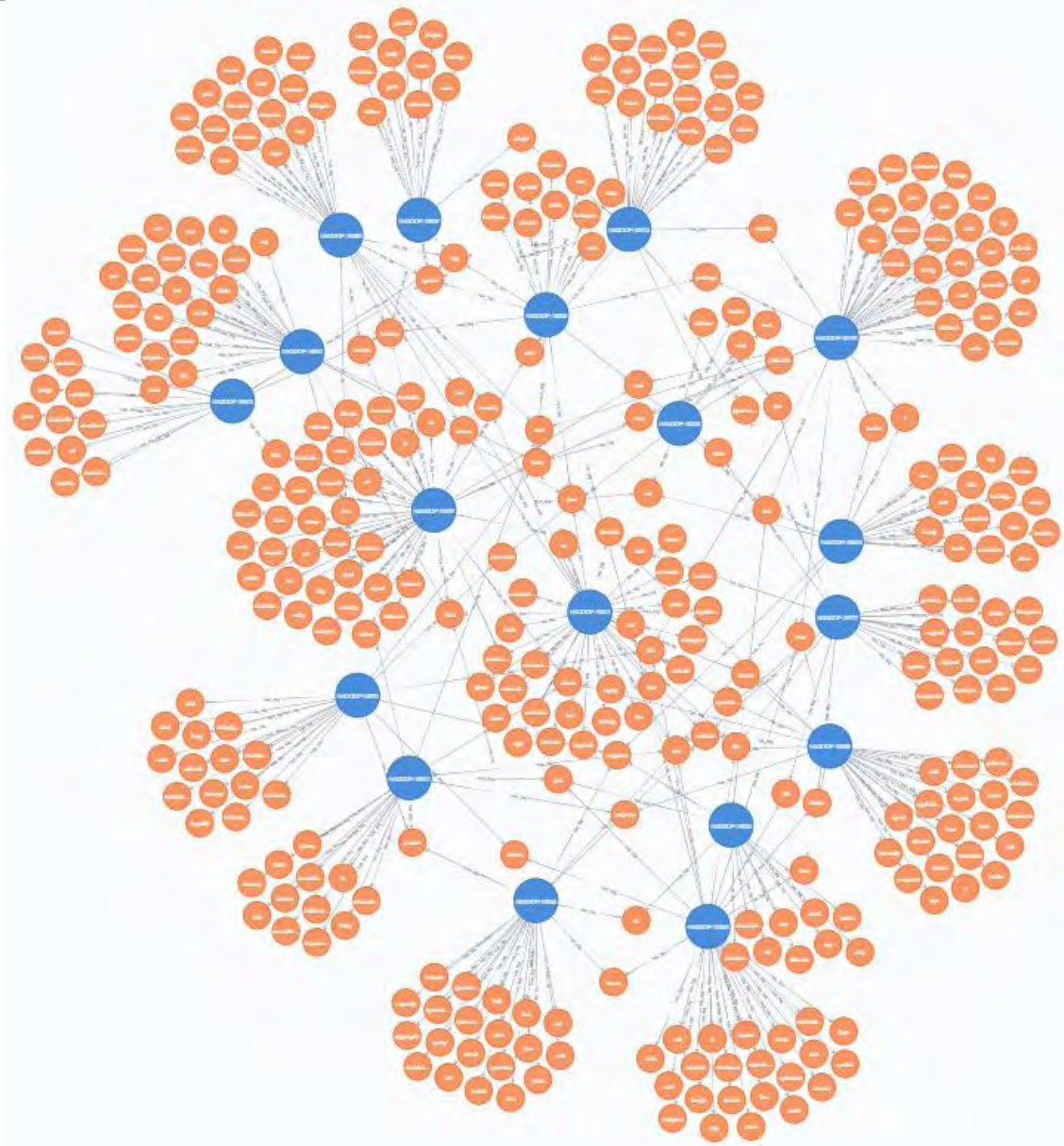
Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής:

```

MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data

```

```
CALL algo.similarity.pearson.stream(data, {topK:1,
similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to
WHERE from = 'HADOOP-15233' OR to = 'HADOOP-15233'
//Only show records that contain the issue's name
WITH collect(from) AS from_list, collect(to) AS to_list
MATCH (i1:Issue)-[r1:has_tag]->(t:Tag)
WHERE i1.key IN (from_list + to_list)
RETURN i1, r1, t;
```



Εικόνα 3.12: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι)

Q14. Ποιά issues είναι όμοια με ποιά άλλα issues;

Αναζητούμε τα issues που είναι παρόμοια μεταξύ τους σύμφωνα με το μέτρο ομοιότητας Pearson με ποσοστό ομοιότητας μεγαλύτερο του 70% και εμφανίζουμε τα ζεύγη όμοιων issues καθώς και το βαθμό ομοιότητας τους.

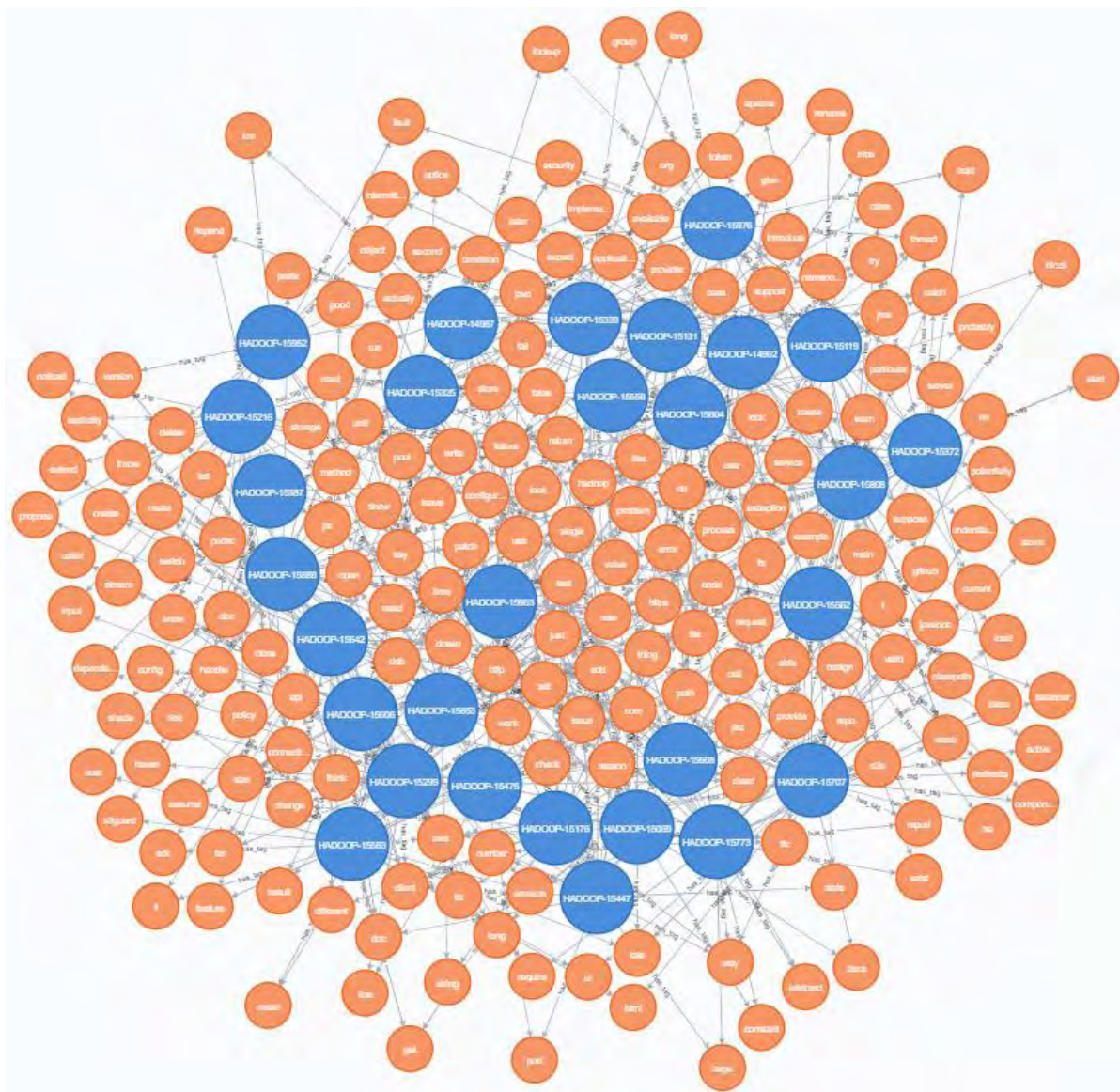
```
MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.pearson.stream(data, {topK:1,
similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
RETURN from, to, round(factor * value)/factor AS similarity;
```

from	to	similarity
"HADOOP-15992"	"HADOOP-15407"	0.87
"HADOOP-15759"	"HADOOP-15268"	1.0
"HADOOP-15991"	"HADOOP-15394"	0.94
"HADOOP-15990"	"HADOOP-15878"	1.0
"HADOOP-15982"	"HADOOP-15394"	0.92
"HADOOP-15409"	"HADOOP-15993"	1.0
"HADOOP-15989"	"HADOOP-15265"	0.85
"HADOOP-15988"	"HADOOP-15649"	1.0
"HADOOP-15987"	"HADOOP-15268"	1.0
"HADOOP-15986"	"HADOOP-15375"	0.99
"HADOOP-15985"	"HADOOP-15974"	1.0
"HADOOP-15966"	"HADOOP-15970"	1.0
"HADOOP-15993"	"HADOOP-15671"	1.0
"HADOOP-15971"	"HADOOP-15763"	1.0
"HADOOP-15981"	"HADOOP-15380"	1.0
"HADOOP-15977"	"HADOOP-15862"	1.0

Εικόνα 3.13: Πίνακας μερικών αποτελεσμάτων – παρόμοια issues και ο βαθμός ομοιότητας τους.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής:

```
MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.pearson.stream(data, {degreeCutoff: 32,
topK: 1, similarityCutoff: 0.7})
YIELD item1, item2, similarity
WITH algo.asNode(item1).key AS from,
      algo.asNode(item2).key AS to
WITH collect(from) AS from_list,
      collect(to) AS to_list
MATCH (i1:Issue)-[r1:has_tag]
      ->(t:Tag)<-[r2:has_tag]-(i2:Issue)
WHERE i1.key IN from_list AND i2.key IN to_list
RETURN i1, r1, t, r2, i2;
```



Εικόνα 3.14: Πίνακας μερικών αποτελεσμάτων – παρόμοια issues και ο βαθμός ομοιότητας τους.

3.3.4 Ευκλείδεια Απόσταση

Q15. Ποιά issues είναι όμοια με ένα συγκεκριμένο issue; (π.χ. HADOOP-15754)

Αναζητούμε τα issues που είναι παρόμοια με το HADOOP-15754 σύμφωνα με την ευκλείδεια απόσταση με μέτρο απόστασης μικρότερο του 2 και εμφανίζουμε τα ζεύγη όμοιων issues καθώς και το βαθμό ομοιότητάς τους.

```
MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.euclidean.stream(data,
{topK:1 , similarityCutoff: 2.0})
YIELD item1, item2, similarity AS distance
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
distance, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(distance) AS
value, 10^precision AS factor //Round by 2 decimal places
WHERE from = 'HADOOP-15233' OR to = 'HADOOP-15233'
//Only show records that contain the issue's name
RETURN from, to, round(factor * value)/factor AS distance;
```

from	to	distance
"HADOOP-15911"	"HADOOP-15233"	0.25
"HADOOP-15777"	"HADOOP-15233"	0.13
"HADOOP-15834"	"HADOOP-15233"	0.2
"HADOOP-15811"	"HADOOP-15233"	0.14
"HADOOP-15801"	"HADOOP-15233"	0.13
"HADOOP-15687"	"HADOOP-15233"	0.16
"HADOOP-15645"	"HADOOP-15233"	0.13
"HADOOP-15570"	"HADOOP-15233"	0.17
"HADOOP-15399"	"HADOOP-15233"	0.23
"HADOOP-15419"	"HADOOP-15233"	0.14

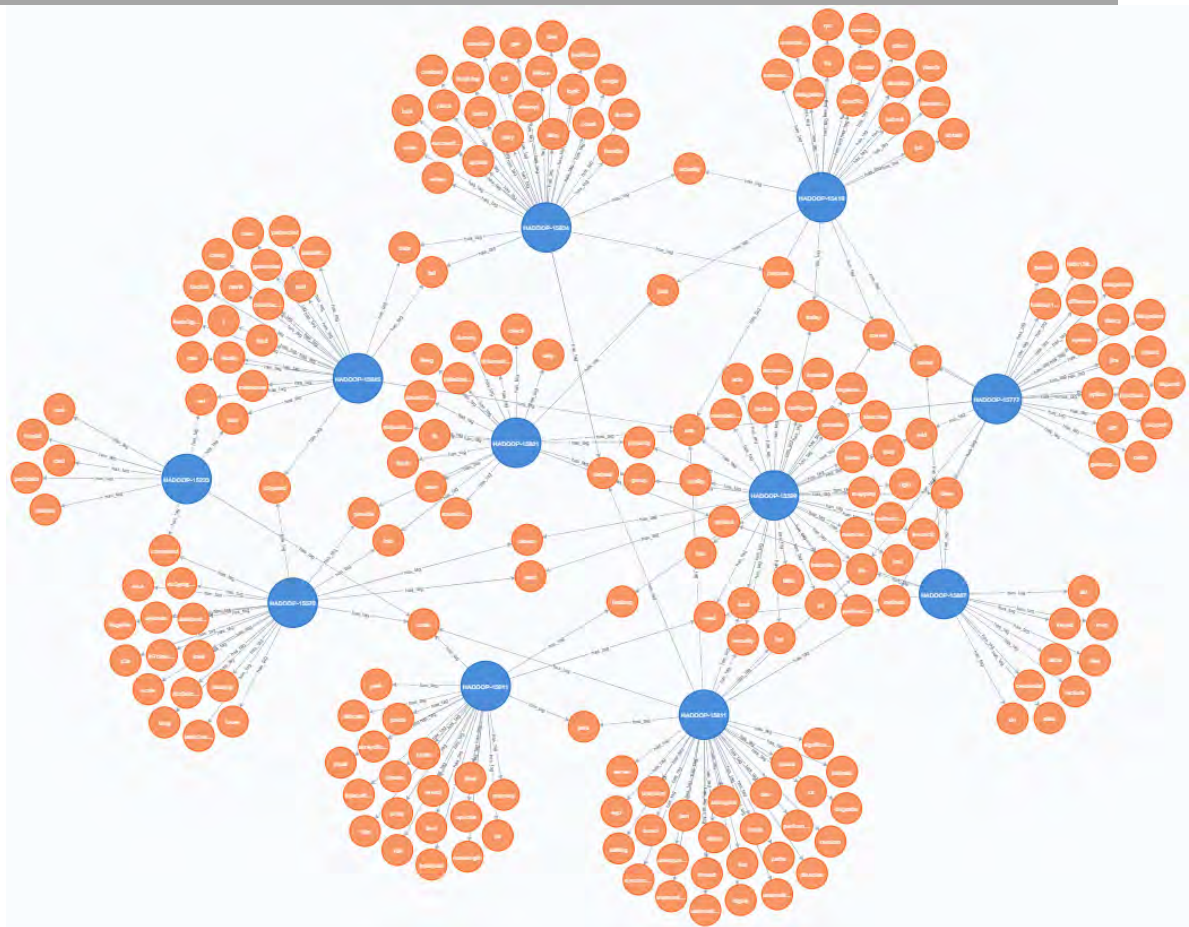
Εικόνα 3.15: Πίνακας αποτελεσμάτων – HADOOP-15754, παρόμοια issues και ο βαθμός ομοιότητάς τους.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής:


```

MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.euclidean.stream(data,
{topK:1 , similarityCutoff: 2.0})
YIELD item1, item2, similarity AS distance
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
distance, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(distance) AS
value, 10^precision AS factor //Round by 2 decimal places
WHERE from = 'HADOOP-15233' OR to = 'HADOOP-15233'
//Only show records that contain the issue's name
WITH collect(from) AS from_list,
collect(to) AS to_list
WITH collect(from) AS from_list, collect(to) AS to_list
MATCH (i1:Issue)-[r1:has_tag]->(t:Tag)
WHERE i1.key IN (from_list + to_list)
RETURN i1, r1, t;

```



Εικόνα 3.16: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι)

Q16. Ποιά issues είναι όμοια με ποιά άλλα issues;

Αναζητούμε τα issues που είναι παρόμοια μεταξύ τους σύμφωνα με την ευκλείδεια απόσταση με μέτρο απόστασης μικρότερο του 2 και εμφανίζουμε τα ζεύγη όμοιων issues καθώς και το βαθμό ομοιότητας τους.

```

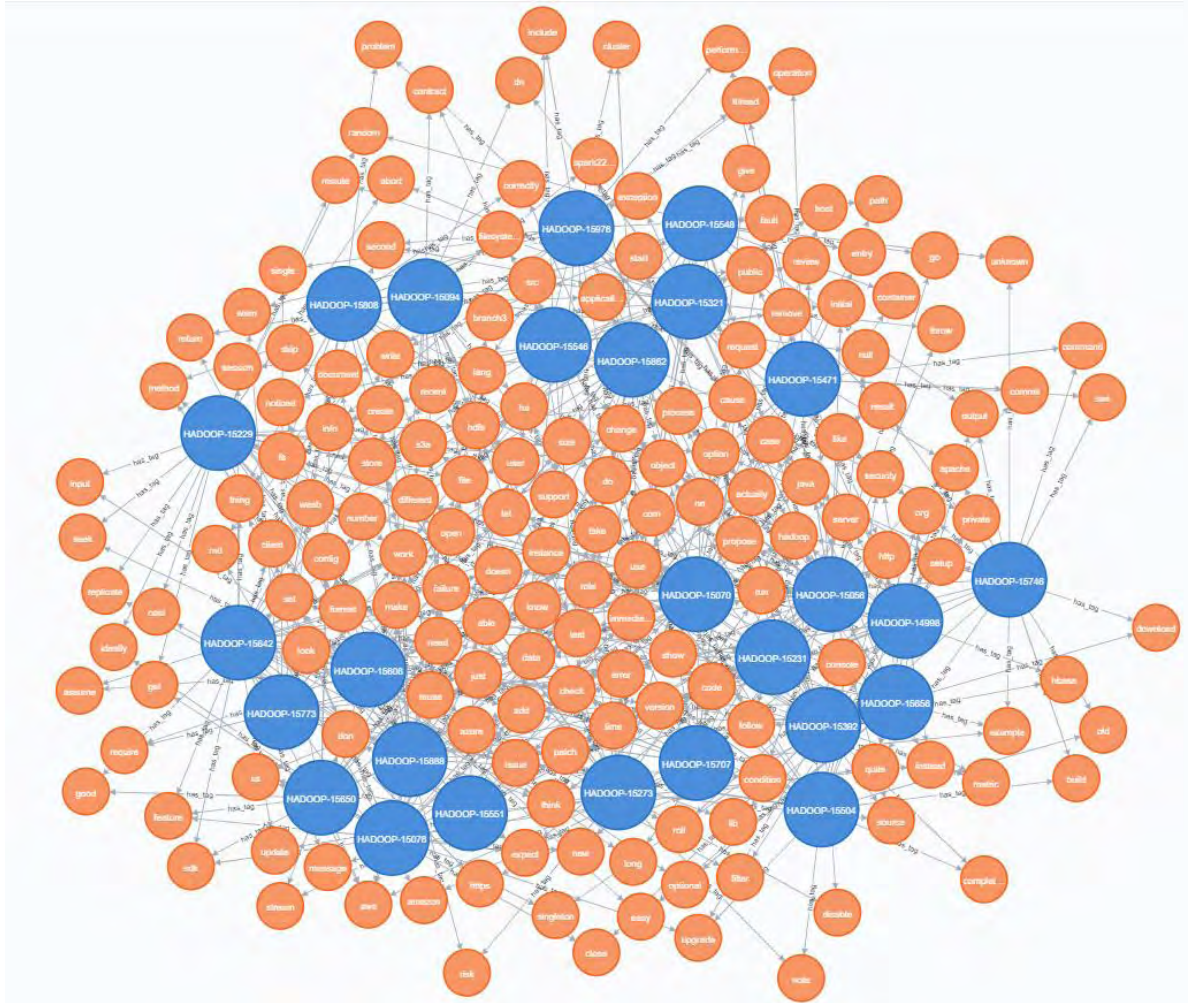
MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.euclidean.stream(data,
{topK:1 , similarityCutoff: 2.0})
YIELD item1, item2, similarity AS distance
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
distance, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(distance) AS
value, 10^precision AS factor //Round by 2 decimal places
RETURN from, to, round(factor * value)/factor AS distance;
    
```

from	to	distance
"HADOOP-15978"	"HADOOP-15979"	0.0
"HADOOP-15992"	"HADOOP-15675"	0.21
"HADOOP-15759"	"HADOOP-15124"	0.17
"HADOOP-15991"	"HADOOP-15780"	0.19
"HADOOP-15990"	"HADOOP-15899"	0.08
"HADOOP-15982"	"HADOOP-15899"	0.08
"HADOOP-15409"	"HADOOP-15096"	0.1
"HADOOP-15989"	"HADOOP-15052"	0.24
"HADOOP-15988"	"HADOOP-15649"	0.07
"HADOOP-15619"	"HADOOP-15984"	0.0
"HADOOP-15987"	"HADOOP-15668"	0.12
"HADOOP-15986"	"HADOOP-15052"	0.21
"HADOOP-15985"	"HADOOP-15899"	0.1
"HADOOP-15966"	"HADOOP-15899"	0.09
"HADOOP-15994"	"HADOOP-15945"	0.0
"HADOOP-15993"	"HADOOP-15363"	0.11

Εικόνα 3.17: Πίνακας μερικών αποτελεσμάτων – παρόμοια issues και ο βαθμός ομοιότητας τους.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής:

```
MATCH (i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(i), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.euclidean.stream(data,
    {degreeCutoff: 45, topK:1 , similarityCutoff: 2.0})
YIELD item1, item2, similarity AS distance
WITH algo.asNode(item1).key AS from, algo.asNode(item2).key AS
to, distance, 2 AS precision
WITH collect(from) AS from_list,
    collect(to) AS to_list
MATCH (i1:Issue)-[r1:has_tag]
    ->(t:Tag)<-[r2:has_tag]-(i2:Issue)
WHERE i1.key IN from_list AND i2.key IN to_list
RETURN i1, r1, t, r2, i2;
```



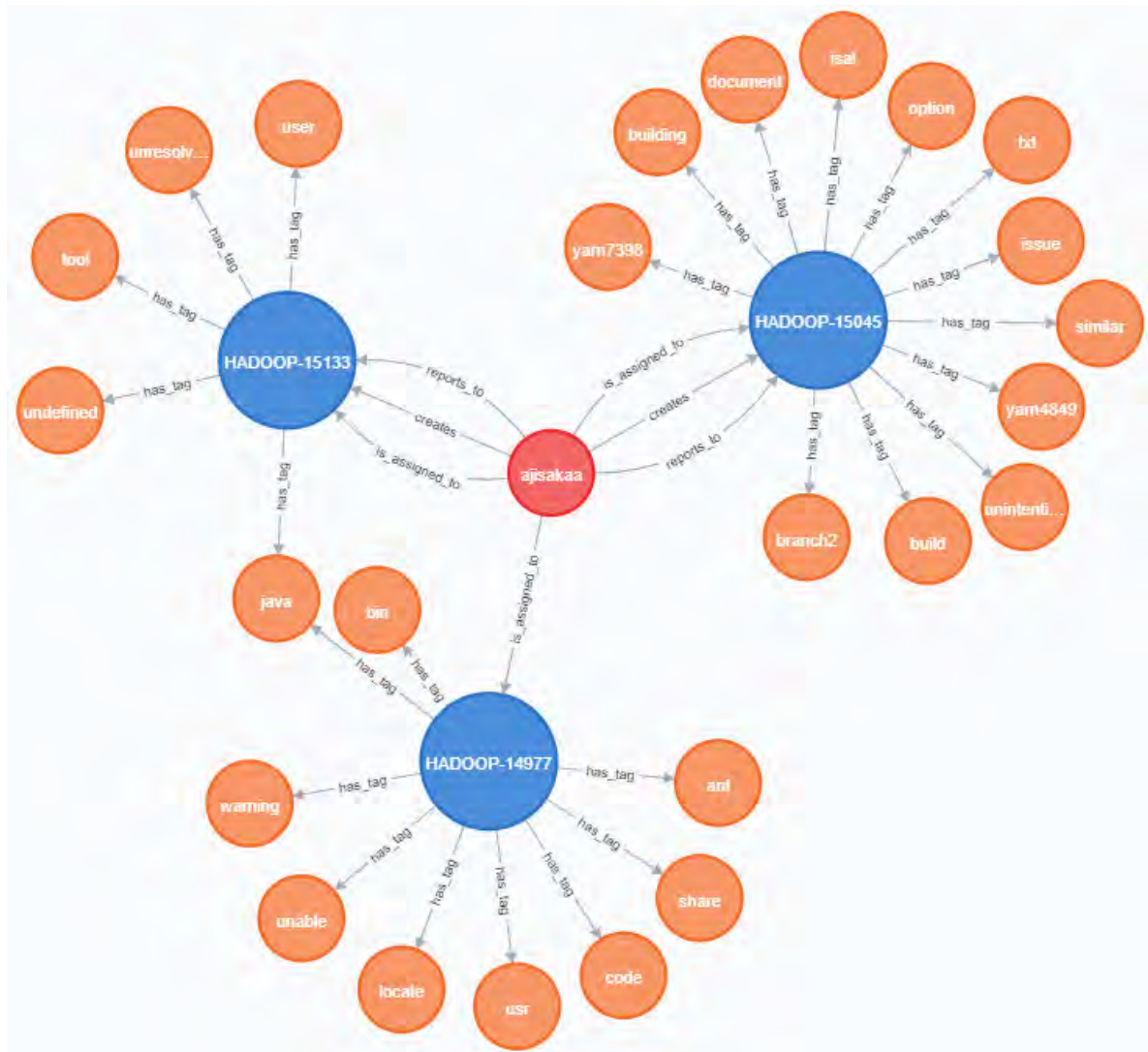
Εικόνα 3.18: Γράφος αποτελεσμάτων – παρόμοια issues (μπλε κόμβοι) και οι κοινές ετικέτες τους (πορτοκαλί κόμβοι)

Σε αυτή την ενότητα παρουσιάσαμε τέσσερα αντιπροσωπευτικά μέτρα ομοιότητας / απόστασης (Jaccard, Cosine, Pearson και ευκλείδεια απόσταση). Αναφέραμε όλα αυτά τα μέτρα για λόγους πληρότητας, όμως στη συνέχεια της εργασίας θα συνεχίσουμε χρησιμοποιώντας μόνο δύο εξ' αυτών τα οποία σύμφωνα με το [13] είναι τα καλύτερα μέτρα με χρήση πολλών διαφορετικών datasets. Τα μέτρα αυτά είναι το Jaccard και το Cosine.

