ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Συστημάτων Μετάδοσης Πληροφορίας και Τεχνολογίας Υλικών

Δημιουργία Αλληλεπιδραστικού Γραφικού Περιβάλλοντος για την οπτική αναπαράσταση των συσχετίσεων μεταξύ οντολογιών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αθανάσιου Χ. Αποστόλου

Επιβλέπων: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Οκτώβρης 2020

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Συστημάτων Μετάδοσης Πληροφορίας και Τεχνολογίας Υλικών

Δημιουργία Αλληλεπιδραστικού Γραφικού Περιβάλλοντος για την οπτική αναπαράσταση των συσχετίσεων μεταξύ οντολογιών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αθανάσιου Χ. Αποστόλου

Επιβλέπων: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Αυγούστου 2020.

…….	……..	……..
Θεοδώρα Βαρβαρίγου	Εμμανουήλ Βαρβαρίγος	Συμεών Παπαβασιλείου
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβρης 2020

……..

Αθανάσιος Χ. Αποστόλου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

δφασδ

γφδς

δφαδφασδφασδσαααααααααααααααααααααααααααααααααααααααααααααααααααααααααα ααααααααααααα

Λέξεις Κλειδιά

Abstract

δφσαδφ

φσδαφ

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε από τον Μάρτιο εώς τον Οκτώβρη του 2020 για την ολοκλήρωση των σπουδών μου στην Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Ο υπεύθυνος τομέας είναι ο Τομέας Επικοινωνιών, Ηλεκτρονικής & Συστημάτων Πληροφορικής υπό την επίβλεψη της καθηγήτριας Θεοδώρας Βρβαρίγου.

Εξαιρετική βοήθεια προήλθε από τον Ευθύμιο Χονδρογιάννη, ο οποίος μου είχε αναθέσει το θέμα της πτυχιακής και ήταν πάντα κατατοπιστικός σε οποιαδήποτε σύνθετα ζητήματα συνάντησα καθ’ όλη την διάρκεια και τον ευχαριστώ θερμά.

Κατά την διάρκεια εκπόνησης της διπλωματικής εντρύφησα στις τεχνολογίες του Σημασιολογικού Ιστού και κατανόησα σε βάθος την έννοια της Οντολογίας. Απέκτησα διάφορες γνώσεις για επιστημονικά θέματα που αφορούν την επεξεργασία φαρμακευτικών και κλινικών δεδομένων. Δούλεψα πάνω σε πραγματικά δεδομένα έχοντας την ευκαιρία να συναντήσω ρεαλιστικά προβλήματα και να καταφέρω να τα επιλύσω. Επιπροσθέτως βελτίωσα και εξειδίκευσα τις γνώσεις μου που αφορούν την δημιουργία πραγματικών εφαρμογών τόσο για frontend γραφικών διεπαφών χρηστών, όσο και για backend/servers. Οι γνώσεις αυτές θα συνεισφέρουν θετικά στην αργότερα επαγγελματική μου πορεία.

Πίνακας Περιεχομένων

Ευρετήριο Σχημάτων

Ευρετήριο Πινάκων

1. Εισαγωγή

Στις μέρες μας η αποθήκευση και η επεξεργασία της κλινικής και φαρμακευτικής γνώσης αποτελεί ζήτημα με μεγάλο ενδιαφέρον. Η γνώση αυτή αποκτάται από διαφορετικούς οργανισμούς (κλινικές, νοσοκομεία, φαρμακεία, κτλ) και αποθηκεύεται σε διαφορετικά σχήματα και δομές όπως απλά αρχεία ή διάφορες βάσεις δεδομένων. Ωστόσο πέρα από την αποθηκευμένη πληροφορία αυτή καθ' αυτή, μεγάλη σημασία έχει να καθορίσουμε τους όρους που χρησιμοποιούνται για να περιγράψουν αυτή την πληροφορία. Για αυτόν τον λόγο χρησιμοποιούμε Οντολογίες οι οποίες θα εξηγηθούν αναλυτικά στα επόμενα κεφάλαια.

Στην περίπτωση της διπλωματικής μας μελετάμε τέτοια δεδομένα που αφορούν φάρμακα και ασθένειες σε μορφή Οντολογιών που είναι αποθηκευμένα σε αρχεία γλώσσας OWL (Web Ontology Language). Αυτά τα αρχεία περιέχουν πολλές ίδιες έννοιες (π.χ. κάποιο συγκεκριμένο φάρμακο) οι οποίες όμως απεικονίζονται με διαφορετικά μοντέλα σε κάθε περίπτωση. Ο καθορισμός των συσχετίσεων μεταξύ τέτοιων διαφορετικών μοντέλων έχει ιδιαίτερη σημασία, καθώς μπορεί να μας οδηγήσει στην λύση διάφορων προβλημάτων. Τέτοια προβλήματα είναι η ενοποίηση δεδομένων και η απάντηση ερωτημάτων παρά την όποια διαφορετική φύση των μοντέλων μας.

Για τον καθορισμό των συσχετίσεων μεταξύ των οντολογιών υπάρχει αρκετή δουλεία στην βιβλιογραφία όσον αφορά τον εντοπισμό των συσχετίσεων μεταξύ των όρων τον οντολογιών αλλά και την έκφραση των κανόνων σε μια μορφή που είναι κατανοητή από τον υπολογιστή. Στην παρούσα διπλωματική εργασία βασιζόμαστε στο εργαλείο *Ontology Alignment Tool* [1] το οποίο πραγματοποιεί αυτοματοποιημένη παραγωγή κανόνων συσχετισμών μεταξύ οντολογιών. Το εργαλείο αυτό θα αναλυθεί στα επόμενα κεφάλαια και θα εξηγηθεί ο τρόπος που προκύπτουν οι κανόνες καθώς και η μορφή τους.

Σκοπός της εργασίας μας είναι η οπτική αναπαράσταση αυτών των κανόνων καθώς και η αναπαράσταση των βασικών όρων των Οντολογιών με τρόπο κατανοητό. Για αυτόν τον σκοπό θα δημιουργήσουμε μια διαδικτυακή εφαρμογή η οποία θα επεξεργάζεται δύο

Οντολογίες *owl* καθώς και τους κανόνες συσχέτισης τους οι οποίοι έχουν παραχθεί με το παραπάνω εργαλείο. Το αποτέλεσμα θα είναι σχηματικές αναπαραστάσεις των παραπάνω με δυνατότητα αλληλεπίδρασης του χρήστη για την πλήρη και εύκολη κατανόηση αυτών των συσχετίσεων. Αυτό θα μπορεί να οδηγήσει στην εύρεση λαθών του εργαλείου όπως για παράδειγμα η απουσία κανόνων που θα έπρεπε να υπάρχουν.

