

Ιόνιο Πανεπιστήμιο
Τμήμα Πληροφορικής



Πτυχιακή Εργασία

Ανάπτυξη Σημασιολογικής Εφαρμογής και
Οντολογίας για Ηλεκτρονικά Παιχνίδια

Ελένη Μαρία Μητσοπούλου
Α.Μ. Π2017018

Επιβλέπων: Μιχαήλ Στεφανιδάκης

Σεπτέμβριος 2021

Επιβλέπων

Μιχαήλ Στεφανιδάκης, Αν Καθηγητής
Τμ. Πληροφορικής Ιονίου Πανεπιστημίου

Τριμελής Επιτροπή

Μιχαήλ Στεφανιδάκης, *Αν. Καθηγητής*
Τμ. Πληροφορικής Ιονίου Πανεπιστημίου

Θεόδωρος Ανδρόνικος, *Αν. Καθηγητής*
Τμ. Πληροφορικής Ιονίου Πανεπιστημίου

Ελένη Χριστοπούλου, *ΕΔΙΠ*
Τμ. Πληροφορικής Ιονίου Πανεπιστημίου

Περίληψη

Με τη διαρκή και αυξανόμενη χρήση του Παγκόσμιου Ιστού, κάθε μέρα δημιουργούνται εκατομμύρια νέα δεδομένα που αποθηκεύονται στις βάσεις δεδομένων και τους εξυπηρετητές που βρίσκονται διάσπαρτοι σε ολόκληρο τον κόσμο. Μέσα σ' έναν τόσο μεγάλο όγκο δεδομένων, είναι εύκολο να χαθούν σημαντικές πληροφορίες αν δεν αρχειοθετηθούν εγκαίρως. Για την αντιμετώπιση αυτού του προβλήματος δημιουργήθηκε ο Σημασιολογικός Ιστός, μια επέκταση του Παγκόσμιου Ιστού και σύνολο προτύπων που δίνει τη δυνατότητα στον υπολογιστή να συμβάλλει με τον δικό του τρόπο στην καλύτερη ανάκτηση των δεδομένων, καθώς αυτά τώρα διαθέτουν δικές τους περιγραφές, ειδικά ορισμένες, ώστε να μπορεί να τις αναγνωρίσει ακόμη και μια μηχανή.

Η παρούσα πτυχιακή εργασία δημιουργήθηκε στα πλαίσια της περιγραφής του τρόπου εφαρμογής του ολοένα και εκτεινόμενου φαινομένου του Σημασιολογικού Ιστού στο σημερινό περιβάλλον του διαδικτύου. Αρχικά δημιουργήθηκε μια οντολογία κατηγοριοποίησης ηλεκτρονικών παιχνιδιών και στη συνέχεια μια διαδικτυακή εφαρμογή για να επικοινωνεί με την συγκεκριμένη οντολογία. Τελικός μας στόχος ήταν να δείξουμε πώς μπορεί να εφαρμοστεί μια παρόμοια οντολογία του Σημασιολογικού Ιστού σε έναν κοινό και σύγχρονο διαδικτυακό ιστότοπο με έναν ιδιαίτερα απλό τρόπο, για την ευκολότερη και ταυτόχρονα πιο ελκυστική προσπέλαση μιας οντολογίας από ανθρώπινο χρήστη.

Για την ανάπτυξη αυτής της σημασιολογικής εφαρμογής, χρησιμοποιήθηκαν διάφορα εργαλεία και συστήματα τα οποία έχουν ευρεία χρήση στον κλάδο του Σημασιολογικού Ιστού, όπως το λογισμικό σύνταξης οντολογιών Protégé, ο ειδικός εξυπηρετητής για τη διαχείριση οντολογιών και ερωτημάτων Apache Jena, και ένα απ' τα πιο διαδεδομένα frameworks για την ανάπτυξη γρήγορων διαδικτυακών εφαρμογών ReactJS. Κατά την ανάπτυξη αυτής της εφαρμογής, δόθηκε ιδιαίτερη προσοχή ώστε η διαδικασία να είναι όσο το δυνατόν πιο απλή γίνεται, για να έχει τη δυνατότητα κανείς, ανεξαρτήτως του επιπέδου δεξιοτήτων του, να ακολουθήσει τα βήματα προς την κατασκευή μιας παρόμοιας εφαρμογής με ευκολία.

Λέξεις-κλειδιά: Σημασιολογικός Ιστός, Οντολογία, Protégé, Apache Jena, Apache Fuseki, ReactJS, ηλεκτρονικά παιχνίδια

Abstract

With the constant and ever-growing use of the World Wide Web, millions of new data are being created every day, which are then stored in databases and servers located in various places all over the world. Amongst such a large volume of data, it is easy for somebody to lose important information if it is not archived in time. To address this problem, the Semantic Web was created as an extension to the World Wide Web and as a set of standards and formats used to enable a computer to contribute with its own abilities in the better retrieval of data, as these are now given their own descriptors, defined in such a way that even a machine will be able to recognize them.