3.4 Ομοιότητα μεταξύ προγραμματιστών

Πέρα από τα issues και τα παραχθείσα εξ' αυτών tags, έχουμε και τους προγραμματιστές που δουλεύουν σε αυτά. Σε μια προσπάθεια αντιστοίχισης των προγραμματιστών σε ομάδες βάσει των ικανοτήτων που πιστεύουμε ότι αποκτούν από τα issues που δουλεύουν γράφουμε τα ακόλουθα ερωτήματα στην cypher.

Πριν από αυτό το βήμα θα δούμε ένα υπο-γράφο που εκφράζει την οργάνωση της βάσης.



Εικόνα 3.19: Μοντέλο οργάνωσης της βάσης, issues που συνδέονται μεταξύ τους (δεν οπτικοποιείται στο συγκεκριμένο παράδειγμα), issues που συνδέονται με ετικέτες (κοινές ή μη) και προγραμματιστές που συνδέονται με issues άμεσα και έμμεσα (μέσω μονοπατιού) με ετικέτες.

Αρχικά, πριν ορίσουμε τα μέτρα ομοιότητας για να βρούμε προγραμματιστές με παρόμοιο τρόπο λειτουργίας, θα εκφράσουμε ερωτήματα σε Cypher τα οποία δείχνουν την ικανότητα για ένα ή πολλά skills όλων των προγραμματιστών.

(Επισημαίνουμε ότι η εκτίμηση ικανότητας γίνεται μέσα από την συχνότητα εμφάνισης ενός tag).

Q17. Ποια είναι τα κορυφαία skills ανά προγραμματιστή;

Αναζητούμε τις ετικέτες των issues που δουλεύει ένας προγραμματιστής, μετράμε την εμφάνιση κάθε ετικέτας για κάθε προγραμματιστή και τις επιστρέφουμε μαζί με το όνομα του προγραμματιστή. (πρώτος τρόπος απεικόνισης)

```

MATCH (p:Person)-[]->(i:Issue)-[:has_tag]->(t:Tag)
WITH p.key AS programmer, t.key AS tag, count(*) AS tagCount
WHERE tagCount > 8 //Remove low score results
RETURN programmer, tag, tagCount
ORDER BY programmer, tagCount DESC
    
```

programmer	tag	tagCount
"ajayydv"	"code"	17
"ajayydv"	"configuration"	10
"ajayydv"	"secure"	9
"ajayydv"	"jira"	9
"ajisakaa"	"org"	45
"ajisakaa"	"apache"	43
"ajisakaa"	"java"	41
"ajisakaa"	"hadoop"	36
"ajisakaa"	"noformat"	28
"ajisakaa"	"issue"	17
"ajisakaa"	"update"	17
"ajisakaa"	"fail"	16
"ajisakaa"	"http"	16
"ajisakaa"	"version"	16
"ajisakaa"	"let"	15

Εικόνα 3.20: Πίνακας μερικών αποτελεσμάτων – όνομα προγραμματιστή, ετικέτες που «κληρονομεί» από τα δεδομένα tasks και η συχνότητα κληρονομής τους.

Q18. Ποια είναι τα κορυφαία skills ανά προγραμματιστή;

Αναζητούμε τις ετικέτες των issue που δουλεύει ένας προγραμματιστής, μετράμε την εμφάνιση κάθε ετικέτας για κάθε προγραμματιστή και τις επιστρέφουμε μαζί με το όνομα του προγραμματιστή. (δεύτερος τρόπος απεικόνισης)

```
MATCH (p:Person)-[]->(i:Issue)-[:has_tag]->(t:Tag)
WITH p.key AS programmer, t.key AS tag, count(*) AS tagCount
WHERE tagCount > 8 //Remove low score results
RETURN programmer, [tag, tagCount] AS tagList
ORDER BY programmer, tagCount DESC
```

programmer	tagList
"ajayydv"	["code", 17]
"ajayydv"	["configuration", 10]
"ajayydv"	["secure", 9]
"ajayydv"	["jira", 9]
"ajisakaa"	["org", 45]
"ajisakaa"	["apache", 43]
"ajisakaa"	["java", 41]
"ajisakaa"	["hadoop", 36]
"ajisakaa"	["noformat", 28]
"ajisakaa"	["issue", 17]
"ajisakaa"	["update", 17]
"ajisakaa"	["fail", 16]
"ajisakaa"	["http", 16]
"ajisakaa"	["version", 16]
"ajisakaa"	["let", 15]
"ajisakaa"	["file", 14]

Εικόνα 3.21: Πίνακας μερικών αποτελεσμάτων – όνομα προγραμματιστή, ετικέτες που «κληρονομεί» από τα δεδουλευμένα tasks και η συχνότητα κληρονομησής τους.

Q19. Κατάταξη προγραμματιστών σε συγκεκριμένα skills;

Αναζητούμε συγκεκριμένες ετικέτες των *issue* που δουλεύει ένας προγραμματιστής (π.χ. *java*, *Hadoop*), μετράμε την εμφάνιση κάθε ετικέτας για κάθε προγραμματιστή και τις επιστρέφουμε μαζί με το όνομα του προγραμματιστή.

```
MATCH (p:Person)-[]->(i:Issue)-[:has_tag]->(t:Tag)
WITH p.key AS programmer, t.key AS tag, count(*) AS tagCount
WHERE tagCount > 8 AND tag IN ['java', 'hadoop']
RETURN programmer, tag, tagCount
ORDER BY tagCount DESC
```

programmer	tag	tagCount
"stevel@apache.org"	"hadoop"	98
"ajisakaa"	"java"	41
"ajisakaa"	"hadoop"	36
"gabor.bota"	"java"	34
"jojochuang"	"hadoop"	33
"tasanuma0829"	"java"	26
"elek"	"hadoop"	26
"gabor.bota"	"hadoop"	25
"jojochuang"	"java"	25
"xiaochen"	"hadoop"	24
"tasanuma0829"	"hadoop"	23
"xiaochen"	"java"	23
"stevel@apache.org"	"java"	22
"mackrorysd"	"hadoop"	21
"wujinhu"	"java"	18
"daryn"	"java"	16

Εικόνα 3.22: Πίνακας μερικών αποτελεσμάτων – όνομα προγραμματιστή, ετικέτα *java* που «κληρονομεί» από τα δεδομένα *tasks* και η συχνότητα κληρονομησής της.

Στη συνέχεια, ενσωματώνουμε το κώδικα που συνδέει τους προγραμματιστές με ετικέτες στους κώδικες των ενοτήτων 3.3.1, 3.3.2, 3.3.3, 3.3.4 για να βρούμε τους όμοιους προγραμματιστές.

3.4.1 Ομοιότητα Jaccard

Q20. Ποιοί προγραμματιστές είναι όμοιοι με έναν συγκεκριμένο προγραμματιστή; (π.χ. 'tom10271')

Αναζητούμε τους προγραμματιστές που είναι παρόμοιοι με τον tom10271 σύμφωνα με το μέτρο ομοιότητας Jaccard με ποσοστό ομοιότητας μεγαλύτερο του 45% και εμφανίζουμε τα ζεύγη όμοιων προγραμματιστών και το βαθμό ομοιότητας τους.

```
MATCH (p:Person)-[]->(i:Issue)-[:has_tag]->(t:Tag)
WITH {item:id(p), categories: collect(id(t))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.jaccard.stream(data,
    {topK:1, similarityCutoff: 0.45})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
WHERE from.key = 'tom10271' OR to.key = 'tom10271' //Only show
records that contain the programmer's name
RETURN from, to, round(factor * value)/factor AS similarity;
```

from	to	similarity
"ferhui"	"tom10271"	0.5
"tom10271"	"echo93"	0.54

Εικόνα 3.23: Πίνακας αποτελεσμάτων – ο tom10271 και οι παρόμοιοι σε αυτόν προγραμματιστές

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής (η οπτική αναπαράσταση δίνεται στην Εικόνα 3.24):

```

MATCH (p:Person)-[]->(i:Issue)-[:has_tag]->(t:Tag)
WITH {item:id(p), categories: collect(id(t))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.jaccard.stream(data,
    {topK:1, similarityCutoff: 0.45})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to
WHERE from.key = 'tom10271' OR to.key = 'tom10271'
//Only show records that contain the programmer's name
WITH collect(from) AS from_list, collect(to) AS to_list
MATCH (p1:Person)-[]->(i1:Issue)-[r1:has_tag]->(t:Tag)<-
[r2:has_tag]->(i2:Issue)<-[]-(p2:Person)
WHERE (p1.key IN from_list) AND (p2.key IN to_list)
RETURN p1, p2, r1, r2, t, i1, i2;
    
```



Εικόνα 3.24: Γράφος αποτελεσμάτων – κόμβοι ετικέτες (πορτοκαλί), issues (μπλε) και προγραμματιστές (κόκκινο).

Q21. Ποιοί προγραμματιστές είναι όμοιοι μεταξύ τους;

Αναζητούμε τους προγραμματιστές που είναι παρόμοιοι μεταξύ τους σύμφωνα με το μέτρο ομοιότητας Jaccard με ποσοστό ομοιότητας μεγαλύτερο του 45% και εμφανίζουμε τα ζεύγη όμοιων προγραμματιστών και το βαθμό ομοιότητας τους.

```
MATCH (p:Person)-[]->(i:Issue)-[:has_tag]->(t:Tag)
WITH {item:id(p), categories: collect(id(t))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.jaccard.stream(data,
    {topK:1, similarityCutoff: 0.45})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
RETURN from, to, round(factor * value)/factor AS similarity;
```

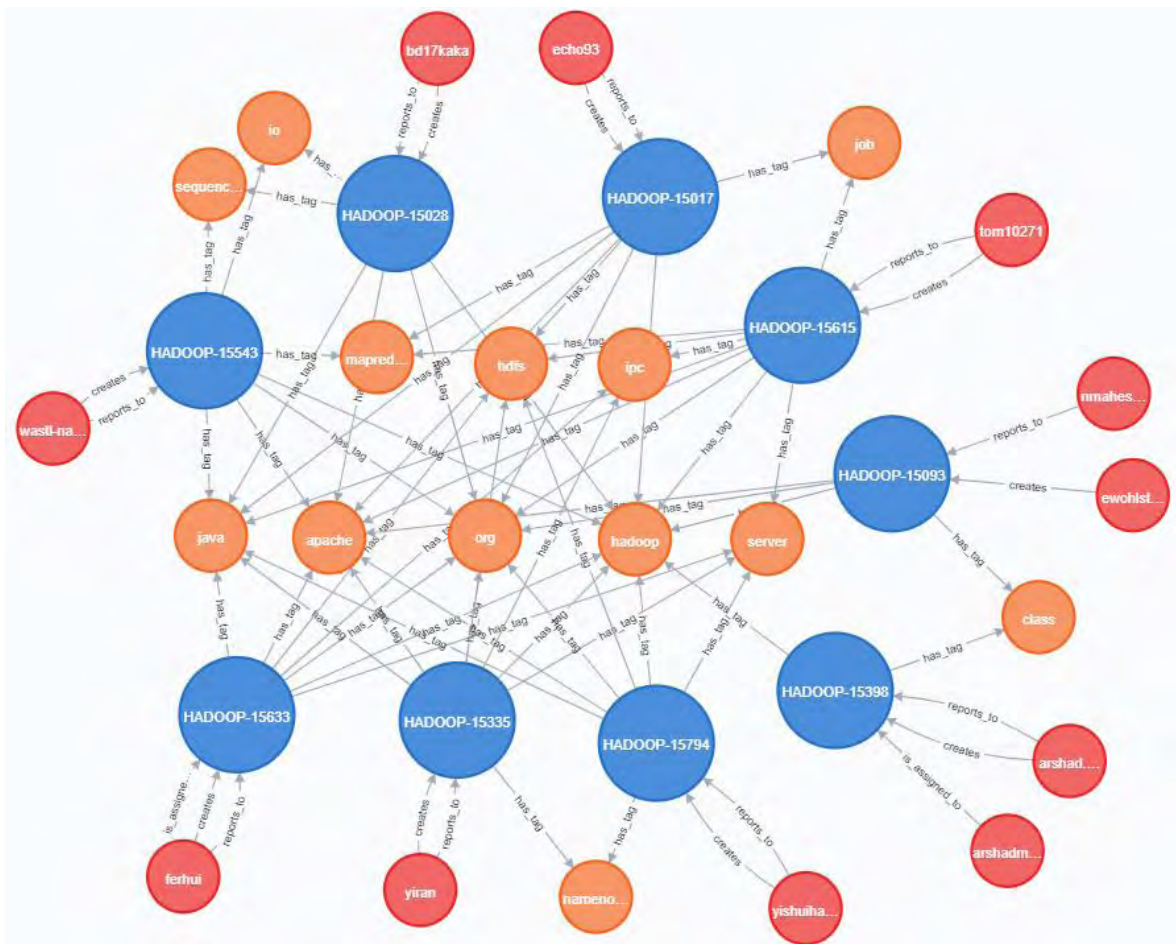
from	to	similarity
"yishuihanxiao"	"yiran"	0.47
"ferhui"	"tom10271"	0.5
"tom10271"	"echo93"	0.54
"wastl-nagel"	"bd17kaka"	0.55
"arshadmohammad"	"arshad.mohammad"	0.5
"ewohlstadter"	"nmaheshwari"	1.0

Εικόνα 3.25: Πίνακας αποτελεσμάτων – ζεύγη όμοιων προγραμματιστών και το ποσοστό ομοιότητας.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής(η οπτική αναπαράσταση δίνεται στην Εικόνα 3.26):

```

MATCH (p:Person)-[]->(i:Issue)-[:has_tag]->(t:Tag)
WITH {item:id(p), categories: collect(id(t))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.jaccard.stream(data,
    {topK:1, similarityCutoff: 0.45})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to
WITH collect(from) AS from_list, collect(to) AS to_list
MATCH (p1:Person)-[]->(i1:Issue)-[r1:has_tag]->(t:Tag)<-
[r2:has_tag]->(i2:Issue)<-[]-(p2:Person)
WHERE (p1.key IN from_list) AND (p2.key IN to_list)
RETURN p1, p2, r1, r2, t, i1, i2;
    
```



Εικόνα 3.26: Γράφος αποτελεσμάτων – κόμβους ετικέτες (πορτοκαλί), issues (μπλε) και προγραμματιστές (κόκκινο).

3.4.2 Ομοιότητα Cosine

Q.22 Ποιοί προγραμματιστές είναι όμοιοι με έναν συγκεκριμένο προγραμματιστή; (π.χ για τον 'shv')

Αναζητούμε τους προγραμματιστές που είναι παρόμοιοι με τον 'shv' σύμφωνα με το μέτρο ομοιότητας Cosine με ποσοστό ομοιότητας μεγαλύτερο του 90% και εμφανίζουμε τα ζεύγη όμοιων προγραμματιστών και το βαθμό ομοιότητας τους.

```
MATCH (p:Person)-[]->(i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(p), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.cosine.stream(data,
{topK: 1, similarityCutoff: 0.9})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
WHERE from = 'shv' OR to = 'shv'
//Only show records that contain the programmer's name
RETURN from, to, round(factor * value)/factor AS similarity;
```

from	to	similarity
"lukmajercak"	"shv"	1.0
"changzc_330"	"shv"	0.95
"eyang"	"shv"	0.96
"yuzhihong@gmail.com"	"shv"	1.0
"leosun"	"shv"	0.96
"jianliang.wu"	"shv"	1.0
"linyiqun"	"shv"	1.0
"pnowojski"	"shv"	0.95
"bhujay"	"shv"	1.0

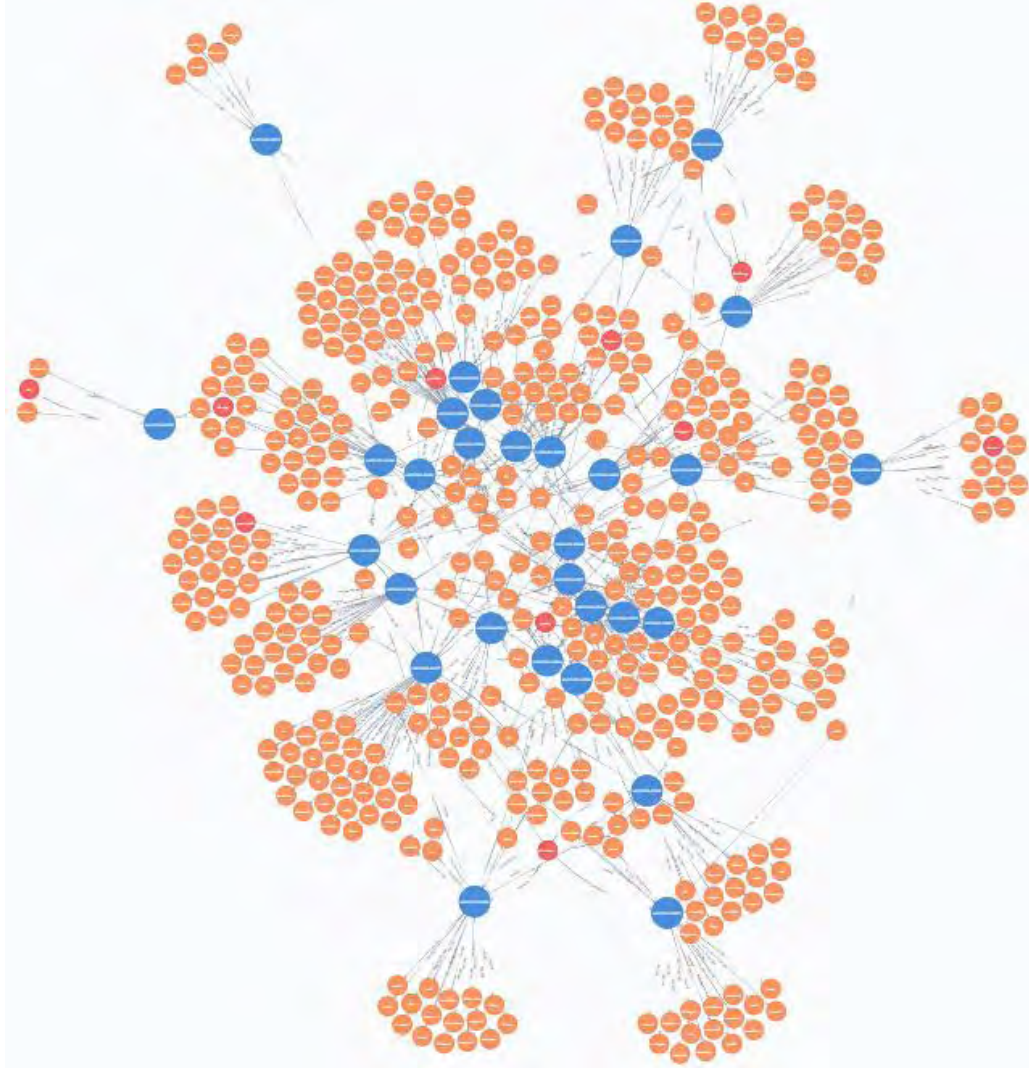
Εικόνα 3.27: Πίνακας αποτελεσμάτων – ο shv και οι παρόμοιοι σε αυτόν προγραμματιστές.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής (η οπτική αναπαράσταση δίνεται στην Εικόνα 3.28):

```

MATCH (p:Person)-[]->(i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(p), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.cosine.stream(data,
{topK: 1, similarityCutoff: 0.9})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to
WHERE from.key = 'shv' OR to.key = 'shv'
//Only show records that contain the programmer's name
WITH collect(from) AS from_list, collect(to) AS to_list
MATCH (p1:Person)-[]->(i1:Issue)-[r1:has_tag]->(t:Tag)
WHERE p1.key IN (from_list + to_list)
RETURN p1, r1, t, i1;

```



Εικόνα 3.28: Γράφος αποτελεσμάτων – κόμβοι ετικέτες (πορτοκαλί), issues (μπλε) και προγραμματιστές (κόκκινο).

Q.23 Ποιοί προγραμματιστές είναι όμοιοι μεταξύ τους;

Αναζητούμε τους προγραμματιστές που είναι παρόμοιοι μεταξύ τους σύμφωνα με το μέτρο ομοιότητας Cosine με ποσοστό ομοιότητας μεγαλύτερο του 90% και εμφανίζουμε τα ζεύγη όμοιων προγραμματιστών και το βαθμό ομοιότητας τους.

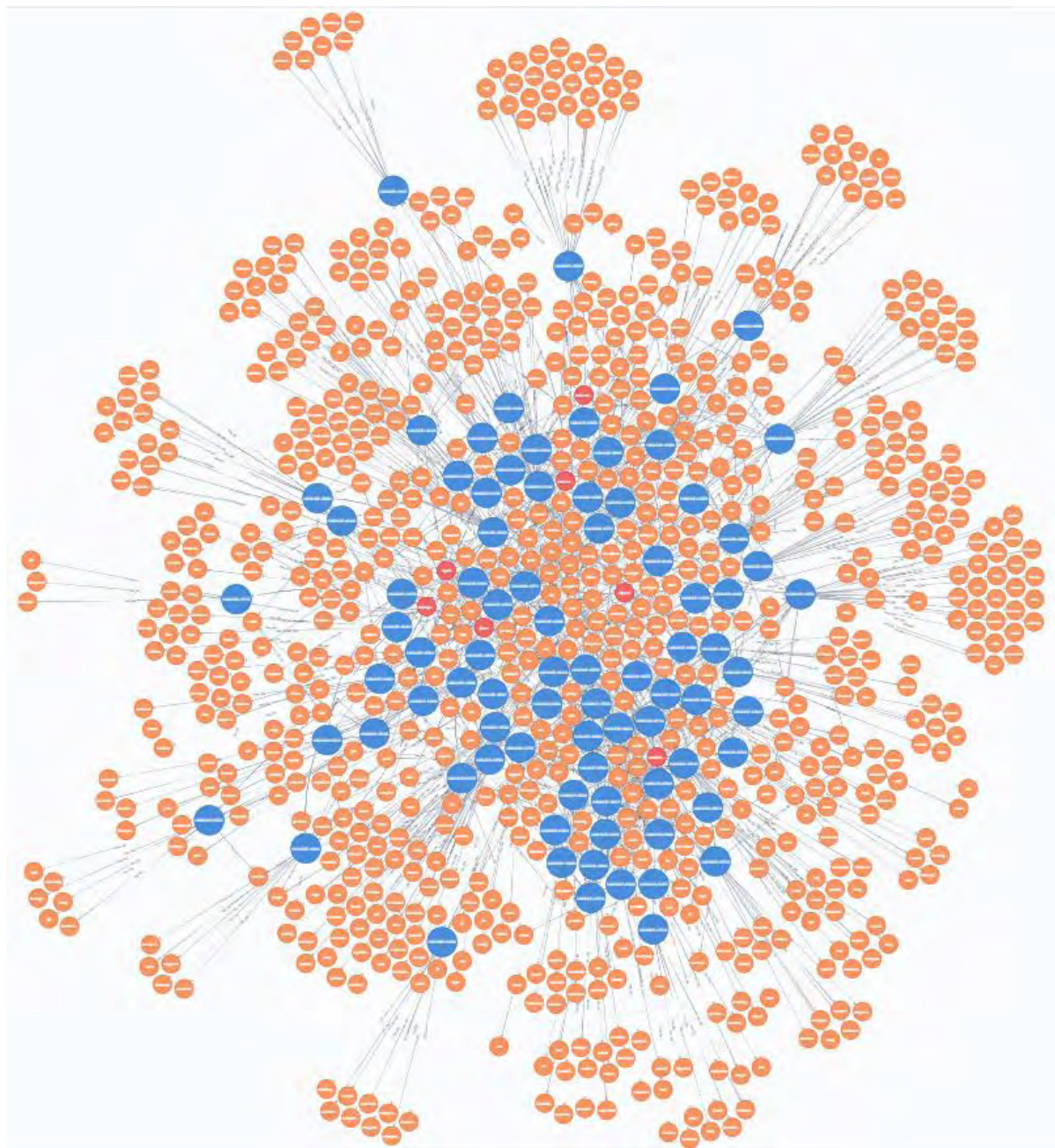
```
MATCH (p:Person)-[]->(i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(p), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.cosine.stream(data,
{topK: 1, similarityCutoff: 0.9})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to,
similarity, 2 AS precision
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to, toFloat(similarity) AS
value, 10^precision AS factor //Round by 2 decimal places
RETURN from, to, round(factor * value)/factor AS similarity;
```

from	to	similarity
"sanxiago"	"kowon2008"	0.97
"echo93"	"andrioni"	0.93
"nchammas"	"jnp"	0.95
"ywskycn"	"yzhangal"	0.96
"grepas"	"kowon2008"	1.0
"cmukka20"	"kowon2008"	1.0
"sunilkc"	"rohithsharma"	0.95
"qingwu.fu"	"jim_brennan"	0.93
"zsombor"	"jonboone"	1.0
"zswing"	"addisonj@gmail.com"	0.96

Εικόνα 3.29: Πίνακας αποτελεσμάτων – ζεύγη όμοιων προγραμματιστών και το ποσοστό ομοιότητας.