Στο κεφάλαιο 2 αναλύουμε τις βασικές θεωρητικές έννοιες που χρειάζονται για την κατανόηση της εργασίας μας καθώς και τις βασικές τεχνολογίες στις οποίες θα βασιστεί η εφαρμογή μας.

2. State of the Art

Εδώ θα αναλύσουμε τις βασικές έννοιες που χρησιμοποιούνται στην μελέτη μας καθώς και τις κύριες τεχνολογίες στις οποίες θα βασιστεί η εφαρμογή μας.

2.1. Διαδικτυακές Εφαρμογές

Θα εξηγήσουμε τις βασικές αρχές στις οποίες κατασκευάζονται οι διαδικτυακές εφαρμογές και θα αναλύσουμε σε λειτουργικό βαθμό τις τεχνολογίες που θα χρησιμοποιήσουμε.

2.1.1 Τεχνολογίες Ιστού και Javascript

Μια *ιστοσελίδα διαδικτύου (web page)* αποτελείται από αρχεία τα οποία έχουν μια συγκεκριμένη δομή ώστε να μπορούν οι *περιηγητές ιστού (web browsers)* να τα διαβάζουν για να τα αναπαραστήσουν στην οθόνη. Η λειτουργία αυτών των αρχείων περιγράφεται από την *προδιαγραφή html (html specification)* [2] . Αυτή η προδιαγραφεί ορίζει μια γενικευμένη γλώσσα για την περιγραφή των εγγράφων και των εφαρμογών, καθώς και κάποιες *προγραμματιστικές διεπαφές (api)* για την αλληλεπίδραση με την αναπαράσταση στην μνήμη των πόρων που χρησιμοποιεί αυτή η γλώσσα.

Γενικά ορίζονται δύο βασικές συντάξεις με τις οποίες μπορούν να γραφούν τέτοια αρχεία. Η πρώτη είναι η *HTML (HyperText Markup Language)* και η δεύτερη είναι η *XML (eXtensible Markup Language)*. Δεν θα μπούμε σε λεπτομέρειες περιγραφής τους αλλά και οι 2 γλώσσες που χρησιμοποιούνται για την σύνταξη τέτοιων αρχείων μπορούν να διαβαστούν από όλους τους σύγχρονους περιηγητές.

Η αναπαράσταση στην μνήμη των αντικειμένων που χρησιμοποιούνται στην ιστοσελίδα ή εφαρμογή ιστού μαζί με την προγραμματιστική διεπαφή που ορίζεται για να ελέγχουμε την κατάσταση αυτών των αντικειμένων είναι γνωστή ως *HTML DOM (Document Object Model)*.

Καταλαβαίνουμε ότι το DOM είναι δύο έννοιες:

- Είναι αρχικά ένα μοντέλο αντικειμένων για ένα έγγραφο html. Για κάθε σελίδα που φορτώνει ο περιηγητής φτιάχνει ένα αντικείμενο DOM. Το αντικείμενο αυτό αποτελείται από όλα τα *html στοιχεία (html elements)* οργανωμένα σε δενδρική δομή. Για κάθε τέτοιο στοιχείο υπάρχει η πληροφορία για τις *ιδιότητες (properties)*, τις *μεθόδους (methods)* και τα *γεγονότα (events)* που σχετίζονται με αυτό το στοιχείο.

- Είναι επίσης μια προγραμματιστική διεπαφή μέσω της οποίας μπορούμε να επηρεάσουμε και να ελέγξουμε την κατάσταση αυτών των αντικειμένων. Η γλώσσα προγραμματισμού που χρησιμοποιείται από όλους τους περιηγητές για αυτή την λειτουργία είναι η *Javascript*.

Εναποθέτουμε ένα παράδειγμα της δενδρικής δομής των στοιχείων ενός html αντικειμένου:

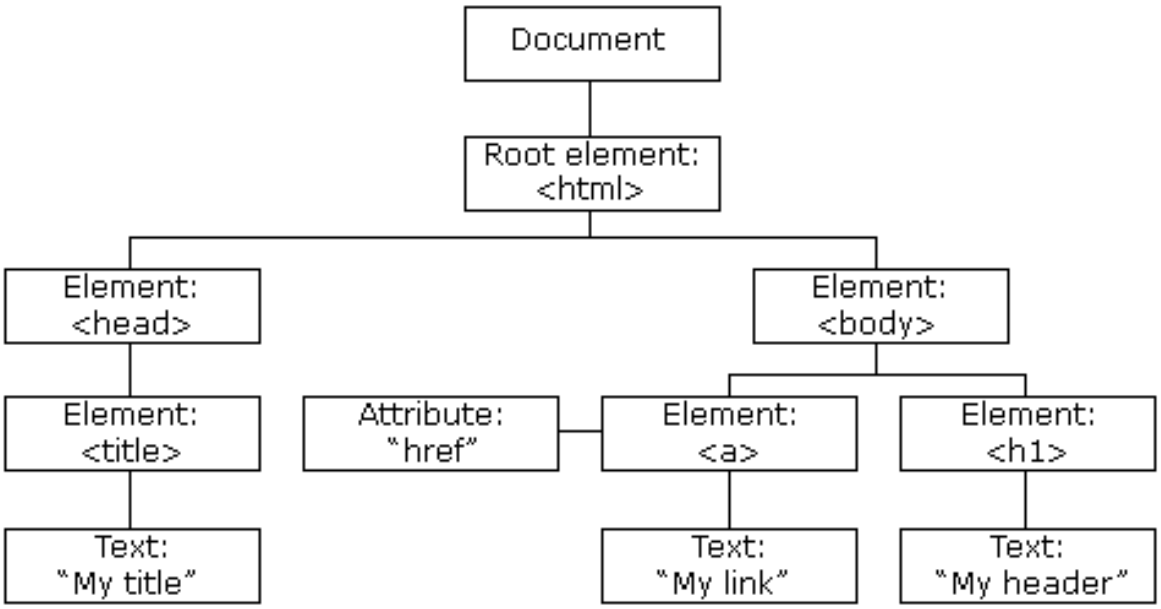

Figure 1: δενδρική δομή HTML DOM [3]

Αναφέραμε ότι η javascript είναι η κύρια γλώσσα επεξεργασίας του DOM. Ο κάθε περιηγητής έχει την δικιά του υλοποίηση javascript για να υποστηρίζει τις απαραίτητες λειτουργίες. Όλες οι υλοποιήσεις τηρούν τις προδιαγραφές για την γλώσσα όπως ορίζονται στο *ECMAScript specification*, τελευταία έκδοση του οποίου είναι αυτή του 2020 [4]. Ο

κώδικας javascript που συμπεριλαμβάνεται από κάποιο αρχείο html εκτελείται από τον περιηγητή κατά την φόρτωση της ιστοσελίδας. Υπάρχουν αρκετές διαφορές από υλοποίηση σε υλοποίηση αλλά για την κατανόηση των επόμενων μας ενδιαφέρουν να εξηγήσουμε κάποιες συμπεριφορές που ορίζονται στο πρότυπο και είναι ίδιες σε κάθε υλοποίηση.