This thesis was created within the context of describing how the ever-expanding phenomenon of the Semantic Web can be applied using the state of the Internet environment of modern day. Firstly, an ontology was created for the classification of video games, and next a web-based application to communicate with this ontology. Our goal was to present how could a similar Semantic Web ontology be applied in a common and modern web site, for easier and at the same time more approachable access to the contents of an ontology by a human user.

For the development of this semantic application, various tools and systems were involved, known to have a wide use in the production of a variety of other existing Semantic Web applications, including Protégé, an ontology editor software, Apache Jena, a server designed for storing and managing ontology data, and ReactJS, one of the leading frameworks for the development of optimized web applications. One notable point regarding this application's creation, was to put emphasis into keeping its development process as simple as possible, with the purpose of allowing anyone, regardless of skill level, to replicate the required steps towards the construction of a similar application.

Keywords: Semantic Web, Ontology, Protégé, Apache Jena, Apache Fuseki, ReactJS, Video Games

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον κ. Μιχάλη Στεφανιδάκη, Αναπληρωτή Καθηγητή του Τμήματος Πληροφορικής του Ιονίου Πανεπιστημίου για την άρτια συνεργασία μας, την πολύτιμη βοήθεια και επιστημονική καθοδήγηση που μου προσέφερε σε όλη την πορεία εκπόνησης της πτυχιακής μου εργασίας. Οι εύστοχες συμβουλές και υποδείξεις του έπαιξαν καταλυτικό ρόλο στη υλοποίηση του συστήματος που αναπτύχθηκε.

Επίσης θα ήθελα να ευχαριστήσω τους καθηγητές μου κ. Θεόδωρο Ανδρόνικο και κα. Ελένη Χριστοπούλου οι οποίοι αποδέχτηκαν να είναι μέλη της τριμελούς επιτροπής, καθώς επίσης και για τις πολύτιμες γνώσεις που μου προσέφεραν στα χρόνια των σπουδών μου.

Τέλος, θα ήθελα να εκφράσω τη βαθιά ευγνωμοσύνη μου προς την οικογένειά μου για την πολύτιμη βοήθεια, την ηθική στήριξη, την αμέριστη συμπαράσταση και τη διαρκή ενθάρρυνση.

Κέρκυρα, Σεπτέμβριος 2021

Ελένη Μαρία Μητσοπούλου

Περίληψη	iii
Abstract	iv
Ευχαριστίες	v
Εισαγωγή	1
Σκοπός της πτυχιακής εργασίας	1
Δομή της εργασίας	2
Θεωρητικό Υπόβαθρο.....	3
1. Σημασιολογικός Ιστός	3
2. Πρότυπα	7
2.1. XML	8
2.2. XML Schema	10
2.3. RDF	10
2.4. RDF Schema.....	14
2.5. Μορφές σειριοποίησης RDF	16
2.6. OWL	21
2.7. SPARQL.....	24
3. Μεταδεδομένα	26
4. Διασυνδεδεμένα Δεδομένα.....	27
5. Πόρος (Resource) & URI	29
6. Οντολογίες.....	30
Τεχνολογικό Υπόβαθρο	33
1. Protégé.....	33
2. Apache Jena.....	35
2.1. Εγκατάσταση Apache Jena.....	35
3. Apache Fuseki	40
3.1. Εγκατάσταση του Apache Fuseki	40
4. ReactJS	43

4.1. Εγκατάσταση του ReactJS	44
Η Οντολογία	47
1. Ανάπτυξη της Οντολογίας.....	48
1.1. Συλλογή και χρήση πηγών	48
1.2. Συγγραφή της οντολογίας	52
2. Δομή της Οντολογίας	54
2.1. Κλάσεις.....	54
2.2. Συσχετίσεις / Ιδιότητες αντικειμένων	58
2.3. Ιδιότητες δεδομένων.....	59
Ανάπτυξη της Εφαρμογής.....	61
1. Δομή του Κώδικα	61
2. Αρχιτεκτονική	78
3. Σενάρια Χρήσης	79
3.1. Αναζήτηση ενός παιχνιδιού.....	79
3.2. Αναζήτηση μιας πλατφόρμας κυκλοφορίας παιχνιδιού	81
3.3. Γενική εξερεύνηση των παιχνιδιών που φιλοξενούνται.....	83
Συμπεράσματα - Μελλοντικές Επεκτάσεις.....	87
Βιβλιογραφία	89