Αν θέλουμε να οπτικοποιήσουμε τα αποτελέσματα του παραπάνω ερωτήματος σε γράφο το μετατρέπουμε ως εξής(η οπτική αναπαράσταση δίνεται στην Εικόνα 3.30):

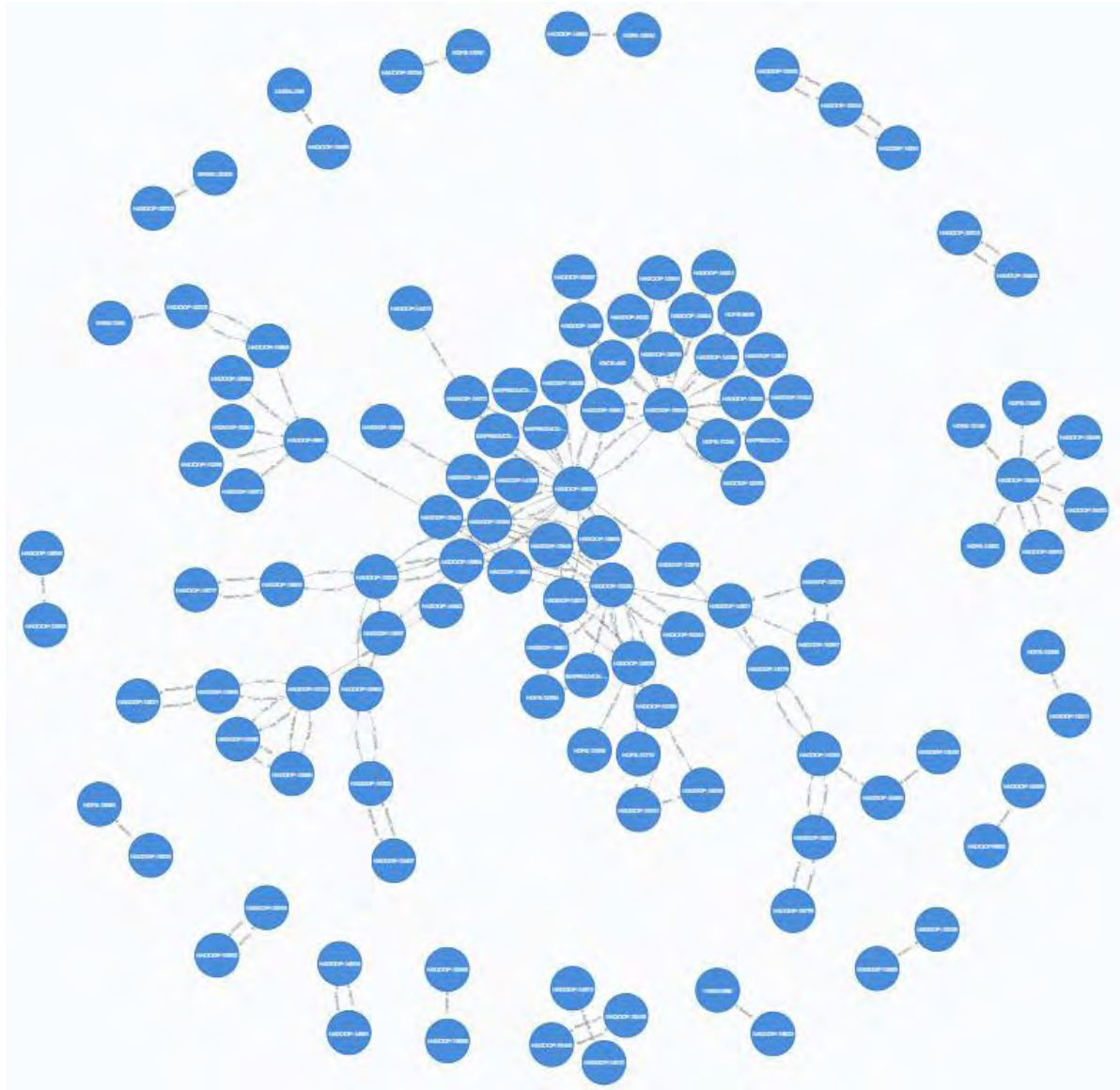
```
MATCH (p:Person)-[]->(i:Issue)-[r:has_tag]->(t:Tag)
WITH {item:id(p), weights: collect(coalesce(r.weight,
algo.NaN()))} AS Data
WITH collect(Data) AS data
CALL algo.similarity.cosine.stream(data,
    {topK: 1, degreeCutoff: 200, similarityCutoff: 0.9})
YIELD item1, item2, similarity
WITH algo.asNode(item1) AS from, algo.asNode(item2) AS to
WHERE id(from) < id(to) //Remove to -> from duplicate record
WITH from.key AS from, to.key AS to
WITH collect(from) AS from_list, collect(to) AS to_list
MATCH (p1:Person)-[]->(i1:Issue)-[r1:has_tag]->(t:Tag)
WHERE p1.key IN (from_list + to_list)
RETURN p1, r1, t, i1;
```

Εικόνα 3.30: Γράφος μερικών αποτελεσμάτων – κόμβοι ετικέτες (πορτοκαλί), issues (μπλε) και προγραμματιστές (κόκκινο).

3.5 Γράφος Εξάρτησης – PageRank

Κάποια από τα issues που έχουμε φορτώσει στην βάση έχουν, κάποιες εξαρτήσεις μεταξύ τους όπως φαίνεται στην Εικόνα 3.31:



Εικόνα 3.31: Γράφος εξάρτησης (υπο-γράφος του συνολικού γραφήματος) – κόμβοι issues (μπλε) με ακμές τις σχέσεις εξάρτησης (depends_upon).

Με χρήση του αλγόριθμου PageRank, μπορούμε να ανιχνεύσουμε προγραμματιστικά τον πιο κεντρικό κόμβο του γράφου εξαρτήσεων, ο οποίος θα έχει το μεγαλύτερο score σύμφωνα με τον αλγόριθμο. Ο εν λόγω κόμβος δημιουργεί ένα κρίσιμο σημείο εξάρτησης στο γράφημα και χρίζει μεγαλύτερης προτεραιότητας σαν issue.

Q24: Ποια είναι τα πιο σημαντικά issues σύμφωνα με τον PageRank;

Εκτελώντας το ακόλουθο ερώτημα σε Cypher, παίρνουμε το όνομα του Issue, μαζί με το σκορ του, όπως αυτό υπολογίζεται από τον PageRank, (εμφανίζουμε issues με τιμή μεγαλύτερη του 0.9).

Επειδή το γράφημα έχει κόμβους που συνδέονται διμερώς μεταξύ τους, χρησιμοποιούμε αμφίδρομη κατεύθυνση αντιμετωπίζοντας το γράφημα σαν μη-κατευθυνόμενο.

```
CALL algo.pageRank.stream('Issue', 'depends_upon',
{direction: 'BOTH', iterations: 100, dampingFactor: 0.85})
YIELD nodeId, score
WITH algo.asNode(nodeId).key AS Issue,
toFloat(score) AS value, 10^2 AS factor
WHERE score > 0.9
RETURN Issue, round(factor * value)/factor AS PageRank_Score
ORDER BY PageRank_Score DESC
```

Issue	PageRank_Score
"HADOOP-15788"	7.8
"HADOOP-15620"	4.02
"HADOOP-15509"	3.29
"HADOOP-9991"	3.29
"HADOOP-15220"	2.37
"HADOOP-15273"	1.46
"HADOOP-15304"	1.46
"HADOOP-14556"	1.46
"HADOOP-15576"	1.46
"HADOOP-15281"	1.38
"HADOOP-15364"	1.37
"HADOOP-15229"	1.32
"HADOOP-15183"	1.3
"HADOOP-14936"	1.3
"HADOOP-15086"	1.0
"HADOOP-14816"	1.0

Εικόνα 3.32: Πίνακας μερικών αποτελεσμάτων – Issues με τα PageRank σκορ.

4

Αξιοποίηση Ψηφιακών Προσωπικών Βοηθών σε ζητήματα Project Management

4.1 Εισαγωγή

Ένας ψηφιακός βοηθός (virtual assistant) ορίζεται ως ένα λογισμικό το οποίο εκπληρώνει εντολές που τίθενται από έναν άνθρωπο μέσω γραπτών ή φωνητικών εντολών. Πολλές φορές αναφέρονται στη βιβλιογραφία και με τον όρο chatbot ή bot, δηλαδή ενός λογισμικού το οποίο διεξάγει συζήτηση (chat) για την εκπλήρωση κάποιου σκοπού με έναν άνθρωπο μέσω γραπτών ή φωνητικών εντολών [16][17].

Το 1950, ο Alan Turing στην εργασία του «Computing Machinery and Intelligence» παρουσίασε το λεγόμενο Turing Test, μια σειρά κριτηρίων τα οποία θα μπορούσαν να πιστοποιήσουν αν ένα πρόγραμμα υπολογιστών διαθέτει ένα είδος νοημοσύνης που προσομοιάζει στην ανθρώπινη. Τα κριτήρια αυτά αφορούν στην ικανότητα του προγράμματος να ξεγελάσει έναν άνθρωπο κριτή –όσον αφορά το περιεχόμενο της μεταξύ τους συζήτησης– ώστε να πιστέψει ότι στην άλλη μεριά της οθόνης του υπολογιστή βρίσκεται ένας άλλος άνθρωπος που πληκτρολογεί τις απαντήσεις έναντι ενός ‘έξυπνου’ προγράμματος [18]. Το διάστημα 1960–1990, αναπτύχθηκαν διάφορα bots, τα οποία είχαν φτιαχτεί με διάφορες μεθόδους. Ιστορικά, ένα από τα πρώτα bots ήταν το ELIZA (1964 – 1966), το οποίο χρησιμοποιούσε NLP (Natural Language Processing για να απαντήσει στα ερωτήματα ταιριάζοντας λεκτικά μοτίβα και μην λαμβάνοντας υπόψη τα συμφραζόμενα μεταξύ διαφορετικών προτάσεων [19].

Ακολούθησε το PARRY (1972) που ήταν υλοποιημένο με παρόμοιες μεθόδους, κατόπιν το Mark V. Shaney (1984) που χρησιμοποιούσε αλυσίδες Markov για να παράγει κείμενο για online posts βάσει υπάρχοντος κειμένου στο Usenet και το Racter(1984), που χρησιμοποίησε τεχνητή νοημοσύνη για να παράγει μορφές κειμένου βάσει υπάρχοντος κειμένου που χρησιμοποιήθηκε σαν είσοδος στο πρόγραμμα αυτό. Τα bots αυτά ανήκουν στην κατηγορία των Classic chatbots και έχουν περιορισμένες δυνατότητες σε σχέση με μελλοντικά chatbots [20].

Το διάστημα 1990–2010, αναπτύχθηκαν διάφορα bots, πιο εξελιγμένα από εκείνα της περιόδου των Classic Chatbots, όπως το A.L.I.C.E. (1995–1998) [21], ένα πρόγραμμα φτιαγμένο με NLP μεθόδους και ευρετικές τεχνικές μοτίβων πάνω στην δοθείσα ανθρώπινη πληροφορία, εμπνευσμένο από το ELIZA. Ένα άλλο αντιπροσωπευτικό bot της περιόδου ήταν το Watson (2006) της IBM [22],[23],[24] το οποίο πρέπει να σημειώσουμε ότι δεν είναι απλά ένα πρόγραμμα υπολογιστών όπως αυτά που αναφέρθηκαν παραπάνω αλλά αποτελείται και από το δικό του υπολογιστικό σύστημα όσον αφορά το υλικό. Το Watson περιγράφεται ως ένα υπολογιστικό σύστημα που απαντά σε ερωτήσεις δομημένες σε φυσική γλώσσα. Αρχικά είχε αναπτυχθεί για να απαντά σε ερωτήσεις του γνωστού αμερικάνικου τηλεπαιχνιδιού Jeopardy! Και το 2011 κατάφερε να νικήσει τους πρωταθλητές του εν λόγω τηλεπαιχνιδιού. Είναι αναπτυγμένο με μεθόδους προχωρημένου NLP, ανάκτησης πληροφορίας, αναπαράστασης γνώσης και εκμάθησης μηχανής (Machine Learning). Ακόμη υπάρχει το Cleverbot (1997 – 2011) το οποίο χρησιμοποιεί έναν αλγόριθμο τεχνητής νοημοσύνης για να μαθαίνει βάσει του κειμένου που εισάγει ο χρήστης όχι μόνο εκείνη τη στιγμή αλλά και από εκατομμύρια κείμενα χρηστών που εισήχθησαν στο bot συνολικά στα χρόνια της λειτουργίας του [25].

Από το 2011 μέχρι σήμερα, δημιουργήθηκαν διάφορα chatbots όπως το Siri (2011) που είναι ένας ψηφιακός βοηθός για τα συστήματα της Apple. Το Siri χρησιμοποιεί ερωτήματα φωνής σε συνδυασμό με επεξεργασία φυσικής γλώσσας (φωνητικά) του χρήστη για να απαντήσει σε ερωτήσεις, να κάνει προτάσεις, ή να πραγματοποιήσει ενέργειες εκ μέρους του χρήστη (π.χ. να καλέσει κάποιον / να ρυθμίσει ραντεβού στην ατζέντα του, κ.τ.λ.).

Το Cortana (2014) που αποτελεί όπως το Siri έναν ψηφιακό βοηθό για τις πλατφόρμες της Microsoft, αλλά και για τις πλατφόρμες των smartphones όπως iOS & Android. Οι δυνατότητες του είναι παρόμοιες με του Siri. Το Alexa (2014) της Amazon επίσης είναι παρόμοιο με τα παραπάνω. Το Google Assistant (2016) είναι ένας ψηφιακός βοηθός της Google ο οποίος χρησιμοποιεί μεθόδους τεχνητής νοημοσύνης και είναι ικανός να εμπλακεί σε διμερείς συζητήσεις σε αντίθεση με άλλους ψηφιακούς βοηθούς που απαντούν ένα αίτημα του χρήστη κάθε φορά. Αυτή η γενιά chatbots θεωρείται το state-of-the-art [26],[27] με τα εξής κοινά κύρια γνωρίσματα:

- Κατανόηση φυσικής γλώσσας των ερωτημάτων του χρήστη. (Natural Language Understanding – NLU)
- A priori προγραμματισμένες ενέργειες, ώστε ο ψηφιακός βοηθός να μπορεί να εκτελέσει πολλές διαφορετικές ενέργειες πέρα από το να απαντήσει γνωστικά σε μια ερώτηση.

- Κατανόηση των συμφραζομένων της συζήτησης, δηλαδή το bot θυμάται προηγούμενα ερωτήματα του χρήστη για να απαντήσει παρεμφερή ή συσχετιζόμενα μελλοντικά ερωτήματα του τελευταίου.
- Χρήση τεχνητής νοημοσύνης για την βελτίωση της λειτουργίας του bot σε μια απόπειρα αυτό-μάθησης (AI self-learning) χρησιμοποιώντας τα εκατομμύρια δεδομένα που έχουν μαζέψει οι παραπάνω εταιρίες.
- Φωνητική αλληλεπίδραση με τον ψηφιακό βοηθό, πλήρη κατανόηση της φωνής του χρήστη σε πολλές διαφορετικές γλώσσες και απάντηση από το bot σε αυτές με τη χρήση συνθετικής φωνής.

Ιδιαίτερα το τελευταίο σημείο έχει οδηγήσει στο να χαρακτηριστεί αυτή η γενιά ψηφιακών βοηθών / bots ως Voice Assistants [17].

4.2 Σύγχρονες εφαρμογές των bots

Όλες οι παρακάτω κατηγορίες εφαρμογών chatbots υλοποιούνται στα πλαίσια κάποιας υπάρχουσας πλατφόρμας επικοινωνίας π.χ. Facebook Messenger, Slack, Telegram, Skype ή πιο σπάνια κάποιας αφοσιωμένης εταιρικής πλατφόρμας.

(α) Για τεχνική υποστήριξη καταναλωτών (έναντι της χρήσης τηλεφωνικού κέντρου). Για παράδειγμα το chatbot της εταιρίας VSpirits Technologies Ltd (Virtual Spirits) μπορεί να προσφέρει 24/7 τεχνική υποστήριξη στους πελάτες της εταιρίας που το ενσωματώνει με συγκεκριμένους διαλόγους προσαρμοσμένους στις ανάγκες της εκάστοτε επιχείρησης που υιοθετεί το chatbot της Virtual Spirits [28].

(β) Για προώθηση προσφορών σε ενδιαφερόμενους καταναλωτές από τράπεζες, εταιρίες ασφάλισης, πωλητές κ.τ.λ. Για παράδειγμα το chatbot της γερμανικής εταιρίας κινητής τηλεφωνίας T-mobile το οποίο παρέχει πληροφορίες για την εταιρία και τα προϊόντα της και κατόπιν προτείνει προσφορές σε συσκευές κινητής ομιλίας [29].

(γ) Για απλοποίηση των παραγγελιών σε αλυσίδες καταστημάτων όπως εστιατόρια, καφέ κ.τ.λ. Για παράδειγμα το chatbot της πολυεθνικής Starbucks μέσω του οποίου ο χρήστης μπορεί εύκολα να παραγγείλει καφέ [29].

(δ) Για συμβουλευτική υποστήριξη πάνω σε θέματα παροχής υπηρεσιών, τραπεζικά, υγείας, ταξιδιών κ.τ.λ. Για παράδειγμα το chatbot της γαλλικής τράπεζας Société Générale, το οποίο πέρα από απλά ερωτήματα των χρηστών σε αυτό, μπορεί να προσφέρει συμβουλευτική υποστήριξη σχετικά με τις καταθέσεις τους [30].

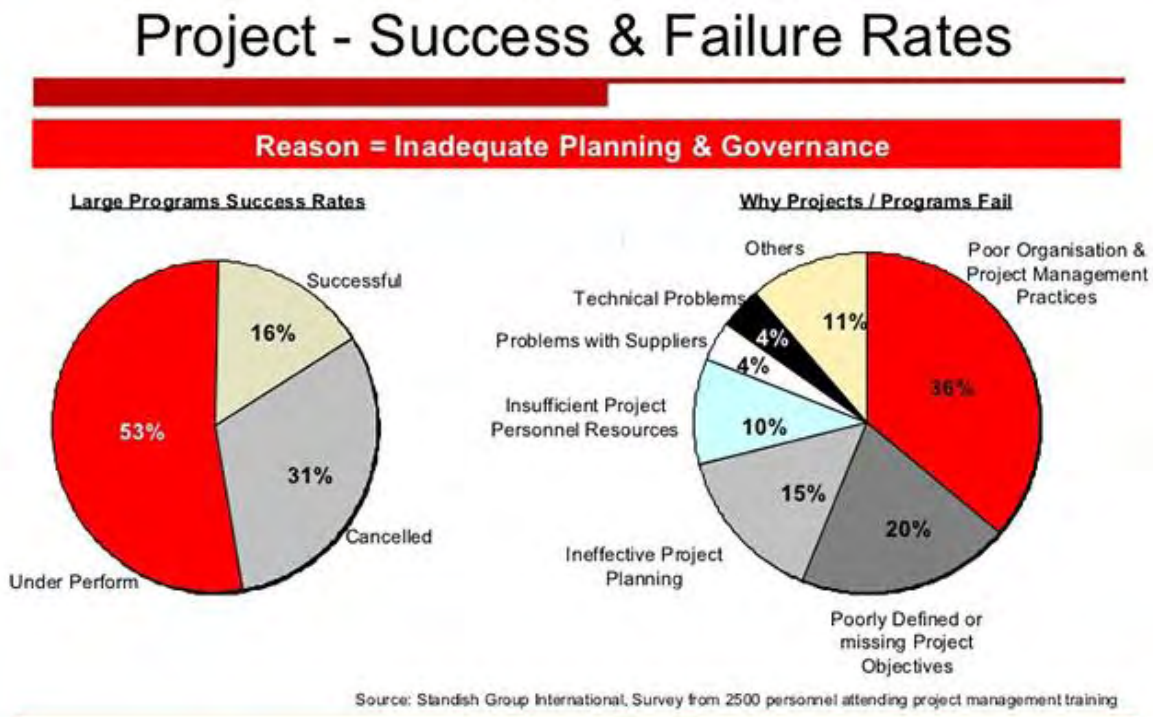
Επίσης η εταιρία Anaamo ισχυρίζεται ότι το chatbot της μπορεί μέσω απλών ερωτημάτων σε έναν ασθενή σχετικά με την κατάσταση της υγείας του να τον κατευθύνει στον σωστό γιατρό [31].

(ε) Για εκπαιδευτικούς σκοπούς, εφαρμογές στις οποίες το bot μπορεί να παρέχει χρήσιμες πληροφορίες στο χρήστη βάσει των ερωτήσεων του. Για παράδειγμα, το chatbot της εταιρίας Wholefoods παρέχει προτεινόμενες συνταγές, γαστρονομικές συμβουλές στο χρήστη βάσει των απαιτήσεων του σε αυτό. Και το chatbot της εταιρίας Duo Lingo (διαθέσιμο εντός της

ομώνυμης εφαρμογής εκμάθησης ξένων γλωσσών), το οποίο προσφέρει ξενόγλωσσους χαρακτήρες που συζητούν σε πραγματικό χρόνο με τον χρήστη με σκοπό να εξασκήσει τις γνώσεις που απέκτησε σε συγκεκριμένες γλώσσες μέσω της εφαρμογής [32].

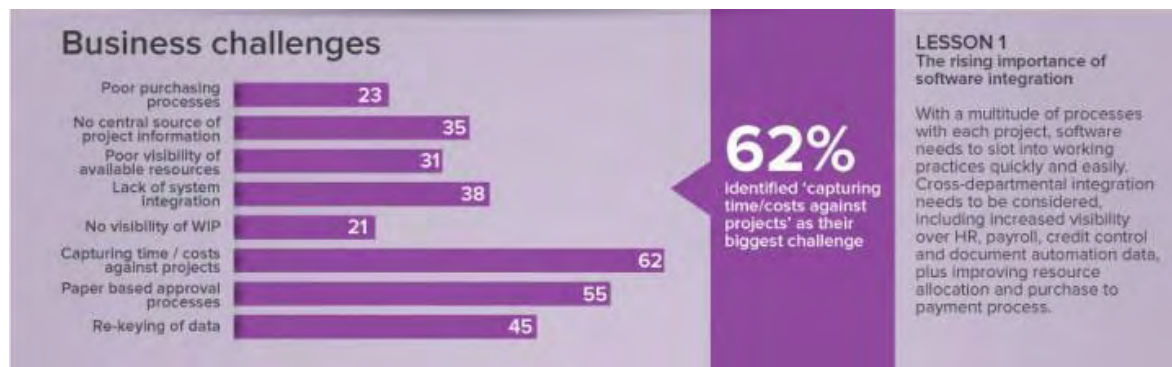
4.3 Ανάπτυξη bots για επίλυση ζητημάτων Project Management

Έναν από τα πιο συνηθισμένα προβλήματα διαχείρισης ενός project, είναι η έλλειψη επικοινωνίας ανάμεσα στον project manager, τους middle managers, και τους προγραμματιστές που δουλεύουν σε συγκεκριμένα υποέργα του συνολικού project, όπως φαίνεται και στην **Εικόνα 4.1**.



Εικόνα 4.1: Στατιστική επί των ποσοστών επιτυχίας / αποτυχίας στο Project Management (Πηγή: [33]).

Όπως φαίνεται και στην **Εικόνα 4.2**, οι πρόσφατες μελέτες έχουν δείξει ότι καταναλώνεται πολύς εταιρικός χρόνος εκ μέρους όλων των συμμετεχόντων στο project, να ενημερώνουν σχετικά με την διαθεσιμότητα τους και πάνω σε τι ακριβώς δουλεύουν με αποτέλεσμα να γίνονται λάθη διαχείρισης, να χάνεται χρόνος και να δουλεύουν όλοι περισσότερο.