Για διάφορους λόγους η javascript εκτελείται σε μόνο έναν νήμα επεξεργασίας. Αυτό σημαίνει ότι δεν υπάρχει δυνατότητα δημιουργίας πολλαπλών νημάτων από τον επεξεργαστή ώστε να μπορούν να εκτελεστούν πολλές λειτουργίες ταυτόχρονα. Για αυτό τον λόγο η javascript λειτουργεί με ένα μοντέλο ταυτοχρονισμού γνωστό ως *βρόγχος γεγονότων (even loop)*. Οι λεπτομέρειες διαφέρουν ανάλογα με την *μηχανή javascript (javascript engine)* του κάθε περιηγητή, αλλά η λογική είναι ότι υπάρχει μια ουρά μηνυμάτων ή γεγονότων καθένα με τα οποία συνδέονται με μια *συνάρτηση (function)*. Αυτά εκτελούνται κάθε φορά με την σειρά καθώς νέα γεγονότα προστίθενται στην ουρά.

Για καλύτερη κατανόηση δείχνουμε μια γενικευμένη εικόνα επεξήγησης του event loop από την javascript engine του περιηγητή mozilla firefox:

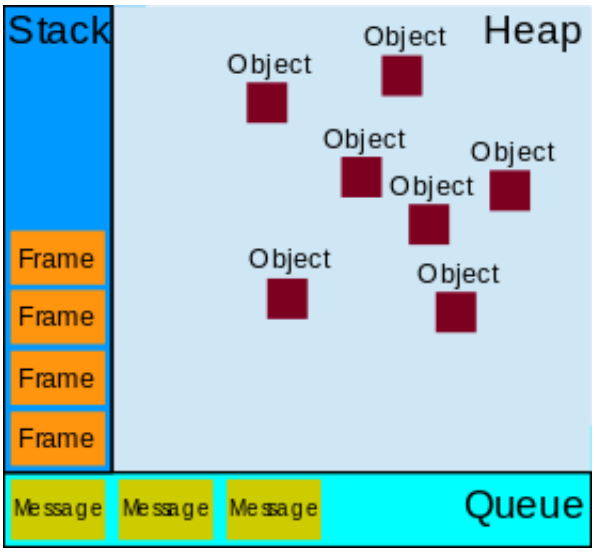

Figure 2: Event loop of firefox browser [5]

καθώς και μια εικόνα των φάσεων που περνάει το event loop του nodejs όπου είναι ένα *περιβάλλον εκτέλεσης javascript (javascript runtime)* βασισμένο στο javascript engine του περιηγητή chrome:

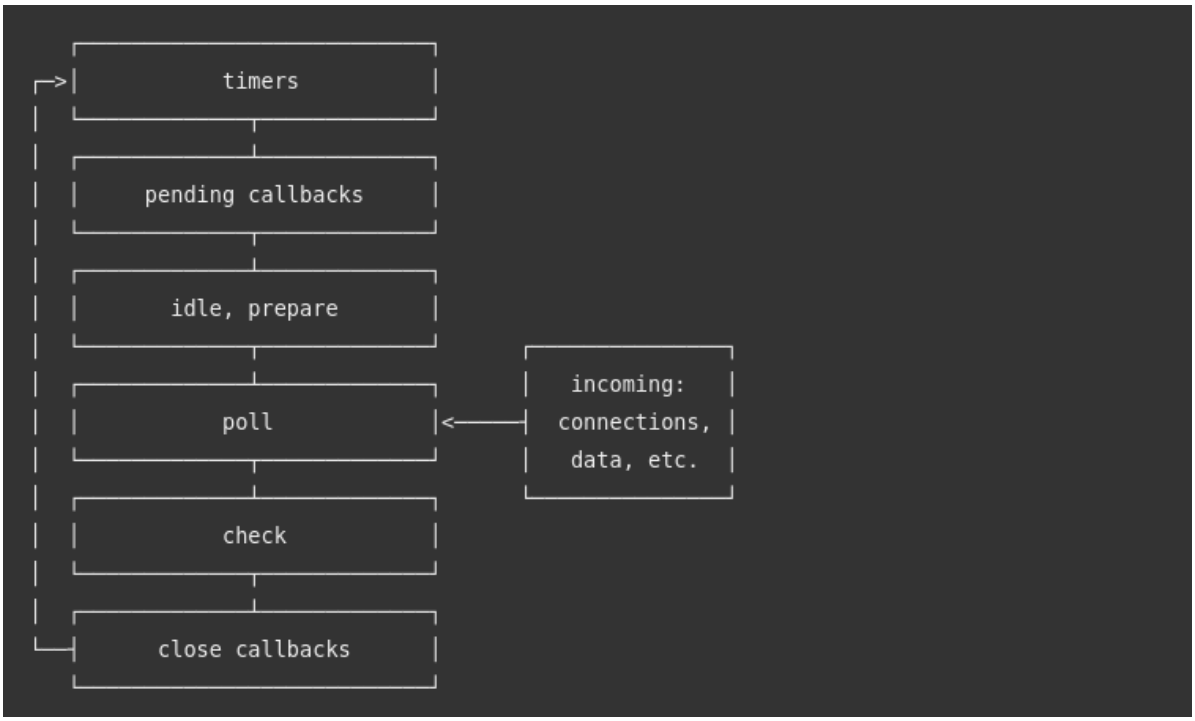

Figure 3: nodejs event loop [6]

Καταλαβαίνουμε ότι αν η συνάρτηση εκτέλεσης κάποιου γεγονότος είναι μεγάλης διάρκειας τότε όλη η ουρά μπλοκάρει και δεν εκτελείται κάποιο άλλο γεγονός μέχρι να τελειώσει το πρώτο. Για αυτό τον λόγο προσπαθούμε να φτιάχνουμε συναρτήσεις με τρόπο ασύγχρονο. Δηλαδή αρχικά τις καταμερίζουμε σε μικρότερα μέρη. Όταν καλούμε κάποια συνάρτηση, τότε αφού τελειώσει το μέρος της που εκτελούταν, αυτή επιστρέφει τον έλεγχο ώστε να μπορέσει να εκτελεστεί κάποια άλλη συνάρτηση από την ουρά του event loop. Καταλαβαίνουμε ότι επειδή ο έλεγχος επιστρέφει μετά από την εκτέλεση ενός μικρού μέρους της αρχικής συνάρτησης που θέλαμε να υλοποιήσουμε και όχι αφού εκτελεστεί ολόκληρη, θέλουμε κάποιον τρόπο για να καταλαβαίνουμε πότε έχει ολοκληρωθεί όλη η εργασία που θέλαμε να κάνουμε. Στην σύγχρονη javascript η υλοποίηση των ασύγχρονων συναρτήσεων γίνονται μέσω του μηχανισμού των *υποσχέσεων (Promises)*. Ένα promise χρησιμοποιείται για να περιμένουμε κάποια τιμή που ακόμα δεν είναι διαθέσιμη. Μπορεί να έχει 3 καταστάσεις:

- *εκκρεμής (pending)* όταν δεν έχει ολοκληρωθεί ακόμα
- *ολοκληρωμένη (fullfiled)* όταν η εκτέλεση ολοκληρώθηκε και η τιμή είναι διαθέσιμη

- *απορριφθείσα (rejected)* όταν η λειτουργία υπολογισμού της τιμής απέτυχε

Έτσι μπορούμε να προγραμματίσουμε τι θα γίνει σε κάθε περίπτωση και να εκτελούμε ταυτόχρονα υπολογισμούς χωρίς να μπλοκάρουμε το event loop. Παραθέτουμε ένα διάγραμμα των καταστάσεων ενός promise σε προγραμματιστικό επίπεδο:

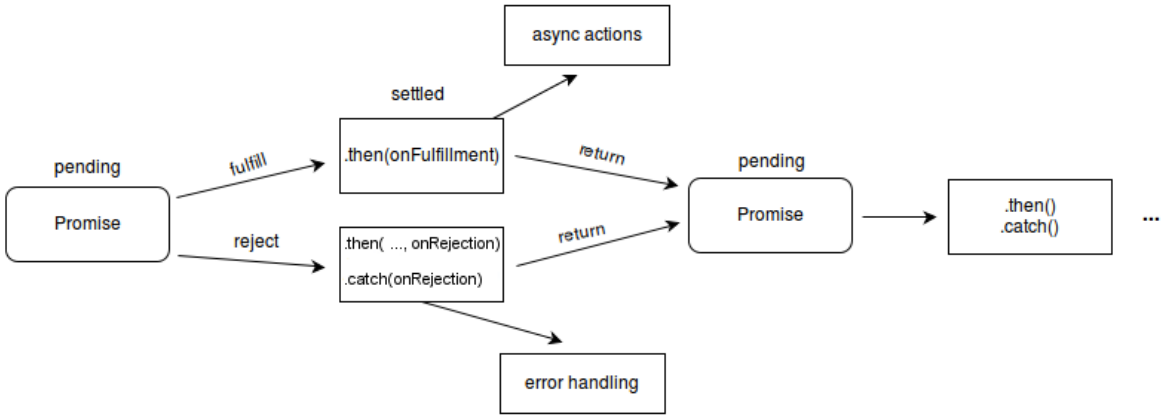

Figure 4: Promise states [7]

2.1.2 Vuejs Framework

Έχουμε εξηγήσει τις βασικές λειτουργίες των διαδικτυακών εφαρμογών και του DOM. Ωστόσο, δεδομένου ότι η βασικές λειτουργίες του σχεδιάστηκαν αρκετά παλιά, παρά τις βελτιώσεις του τα τελευταία χρόνια υπάρχουν ορισμένα μειονεκτήματα. Κύριο μειονέκτημα είναι η κακή απόδοσή του. Έχουμε αναφέρει ότι το DOM κρατάει μια δενδρική δομή των αντικειμένων html. Κάθε φορά που θέλουμε να αλλάξουμε κάποιο html element τότε πρέπει να βρεθεί αυτό το στοιχείο από το δέντρο, να επεξεργαστεί και μετά ο περιηγητής να το απεικονίσει στην οθόνη (render). Όταν αλλάζουμε πολλά στοιχεία προγραμματιστικά με javascript υπάρχει μεγάλο υπολογιστικό κόστος. Το δεύτερο κύριο μειονέκτημα είναι ότι η προγραμματιστική διεπαφή που ορίζεται για την επεξεργασία του DOM δεν είναι πολύ εύχρηστη. Υπάρχουν βιβλιοθήκες όπως η γνωστή jQuery όπου προσφέρουν λίγο πιο εύχρηστες προγραμματιστικές διεπαφές, ωστόσο αν και έχουν βρει μεγάλη επιτυχία στην δημιουργία απλών ιστοσελίδων όταν πρόκειται για πιο σύνθετες εφαρμογές υστερούν σημαντικά. Για αυτόν τον σκοπό έχουν δημιουργηθεί διάφορα

πιο σύνθετα frameworks που προσφέρουν αυξημένες δυνατότητες στους προγραμματιστές.

Ιστορικά τα παλιότερα χρόνια είχαν επικρατήσει τα *frameworks από την μεριά του διακομιστή (server side frameworks)*. Η λογική είναι ότι οι ιστοσελίδες προσφέρονται από έναν υπολογιστή που έχει ρόλο server. Ωστόσο, πριν οι σελίδες σταλούν στου περιηγητές έχουν επεξεργαστεί με διάφορες template engines όπου έχουν δημιουργήσει από πριν το τελικό DOM. Έτσι ελαχιστοποιείται η χρήση javascript και η επεξεργασία του DOM από τους περιηγητές. Αν και αυτή η μέθοδος προτιμάται ακόμα σε πολλές περιπτώσεις, το κύριο μειονέκτημά της είναι η συνεχής εξάρτηση από τον server και οι συνεχείς κλήσεις για καινούριες σελίδες κάθε φορά που υπάρχει ανάγκη για επεξεργασία του DOM.

Ως εναλλακτική λύση έχουν δημιουργηθεί τα *frameworks από την μεριά του πελάτη (client side frameworks)*. Αυτά προσφέρουν εξελιγμένες δυνατότητες επεξεργασίας του DOM με χρήση javascript χωρίς να χρειάζεται κλήση σε κάποιον server για κάθε σελίδα. Έτσι ο server μπορεί να χρησιμοποιείται αποκλειστικά για να προσφέρει δεδομένα όταν αυτό χρειάζεται χωρίς να έχει κάποιο ρόλο στην επεξεργασία των σελίδων και του DOM. Η διαδικτυακή εφαρμογή χρειάζεται τότε απλά να γίνει διαθέσιμη ως στατικές σελίδες όπου ο περιηγητής μπορεί να κατεβάσει. Η δυνατότητα αυτή της προσφοράς στατικών σελίδων υπάρχει σε πολλούς μικρούς και οικονομικούς από άποψης πόρων servers, αλλά επίσης προσφέρεται πάντα από όλους τους γνωστούς servers που συνήθως χρησιμοποιούνται για δυναμικές σελίδες σε συνδυασμό με κάποια γλώσσα για server side επεξεργασία (π.χ. Apache ή nginx με γλώσσα php, Tomcat με γλώσσα java, κτλ.). Έτσι υπάρχει ευελιξία στον τρόπο με τον οποίο θα γίνει διαθέσιμη η εφαρμογή στους χρήστες.