1

Εισαγωγή

Ο Παγκόσμιος Ιστός δημιουργήθηκε αρχικά για χρήση από ανθρώπους και παρόλο που οτιδήποτε διακινείται σε αυτόν είναι μηχαναναγνώσιμο και προσπελάσιμο δεν είναι ωστόσο μηχανικά κατανοητό. Ο όγκος των διεργασιών που λαμβάνουν χώρα στον Παγκόσμιο Ιστό, επιβάλλει την αυτοματοποίηση τους, προκειμένου να διεκπεραιώνονται σε χρονικά πλαίσια που να δικαιολογούν την εκτέλεση τους.

Ο κύριος σκοπός του Σημασιολογικού Ιστού είναι να αποδώσει σημασία στα δεδομένα που προέρχονται από διαφορετικές πηγές του Διαδικτύου, ώστε να μπορούν οι μηχανές να ερμηνεύσουν και να κατανοήσουν αυτά τα εμπλουτισμένα δεδομένα, να απαντήσουν με ακρίβεια και να ικανοποιήσουν τα αιτήματα του χρήστη, επιτρέποντας τη διαχείριση της γνώσης με αυτόματο τρόπο.

Σκοπός της πτυχιακής εργασίας

Η παρούσα εργασία έχει σκοπό την ανάλυση ενός κλάδου της επιστήμης του διαδικτύου, τον Σημασιολογικό Ιστό, καθώς και την περιγραφή του τρόπου της πρακτικής χρήσης του με την τεκμηρίωση μιας εφαρμογής που ενσωματώνει τον Σημασιολογικό Ιστό στις μεθόδους της. Επίσης αναδεικνύει τη σημασία των οντολογιών στην οργάνωση της γνώσης και των πληροφοριών οι οποίες βασίζονται σε ένα κοινό λεξιλόγιο. Αυτό επιτρέπει την ανταλλαγή πληροφοριών μεταξύ των εφαρμογών του Σημασιολογικού Ιστού. Οι οντολογίες αποτελούν, λοιπόν, ένα από τα βασικότερα εργαλεία σημασιολογικής διαλειτουργικότητας (semantic interoperability), γιατί προσφέρουν λεξιλόγια για την περιγραφή των εννοιών και των σχέσεών τους και λειτουργούν ως εργαλεία αυτόματης “διερμηνείας” και συσχέτισης των μεταδεδομένων που χρησιμοποιούνται από διάφορες εφαρμογές και υπηρεσίες πληροφόρησης.

Αντικείμενο της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη μιας σημασιολογικής εφαρμογής και μιας οντολογίας με θέμα τα ηλεκτρονικά παιχνίδια ή βιντεοπαιχνίδια και την περιγραφή τους ως προς τα στοιχεία που τα θέτουν στο φυσικό περιβάλλον, όπως ποιοι συνέβαλαν στη δημιουργία τους, ή ποιο είναι το είδος τους. Άλλες παρόμοιες οντολογίες που έχουν δημιουργηθεί με τη συγκεκριμένη θεματολογία των ηλεκτρονικών παιχνιδιών εστιάζουν περισσότερο στην καταγραφή του ψηφιακού περιβάλλοντος ενός παιχνιδιού (για παράδειγμα οντότητες όπως χαρακτήρες ή συμβάντα κατά τη διάρκεια του παιχνιδιού), ενώ η δική μας οντολογία εστιάζει στο φυσικό περιβάλλον του. Αφορμή της επιλογής των ηλεκτρονικών

παιχνιδιών ως θέμα της οντολογίας ήταν το πόσο κοινό και συνηθισμένο είναι σαν θέμα στη σημερινή εποχή, η στενή σχέση του με το ευρύτερο πεδίο της πληροφορικής, η απλότητα της αποτύπωσής του σε πλαίσια σημασιολογικών δεδομένων, και μια προϋπάρχουσα προσωπική γνώση της συγκεκριμένης θεματολογίας.

Η συνεισφορά και οι καινοτομίες που παρουσιάζονται σ' αυτή την πτυχιακή εργασία βασίζονται στην περαιτέρω επέκταση των τρόπων εφαρμογής σημασιολογικών δεδομένων σε διαδικτυακά περιβάλλοντα διεπαφών χρήστη, ειδικότερα με τέτοιο τρόπο ώστε να είναι ευκολότερα κατανοητή η χρήση τους από το κοινό, ακόμα και αυτών που δεν έχουν ιδιαίτερες γνώσεις σε ό,τι έχει να κάνει με τον Σημασιολογικό Ιστό.