Εικόνα 4.2: Στατιστική επί των μεγαλύτερων δυσκολιών στο Project Management (Πηγή: [34]).

Αυτό οφείλεται τόσο στην εναλλαγή προσοχής (Switch of Attention) ένα από τα αρνητικά αποτελέσματα της έννοιας του multitasking (δηλαδή να εκτελεί κανείς πολλές ταυτόχρονες διεργασίες), όσο και στον χρόνο που χρειάζεται οι αλλαγές να ενσωματωθούν από τον project manager όσον αφορά τα ανοιχτά tasks στο project και την διαθεσιμότητα σε ανθρώπινο δυναμικό, καθώς και να ενημερωθεί η ομάδα επί αυτών. Από όλα τα παραπάνω είναι εμφανές το γεγονός ότι η επικοινωνία μέσω email και η χειροκίνητη καταγραφή της δουλειάς και των διαθέσιμων πόρων για αυτή δεν είναι αποτελεσματική [35].

Σύμφωνα με την γνώμη του γράφοντος, μια πιθανή λύση στο παραπάνω ζήτημα θα ήταν η εφαρμογή ενός chatbot στον οργανισμό με το οποίο θα αλληλεπιδρούν όλοι οι εργαζόμενοι ενός project, και αυτό θα αναλαμβάνει αποτελεσματικά την καταγραφή της προόδου σε ένα task, την διαθεσιμότητα των πόρων ή του ανθρώπινου δυναμικού, καθώς και την καταγραφή των ανοιχτών tasks στο project.

Δύο αντιπροσωπευτικά use cases για την εφαρμογή chatbot σε έναν οργανισμό είναι τα εξής [36]:

Use Case 1: Task Assignment (Ανάθεση υποέργου)

Διαφορετικοί εργαζόμενοι σε μια επιχείρηση έχουν διαφορετικό σετ δεξιοτήτων, το οποίο –πέρα από πρότερη εργασιακή εμπειρία– καλλιέργησαν δουλεύοντας σε διάφορα tasks εντός της επιχείρησης, επίσης οι εν λόγω εργαζόμενοι θα προτιμήσουν tasks τα οποία σχετίζονται με δεξιότητες που είναι ήδη καλοί.

Ένα chatbot θα μπορούσε να σαρώσει τα παρελθοντικά δεδομένα (ολοκληρωμένα tasks) και να αποφανθεί βάσει της πρότερης εργασίας τους σχετικά με το ποιος εργαζόμενος είναι κατάλληλος για ποια δουλειά.

Use Case 2: Project Tasks Automation (Αυτοματοποίηση υποέργων του Project)

Το chatbot θα μπορούσε να βοηθήσει τους εργαζομένους ενός project να αυτοματοποιήσουν επαναλαμβανόμενα και τετριμμένα tasks όπως π.χ. το κανόνισμα ενός meeting.

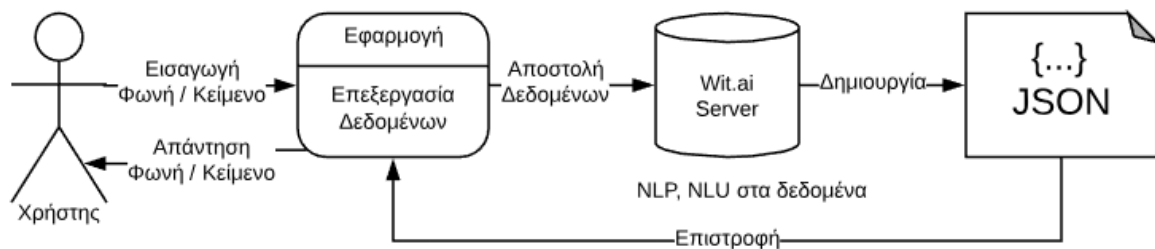
Ένας εργαζόμενος ενός project της επιχείρησης μπορεί να ζητήσει από το chatbot να θέσει ένα meeting στην επιχείρηση το οποίο βολεύει όλους όσους εργάζονται στο project αυτό. Το chatbot θα εξετάσει τις διαθέσιμες ημερομηνίες / ώρες στο ηλεκτρονικό ημερολόγιο της επιχείρησης και κατόπιν θα ρωτήσει τους υπόλοιπους εργαζομένους του project αν μπορούν να παρίστανται (ή όχι) εκείνη την ώρα. Στο τέλος θα συγκεντρώσει αυτά τα δεδομένα και θα βρει τις ώρες που είναι οι περισσότεροι διαθέσιμοι θα τις προτείνει και θα επιλεγεί μία εξ' αυτών από τον Project Manager.

4.4 Εισαγωγή στο wit.ai

Το wit.ai είναι μια πλατφόρμα NLP (Natural Language Processing) και NLU (Natural Language), η οποία ενσωματώνεται εύκολα σε εφαρμογές chatbot. Η πλατφόρμα είναι ελεύθερη τόσο για προσωπική όσο και εμπορική χρήση [37].

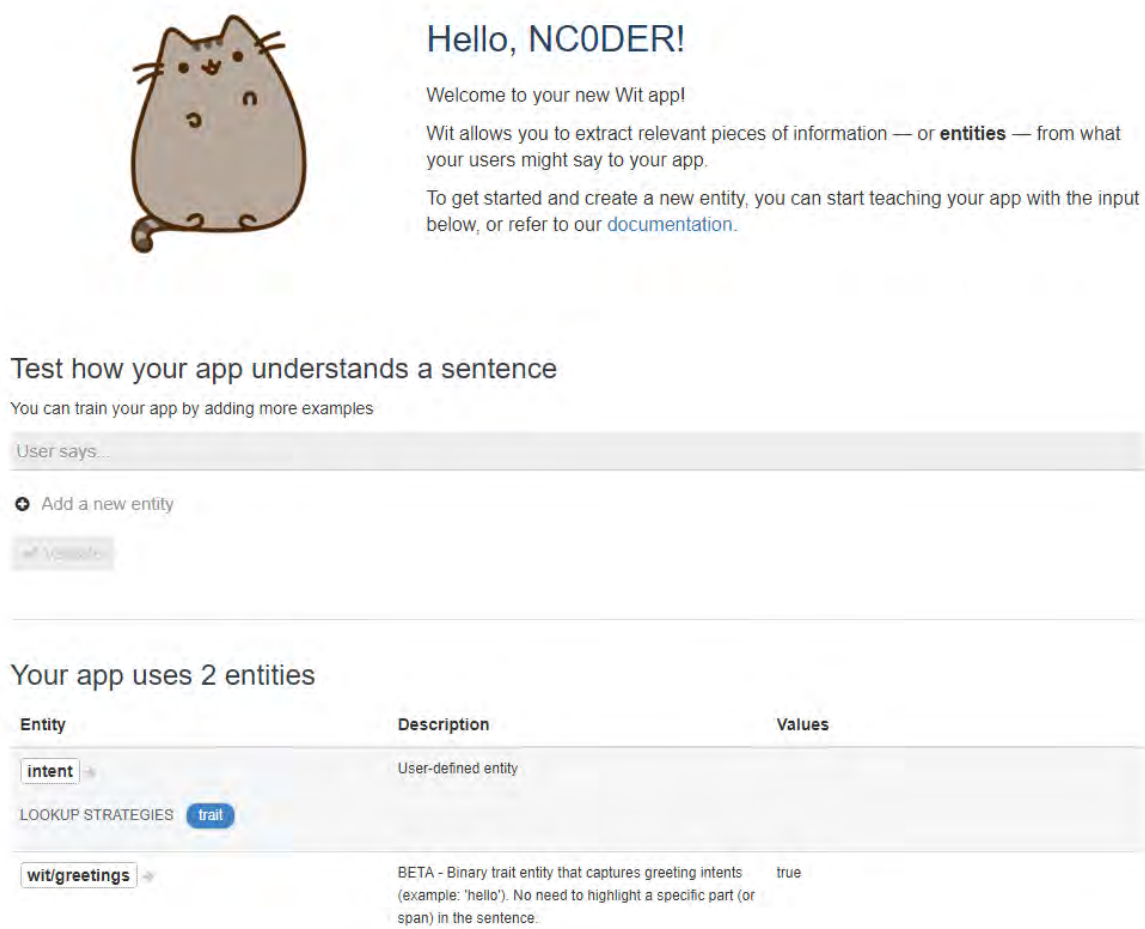
Επιπροσθέτως, η πλατφόρμα υποστηρίζει 136 φυσικές γλώσσες και ενσωματώνεται εύκολα με διαφορετικές γλώσσες προγραμματισμού όπως Node.JS, Python, Ruby και Go. Στην περίπτωση που επιθυμείται η χρήση κάποιας άλλης γλώσσας, υπάρχει το HTTP API, στο οποίο ο προγραμματιστής μπορεί να στείλει και να λάβει αιτήματα του bot του μέσω του πρωτοκόλλου HTTP. Η πλατφόρμα υποστηρίζει δεδομένα κειμένου αλλά και φωνής [38].

Η πλατφόρμα δουλεύει με το μοντέλο που περιγράφεται στην Εικόνα 4.3.



Εικόνα 4.3: Γενικευμένο Μοντέλο εφαρμογής bot με χρήση του Wit.ai.

Η δημιουργία / διαχείριση του bot γίνεται από το site του wit. Εκεί ο προγραμματιστής κάνει login και βλέπει την ακόλουθη οθόνη:



The screenshot shows the Wit.ai interface. On the left is a cartoon cat. To its right, a message says "Hello, NCODER!". Below this, there are instructions: "Welcome to your new Wit app!", "Wit allows you to extract relevant pieces of information — or **entities** — from what your users might say to your app.", and "To get started and create a new entity, you can start teaching your app with the input below, or refer to our [documentation](#)." Below the instructions is a section titled "Test how your app understands a sentence" with a text input field "User says...". Underneath is a button "Add a new entity" and a "Yes/no" button. At the bottom, a table titled "Your app uses 2 entities" lists the entities.

Entity	Description	Values
<code>intent</code>	User-defined entity	
LOOKUP STRATEGIES: <code>trait</code>		
<code>wit/greetings</code>	BETA - Binary trait entity that captures greeting intents (example: 'hello'). No need to highlight a specific part (or span) in the sentence.	true

Εικόνα 4.4: Κεντρική οθόνη προγραμματιστή bot στο wit.ai

Στο πεδίο ‘user says...’ ο προγραμματιστής γράφει προτάσεις που πιστεύει ότι θα ρωτήσει ο χρήστης και το wit της επιστρέφει αναγνωρίζοντας κάποιες οντότητες (entities) τις οποίες μπορεί να διατηρήσει ή να αλλάξει ή να προσθέσει και άλλες. Όσες περισσότερες προτάσεις εισάγει ο προγραμματιστής και αντιστοιχίζει τα συμφραζόμενα τους σε οντότητες τόσο περισσότερο ‘μαθαίνει’ το bot του, με αποτέλεσμα σε παρόμοιες προτάσεις να απαντήσει με σωστές προθέσεις ακόμη και αν ο προγραμματιστής του δεν έχει προβλέψει την εν λόγω ερώτηση.

Παραδείγματος χάριν, το wit.ai όταν εισάγουμε την πρόταση: “What’s the weather tomorrow in Athens?” (Πως είναι ο καιρός αύριο στην Αθήνα) μας επιστρέφει την οντότητα wit/datetime (για την χρονική αναφορά στο αύριο) και εμείς ορίζουμε την οντότητα wit/location (για την τοποθεσία) και την οντότητα get_weather (που εν δυνάμει παραπέμπει σε μια λειτουργία της εφαρμογής μας).

Test how your app understands a sentence

You can train your app by adding more examples

What's the weather tomorrow in Athens ?

<input type="radio"/> get_weather	true	1.000
<input type="radio"/> wit/datetime	4/26/2019, 12:00:00 AM	0.958
<input type="radio"/> wit/location	Resolved value of "Athens" to Athens	0.933

Εικόνα 4.5: Εκμάθηση προθέσεων από προτάσεις στο wit.ai

Αφού εκπαιδύσουμε το bot για την ερώτηση, το επιστρεφόμενο JSON αρχείο για την εφαρμογή μας είναι στην ακόλουθη μορφή:

```
{
  "_text": "What's the weather tomorrow in Athens ?",
  "entities": {
    "get_weather": [
      {
        "confidence": 1,
        "value": "true",
        "type": "value"
      }
    ],
    "datetime": [
      {
        "confidence": 0.95789,
        "values": [
          {
            "value": "2019-04-26T00:00:00.000-07:00",
            "grain": "day",
            "type": "value"
          }
        ],
        "value": "2019-04-26T00:00:00.000-07:00",
        "grain": "day",
        "type": "value"
      }
    ],
    "location": [
      {
        "confidence": 0.93261,
        "value": "Athens",
        "resolved": {
          "values": [
            {
              "name": "Athens",
              "grain": "locality",
              "type": "resolved",
              "timezone": "Europe/Athens",
              "coords": {
                "lat": 37.98376083374,
                "long": 23.727840423584
              }
            }
          ]
        }
      }
    ]
  }
}
```

```

        "external": {
          "geonames": "264371",
          "wikidata": "Q1524",
          "wikipedia": "Athens"
        }
      },
      {
        "name": "Athens",
        "grain": "locality",
        "type": "resolved",
        "timezone": "America/New_York",
        "coords": {
          "lat": 33.960948944092,
          "long": -83.377937316895
        },
        "external": {
          "geonames": "4180386",
          "wikidata": "Q203263",
          "wikipedia": "Athens, Georgia"
        }
      },
      {
        "name": "Athens",
        "grain": "locality",
        "type": "resolved",
        "timezone": "America/New_York",
        "coords": {
          "lat": 39.329238891602,
          "long": -82.101257324219
        },
        "external": {
          "geonames": "4505542",
          "wikidata": "Q755420",
          "wikipedia": "Athens, Ohio"
        }
      }
    ]
  },
  "msg_id": "1pUDeBYObkTBxZX1M"
}

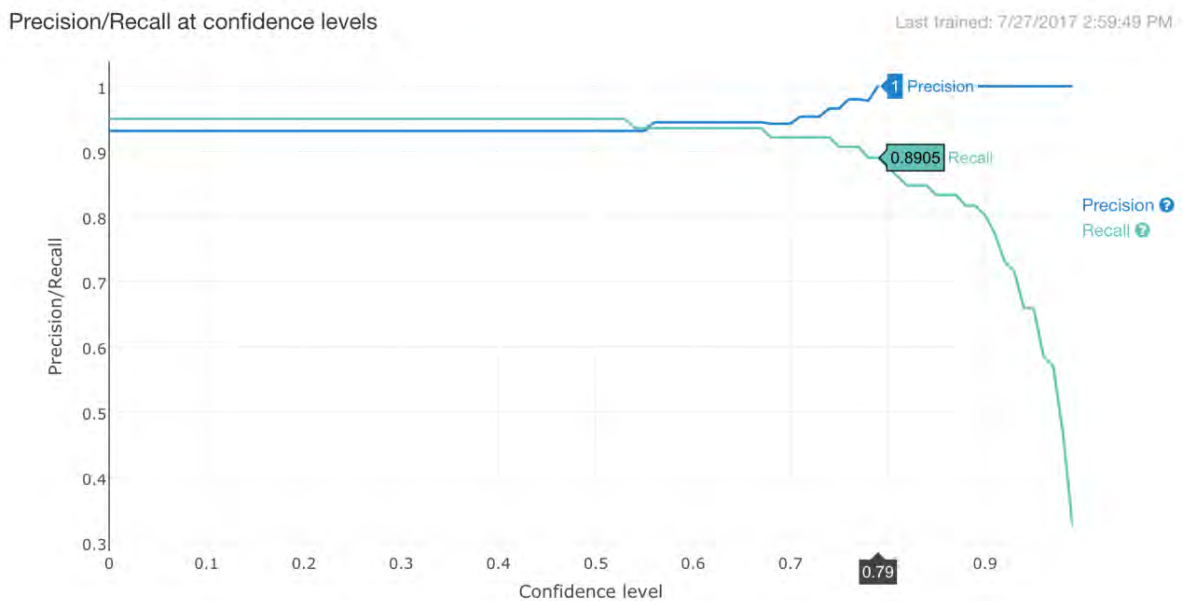
```

Όπως φαίνεται και στο παραπάνω JSON αρχείο κάθε πρόθεση έχει ένα πεδίο confidence (το οποίο εκφράζει το confidence score) δηλαδή το ποσοστό βεβαιότητας με το οποίο το wit.ai πιστεύει ότι εντόπισε σωστά την πρόθεση.

Ένα κλασσικό ερώτημα που μπορεί να σκεφτεί κανείς είναι το εξής:

Ποιο είναι το ελάχιστο ποσοστό βεβαιότητας για το οποίο θεωρούμε ότι το wit.ai έχει βρει σωστά μια πρόθεση μιας πρότασης;

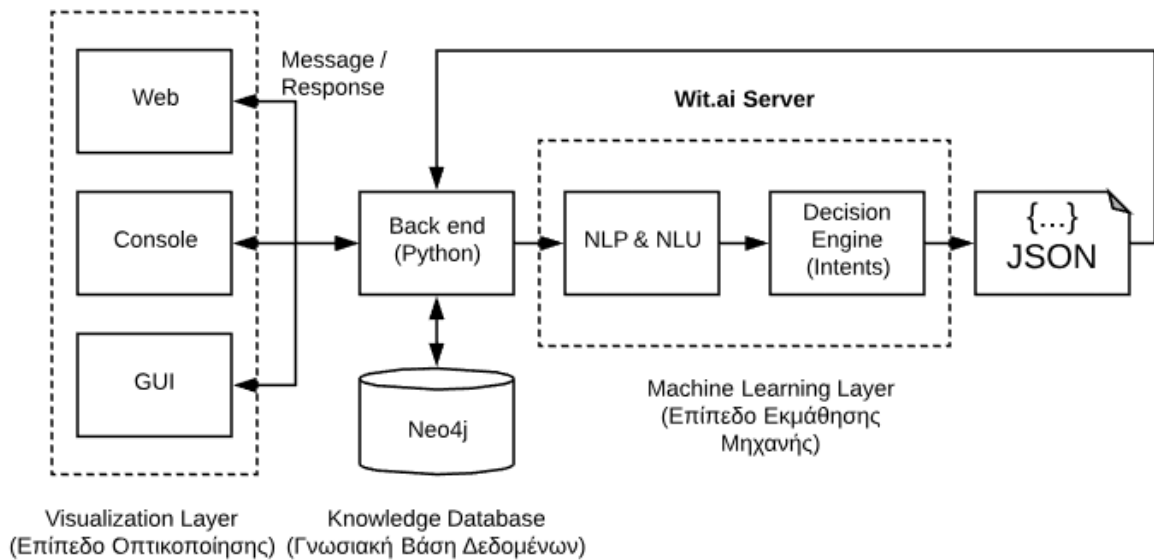
Κοιτώντας το παράδειγμα στο site του wit.ai για ένα bot, συμπεραίνουμε ότι για να έχουμε επίπεδο ακρίβειας 100% (precision 1.0) δηλαδή ότι βρέθηκε σωστά μια εκ των προθέσεων της πρότασης θα πρέπει να δούμε το ελάχιστο επίπεδο εμπιστοσύνης (confidence score) στον οριζόντιο άξονα της γραφικής παράστασης, το οποίο για την συγκεκριμένη πρόθεση είναι το 0.79, οπότε στην εν λόγω εφαρμογή θα αγνοούσαμε την εν λόγω πρόθεση αν ανιχνευόταν με επίπεδο εμπιστοσύνης μικρότερο του 0.79.



Εικόνα 4.6: Γραφική παράσταση μεταξύ του επιπέδου ακρίβειας & εμπιστοσύνης (Πηγή: [39]).

4.5 Ανάπτυξη ενός chatbot για επίλυση ζητημάτων Project Management

Στα πλαίσια της παρούσας εργασίας υλοποιούμε ένα chatbot με σκοπό να καλύψουμε το Use Case 1 που αναφέρθηκε στην Ενότητα 4.3. Το bot αυτό υλοποιείται με χρήση της πλατφόρμας wit.ai που αναλύσαμε παραπάνω ακολουθώντας την αρχιτεκτονική της Εικόνας 4.7, η οποία είναι προσαρμογή της αρχιτεκτονικής που εμφανίζεται στο [40].



Εικόνα 4.7: Αρχιτεκτονική του Project Management bot.

4.5.1 Απαιτήσεις εφαρμογής

Για την υλοποίηση του chatbot θα χρειαστούμε τα ακόλουθα χαρακτηριστικά συστήματος, λογισμικού, βιβλιοθηκών στις εκδόσεις που αναφέρονται (οι εκδόσεις αυτές, θα είναι οι ελάχιστες δυνατές ώστε το chatbot να έχει πλήρη λειτουργικότητα).

Η εφαρμογή δοκιμάστηκε σε Windows 10 64-bit και Ubuntu Linux 64-bit. Έχει γραφεί σε Python 3 με ελάχιστη υποστηριζόμενη έκδοση την 3.6. Η βάση δεδομένων που χρησιμοποιείται είναι η Neo4j με ελάχιστη υποστηριζόμενη έκδοση βάσης την 3.5.3.

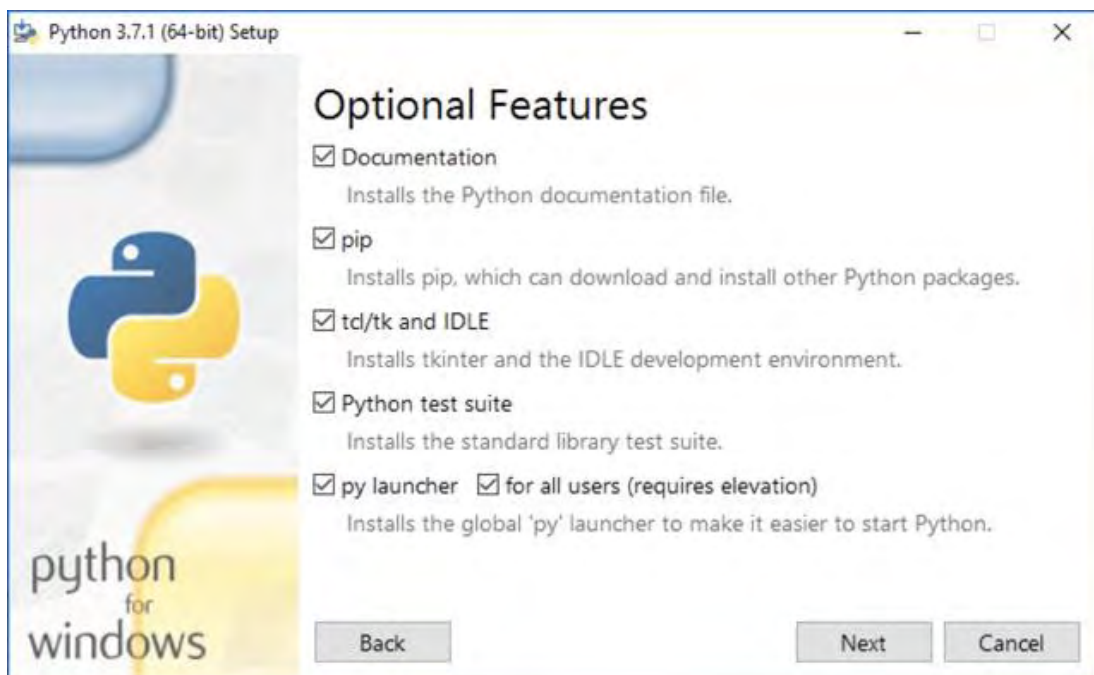
Οδηγίες εγκατάστασης σε Windows

Αρχικά εγκαθιστούμε την [python](#) κατεβάζοντας και εκτελώντας το εκτελέσιμο στο σύνδεσμο (εκτός και αν έχουμε ήδη εγκαταστήσει έκδοση μεγαλύτερης της 3.6.8).



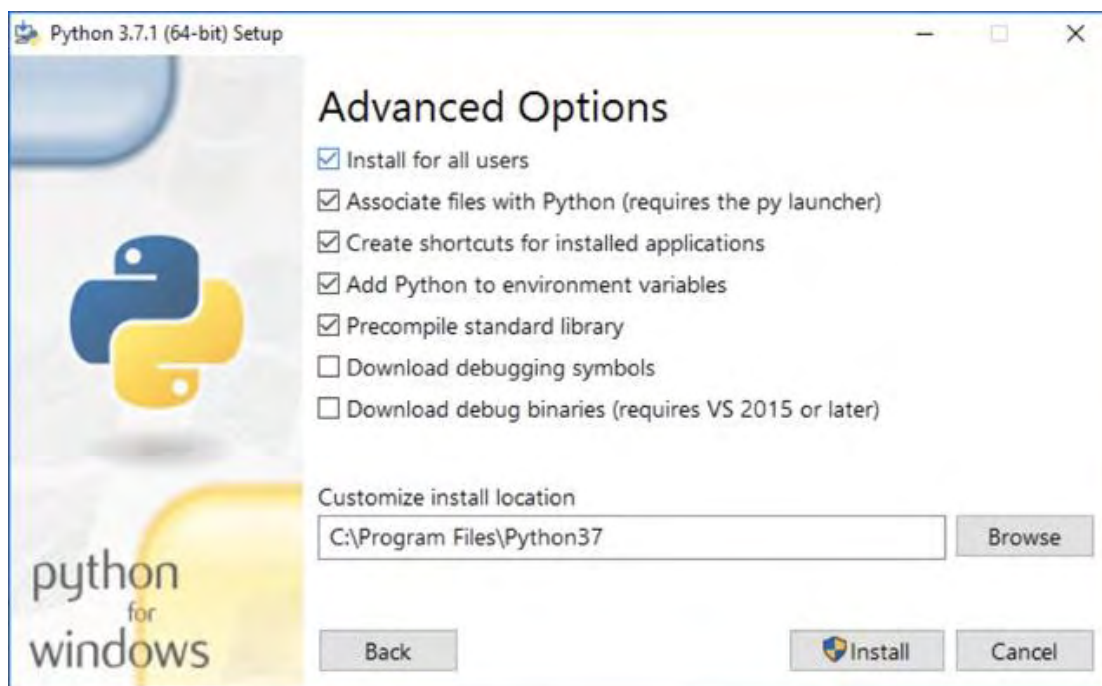
Εικόνα 4.8: Βήμα 1/5 της εγκατάστασης (Πηγή: [41]).