Ένα τέτοιο client side javascript framework είναι το VueJs το οποίο θα αναλύσουμε εδώ. Πρόκειται καταρχάς για ένα, όπως αποκαλείται, *προοδευτικό framework (progressive framework)*. Η έννοια αυτή σημαίνει ότι δεν είναι απαραίτητο να φτιαχτεί μια ολόκληρη ιστοσελίδα ή διαδικτυακή εφαρμογή βάση αυτού, αλλά μπορεί να χρησιμοποιηθεί μόνο για ένα συγκεκριμένο μέρος της εφαρμογής όπου χρειάζονται αυξημένες δυνατότητες επεξεργασίας όταν αυτό κρίνεται απαραίτητο. Η

βασική ιδέα που διέπει την VueJs είναι αυτή του *εικονικού DOM (Virtual DOM)*. Ο προγραμματιστής δεν ασχολείται με το να επεξεργαστεί απευθείας το DOM, αν και εξακολουθεί να υπάρχει αυτή η δυνατότητα. Αντιθέτως, ορίζονται μεταβλητές σε javascript οι οποίες δένονται με συγκεκριμένα elements του DOM *(data binding)*. Λέμε ότι τα δεδομένα αυτά που ορίζονται με αυτόν τον τρόπο είναι *αντιδραστικά (reactive)*. Αυτό σημαίνει ότι το framework παρακολουθεί αυτά τα δεδομένα για αλλαγές. Όταν αυτά αλλάξουν τότε αυτόματα ειδοποιείται η συνάρτηση απεικόνισης (rendering) και δημιουργείται το καινούριο πραγματικό DOM με τα νέα δεδομένα. Οι δυνατότητές του είναι αρκετά εξελιγμένες ώστε να γίνονται rendered μόνο οι περιοχές όπου άλλαξαν εξοικονομώντας έτσι επεξεργαστικούς πόρους. Επίσης, όταν αλλάζουν πολλές τέτοιες μεταβλητές ταυτόχρονα αλλά στην ίδια φάση λειτουργίας, θα γίνει μόνο μια φορά render μετά από όλες τις αλλαγές κάνοντας σαφές το πλεονέκτημα στην απόδοση σε σχέση με την απευθείας επεξεργασία του DOM. Παραθέτουμε μια εικόνα για την καλύτερη κατανόηση της παρακολούθησης αυτών των δεδομένων και του rendering:

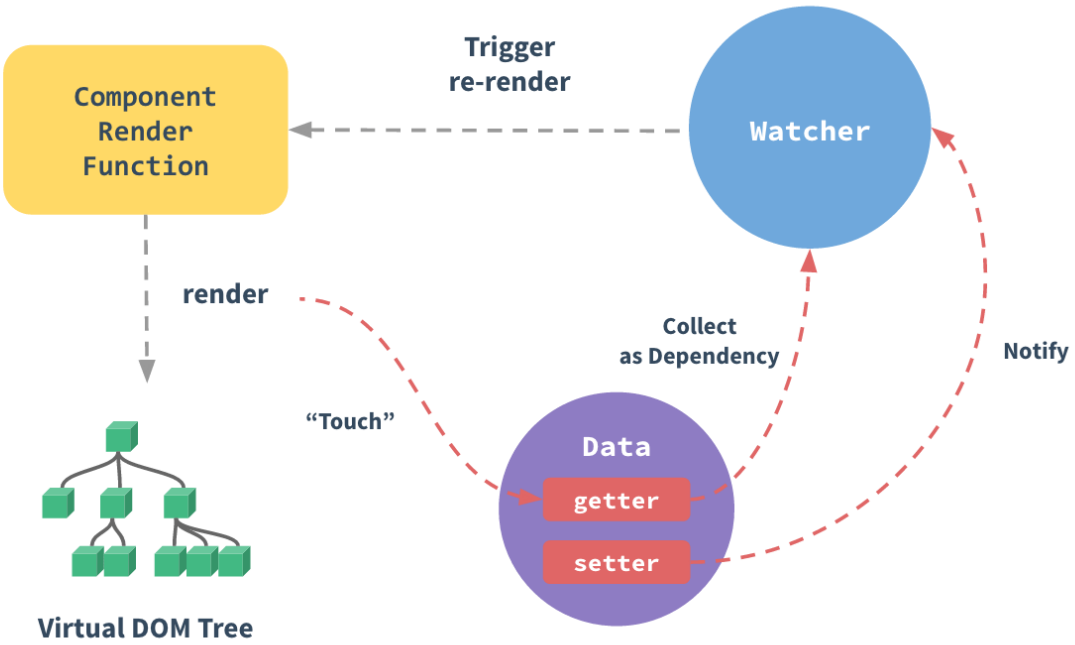

Figure 5: Reactivity στην VueJs [8]

Μια VueJs εφαρμογή οργανώνεται σε ξεχωριστά *δομικά στοιχεία (components)*. Κάθε τέτοιο component έχει τον δικό του κώδικα html, την δικό του κώδικα javascript και την δικιά του css για την επεξεργασία της εμφάνισής του. Το αρχικό component είναι το κύριο και κάθε επόμενο προστίθεται σε κάποιο ήδη υπάρχων με σχέση πατέρα παιδιού. Ο πατέρας μπορεί να επικοινωνεί με το παιδί περνώντας του *ιδιότητες (properties)*, ενώ το παιδί επικοινωνεί με τον πατέρα στέλνοντάς του *γεγονότα (events)*. Κάθε component που δημιουργείται περνάει από διάφορα στάδια ενός κύκλου ζωής όπως φαίνεται στο παρακάτω σχήμα:

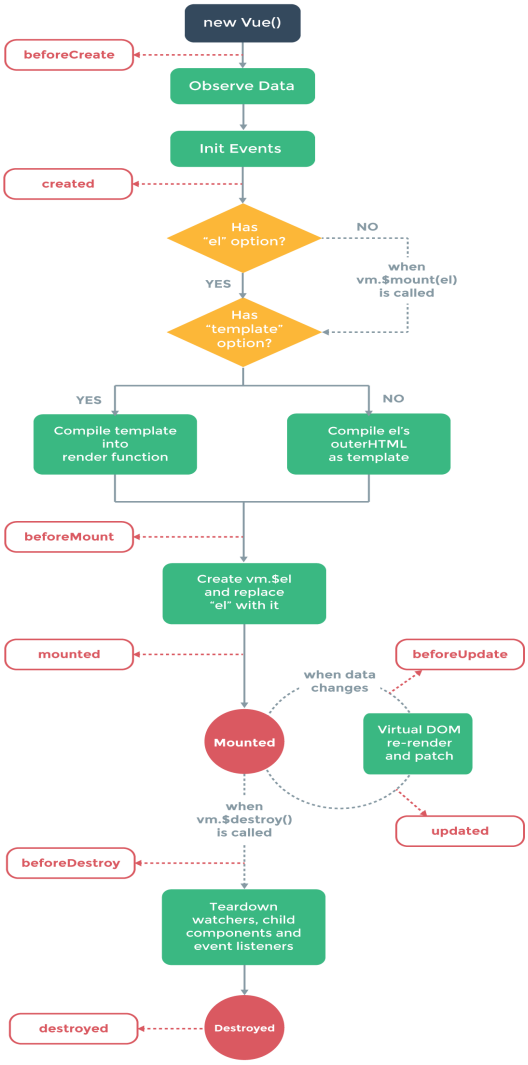

Figure 6: Vue Instance Life Cycle [9]