Δομή της εργασίας

Στο *Κεφάλαιο 1* πραγματοποιείται μια εισαγωγή σχετικά με το θέμα που πραγματεύεται η εργασία, το ρόλο των Σημασιολογικών δεδομένων στη σημερινή εποχή και την ανάπτυξη μιας εφαρμογής που χρησιμοποιεί μια οντολογία. Επιπρόσθετα, παρουσιάζονται οι στόχοι καθώς και η δομή της συγκεκριμένης πτυχιακής.

Το *Κεφάλαιο 2* περιγράφει το θεωρητικό υπόβαθρο στο οποίο βασίζεται η παρούσα πτυχιακή εργασία και λειτουργεί ως εισαγωγή στο θέμα του Σημασιολογικού Ιστού. Αναλύονται διάφοροι ορισμοί που συναντώνται συχνά στην επιστήμη του Σημασιολογικού Ιστού, όπως τα μεταδεδομένα, τα διασυνδεδεμένα δεδομένα και οι οντολογίες. Επιπλέον στο κεφάλαιο αυτό μελετάται ο τρόπος με τον οποίο ελέγχεται η ορθότητα μιας οντολογίας και τα SPARQL ερωτήματα με τα οποία δημιουργούνται ερωτήσεις στις τριάδες RDF.

Στο *Κεφάλαιο 3*, περιγράφονται οι τεχνολογίες και οι μέθοδοι που χρησιμοποιήθηκαν για την ανάπτυξη τόσο της οντολογίας όσο και της εφαρμογής με την επεξήγηση των εργαλείων που χρησιμοποιήθηκαν για την ανάπτυξή της.

Το *Κεφάλαιο 4* περιγράφει τον τρόπο με τον οποίο δομείται η οντολογία, μαζί με αναλυτικές περιγραφές κάθε οντότητας που περιλαμβάνει.

Στο *Κεφάλαιο 5* παρουσιάζεται η δομή της εφαρμογής με αναλυτικές περιγραφές κάθε δομικού στοιχείου της, μέρη του κώδικα που αναπτύχθηκε καθώς και ενδεικτικά σενάρια χρήσης της.

Στο *Κεφάλαιο 6* παρουσιάζονται τα συμπεράσματα που προέκυψαν μετά την εκπόνηση της συγκεκριμένης πτυχιακής εργασίας, καθώς και κάποιες προτάσεις για μελλοντική βελτίωση και επέκτασή της.

Τέλος, αναγράφονται όλες οι βιβλιογραφικές αναφορές που χρησιμοποιήθηκαν κατά τη συγγραφή της παρούσας εργασίας.

2

Θεωρητικό Υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζεται το θεωρητικό υπόβαθρο της παρούσας διπλωματικής εργασίας. Στόχος του κεφαλαίου είναι να πραγματοποιήσει μια συνοπτική περιγραφή του Σημασιολογικού Ιστού και των τεχνολογιών (προτύπων) στις οποίες βασίζεται, όπως, XML, RDF, RDF Schema, OWL καθώς και στη γλώσσα ερωτημάτων SPARQL. Τέλος, αναλύονται έννοιες όπως μεταδεδομένα, διασυνδεδεμένα δεδομένα, πόροι, το ενιαίο αναγνωριστικό πόρων URI και φυσικά οι οντολογίες.

1. Σημασιολογικός Ιστός

Τα τελευταία χρόνια, ο Παγκόσμιος Ιστός έχει γνωρίσει μια κατακόρυφη αύξηση στη συχνότητα χρήσης του, με χιλιάδες νέες πληροφορίες να αποστέλλονται και να αποθηκεύονται σ' αυτόν κάθε δευτερόλεπτο. Μια τέτοια αύξηση οφείλεται στην ανάγκη των ανθρώπων για πληροφόρηση και επικοινωνία, όπου κάθε νέα πληροφορία που εμφανίζεται στον Ιστό κινεί όλο και περισσότερο το ενδιαφέρον τους. Όσες είναι οι πληροφορίες που υπάρχουν στον Παγκόσμιο Ιστό, τόσες είναι και οι καθημερινές αναζητήσεις τους, με τη χρήση κάθε είδους συνδυασμού λέξεων-κλειδιών (key-words) στις μηχανές αναζήτησης, όπως το Google Search, για την ανεύρεση πληροφορίας από τον Ιστό, ανάμεσα στην ολοένα αναπτυσσόμενη “θάλασσα” δεδομένων. Όμως, όσο και να βελτιώνονται αυτές οι μηχανές αναζήτησης, σίγουρα θα υπάρχει κι ένα μέρος της πληροφορίας που δε θα μπορέσουν ν' ανακτήσουν ή που θα εμφανίζεται εσφαλμένα και εκτός θέματος, λόγω του ότι στην πραγματικότητα, ένα μεγάλο μέρος της πληροφορίας του Παγκόσμιου Ιστού είναι ανοργάνωτο και χωρίς καλή κατηγοριοποίηση. Και βέβαια, ρεαλιστικά, δεν μπορεί κανείς να κάνει κάτι για να αντιμετωπιστεί πλήρως αυτό το πρόβλημα.