Μαρκάρουμε την επιλογή “Add Python 3.7 to PATH”, μετά πατάμε αριστερό κλικ στην επιλογή “Customize Installation”. Εμφανίζεται η οθόνη της Εικόνας 4.9.



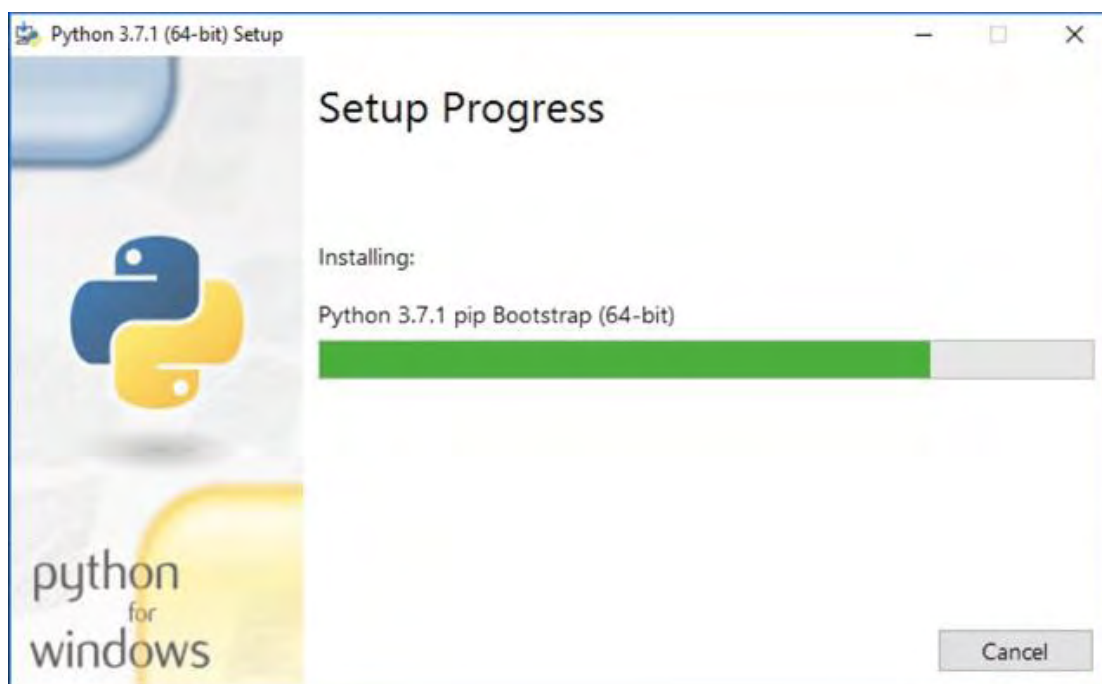
Εικόνα 4.9: Βήμα 2/5 της εγκατάστασης (Πηγή: [42]).

Μαρκάρουμε τις επιλογές “pip” και “py launcher”, οι οποίες θα μας χρειαστούν στην εγκατάσταση της εφαρμογής. Μετά πατάμε αριστερό κλικ στο κουμπί “Next” και μεταβαίνουμε στην την οθόνη της Εικόνας 4.10.



Εικόνα 4.10: Βήμα 3/5 της εγκατάστασης (Πηγή: [43]).

Μετά κλικάρουμε με αριστερό κλικ το κουμπί “Install” και μεταβαίνουμε στην οθόνη της Εικόνας 4.11.



Εικόνα 4.11: Βήμα 4/5 της εγκατάστασης (Πηγή: [44]).

Όταν ολοκληρωθεί η εγκατάσταση βλέπουμε την οθόνη της Εικόνας 4.12.



Εικόνα 4.12: Βήμα 5 / 5 της εγκατάστασης (Πηγή: [45]).

Αριστερό κλικ στο κουμπί “Close” και ολοκληρώσαμε επιτυχώς την εγκατάσταση της python.

Στη συνέχεια εγκαθιστούμε την βάση του [neo4j](#). Κατεβάζουμε και τρέχουμε το εκτελέσιμο, διαλέγουμε το φάκελο εγκατάστασης της βάσης πατάμε με αριστερό κλικ “Next” μέχρι το τέλος της εγκατάστασης (βλέπε Ενότητα **Ρυθμίσεις & Αρχικοποίηση βάσης Neo4j (Windows & Linux)**).

Μετά την εγκατάσταση της python και της neo4j θα εγκαταστήσουμε το project και τις βιβλιοθήκες (σε python) που χρειάζεται. Αρχικά κλικάρουμε το παρακάτω σύνδεσμο για να κατεβάσουμε το project. <https://github.com/NCODER/NeoJira.ai/archive/master.zip>

Αποσυμπιέζουμε το .zip αρχείο με οποιοδήποτε πρόγραμμα αποσυμπίεσης. Μπαίνουμε στον παραγόμενο φάκελο NeoJira.ai-master μέσω τερματικού (cmd) και εκτελούμε την εντολή: `py setup.py install -user`, η οποία εγκαθιστά τις απαραίτητες βιβλιοθήκες στην python του συνδεδεμένου χρήστη. Αν κάποια από τα πακέτα αυτά υπάρχουν ήδη εγκατεστημένα στο σύστημα σε νεότερες εκδόσεις, τότε το script εγκατάστασης τα διατηρεί ανέπαφα και εγκαθιστά τα υπόλοιπα. Τα πακέτα που εγκαθίστανται συνολικά φαίνονται στον παρακάτω πίνακα.

Όνομα βιβλιοθήκης	Ελάχιστη έκδοση
certifi	2019.3.9
chardet	3.0.4
Click	7.0
Flask	1.0.2
idna	2.8
itsdangerous	1.1.0
Jinja2	2.10.1
MarkupSafe	1.1.1
neo4j	1.7.2
neobolt	1.7.4
neotime	1.7.4
nltk	3.4
numpy	1.16.2
pip	19.0.3
prompt-toolkit	2.0.9
pytz	2018.9
requests	2.21.0
scikit-learn	0.20.3
scipy	1.2.1
setuptools	41.0.0
singledispatch	3.4.0.3
six	1.12.0
urllib3	1.24.2
wcwidth	0.1.7
Werkzeug	0.15.2
wit	5.1.0

Πίνακας 4.1: Προαπαιτούμενες βιβλιοθήκες του project προς εγκατάσταση.

Οδηγίες εγκατάστασης σε Debian-Based Linux

Όλη η εγκατάσταση της εφαρμογής γίνεται από το τερματικό. Αρχικά εγκαθιστούμε την python 3.6 (εκτός και αν υπάρχει η ίδια ή νεότερη έκδοση) κατόπιν εγκαθιστούμε την νεότερη έκδοση του εργαλείου git, κατεβάζουμε το project στον τρέχοντα φάκελο, μπαίνουμε στον φάκελο του project και εγκαθιστούμε τις βιβλιοθήκες της εφαρμογής στον λογαριασμό του χρήστη (οι οποίες φαίνονται στον Πίνακα 4.1.)

```
sudo apt install python3.6
sudo apt install git-all
git clone https://github.com/NC0DER/NeoJira.ai
cd Neojira.ai
python3 setup.py install --user
```

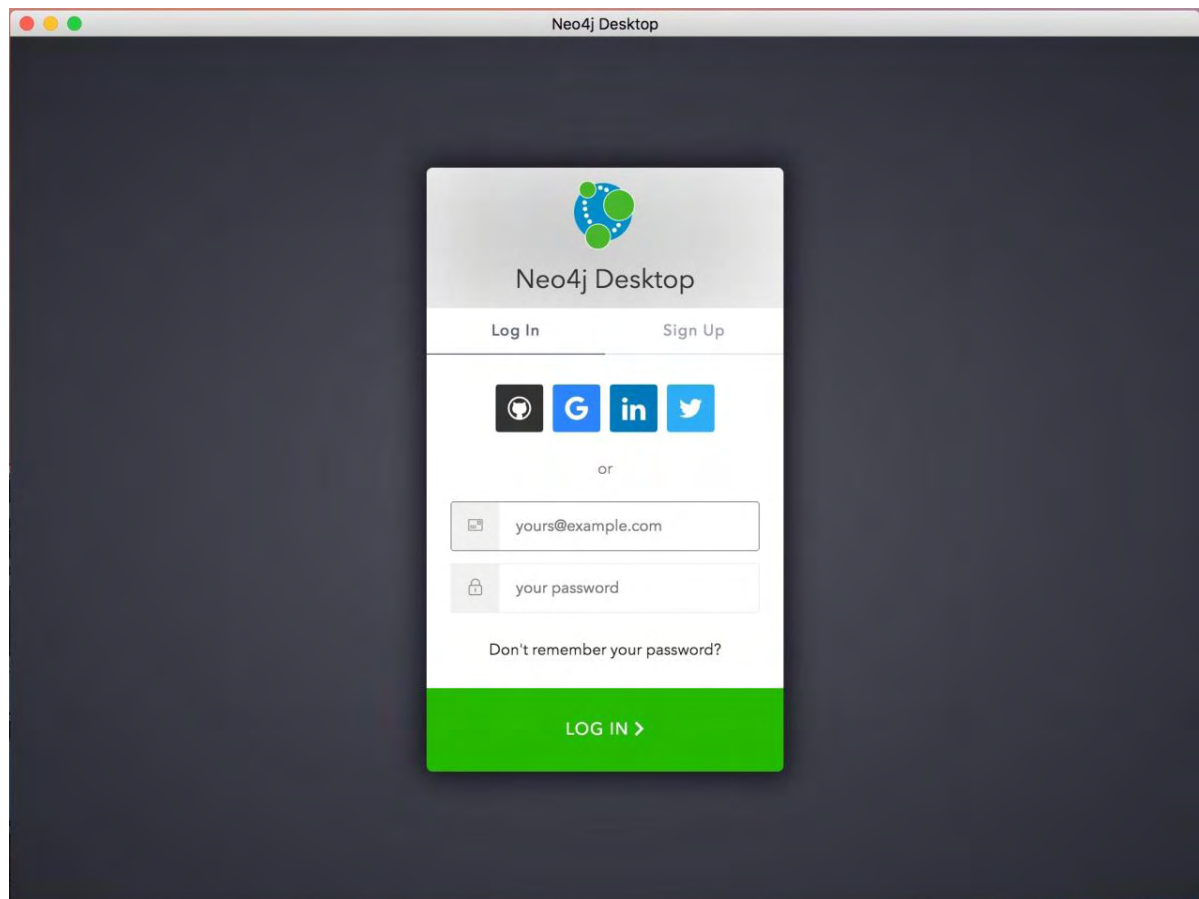
Στη συνέχεια κατεβάζουμε το εκτελέσιμο της βάσης του [neo4j](#) ανοίγουμε το τερματικό και γράφουμε την εντολή η οποία δίνει το δικαίωμα εκτέλεσης στο αρχείο. Μετά την ολοκλήρωση της εντολής τρέχουμε το εκτελέσιμο της βάσης είτε με διπλό κλικ είτε από το τερματικό.

```
chmod +x neo4j-desktop-offline-1.1.22-x86_64.AppImage
```

Ρυθμίσεις & Αρχικοποίηση βάσης Neo4j (Windows & Linux)

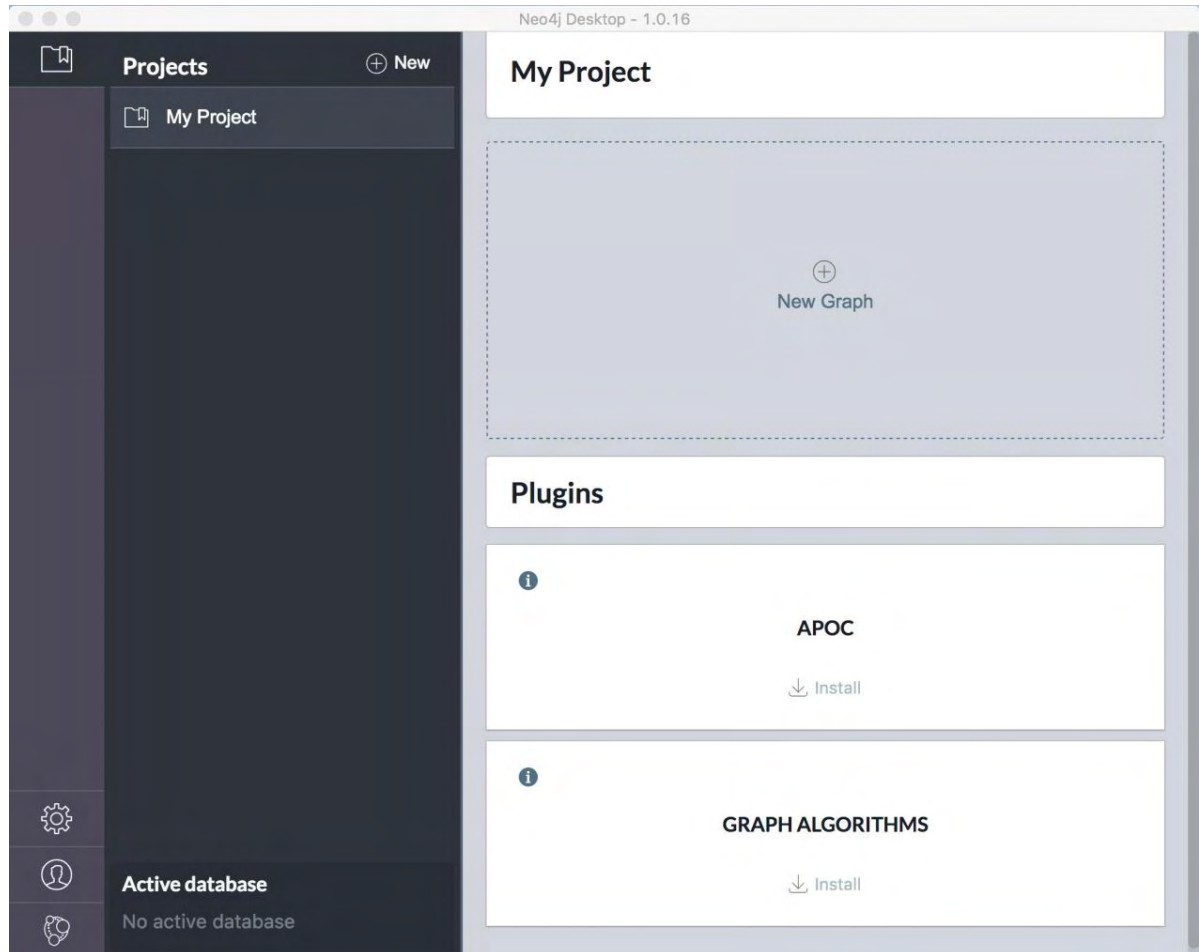
Όταν τρέχουμε για πρώτη φορά το Neo4j Desktop app (δηλαδή το λογισμικό διαχείρισης της βάσης) πρέπει να ρυθμίσουμε κάποια πράγματα.

Αρχικά θα δημιουργήσουμε λογαριασμό για το neo4j ή θα κάνουμε login σε περίπτωση που έχουμε ήδη. Η εγγραφή γίνεται με λογαριασμό Google ή προφίλ σε social media ή λογαριασμό GitHub.

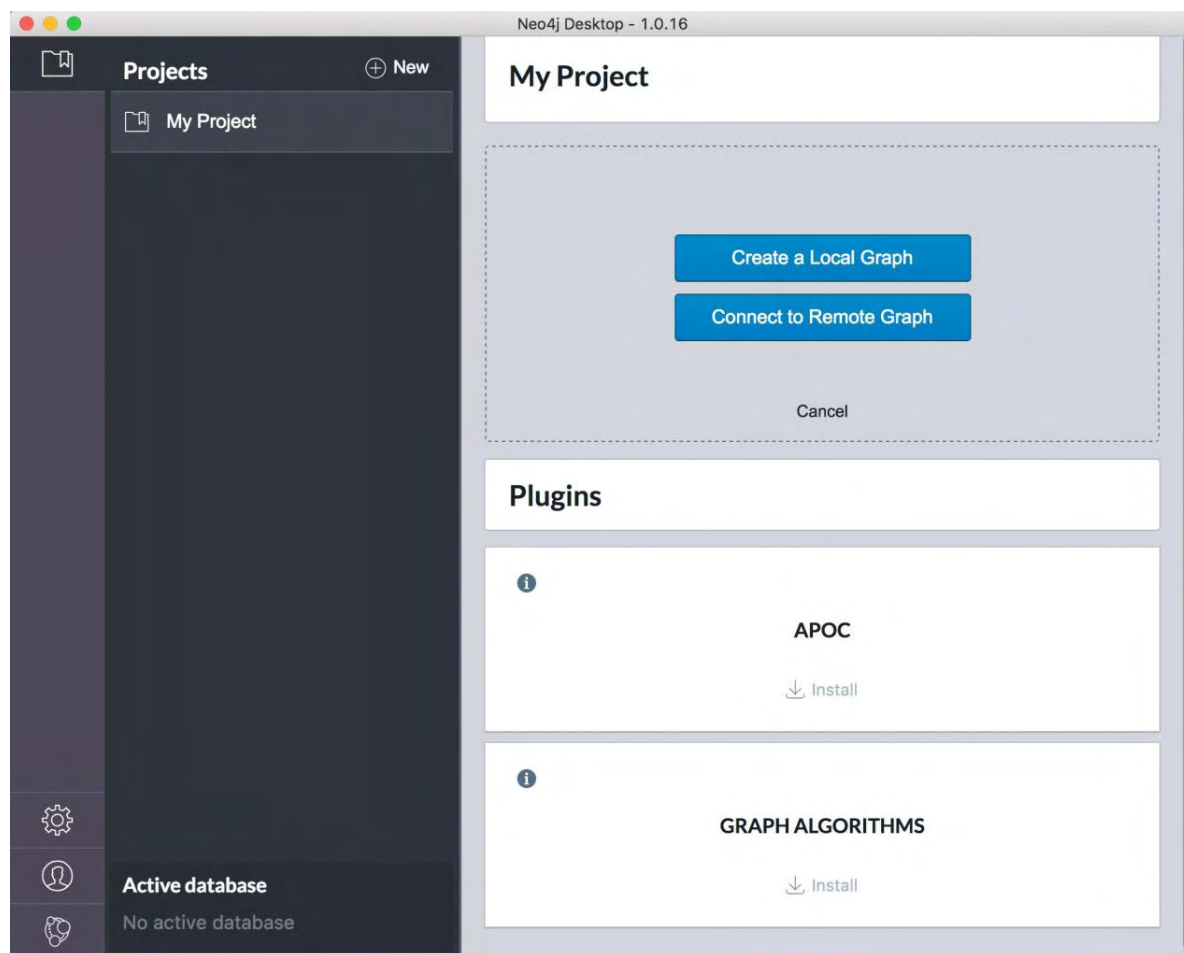


Εικόνα 4.13: Οθόνη εγγραφής / σύνδεσης χρήστη (Πηγή: [46]).

Στη συνέχεια θα δημιουργήσουμε μια τοπική βάση πατώντας την επιλογή “Create new graph” και επιλέγοντας το μπλε κουμπί με τίτλο “Create a local graph” όπως φαίνεται στις Εικόνες 4.14, 4.15.

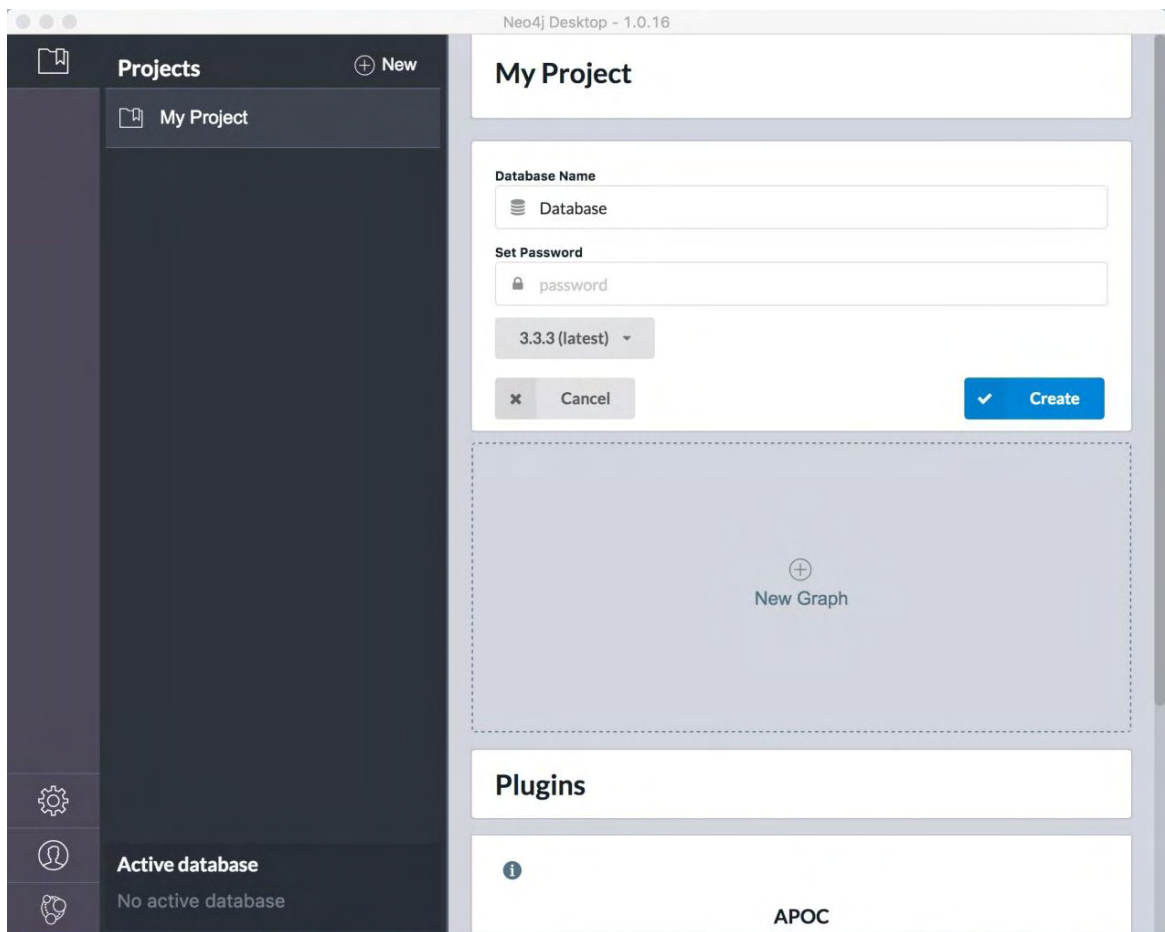


Εικόνα 4.14: Οθόνη διαχείρισης βάσεων (Πηγή: [47]).