2.1.3 Http requests, Ajax και βιβλιοθήκη axios

Έχουμε εξηγήσει ότι με τα client side frameworks δεν χρειάζεται να υπάρχει κάποιος server που να επεξεργάζεται τις διαδικτυακές σελίδες με νέα δεδομένα. Ωστόσο, σε πολλές εφαρμογές, όπως θα δούμε και στην δική μας, χρησιμοποιείται ένας server για να επεξεργαστεί και να μας στείλει δεδομένα. Καταλαβαίνουμε ότι θέλουμε έναν τρόπο να μπορούμε να αλληλεπιδράσουμε με έναν server από την μεριά του client αφού έχει φορτωθεί μια σελίδα.

Τα αιτήματα που μας ενδιαφέρουν στην περίπτωσή μας είναι τα *http requests*. Το *http (HyperText Transfer Protocol)* είναι ένα πρωτόκολλο δικτύου. Χρησιμοποιείται συνήθως από εφαρμογές για να στέλνουμε ή να λαμβάνουμε δεδομένα από έναν server. Δεν θα αναλύσουμε λεπτομέρειες καθώς την *προδιαγραφή* του *(specification)* μπορεί κανείς να την βρει στο διαδίκτυο [10]. Επιγραφικά αναφέρουμε ότι υποστηρίζει διάφορους μεθόδους ανάλογα με το τι θέλουμε να επιτύχουμε όπως οι μέθοδοι:

- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- CONNECT
- OPTIONS

Η καλύτερη μέθοδος για να στείλουμε http αιτήματα όπως τα περιγράψαμε παραπάνω ονομάζεται *AJAX (Asynchronous JavaScript and XML)*. Δεν πρόκειται για κάποια συγκεκριμένη υλοποίηση αλλά για μια τεχνική. Ο όρος XML

περιλαμβάνεται για ιστορικούς λόγους αλλά δεν η ανάγκη η χρήση της XML. Με αυτή την τεχνική μπορούμε να στέλνουμε http αιτήματα σε κάποιον server ασύγχρονα χωρίς να μπλοκάρουμε το event loop (έχουμε εξηγήσει τη σημασία αυτού στις προηγούμενες ενότητες). Όταν το αίτημα ολοκληρωθεί καλείται κάποιος κώδικας που έχουμε ορίσει ανάλογα με το αν ήταν επιτυχές ή αν απέτυχε αυτό το αίτημα. Οι υλοποιήσεις javascript των περιηγητών διαθέτουν υποστήριξη για αυτή την λειτουργία μέσω του *αντικειμένου XMLHttpRequest (XMLHttpRequest object)*. Δεν θα αναλύσουμε τις λεπτομέρειες αυτού καθώς πρόκειται για μια συγκεκριμένη υλοποίηση της παραπάνω τεχνικής.

Εμάς μας ενδιαφέρει η βιβλιοθήκη *Axios*. Πρόκειται για μια βιβλιοθήκη javascript η οποία έχει δημιουργηθεί για την δημιουργία τέτοιων ασύγχρονων αιτημάτων. Υποστηρίζει πολλές διαφορετικές μεθόδους http και είναι αρκετά ευέλικτο ως προς την παραμετροποίηση αυτών των αιτημάτων. Ο μηχανισμός με τον οποίο πετυχαίνει τα ασύγχρονα αιτήματα χωρίς να μπλοκάρει το event loop είναι αυτός των *Promises* τον οποίο έχουμε εξηγήσει σε προηγούμενη ενότητα. Αυτή η χρήση τους είναι πολύ σημαντική αφού επιτρέπει την εύκολη συνεργασία με άλλες βιβλιοθήκες όπου χρησιμοποιούν Promises ή και ακόμα να φτιάχνουμε δικές μας αφαιρετικές συναρτήσεις (abstractions) όπου θα χρησιμοποιούν αυτές την axios και θα επιστρέφουν το αντίστοιχο Promise, κρύβοντας τις λεπτομέρειες από το υπόλοιπο πρόγραμμά μας.

2.1.4 Java, Http server και Vertx Framework

Έχουμε αναφέρει ότι πολλές διαδικτυακές εφαρμογές χρησιμοποιούν κάποιον server είτε για να προσφέρει ιστοσελίδες είτε για να δέχεται και να στέλνει δεδομένα. Εμείς θα αναφερθούμε στους http servers όπου υλοποιούνται με την γλώσσα προγραμματισμού java και χρησιμοποιούνται από τους clients για να λαμβάνουν και να στέλνουν δεδομένα με τους τρόπους που εξηγήσαμε παραπάνω.

Η java είναι μια γλώσσα προγραμματισμού που έχει σχεδιαστεί κυρίως για να τρέχει σε μια εικονική μηχανή την λεγόμενη *JVM (Java Virtual Machine)*. Ο κώδικας πρώτα μεταγλωτίζεται (compiling) σε αρχεία που περιέχουν java bytecode η οποία είναι μια

δυαδική μορφή αναπαράστασης του κώδικα αναγνωρίσιμη από το JVM για να μπορεί να εκτελεστεί. Λόγω της ιστορίας της υπάρχον πληθώρα από βιβλιοθήκες και πολλές υλοποιήσεις διαφορετικών http servers. Υπάρχουν επίσης πολλές υλοποιήσεις με αρκετές διαφορές τέτοιων εικονικών μηχανών που αναγνωρίζουν java bytecode. Κύριο σημείο ενδιαφέροντος είναι ότι εδώ δεν έχουμε τον περιορισμό του μοναδικού thread ενός προγράμματος που υπάρχει στις υλοποιήσεις javascript των περιηγητών. Άρα ένα πρόγραμμα μπορεί να χρησιμοποιεί πολλά threads για να πετύχει τον σκοπό του.

Ένας http server πρέπει να μπορεί να δέχεται ένα http αίτημα με κάποια από τις http μεθόδους που έχουμε αναφέρει παραπάνω. Κάθε αίτημα θα δέχεται κάποια δεδομένα, θα εκτελεί κάποια συγκεκριμένη εργασία και θα επιστρέφει κάποιο αποτέλεσμα. Στόχος μας είναι να μπορούμε να εξυπηρετήσουμε πολλά τέτοια αιτήματα ταυτόχρονα. Κάθε αίτημα θα πρέπει να εκτελείται απομονωμένα χωρίς τα ενδιάμεσα αποτελέσματα κάποιας τρέχουσας εργασίας ενός αιτήματος να επηρεάζει τα αποτελέσματα κάποιας άλλης. Θα αναφέρουμε περιληπτικά 3 γνωστά μοντέλα με τα οποία πετυχαίνουμε αυτόν τον ταυτοχρονισμό (concurrency).