Ο *Σημασιολογικός Ιστός* είναι μια επέκταση του διαδικτύου που ήδη υπάρχει ή μια εξέλιξή του. Οι πληροφορίες που παρουσιάζονται στο χρήστη ορίζονται με έναν τρόπο ώστε να είναι αναγνωρίσιμες όχι μόνο από ανθρώπους αλλά και από «μηχανές» μέσα από τους ευφυείς-πράκτορες. Έχουν προκύψει πολλές μέθοδοι για την περαιτέρω διευκόλυνση της χρήσης του διαδικτύου από τους ανθρώπους, όπως η διαλειτουργική συνεργασία συστημάτων, οι αυτοματοποιημένες διεργασίες και η δυναμική επεξεργασία πληροφοριών και δεδομένων. Οι πληροφορίες που ήδη υπάρχουν στον Ιστό οργανώνονται και διασυνδέονται με τέτοιο τρόπο ώστε να μπορούν να αξιοποιούνται από οποιαδήποτε εφαρμογή, όπως ιστοσελίδες που

χρησιμοποιούνται καθημερινά από χρήστες, ακόμη και οικιακές συσκευές σύγχρονης τεχνολογίας, οι οποίες έχουν την ικανότητα να διασυνδέονται μεταξύ του ολοένα και περισσότερο. Αυτό το φαινόμενο είναι κοινώς γνωστό ως Διαδίκτυο των Πραγμάτων, ή *Internet of Things*. Ο απώτερος στόχος είναι η δημιουργία έξυπνων συστημάτων που θα διαχειρίζονται αξιόπιστα, αποδοτικά και αποτελεσματικά την πληθώρα δεδομένων και πληροφοριών που προέρχονται από το περιβάλλον.

Ο Σημασιολογικός Ιστός επινοήθηκε από τον Tim Berners-Lee που είναι ταυτόχρονα και ιδρυτής του Παγκόσμιου Ιστού (world wide web ή www). Σύμφωνα με τη θεωρία του, για την αποδοτικότερη συνεργασία μεταξύ ανθρώπων και υπολογιστών είναι απαραίτητος ο καθορισμός του πραγματικού νοήματος της πληροφορίας. Ο δημιουργός του τον φαντάστηκε ως έναν πιο ευέλικτο, ολοκληρωμένο, αυτόματο και ευπροσάρμοστο ιστό, που θα παρέχει αλληλεπιδραστική εμπειρία στον χρήστη.

Πρωταρχικός στόχος στη δημιουργία του Ιστού ήταν ο σχεδιασμός ενός περιβάλλοντος που ωθεί την ελεύθερη ανταλλαγή πληροφοριών, κάτι που όμως δεν αξιοποιούσε τις δυνατότητες που παρέχουν τα σημασιολογικά δεδομένα ως προς την ερμηνεία τους. Μόνος ρόλος των έγγραφων υπερκειμένων (hypertext documents) είναι να δείχνουν με ποιό τρόπο συσχετίζονται αυτά μεταξύ τους με απλούς συνδέσμους συνδέσμων, και δεν συμβάλλουν περαιτέρω στον προσδιορισμό κάποιου νοήματος του κειμένου που περιέχουν αυτά τα έγγραφα. Για να λυθεί αυτό με ρεαλιστικό τρόπο, υπάρχει ανάγκη σύνθεσης ειδικού λογισμικού και διαδικτυακών υπηρεσιών που μελετούν και ερμηνεύουν το περιεχόμενο με τη χρήση σημασιολογικών όρων, καταλήγοντας έτσι να μπορεί να γίνει κατανοητό και από μηχανές.