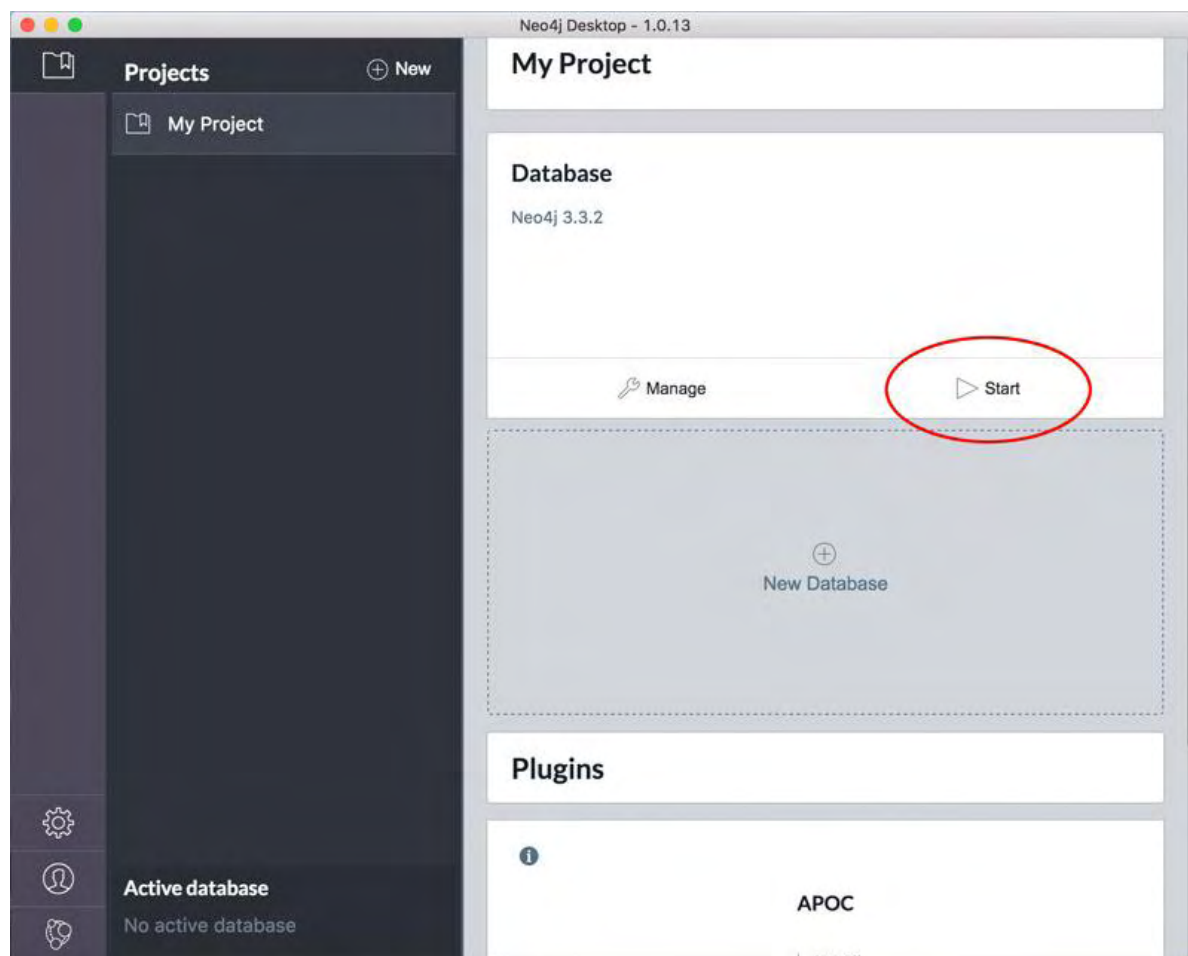


Εικόνα 4.15: Οθόνη διαχείρισης βάσεων, δημιουργία βάσης 1 / 2 (Πηγή: [48]).

Στην Εικόνα 4.16 φαίνεται η οθόνη δημιουργίας βάσεων στην οποία συμπληρώνουμε τα στοιχεία όπως το όνομα της βάσης (προτεινόμενο Jira) και το πεδίο password στο οποίο βάζουμε την τιμή 123 (όπως υπάρχει και στα python scripts της εφαρμογής). Δεδομένου ότι η βάση τρέχει μόνο τοπικά στην τρέχουσα εγκατάσταση δεν χρειάζεται ένα σύνθετο password. Το τελευταίο στοιχείο είναι η έκδοση της βάσης η οποία πρέπει κατ' ελάχιστον να είναι η 3.5.3 ή και νεότερη, (ο περιορισμός αυτός οφείλεται σε μια σειρά αλγορίθμων ομοιότητας που είναι μόνο διαθέσιμοι από την 3.5.3 και έπειτα). Κατόπιν κλικάρουμε το μπλε κουμπί “Create” και ολοκληρώνουμε την δημιουργία της βάσης (βλέπε Εικόνα 4.17).

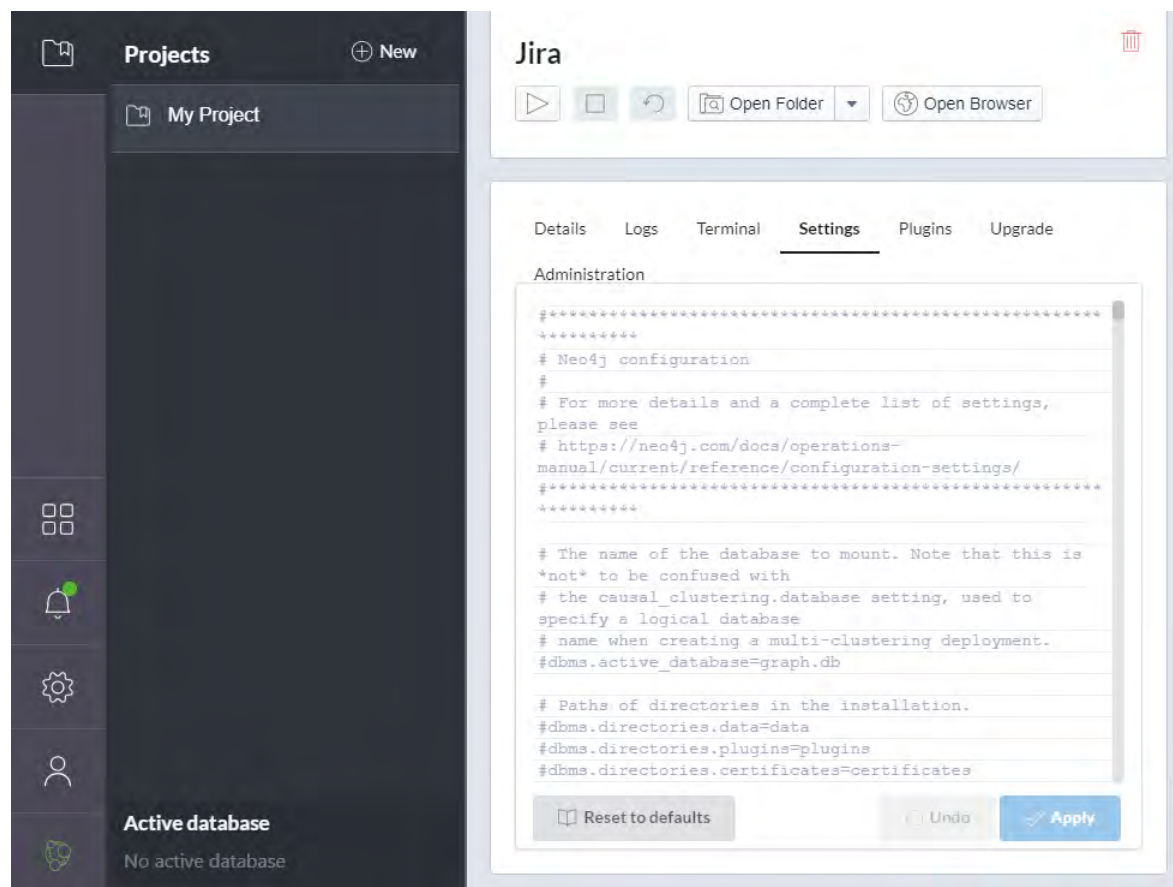


Εικόνα 4.16: Οθόνη διαχείρισης βάσεων, δημιουργία βάσης 2 / 2 (Πηγή: [49]).



Εικόνα 4.17: Οθόνη διαχείρισης βάσεων, κατόπιν δημιουργίας βάσης (Πηγή: [50]).

Πριν ξεκινήσουμε την βάση θα πρέπει να εγκαταστήσουμε τα πρόσθετα (**Plugins**) που βρίσκονται λίγο πιο κάτω στην οθόνη, τα **APOC** και **Graph Algorithms**, τα οποία χρειάζονται στην εφαρμογή, πατώντας την ένδειξη “Install” και περιμένοντας μέχρι να εγκατασταθούν επιτυχώς. Επιπροσθέτως, θα πρέπει να κάνουμε ορισμένες ρυθμίσεις στην βάση για λόγους απόδοσης. Κλικάρουμε το κουμπί “Manage” κατόπιν την ένδειξη “Settings” και μεταβαίνουμε στην οθόνη της Εικόνας 4.18.



Εικόνα 4.18: Οθόνη διαχείρισης βάσης Jira, ενότητα Settings (Ρυθμίσεις).

Σε αυτή την οθόνη μπορούμε εύκολα να περάσουμε τις ρυθμίσεις που βρίσκονται στο επισυναπτόμενο αρχείο `neo4j_config.txt`, ανοίγουμε το αρχείο πατάμε την συντόμευση πληκτρολογίου (CTRL + A: επιλογή όλου του κειμένου) κατόπιν στην ενότητα setting πατάμε στο κείμενο ώστε να αναβοσβήνει ο κέρσορας και εν συνεχεία πατάμε πάλι CTRL + A και μετά CTRL + V: επικόλληση και τέλος το κουμπί “Apply” για να εφαρμοστούν οι νέες ρυθμίσεις.

Σε αυτό το στάδιο είμαστε έτοιμοι να τρέξουμε την βάση πατώντας το κουμπί “Run” όπως φαίνεται στην Εικόνα 4.17 και να την γεμίσουμε με τα δεδομένα του αρχείου `hadoop_issues.json` κάνοντας χρήση του εκτελέσιμου που βρίσκεται στο φάκελο `neojira` του project με όνομα `jira_issues_parser.py` το οποίο σε 2 με 5 λεπτά (αναλόγως της απόδοσης του συστήματος) θα δημιουργήσει τα χιλιάδες issues και τις μεταξύ τους σχέσεις.

Τέλος εκτελούμε το αρχείο `neo4jira_cli.py` αν θέλουμε να χρησιμοποιήσουμε την εφαρμογή σε περιβάλλον κονσόλας ή εναλλακτικά εκτελούμε το αρχείο `flask_server.py` για να ανοίξουμε τον server που αλληλεπιδρά με την βάση και με το περιβάλλον γραφικής διεπαφής που δημιουργείται όταν κλικάρουμε το αρχείο `neojira.pyw` το οποίο ανοίγει το περιβάλλον γραφικής διεπαφής σε έναν από τους εγκατεστημένους περιηγητές του συστήματος.

4.5.2 Βήματα υλοποίησης εφαρμογής

Σε αυτή την ενότητα θα κάνουμε μια επισκόπηση της συλλογιστικής με την οποία αναπτύξαμε την εφαρμογή. Στο πρώτο βήμα στήσαμε μια βάση δεδομένων στο neo4j και πειραματιστήκαμε με ερωτήματα (εντολές που ανακτούν δεδομένα της βάσης) τα οποία παράγουν γνώση εκ των υπαρχόντων δεδομένων. Στη συνέχεια βελτιώσαμε τα ερωτήματα αυτά για λόγους απόδοσης και καταγράψαμε τα αποτελέσματα τους.

Στο δεύτερο βήμα διασυνδέσαμε την Python3 με την βάση του neo4j και φτιάξαμε εκτελέσιμα που τρέχουν εντολές στην βάση για δημιουργία οντοτήτων και σχέσεων από το αρχείο και τα ερωτήματα που φτιάξαμε. Σε αυτό το σημείο προσθέσαμε μια νέα κατηγορία οντότητας στα δεδομένα, τις ‘ετικέτες’ οι οποίες είναι οι πιο σημαντικοί όροι που προέκυψαν μέσω NLP ανάλυσης από την περιγραφή κειμένου οντοτήτων της βάσης.

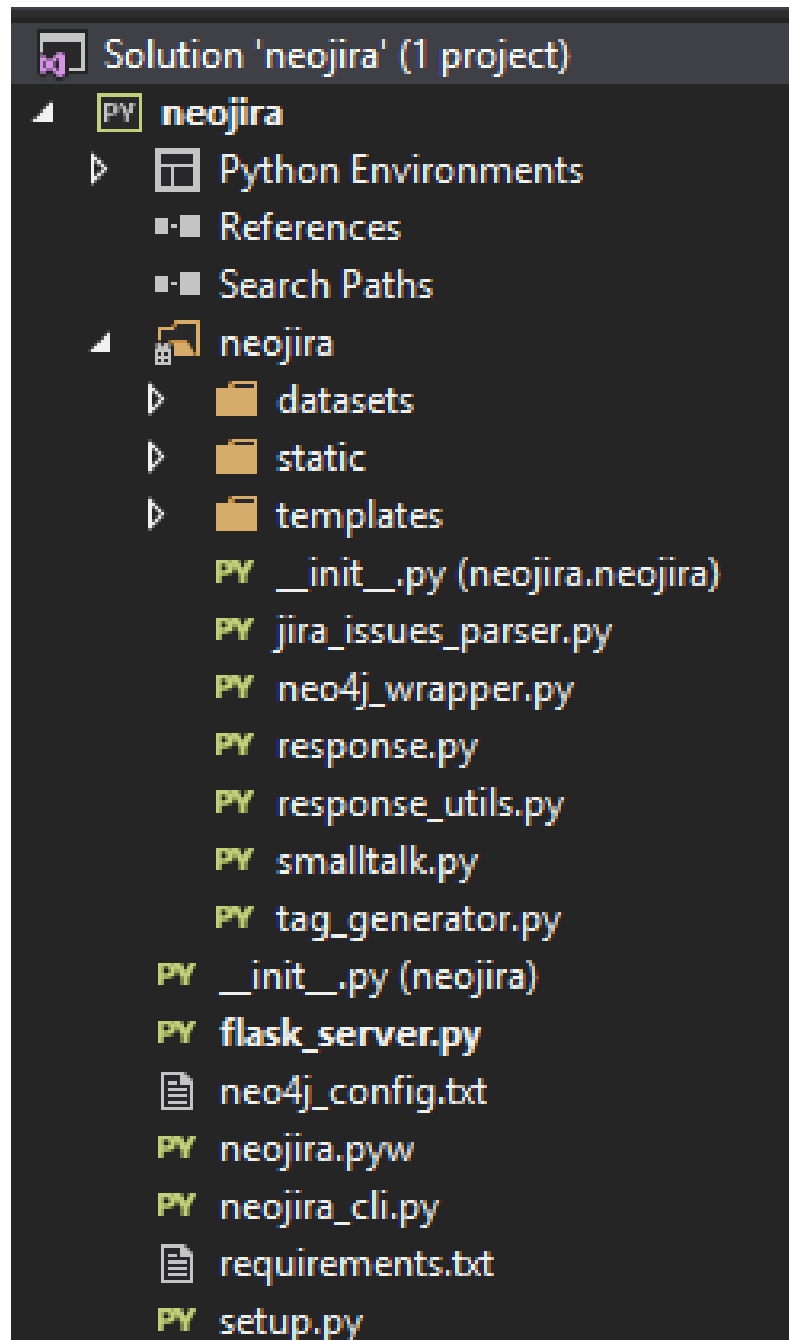
Στο τρίτο βήμα προσθέσαμε την διασύνδεση της Python3 και της βάσης με την πλατφόρμα του wit.ai, η τελευταία μας επέτρεψε να φτιάξουμε ένα chatbot και να το εκπαιδεύσουμε με γνωστές προτάσεις / ερωτήματα για τις ανάγκες της εφαρμογής μας δηλαδή την 1-1 διασύνδεση ανάμεσα σε ένα οποιαδήποτε ερώτημα (που ανήκει σε ένα σύνολο παρόμοιων ερωτημάτων) και σε ένα κομμάτι γνώσης το οποίο προκύπτει από την εκτέλεση ενός εκ των παραπάνω ερωτημάτων.

Στο τέταρτο βήμα προσθέσαμε την βιβλιοθήκη του Flask στην Python3, η οποία μας επέτρεψε να στήσουμε ένα τοπικό webserver ο οποίος δημιουργεί ένα γραφικό περιβάλλον για την εφαρμογή μας μέσω του browser του συστήματος με την χρήση γνωστών τεχνολογιών web όπως HTML5, CSS, JavaScript κ.τ.λ.

4.5.3 Δομή λογισμικού εφαρμογής

Αρχικά ξεκινάμε με την οπτικοποίηση του project σε φακέλους όπως φαίνεται στο Visual Studio. Το project έχει την εξής δομή: Τα κύρια εκτελέσιμα scripts είναι στο top level του project. Τα δευτερεύοντα scripts που έχουν βοηθητικό ρόλο στα κύρια scripts καθώς και τα datasets (αρχεία δεδομένων) βρίσκονται στον φάκελο neojira.

Στον Πίνακα 4.2 θα εξηγήσουμε περιληπτικά την λειτουργία του κάθε αρχείου ή φακέλου.



Εικόνα 4.19: Ιεραρχία αρχείων του project.

Όνομα Αρχείου / Φακέλου	Περιγραφή Λειτουργίας
setup.py	Εκτελέσιμο που εγκαθιστά όλες τις απαραίτητες προαπαιτούμενες βιβλιοθήκες της εφαρμογής.
requirements.txt	Αρχείο κειμένου που περιέχει την λίστα των ακριβή εκδόσεων που χρησιμοποιήθηκαν όταν φτιάχτηκε το project.
neo4j_config.txt	Αρχείο κειμένου που περιέχει τις ρυθμίσεις για την βάση δεδομένων.
neojira_cli.py	Εκτελέσιμο που τρέχει την εφαρμογή σε κονσόλα.
flask_server.py	Εκτελέσιμο που τρέχει τον server που στέλνει αιτήματα στην βάση και σηκώνει το web περιβάλλον γραφικών της εφαρμογής.
neojira_pyw	Εκτελέσιμο που ανοίγει ένας από τους εγκατεστημένους περιηγητές του συστήματος στην τοπική σελίδα της εφαρμογής που περιέχει το web περιβάλλον γραφικών της εφαρμογής.
templates	Φάκελος που περιέχει το index.html το οποίο είναι το αρχείο που περιγράφει την δομή του web περιβάλλον γραφικών της εφαρμογής.
static	Φάκελος που περιέχει “στατικά” αρχεία της εφαρμογής που σχετίζονται με το web περιβάλλον γραφικών της π.χ. .css ή .js αρχεία.
css	Φάκελος που περιέχει τα .css αρχεία (βιβλιοθήκη Bootstrap) και ένα προσαρμοσμένο, τα οποία περιγράφουν το στυλ των γραφικών αντικειμένων του web περιβάλλον γραφικών της εφαρμογής.
js	Φάκελος που περιέχει τα .js αρχεία (βιβλιοθήκη Bootstrap & JQuery) και ένα προσαρμοσμένο, τα οποία περιγράφουν την συμπεριφορά (ως προς τις ενέργειες του χρήστη) των γραφικών αντικειμένων του web περιβάλλον γραφικών της εφαρμογής.
neojira	Φάκελος που περιέχει βοηθητικά εκτελέσιμα της εφαρμογής και αρχεία δεδομένων.
datasets	Φάκελος που περιέχει αρχεία δεδομένων όπως το hadoop_issues.json.
jira_issues_parser.py	Εκτελέσιμο που δημιουργεί τους κόμβους

	και τις σχέσεις στην βάση του neo4j, διαβάζοντας το αρχείο <code>hadoop_issues.json</code>
<code>neo4j_wrapper.py</code>	Εκτελέσιμο που ενθυλακώνει την λειτουργικότητα αλληλεπίδρασης με την βάση εντός μιας κλάσης και των μεθόδων / συναρτήσεων της.
<code>response.py</code>	Εκτελέσιμο που χειρίζεται την δομή του διαλόγου μεταξύ της εφαρμογής και του χρήστη.
<code>response_utils.py</code>	Εκτελέσιμο που περιέχει τα ερωτήματα στη βάση και τις βοηθητικές συναρτήσεις για κάθε σενάριο διαλόγου του <code>response.py</code>
<code>smalltalk.py</code>	Εκτελέσιμο που διαλέγει τυχαίες εκφράσεις κουβέντας για να φαίνεται πιο “ανθρώπινος” ο τρόπος έκφρασης του chatbot.
<code>tag_generator.py</code>	Εκτελέσιμο το οποίο παράγει ετικέτες κόμβους που χαρακτηρίζουν τους κόμβους <code>issues</code> της βάσης. Προκύπτουν με NLP μεθόδους οι οποίες εφαρμόζονται πάνω στην περιγραφή του <code>issue</code> .

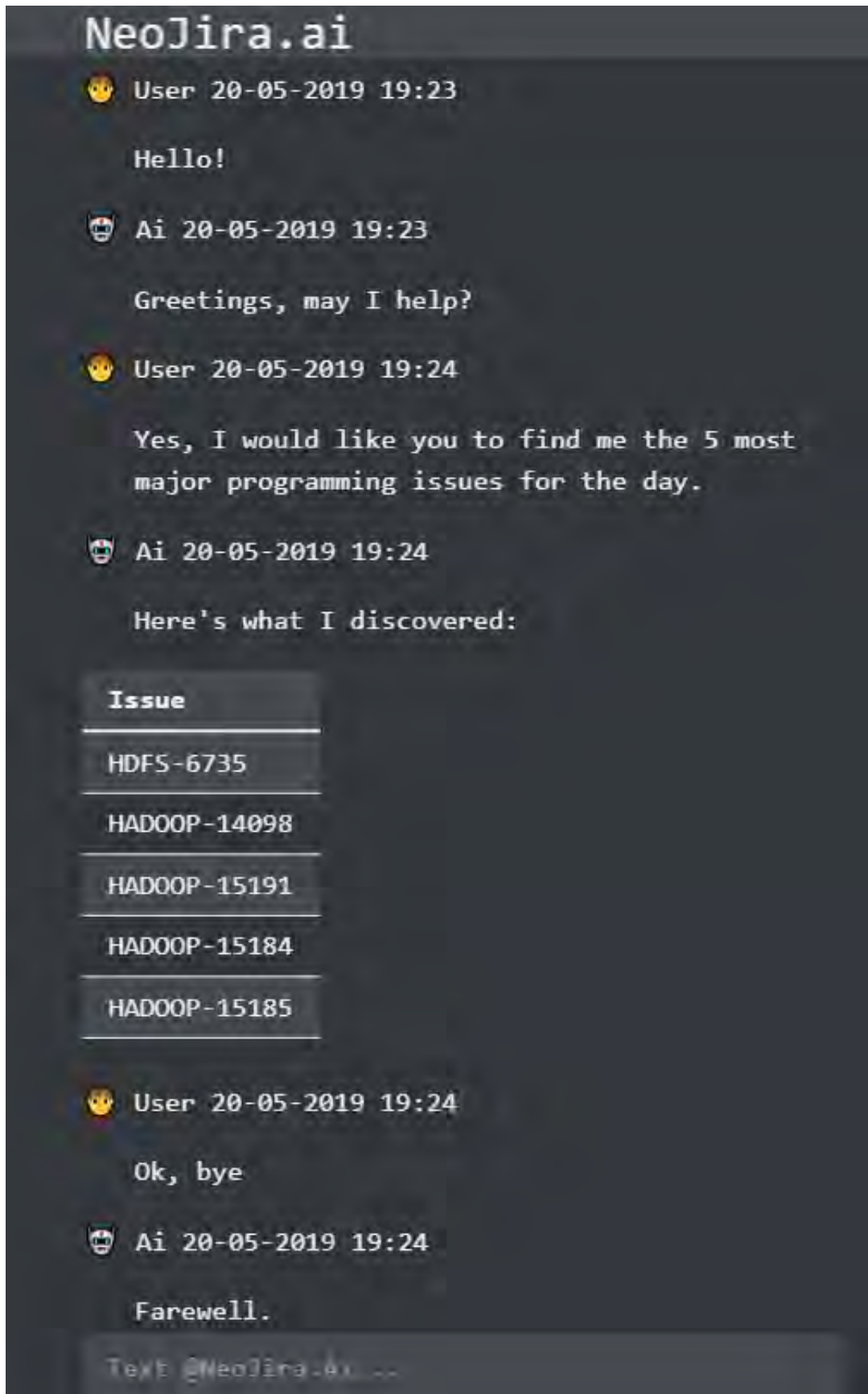
Πίνακας 4.2: Περιγραφή αρχείων / φακέλων εφαρμογής.

4.6 Γραφική Διεπαφή εφαρμογής

Με την χρήση της βιβλιοθήκης `bootstrap`, η γραφική διεπαφή της εφαρμογής προσαρμόζεται σε διάφορων ειδών οθόνες: π.χ. μεγάλη οθόνη (ανάλυση: 1920 x 1080) ή μεσαία οθόνη (ανάλυση: 1280 x 900) ή μικρή οθόνη (ανάλυση: 320 x 568).