- Ο πιο ιστορικός τρόπος να πετύχουμε τον ταυτοχρονισμού είναι με *νήματα (threads)*. Ένα νήμα είναι απομονωμένο και έχει τον δικό χώρο μνήμης για μεταβλητές αν και μπορεί να διαμοιράζεται πόρους με διάφορους τρόπους. Με αυτό το μοντέλο επιλέγουμε να δημιουργούμε ένα καινούριο νήμα για κάθε καινούριο αίτημα που καλούμε να απαντήσουμε. Έτσι κάθε εργασία που πρέπει να εκτελεστεί είναι απομονωμένη από τις υπόλοιπες. Αναφέρουμε για παράδειγμα ότι ο Tomcat server χρησιμοποιεί αυτό το μοντέλο, αν και υπάρχουν αρκετοί τρόποι παραμετροποίησης του. Μειονέκτημα αυτού του μοντέλου είναι ότι ένας υπολογιστής μπορεί να εκτελεί τόσα παράλληλα νήματα όσα μπορεί να υποστηρίξει ο επεξεργαστής του. Σε κρίσιμους servers με μεγάλο αριθμό αιτημάτων, τα νήματα που δημιουργούνται είναι αρκετές φορές πολλαπλάσια των νημάτων του επεξεργαστή, επιφέροντας μεγάλη μείωση στην απόδοση.

- Ένα άλλο πιο πρόσφατο μοντέλο είναι το *μοντέλο των ηθοποιών (actor model)*. Σε αυτό μια εργασία γίνεται σε κάποιον actor. Κάθε actor είναι απομονωμένος έχοντας την δικιά του *ιδιωτική κατάσταση (private state)* που δεν μπορεί να αλλάξει από κάποιον άλλον. Ο κάθε actor έχει ένα *γραμματοκιβώτιο (mailbox)* και επικοινωνούν ανταλλάσοντας μηνύματα μεταξύ τους. Οι υλοποιήσεις διαφέρουν από σύστημα σε σύστημα αλλά οι actors πάντα χρειάζονται λιγότερους πόρους από τα threads και μπορούν να υπάρχουν σε μεγάλο βαθμό ταυτόχρονα. Μια τέτοια υλοποίηση βασισμένη στο actor model είναι αυτή της βιβλιοθήκης Akka. Δείχνουμε μια εικόνα για καλύτερη κατανόηση αυτού του μοντέλου.

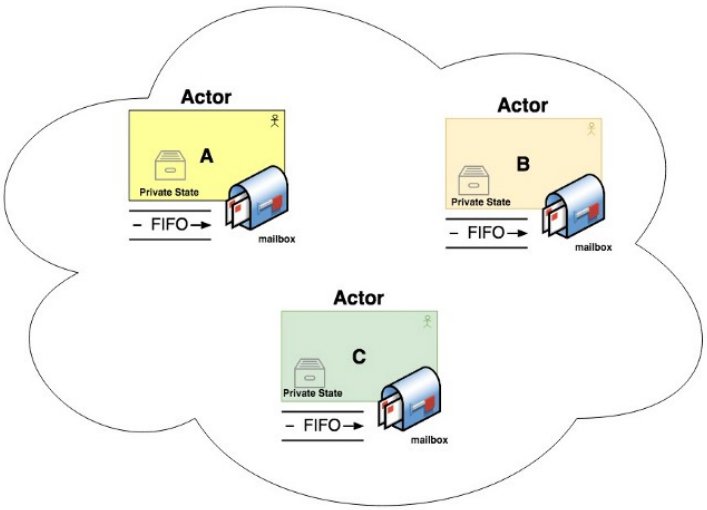

Figure 7: Actor model [11]

- Το τελευταίο μοντέλο που θα αναφέρουμε είναι αυτό του ασύγχρονου προγραμματισμού με κάποιο event loop. Το έχουμε ήδη εξηγήσει καθώς οι υλοποιήσεις javascript βασίζονται σε αυτό το μοντέλο. Πολλές φορές έχει την καλύτερη απόδοση από όλα τα μοντέλα. Ωστόσο, μειονέκτημά του είναι ότι δεν πρέπει ποτέ να μπλοκάρουμε το event loop με συνεχόμενες εργασίες που διαρκούν αρκετή ώρα (π.χ. επαναλήψεις). Έτσι, κάνει τον προγραμματισμό δυσκολότερο αφού πρέπει ο προγραμματιστής να

μετατρέπει τις συναρτήσεις του σε ασύγχρονες καθώς και να χρησιμοποιεί όσο μπορεί βιβλιοθήκες με ασύγχρονη λογική.

Εμείς θα επικεντρωθούμε στο Vert.x framework [12]. Πρόκειται για ένα framework που είναι γραμμένο στην γλώσσα Polyglot, η οποία μπορεί να μεταφραστεί σε πολλές άλλες γλώσσες. Έτσι είναι εύκολα διαθέσιμο για όλες τις γλώσσες που μπορούν να τρέξουν στο JVM όπως Java, Scala, Koltin, κτλ. Έχει εμπνευστεί από την NodeJs που είχαμε αναφέρει ως ένα περιβάλλον για εκτέλεση javascript. Βασίζεται στο μοντέλο του ασύγχρονου προγραμματισμού και μας προσφέρει την δυνατότητα να δημιουργούμε ασύγχρονες συναρτήσεις. Όπως όλες οι υλοποιήσεις αυτού του μοντέλου διαθέτει ένα event loop όπου εκτελεί τα γεγονότα που εισέρχονται σε μια ουρά:

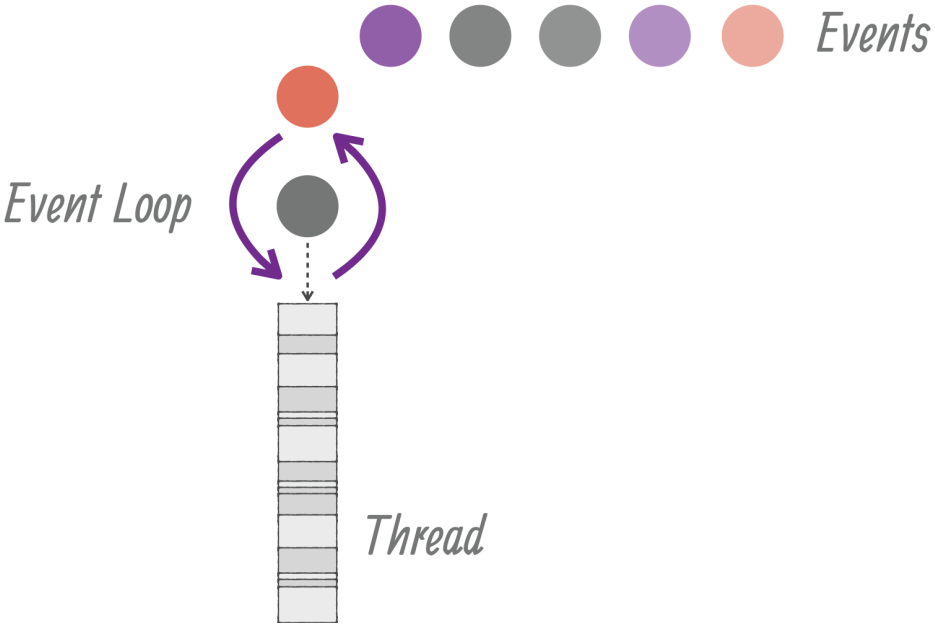

Figure 8: Vert.x event loop [13]

Ισχύουν όλα αυτά που έχουμε εξηγήσει σε προηγούμενες ενότητες για το event loop. Ο τρόπος για να περιμένουμε κάποιο ασύγχρονο αποτέλεσμα έχει την ίδια λογική, απλά οι ορολογίες είναι λίγο διαφορετικές καθώς χρησιμοποιούμε *Promises* και *Futures* για τα οποία δεν θα μπούμε σε λεπτομέρειες αφού αφορούν την υλοποίηση της προγραμματιστικής διεπαφής.

Κύριο πλεονέκτημα αυτού του framework είναι ότι λόγω του ότι εκτελείται πάνω στο JVM είναι δυνατόν να χρησιμοποιήσει και πολλαπλά νήματα. Ο τρόπος που το πραγματοποιεί αυτό είναι με την έννοια του Verticle. Κάθε Verticle έχει το δικό του event loop και μπορεί να τρέχει σε ξεχωριστό νήμα αν υπάρχει διαθέσιμο στο σύστημα. Συνήθως υπάρχει ένα κύριο (main) Verticle το οποίο κάνει deploy τα υπόλοιπα.

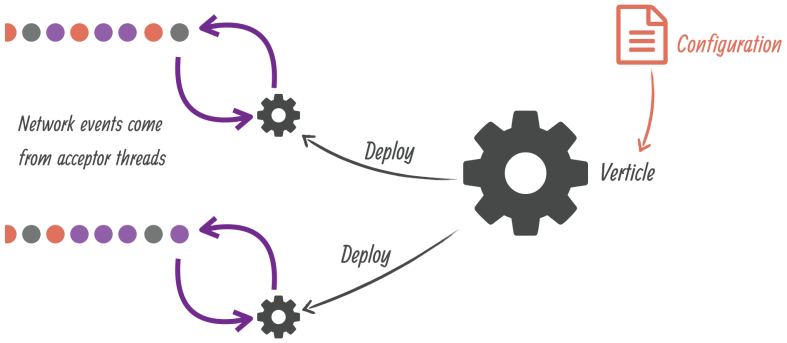

Figure 9: Vert.x verticles [13]

Τα verticles επικοινωνούν μεταξύ τους μέσω ανταλλαγής μηνυμάτων. Όλα τα μηνύματα περνούν από έναν *διάδρομο γεγονότων (event bus)* ο οποίος ελέγχεται από το κύριο Verticle. Ένα παράδειγμα φαίνεται στην παρακάτω εικόνα

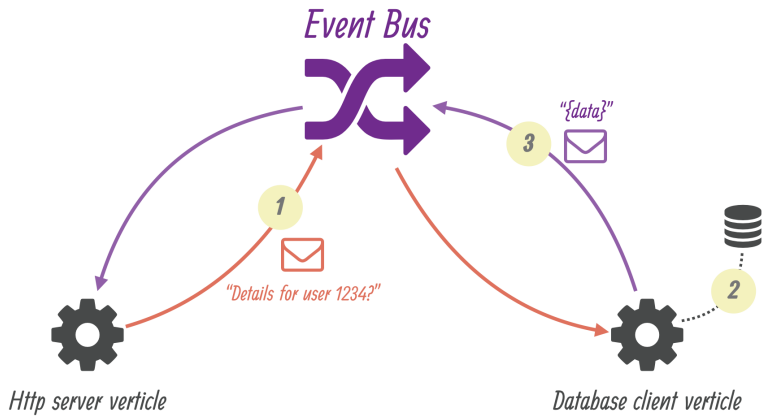

Figure 10: Vert.x event bus [13]

Το framework έχει ενσωματωμένη δυνατότητα δημιουργίας ενός http server όπου λειτουργεί με το ασύγχρονο μοντέλο βάση όλων των παραπάνω.

2.1.5 Βιβλιοθήκη Owlapi

δ

2.2. Σημασιολογικός Ιστός και Οντολογίες

2.3. Καθορισμός Συσχετίσεων μεταξύ Οντολογιών

Δφασδφα

ασδφ

3. Περιγραφή Συστήματος

Asdfa

asdf

4. Υλοποίηση Συστήματος και Παραδείγματα

Φσαδφα

ασδφ

5. Συμπεράσματα και μελλοντική εξέλιξη

Δφασδ

δασφς

6. Σύνοψη

Δφασδ

7. Βιβλιογραφικές Αναφορές

[1]: Efthymios Chondrogiannis, Vassiliki Andronikou, Efstathios Karanastasis, Theodora Varvarigou, Ontology Alignment Tool, , https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.664.4075&rep=rep1&type=pdf
[2]: WHATWG, HTML Living Standard, 2020, https://html.spec.whatwg.org/print.pdf
[3]: , The HTML DOM, , https://www.w3schools.com/whatis/whatis_htmldom.asp
[4]: , ECMAScript Specification, 2020, https://www.ecma-international.org/ecma-262/
[5]: , Concurrency model and the event loop, , https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop
[6]: , The Node.js Event Loop, , https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/
[7]: , Promise, , https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
[8]: , Reactivity in VueJs, , https://vuejs.org/v2/guide/reactivity.html
[9]: , The Vue Instance, , https://vuejs.org/v2/guide/instance.html
[10]: , Hypertext Transfer Protocol -- HTTP/1.1, , https://www.ietf.org/rfc/rfc2616.txt
[11]: , Actor Model in Nutshell, , https://medium.com/@KtheAgent/actor-model-in-nutshell-d13c0f81c8c7
[12]: , Vertx, , https://vertx.io/
[13]: Julien Ponge, Thomas Segismont, Julien Viet, A gentle guide to asynchronousprogramming with Eclipse Vert.x forJava developers, 2019, https://vertx.io/docs/guide-for-java-devs/guide-for-java-devs.pdf