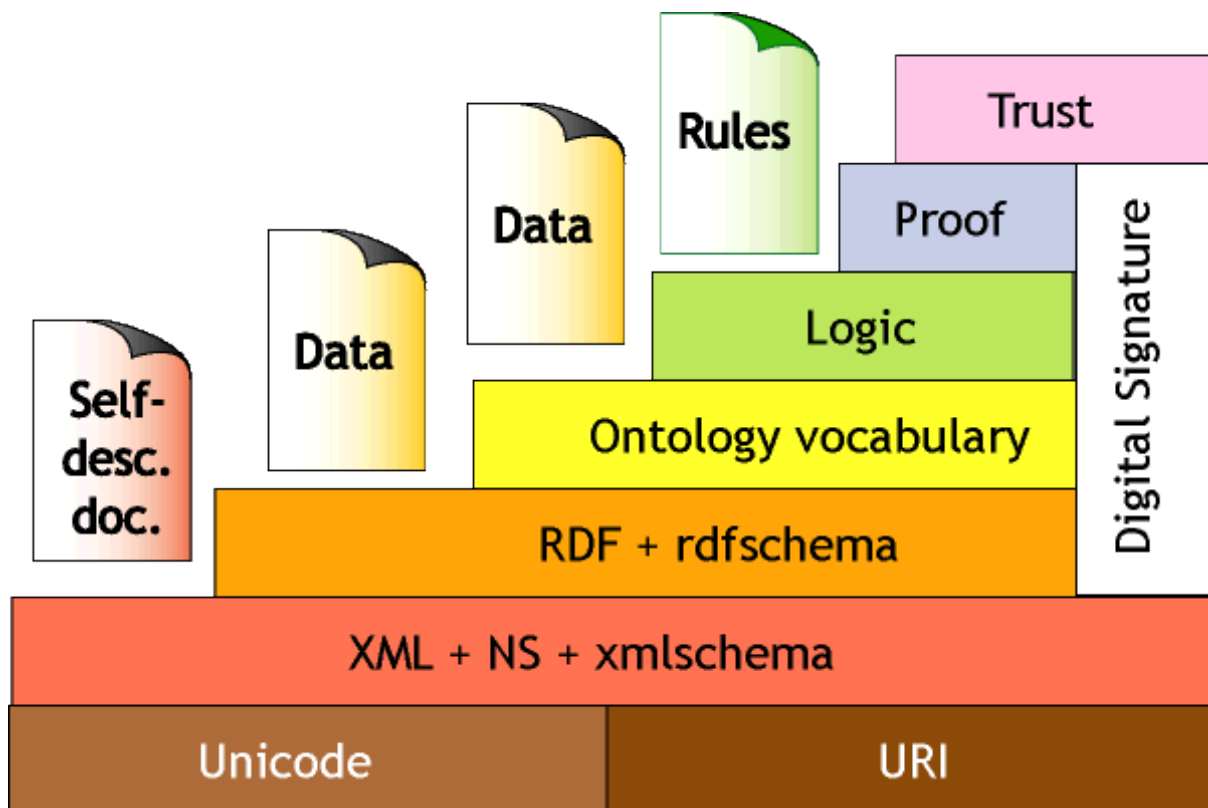
Το όραμα αυτό εκφράστηκε σε ένα άρθρο στο περιοδικό Scientific American το 2001 από τους Tim Berners-Lee, Jim Hendler και Ora Lassila. Στο άρθρο αυτό περιγράφουν το όραμα ενός κόσμου στον οποίο οι άνθρωποι δεν χάνονται πλέον μέσα στις διάφορες πληροφορίες τις οποίες αναζητούν, ούτε είναι απαραίτητο να αλληλεπιδρούν μεταξύ τους για να εκτελέσουν καθημερινές εργασίες όπως προγραμματισμό ραντεβού, εύρεση εγγράφων κλπ. Αντίθετα το διαδίκτυο θα μπορεί να εκτελεί τις πολύπλοκες και χρονοβόρες αυτές διαδικασίες. Αυτό μπορεί να επιτευχθεί με την παροχή πληροφοριών, οι οποίες είναι σχετικές με τους πόρους του διαδικτύου, καθώς και με τη χρήση εργαλείων για την επεξεργασία των πληροφοριών αυτών, με αποτέλεσμα οι εφαρμογές να είναι σε θέση να βρουν τις απαραίτητες πληροφορίες για να πάρουν τις σωστές αποφάσεις.[1]

Η ρίζα της λέξης «σημασιολογία» προέρχεται από τις λέξεις «σημαίνω», «σημάδι» ή «σημαντικός» και ετυμολογικά χρησιμοποιείται για να αποδώσει νόημα σε κάποια σύμβολα-λέξεις. Ο Σημασιολογικός Ιστός είναι λοιπόν, ένα σύστημα που επιτρέπει στις μηχανές να κατανοήσουν και να ανταποκριθούν σε πολύπλοκα αιτήματα με βάση τη σημασία τους, πράγμα που προϋποθέτει ότι οι σχετικές πηγές πληροφοριών πρέπει να είναι σημασιολογικά δομημένες. Με την εισαγωγή μεταδεδομένων στις σελίδες για την περιγραφή του τρόπου με τον οποίο αυτές συσχετίζονται, δίνεται η δυνατότητα στους υπολογιστές να αναγνώσουν τις σελίδες αυτές και σε πράκτορες λογισμικού να έχουν «έξυπνη» πρόσβαση στον Ιστό και να πραγματοποιούν λειτουργίες, τις οποίες σε άλλες περιπτώσεις θα έκαναν οι χρήστες