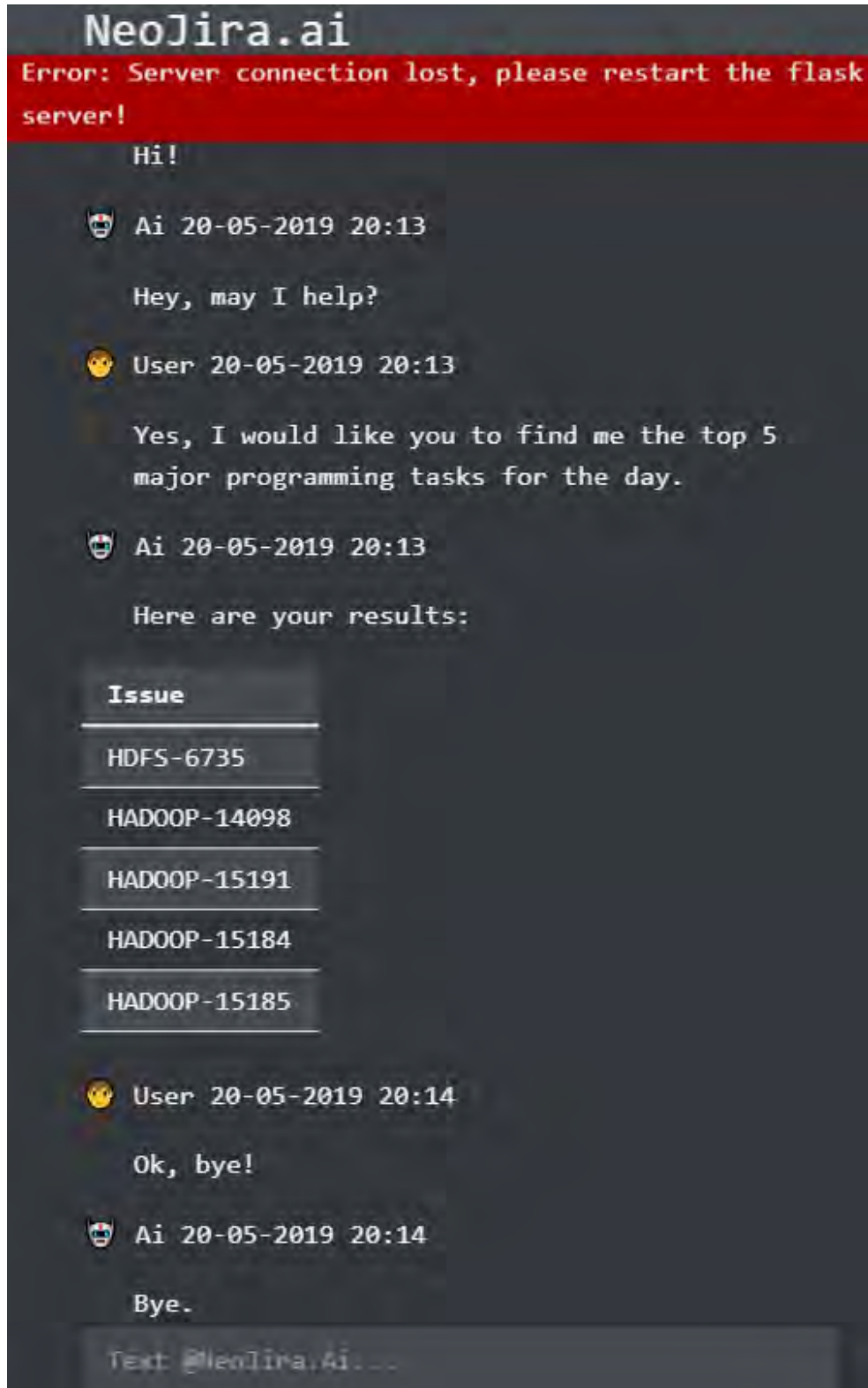
Με την είσοδό μας στο περιβάλλον της γραφικής διεπαφής βλέπουμε την οθόνη της Εικόνας 4.20 (χωρίς τα μηνύματα). Στην κορυφή είναι η κύρια μπάρα της εφαρμογής που φέρει το όνομα της εφαρμογής. Στην βάση είναι η περιοχή που κλικάρουμε και πληκτρολογούμε το μηνυμά μας και πατάμε `Enter` για να το στείλουμε στο `Neojira.ai` για να μας απαντήσει βάσει της γνώσης που εξάγει από την βάση.

Ανεξαρτήτως του ποσοστού που θα κάνει `scroll` ο χρήστης για να βλέπει την συζήτηση αυτές οι δύο περιοχές της εφαρμογής θα μένουν πάντα στις παραπάνω θέσεις για την διευκόλυνση του χρήστη.

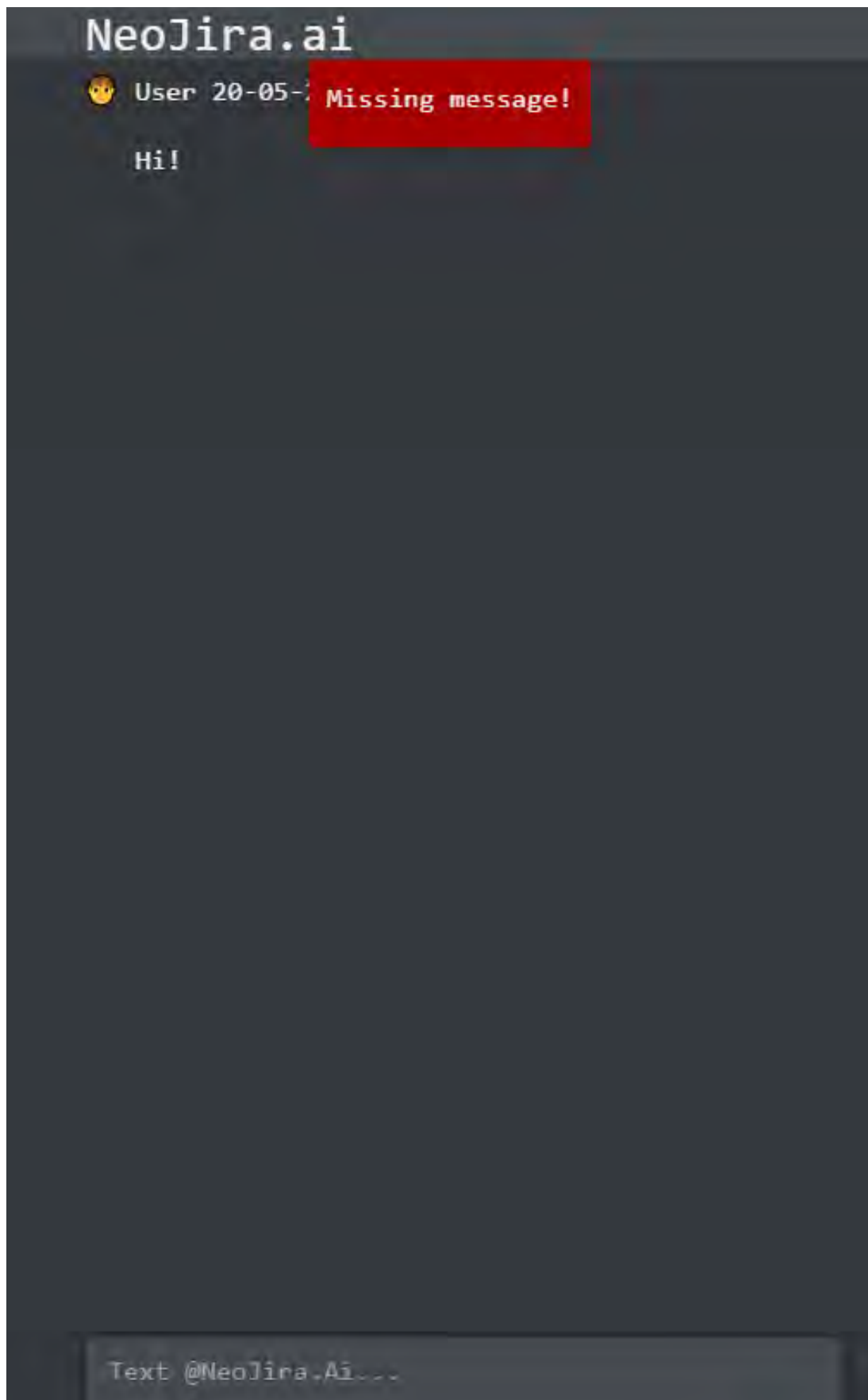


Εικόνα 4.20: Κύρια οθόνη γραφικής διεπαφής – Διάλογος με το chatbot.

Σε περίπτωση που γίνει κάποιος λάθος στην επικοινωνία της γραφικής διεπαφής με τον server π.χ. χαθεί το μήνυμα ή ο server να τερματίσει απότομα την λειτουργία του θα λάβουμε 1 εκ των 2 ακόλουθων μηνυμάτων λάθους στην κορυφή της εφαρμογής.



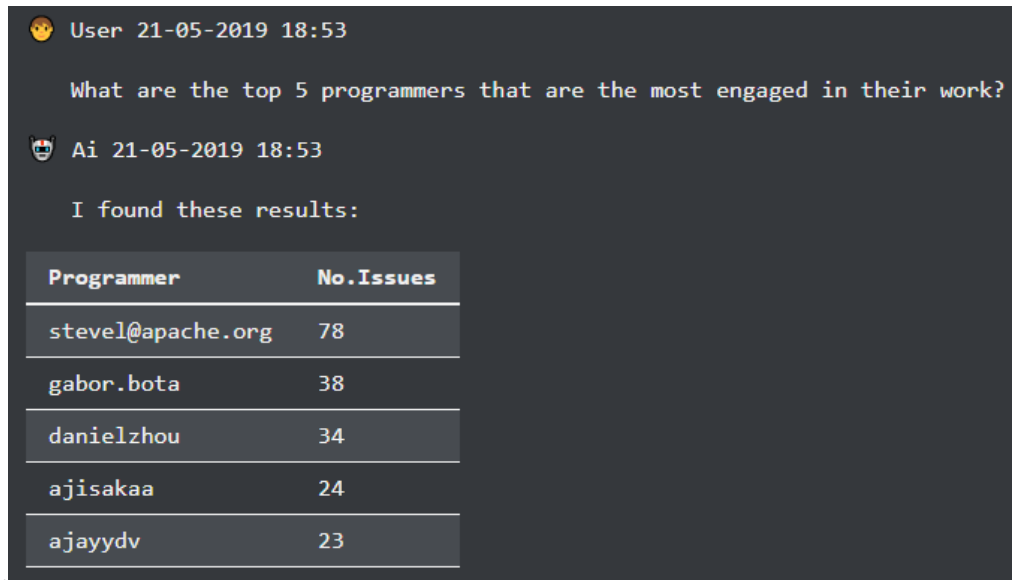
Εικόνα 4.21: Κύρια οθόνη γραφικής διεπαφής – Σφάλμα σύνδεσης με τον server.



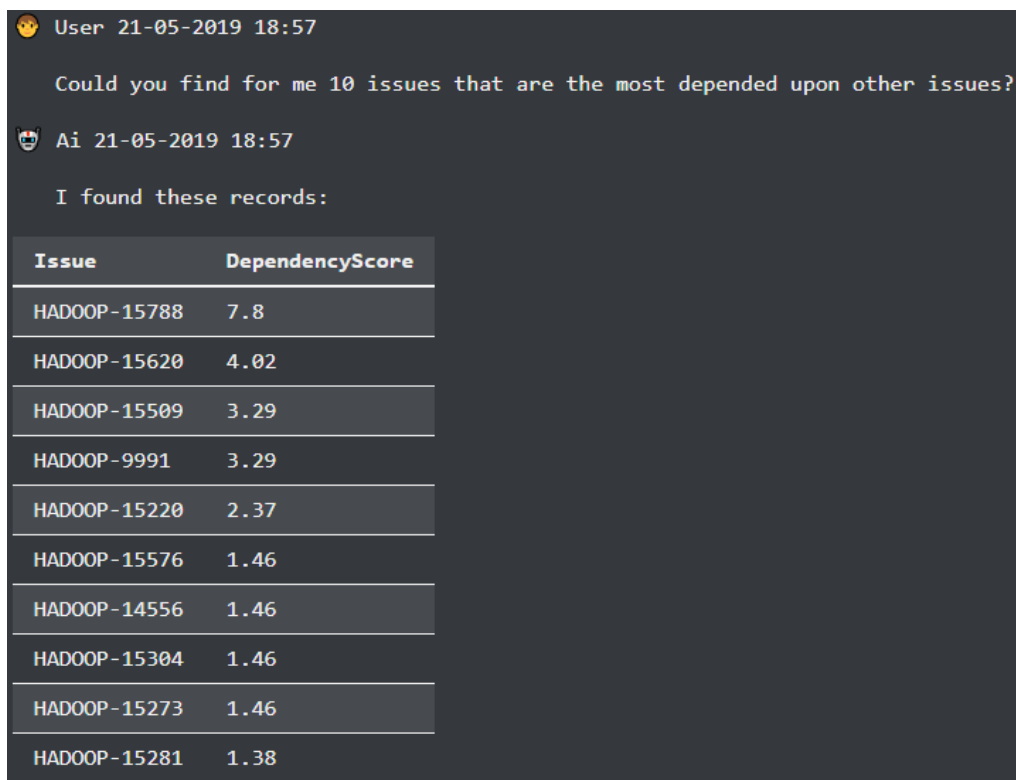
Εικόνα 4.22: Κύρια οθόνη γραφικής διεπαφής – Σφάλμα αποστολής μηνύματος στον server.

4.7 Παραδείγματα χρήσης

Στην υποενότητα αυτή παραθέτουμε τρία παραδείγματα που αφορούν απλά queries του χρήστη καθώς και δύο user stories που αφορούν σε συχνά προβλήματα στο χώρο του project management. Τα queries φαίνονται στις Εικόνες 4.23, 4.24 και 4.25.



Εικόνα 4.23: Αποτελέσματα εύρεσης προγραμματιστών με την μεγαλύτερη δραστηριότητα.



Εικόνα 4.24: Αποτελέσματα εύρεσης των πιο εξαρτημένων έργων του συνολικού έργου.

Query 3: Εύρεση του πλήθους και τύπου των έργων που έχει ανατεθεί στον κάθε προγραμματιστή.

👤 User 21-05-2019 19:19

Please find me the type and amount of issues for every employee.

🤖 Ai 21-05-2019 19:19

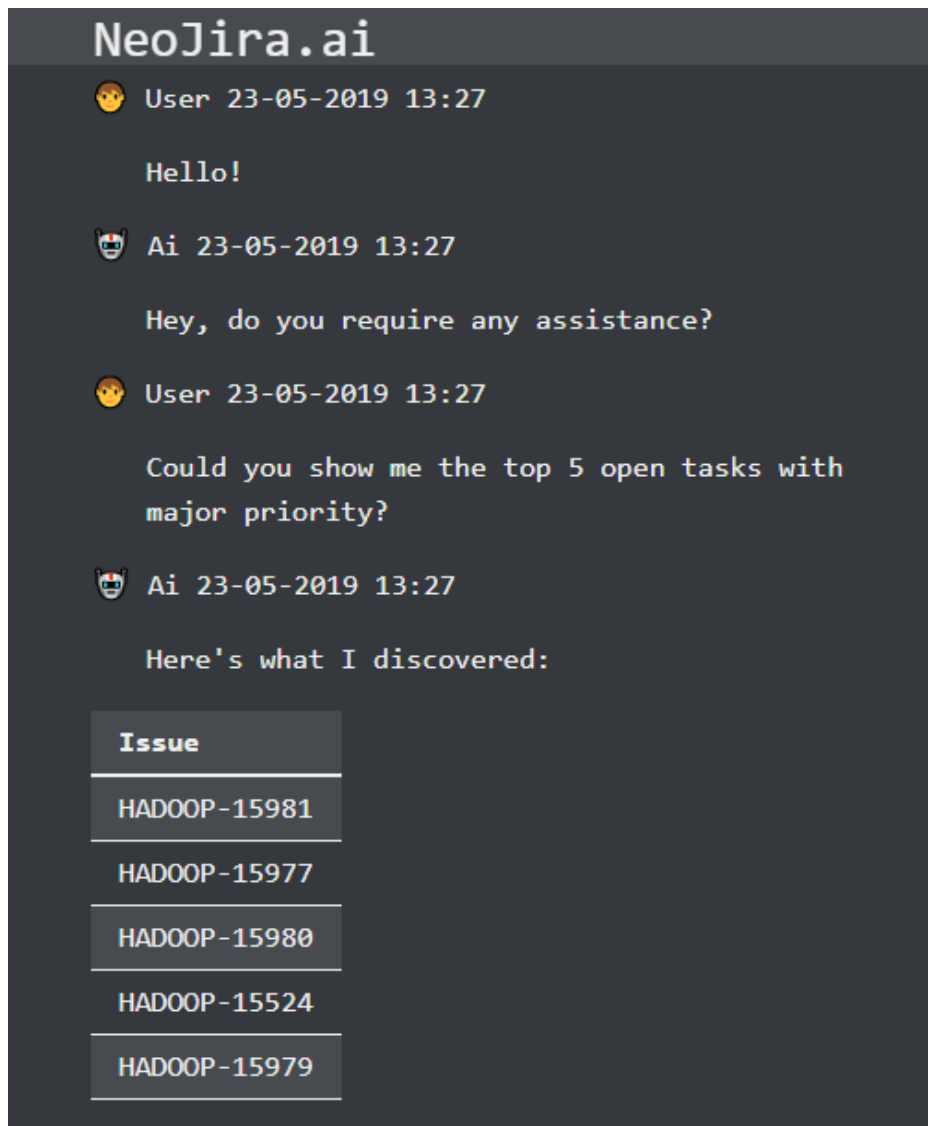
I found these results:

Programmer	No.IssueTypes
ghuangups	[['Open', 3]]
pnpie	[['Resolved', 2]]
daryn	[['Resolved', 6], ['Open', 16], ['Patch Available', 9]]
tlevine	[['Resolved', 3]]
kasha	[['Open', 1]]
jack-lee	[['Resolved', 2], ['Patch Available', 3], ['Open', 4], ['In Progress', 1]]
jesmith3	[['Open', 2]]
ajisakaa	[['Resolved', 91], ['Patch Available', 3], ['Open', 24]]
fengyongshe	[['Patch Available', 2]]
danielzhou	[['Resolved', 55], ['Closed', 6], ['Open', 4], ['Reopened', 1], ['Patch Available', 10]]

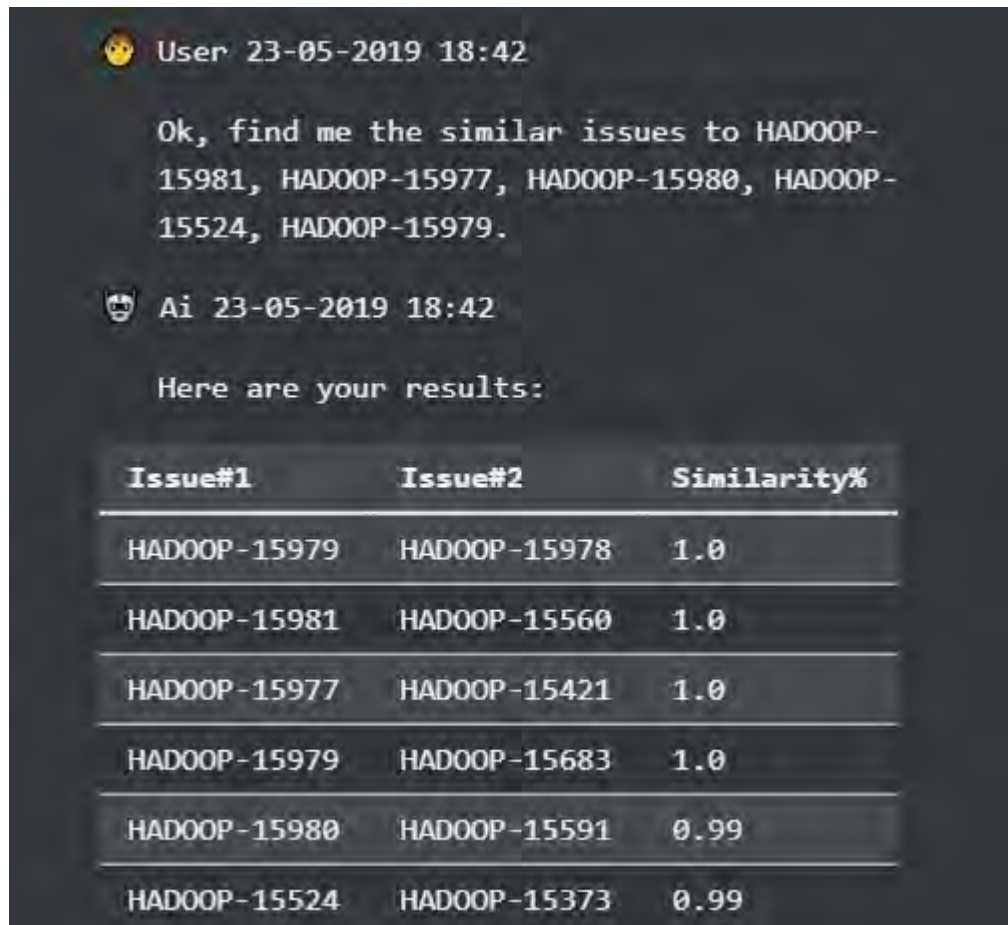
Εικόνα 4.25: Μερική λίστα – παραγόμενων αποτελεσμάτων του bot.

User story 1: Δυσκολία στην οργάνωση του έργου

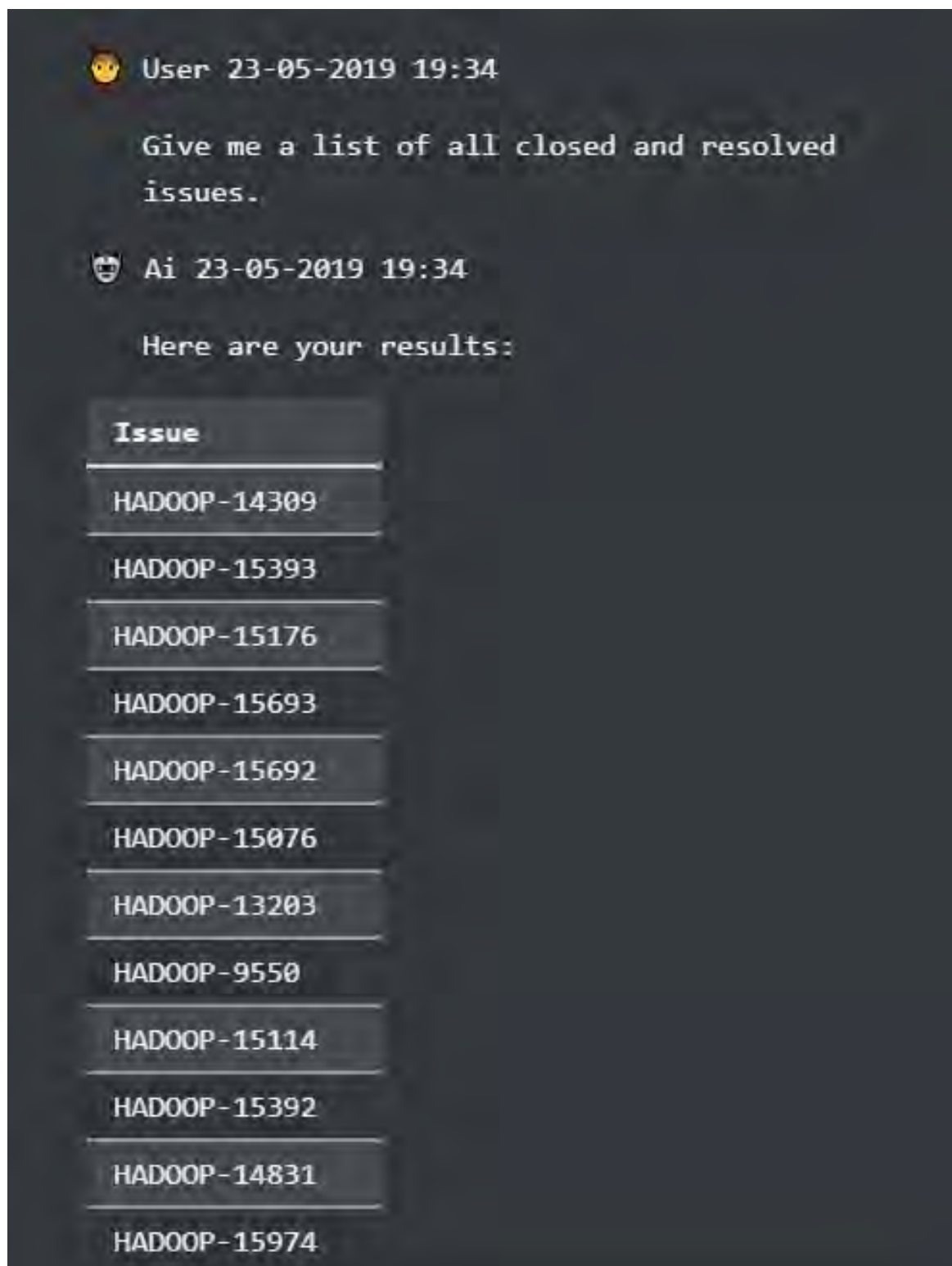
Έστω ότι ένα project βρίσκεται σε ένα κρίσιμο σημείο του έργου, στο οποίο μπορεί να καθυστερήσει. Ο Project Manager, αναζητά να βρει τα ανοιχτά υποέργα που έχουν την μέγιστη προτεραιότητα τα οποία πρέπει να επιλυθούν για να προχωρήσει το έργο. Μόλις βρίσκει τα εν λόγω υποέργα διαπιστώνει ότι υπάρχει μια δυσκολία στην ενσωμάτωση τους με το υπόλοιπο έργο για αυτό άλλωστε παρέμειναν ανοιχτά όλο αυτό το διάστημα. Σε μια προσπάθεια επίλυσης τους ο Project Manager βρίσκει παρόμοια υποέργα του ίδιου project ή άλλων περασμένων project για να δει αν μπορεί να αναπαράγει τις μεθόδους που ακολουθήθηκαν στα τελευταία με σκοπό να κλείσουν και αυτά. Μια βασική προϋπόθεση είναι ότι τα παρόμοια υποέργα θα πρέπει να έχουν κλείσει, κάτι το οποίο θα ελέγξει ο project manager, πριν προχωρήσει στο να διαβάσει την περιγραφή των εν λόγω υποέργων. Στην προσπάθεια του να δει αν το πρόβλημα οφείλεται σε κακή ανάθεση προσωπικού, ρωτάει επίσης το bot σχετικά με τα άτομα που δούλεψαν στα ήδη επιλυθέντα έργα και τα συγκρίνει με τα άτομα που έχει αναθέσει στα ανοιχτά έργα.