χειροκίνητα. Επίσης, παρέχεται ένα κοινό πλαίσιο για να διαμοιράζονται και να επαναχρησιμοποιούνται δεδομένα ανεξάρτητα από τα όρια εφαρμογών, υπηρεσιών, επιχειρήσεων ή κοινοτήτων. Η βασική του αρχή είναι ότι, σε ένα κοινό και διαλειτουργικό πλαίσιο αναπαράστασης των δεδομένων, η πληροφορία που παρουσιάζεται στο χρήστη αποκτά δομή και νόημα, δημιουργώντας ένα περιβάλλον, όπου άνθρωποι και μηχανές θα επικοινωνούν με σημασιολογικές βάσεις.

Τελικά ο στόχος του Σημασιολογικού Ιστού είναι να παρέχει τα πρότυπα και τις τεχνολογίες που απαιτούνται έτσι ώστε να προστεθεί σημασία στον ήδη υπάρχοντα Ιστό, με αποτέλεσμα την καλύτερη αξιοποίησή του και κατ' επέκταση την αυτοματοποίηση των διεργασιών. Εν ολίγοις, η προσπάθεια του Σημασιολογικού Ιστού εστιάζει στο να αναπτυχθούν νέα υπολογιστικά συστήματα τα οποία θα είναι ικανά να υποστηρίξουν τη διαδραστικότητα του διαδικτύου.

Για την καλύτερη κατανόηση της δομής του Σημασιολογικού Ιστού, ο Tim Berners-Lee έφτιαξε ένα σχεδιάγραμμα με τα πολλαπλά στρώματα από τα οποία αποτελείται, το οποίο ονομάστηκε Στοίβα του Σημασιολογικού Ιστού (Semantic Web Stack):



Η δομή του Σημασιολογικού Ιστού, όπως απεικονήθηκε απ' τον Tim Berners-Lee

- Στο πρώτο και βασικότερο επίπεδο, βρίσκεται το πρότυπο κωδικοποίησης χαρακτήρων *Unicode* για την ονομασία κειμενικής πληροφορίας σε οποιαδήποτε γλώσσα ή γραφικό

χαρακτήρα, μαζί με τα αναγνωριστικά *Uniform Resource Identifiers* (Ενιαίο αναγνωριστικό πόρων ή *URI*) για τον εντοπισμό οντοτήτων οπουδήποτε στον Ιστό, σε φυσικό και λογικό επίπεδο, για τη χρήση τους ως πόρους. Πρόσφατα, πολλές οντότητες χρησιμοποιούν τα *Internationalized Resource Identifier* (Διεθνοποιημένο αναγνωριστικό πόρων ή *IRI*) ως αναγνωριστικά τους, τα οποία είναι μια γενίκευση των *URI* που υποστηρίζουν περισσότερους γραφικούς χαρακτήρες της *Unicode*. Αυτά τα δύο πρότυπα αποτελούν τη βάση πάνω στην οποία θα κτιστεί όλο το οικοδόμημα του Σημασιολογικού Ιστού.