Εικόνα 4.26: User story 1 – Διάλογος με το bot μέρος 1 / 3.



Εικόνα 4.27: User story 1 – Διάλογος με το bot μέρος 2 / 3.

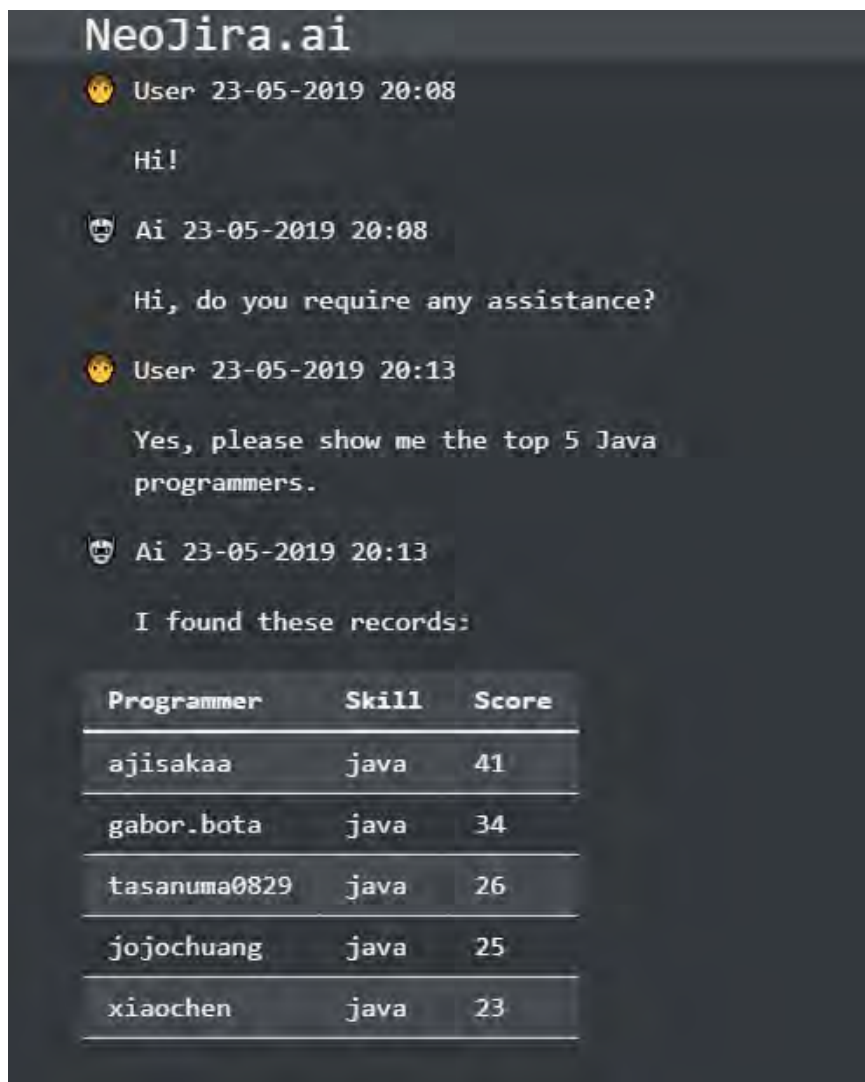


Εικόνα 4.28: User story 1 – Μερική λίστα αποτελεσμάτων – Διάλογος με το bot μέρος 3 / 3.

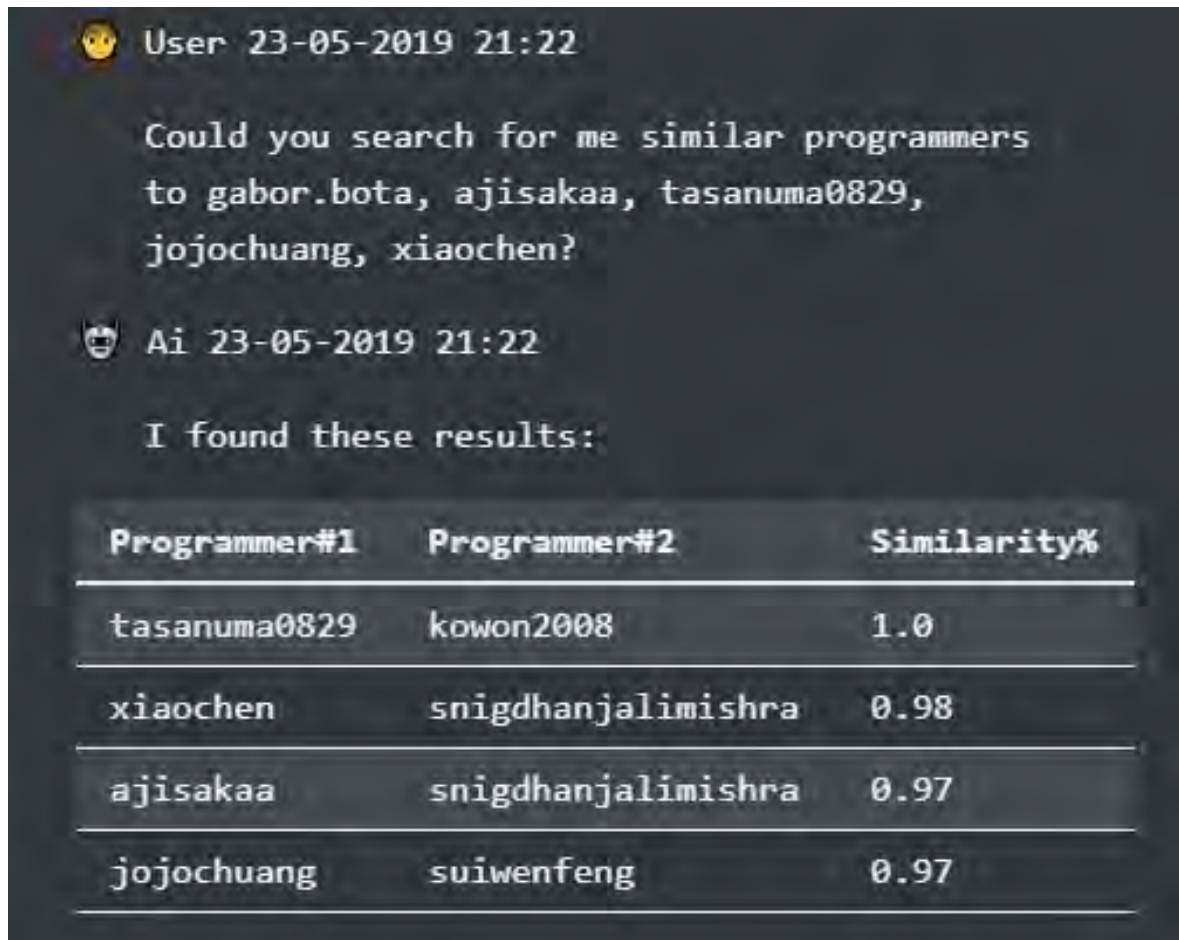
Πράγματι αν εξαιρέσουμε τα issues HADOOP-15421 και HADOOP-15978 από τη λίστα αποτελεσμάτων της Εικόνας 4.27, τα υπόλοιπα παρόμοια issues που βρήκαμε είναι είτε κλειστά είτε επιλυθέντα. Οπότε ο project manager μπορεί να διαβάσει την περιγραφή τους και να ακολουθήσει μια παρόμοια μεθοδολογία που θα επιλύσει τα ανοιχτά issues.

User story 2: Bonus project

Έστω ότι τον τελευταίο χρόνο οι εργαζόμενοι της εταιρίας εργάζονται σε ένα κρίσιμο project σε Java, στο οποίο βρίσκονται οι περισσότεροι προγραμματιστές της εταιρίας που έχουν εξοικείωση με τη γλώσσα αυτή. Απροσδόκητα όμως, ένας καινούριος πελάτης της εταιρίας ζητάει την υλοποίηση ενός μικρότερου project έναντι μιας σχετικά μεγάλης αμοιβής σε σχέση με το project αυτό, το οποίο εκτιμάται από την εταιρία ότι έχει χρόνο περάτωσης το πολύ 2 μηνών με χρήση μιας ομάδας των 4 ατόμων. Το πρόβλημα όμως είναι ότι πέρα από ένα από τους κορυφαίους προγραμματιστές Java της εταιρίας γενικά όλοι οι υπόλοιποι εργαζόμενοι που έχουν εξοικείωση σε Java είναι δεσμευμένοι στο μεγάλο project. Για αυτό το λόγο ο Project Manager αρχικά ρωτάει το bot να βρει τους προγραμματιστές Java της εταιρίας ταξινομημένους σε φθίνουσα σειρά ικανότητας για να δει πόσοι από αυτούς δεν έχουν ανατεθεί στο μεγάλο project, κατόπιν αναζητά παρόμοιους προγραμματιστές που ο τρόπος εργασίας τους είναι παρόμοιος με αυτούς (έμπειροι προγραμματιστές), και στη συνέχεια με την βοήθεια του κορυφαίου προγραμματιστή τους κάνει σύντομα σεμινάρια σε Java και σχηματίζουν μια ομάδα για να υλοποιήσουν το μικρό project.



Εικόνα 4.29: User story 2 – Διάλογος με το bot μέρος 1 / 3.



Εικόνα 4.30: User story 2 – Διάλογος με το bot μέρος 2 / 3.

👤 User 23-05-2019 21:34

Please find for me the number of tasks for all programmers grouped by task type.

🤖 Ai 23-05-2019 21:34

I found these records:

Programmer	No. IssueTypes
jojochuang	[['Resolved', 33], ['Open', 17], ['Patch Available', 2]]
gabor.bota	[['Resolved', 48], ['Patch Available', 11], ['Open', 14], ['In Progress', 3]]
xiaochen	[['Resolved', 38], ['Open', 5], ['Patch Available', 2]]
ajisakaa	[['Resolved', 91], ['Patch Available', 3], ['Open', 24]]
tasanuma0829	[['Resolved', 55]]

Εικόνα 4.31: User story 2 – Πίνακας αποτελεσμάτων – Διάλογος με το bot μέρος 3 / 3.

Από την Εικόνα 4.31, ο Project Manager διαπιστώνει ότι ο `tasanuma0829` είναι ο μόνος έμπειρος προγραμματιστής που δεν βρίσκεται σε ανοιχτά issues και βάσει της λίστας ομοιότητας της Εικόνας 4.30 οι 3 προγραμματιστές που θα συμμετέχουν στο σεμινάριο Java με σκοπό να ενσωματωθούν στην ομάδα των 4, που θα υλοποιήσει το μικρό project, είναι οι `kowon2008`, `snigdhanjalimishra` και `suiwenfeng`.

5

Σύνοψη – Συμπεράσματα

Στα πλαίσια της παρούσας εργασίας, μελετήσαμε έννοιες της διαχείρισης έργου και δη λογισμικού, και σύγχρονες μεθοδολογίες στην διαχείριση έργου καθώς και δημοφιλή πληροφοριακά συστήματα διαχείρισης έργου. Στη συνέχεια μελετήσαμε γραφικές βάσεις δεδομένων και εξετάσαμε τα σαφή πλεονεκτήματα τους έναντι των παραδοσιακών βάσεων δεδομένων στις σύγχρονες εφαρμογές που απαιτούν μεγάλο όγκο ισχυρά διασυνδεδεμένων δεδομένων. Κατόπιν αναπτύξαμε κώδικα python για την διασύνδεση με την βάση, φορτώνοντας την με ένα σετ δεδομένων από το Jira, ένα από τα γνωστά state-of-the-art πληροφοριακά συστήματα διαχείρισης έργου. Αναπτύξαμε αποδοτικά ερωτήματα σε Cypher, τα οποία απαντούν σε κλασσικά ερωτήματα διαχείρισης έργου. Μελετήσαμε επίσης γνωστά κριτήρια ομοιότητας για κατηγοριοποίηση κειμένου όπως τα μέτρα Jaccard, Cosine, Pearson και ευκλείδεια απόσταση, τα οποία και ενσωματώσαμε σε ερωτήματα cypher. Τέλος, μελετήσαμε τα chatbots από το παρελθόν μέχρι σήμερα, αναδείξαμε use cases τους στην διαχείριση έργου και αναλύσαμε την πλατφόρμα του wit.ai που χρησιμοποιήσαμε για να φτιάξουμε ένα conversational chatbot σε Python, το οποίο καλεί τα ερωτήματα σε cypher που αναπτύξαμε για το Project Management και επιστρέφει τα αποτελέσματα στον χρήστη.

Στην εργασία εντοπίσαμε κάποιους περιορισμούς. Αρχικά, στα ερωτήματα που γράψαμε σε Cypher υπήρξαν κάποια συνήθη προγραμματιστικά ιδιώματα [67] (σε άλλες γλώσσες προγραμματισμού), τα οποία θα ήταν καλύτερο να αυτοματοποιούνται με την κλήση μιας ενσωματωμένης συνάρτησης έναντι του να υλοποιούνται χειροκίνητα από τον προγραμματιστή. Αυτό αξίζει να υλοποιηθεί από τους προγραμματιστές του Neo4j σε κάποια μελλοντική έκδοση για λόγους απόδοσης και ευκολότερης αναγνωσιμότητας των ερωτημάτων. Στη συνέχεια, υπήρξαν κάποιοι περιορισμοί με την πλατφόρμα του wit.ai. Επειδή είναι μια πλατφόρμα κλειστού λογισμικού, δεν μπορούσαμε να επέμβουμε στο μοντέλο τεχνητής νοημοσύνης που αναγνωρίζει τις προθέσεις του χρήστη βάσει του κειμένου, είτε για να το αλλάξουμε σε ένα μοντέλο που εξυπηρετεί τις ειδικές ανάγκες της δικής μας εφαρμογής είτε για να προσθέσουμε εξαιρέσεις στην αναγνώριση προθέσεων, (π.χ. φράσεις οι οποίες σε συγκεκριμένα συμφραζόμενα πρέπει να αναγνωριστούν με διαφορετική πρόθεση).

Ένας άλλος περιορισμός της πλατφόρμας του wit.ai είναι ότι αξιολογεί μια πρόταση την φορά, χωρίς να ανακαλεί τα συμφραζόμενα των περασμένων προτάσεων. Ανήκει δηλαδή στην κατηγορία πλατφορμών για chatbots που λέγονται stateless ή memoryless (δεν θυμούνται). Σε διαλόγους που εξετάζουμε στα πλαίσια αυτής της εργασίας αυτό το πρόβλημα δεν είναι εμφανές. Σε μια εφαρμογή όμως που απαιτεί μεγαλύτερους διαλόγους με τον χρήστη (με περισσότερα παρελθοντικά συμφραζόμενα) το πρόβλημα είναι σημαντικό. Για αυτό, για τέτοιου είδους εφαρμογές που απαιτείται μνήμη, προτείνεται η χρήση πλατφορμών που υποστηρίζουν τα λεγόμενα user stories, οι οποίες είναι περιγραφές διαλόγου σχετικά με τι μπορεί να ρωτήσει ο χρήστης και όλες τις διαφορετικές απαντήσεις που θα μπορούσε να πάρει. Τα user stories, όπως υλοποιούνται σε γνωστές πλατφόρμες, υποστηρίζουν και την αποθήκευση στοιχείων από τα διάφορα βήματα του διαλόγου που μπορεί να ζητήσει ο χρήστης.

Σαν μελλοντική επέκταση της εργασίας, προτείνουμε την σύγκριση και ενσωμάτωση εναλλακτικών μέτρων ομοιότητας. Εμείς αναπτύξαμε μια υλοποίηση που συνδυάζει σε μια ενιαία λίστα τα αποτελέσματα των μέτρων ομοιότητας Jaccard & Cosine, αλλά υπάρχουν πολλά άλλα μέτρα ομοιότητας που θα μπορούσαν να ενσωματωθούν. Το chatbot θα μπορούσε να εμπλουτιστεί και με άλλα πιο σύνθετα ερωτήματα cypher (για project management), τα οποία μπορούν να ενσωματωθούν στην υπάρχουσα δομή διαλόγου στο αρχείο response.py. Επίσης, χάριν της χρήσης της πλατφόρμας wit.ai, το bot θα μπορούσε να αποκτήσει ικανότητες φωνητικής αναγνώρισης και απάντησης καθώς και να ενσωματωθεί σε γνωστές πλατφόρμες όπως το Messenger ή το Slack που χρησιμοποιείται κατά κόρον στην ενδοεταιρική επικοινωνία, προσφέροντας ουσιαστικά μία ενιαία λύση, μέσω της οποίας συνομιλούν μεταξύ τους οι εργαζόμενοι, αλλά και οι εργαζόμενοι με το σύστημα.

Συμπερασματικά, μπορούμε να πούμε ότι όσο μεγαλώνουν τα projects και ο αριθμός των εργαζόμενων, και όσο δυσχεραίνεται γενικότερα το project management, τόσο πιο απαραίτητη γίνεται η χρήση των Graph Databases για λόγους οπτικοποίησης των δεδομένων και παραγωγής γνώσης από αυτήν, αλλά και η χρήση Chatbots τα οποία απομονώνουν τον “θόρυβο” του μεγάλου όγκου πληροφοριών και επιστρέφουν στον χρήστη αυτό ακριβώς που θέλει να μάθει.

Βιβλιογραφία - Αναφορές

- [1] [A Comparison of a Graph Database and a Relational Database: a data provenance](#)
–C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, D. Wilkins
- [2] [Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j](#)
–F. Holzschuher, R. Peinl
- [3] [More Connections, More Value: The Return on Connected Data](#)
–J. Morris
- [4] [An Overview of Graph Algorithms in Neo4j](#)
–A.E. Hodler
- [5] [Overcoming SQL Strain and SQL Pain](#) – Neo4j
- [6] [Why Graph Databases?](#) – Neo4j
- [7] [Principles of transaction-oriented database recovery](#)
–T. Haerder, A. Reuter
- [8] Goodrich, Michael T.; Tamassia, Roberto (2015), "Section 13.1: Graph terminology and representations", Algorithm Design and Applications, Wiley, pp. 355–364.
- [9] [Neo4j Product](#)
- [10] <https://s3.amazonaws.com/dev.assets.neo4j.com/wp-content/uploads/20171212132649/graph-platform-1600x800.jpg>
- [11] <https://s3.amazonaws.com/dev.assets.neo4j.com/wp-content/uploads/20170731095054/Property-Graph-Concepts-Simple.svg>
- [12] <https://image.slidesharecdn.com/larnpvsrv1ldpotzjmjw-signature-97ac0cba55d83fa6f2a2f253a387270677bf11da9177d0b7d555fb911fc34ba2-poli-160526084212/95/managing-connected-big-data-in-art-with-neo4j-graph-database-lorenzo-speranzoni-codemotion-amsterdam-2016-52-638.jpg?cb=1464259832>
- [13] [Similarity Measures for Text Document Clustering](#)
–A. Huang
- [14] [What are the key challenges in Project Management?](#)
- [15] [ISO 3100: Risk Management](#)
- [16] [What does Chatbot mean?](#)

- [17] [Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants](#)
– Matthew B. Hoy
- [18] [Computing Machinery and Intelligence. Mind 49: 433-460.](#)
– A. M. Turing
- [19] [CHATTERBOTS, TINYMUDS, and the Turing Test
Entering the Loebner Prize Competition](#)
– M. L. Mauldin
- [20] [Getting A Computer To Write About Itself](#)
- [21] [The Anatomy of A.L.I.C.E.](#)
– R. S. Wallace
- [22] [Is Watson the smartest machine on earth?](#)
- [23] [This is Watson](#)
- [24] [Building Watson: An Overview of the DeepQA Project](#)
–D. Ferruchy, et al.
- [25] [Cleverbot Data for Machine Learning](#)
- [26] [What is the State-of-the-art in Chatbots](#)
- [27] [Industry Watch - The return of the chatbots](#)
–R. Dale
- [28] [Chatbot for technical support](#)
- [29] [10 Best Chatbot Examples to Improve Customer Service in 2019](#)
- [30] [Gagner Du Temps Avec Le Chatbot Bancaire Pour Gagner En Intelligence Avec Les Conseillers](#)
- [31] [Avaamo AI Healthcare](#)
- [32] [9 Most Innovative Chatbot Examples in 2019 from Top Brands](#)
- [33] <https://quartsoft.com/sites/default/files/project-success-failure-rate-survey.jpg>
- [34] <https://blog.capterra.com/wp-content/uploads/2014/07/project-managemetn-2-720x223.jpg>
- [35] [Critical Chain Project Management, Third Edition, p.30](#)
–L. P. Leach
- [36] [3 Small Business Use Cases of AI Chatbots in Project Management](#)
- [37] [wit.ai FAQ](#)
- [38] [wit.ai Quickstart](#)

- [39] https://wit.ai/docs/images/recipes/pr_curves-b311119f.png
- [40] https://cdn-images-1.medium.com/max/2600/1*HiI8h-RlhV8mNu5gn00vew.jpeg
- [41] <https://www.techcoil.com/blog/wp-content/uploads/Python-3.7.1-64-bit-Windows-Setup-first-page.jpg>
- [42] <https://www.techcoil.com/blog/wp-content/uploads/Python-3.7.1-64-bit-Windows-setup-Optional-Features-screen.jpg>
- [43] <https://www.techcoil.com/blog/wp-content/uploads/Python-3.7.1-64-bit-Windows-setup-Advanced-Options-screen.jpg>
- [44] <https://www.techcoil.com/blog/wp-content/uploads/Python-3.7.1-64-bit-Windows-setup-progress.jpg>
- [45] <https://www.techcoil.com/blog/wp-content/uploads/Python-3.7.1-64-bit-Windows-setup-successful-screen.jpg>
- [46] <https://s3.amazonaws.com/dev.assets.neo4j.com/wp-content/uploads/20171030144149/oAuth.jpg>
- [47] https://s3.amazonaws.com/dev.assets.neo4j.com/wp-content/uploads/20180301115235/new_graph.jpg
- [48] https://s3.amazonaws.com/dev.assets.neo4j.com/wp-content/uploads/20180301115550/local_or_remote1.jpg
- [49] <https://s3.amazonaws.com/dev.assets.neo4j.com/wp-content/uploads/20180301120933/create1.jpg>
- [50] <https://s3.amazonaws.com/dev.assets.neo4j.com/wp-content/uploads/20180207153704/start.jpg>
- [51] Μάνατζμεντ / Αρχές Διοίκησης Επιχειρήσεων σελ. 30
–Α.Σ. Χυτήρης
- [52] <https://d3tvpxjako9ywy.cloudfront.net/blog/content/uploads/2016/02/wrike-custom-statuses-dashboard.png?av=c63a6ce1a89e6fca280adf1d3efd3f7f>
- [53] <https://www.wrike.com/>
- [54] <https://monday.com/>
- [55] <https://financesonline.com/uploads/2018/11/vp862d3a5ed1d9346b9bdf4edee38ca03a.png>
- [56] <https://confluence.spartez.com/download/attachments/20547484/enable-estimates.png?version=2&modificationDate=1429091950093&api=v2>
- [57] <https://confluence.atlassian.com/jirasoftwarecloud/files/946023490/946023491/1/1519281781266/Scrum+board.png>
- [58] https://a.slack-edge.com/73267c/marketing/img/downloads/screenshots/win10_screenshot.png

[59] A Guide to the Project Management Body of Knowledge (PMBOK Guide). (2013). 5th ed. Newtown Square, Pennsylvania: Project Management Institute, Inc, pp.6, 533.

[60] ["Kanban in software development: A systematic literature review," 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, Santander, 2013, pp. 9-16.](#)

–M. O. Ahmad, J. Markkula and M. Oivo,

[61] ["The Scrum software development process for small teams," in IEEE Software, vol. 17, no. 4, pp. 26-32, July-Aug. 2000.](#)

–L. Rising, N. S. Janoff

[62] <https://static.kanbantool.com/kanban+landing/kanban-board-examples/basic-team-kanban-board.png>

[63] <http://manifesto.co.uk/wp-content/uploads/2014/10/scrum-task-board.png>

[64] Ladas Corey, “Scrumban: Essays on Kanban Systems for Lean software development”, Seattle, WA, USA: Modus Cooperandi Press, 2009.

[65] ["From scrum to scrumban: A case study of a process transition," 2012 International Conference on Software and System Process \(ICSSP\), Zurich, 2012, pp. 140-149.](#)

–N. Nikitina, M. Kajko-Mattsson and M. Stråle

[66] Ramez Elmasri and Shamkant Navathe, pp.33, 2010. Fundamentals of Database Systems (6th ed.). Addison-Wesley Publishing Company, USA.

[67] [Pattern-Oriented Software Architecture, Volume 1 Chapter 4, A System of Patterns by Michael Stal, Peter Sommerlad, Hans Rohnert, Regine Meunier, Frank Buschmann](#)

[68] Graph databases, " O'Reilly Media, Inc.", pp23, 2013.

–I. Robinson, J. Webber, E.Eifrem