- Στο δεύτερο επίπεδο, βρίσκονται τα πρότυπα *eXtensive Markup Language* (Επεκτάσιμη Γλώσσα Σήμανσης ή *XML*), τα *Namespaces (NS)* και το *XML Schema*. Η γλώσσα *XML* χρησιμοποιεί το πρότυπο *Unicode* από το προηγούμενο επίπεδο για τη λογική αναπαράσταση των εγγράφων στο Σημασιολογικό Ιστό. Η επέκτασή της, *XMLSchema*, είναι γλώσσα σήμανσης και δημιουργίας πληροφοριακών πόρων με πολύπλοκη και ευέλικτη δομή, που αναπαριστά δομημένα έγγραφα και ακολουθεί ένα ορισμένο λεξιλόγιο σε μορφή *XML elements*, ώστε να ενσωματωθούν πληροφορίες και έννοιες του Σημασιολογικού Ιστού. Τα *Namespaces*, τα οποία αποτελούν σύνολα από χαρακτήρες, χρησιμοποιούνται για να οργανώσουν διαφόρων ειδών αντικείμενα, έτσι ώστε για την αναφορά τους να απαιτείται μόνο η σύντομη ονομασία τους.
- Στο τρίτο επίπεδο, βρίσκεται η *Resource Description Framework* (Πλαίσιο Περιγραφής Πόρων ή *RDF*) και το *RDF Schema*, τα κύρια πρότυπα ορισμού νοήματος και μεταδεδομένων για τους πληροφοριακούς πόρους. Συμβάλλουν στη συγγραφή αναγνώσιμων από υπολογιστή κατηγορημάτων σχετικά με τα δεδομένα και τις οντότητες του Ιστού, και στην οργάνωση αυτών σε ιεραρχικές κλάσεις (τάξεις) και ιδιότητες. Σε νεότερες ανασκευές της Στοιβάς του Σημασιολογικού Ιστού, σ' αυτό το επίπεδο προστίθεται και η γλώσσα ερωτημάτων (*Simple Protocol and RDF Query Language*) *SPARQL*, η οποία αναθέτει ερωτήματα πάνω σε δομές *RDF* για την εξόρυξη των πληροφοριών που περιέχουν.
- Στο τέταρτο επίπεδο, βρίσκεται το *λεξιλόγιο οντολογιών (Ontology vocabulary)* για τον ορισμό των λεξιλογίων που χρησιμοποιούνται στη σύνθεση οντολογιών και τη συσχέτιση των πληροφοριών που περιέχουν, και για την μετέπειτα εξέλιξή τους.
- Στο πέμπτο επίπεδο, βρίσκεται η *λογική (Logic)* που αναπαριστά κανόνες κατηγορηματικής λογικής και τα συμπεράσματα που προκύπτουν, με βάση των συσχετίσεων που υπάρχουν μεταξύ των δεδομένων.
- Στο έκτο επίπεδο, βρίσκονται οι *αποδείξεις (Proof)* οι οποίες αξιολογούν τους τρόπους χρήσης των δεδομένων του Ιστού. Σε συνδυασμό με το επόμενο επίπεδο της αξιοπιστίας διασφαλίζουν ότι αυτά τα δεδομένα έχουν τεκμηριωμένη χρήση.
- Στο έβδομο και τελευταίο επίπεδο, βρίσκεται η *αξιοπιστία (Trust)* όπου κρίνεται το πόσο αξιόπιστες είναι οι πηγές δεδομένων που χρησιμοποιούνται σε μια οντολογία. Το συγκεκριμένο επίπεδο δεν έχει αναπτυχθεί πλήρως μέχρι σήμερα, καθώς τα προαπαιτούμενά του δεν είναι σε θέση να υλοποιηθούν ακόμη. Τέτοια προαπαιτούμενα

είναι ειδικές γλώσσες προγραμματισμού, ικανές για την κατηγοριοποίηση του βαθμού αξιοπιστίας, και την αξιολόγηση διαδικτυακών πηγών, με βάση ψηφιακών υπογραφών και πιστοποιητικών.

- Το ημιεπίπεδο της *ψηφιακής υπογραφής (Digital signature)* υποστηρίζει το τρίτο (RDF και RDF Schema), το τέταρτο (λεξιλόγιο οντολογιών) και το πέμπτο επίπεδο (λογική), και αποτυπώνει την απαίτηση κάποιας επαλήθευσης της ταυτότητας χρηστών, είτε ανθρώπινων είτε υπολογιστικών, για την πρόσβαση στις λειτουργίες και τους εσωτερικούς μηχανισμούς μιας οντολογίας. Σήμερα, αυτή η απαίτηση συνδυάζεται και με την απαίτηση της κρυπτογράφησης της ψηφιακής ταυτότητας, με την επικράτηση της εποχής της πληροφορίας στον Ιστό.

Από τα παραπάνω επτά επίπεδα, τα τρία τελευταία (Logic, Proof και Trust) μέχρι σήμερα είναι σε ερευνητικό στάδιο, με τους ακριβείς ορισμούς τους να μην είναι ακόμα σαφείς, ενώ το έκτο και έβδομο επίπεδο (Proof και Trust) δεν έχουν καταφέρει να εφαρμοστούν ακόμη.

2. Πρότυπα

Καθημερινά, ο όγκος δεδομένων που παράγεται ολοένα και αυξάνεται, κάτι που απαιτεί την αναζήτηση ενός αποτελεσματικότερου τρόπου για τη διαχείρισή του. Ο ορισμός των URIs για την ταυτοποίηση πόρων δεδομένων ήταν ένα πρώτο βήμα στην καλύτερη αναγνώρισή τους, αλλά ακόμα μένει το πρόβλημα της οργάνωσης και ιδιαίτερα της συσχέτισής τους. Σημαντικότερη ακόμα είναι η δυνατότητα αναγνώρισης και επεξεργασίας όλων αυτών των στοιχείων και από ανθρώπινους χρήστες και από υπολογιστές. Γι' αυτόν το λόγο, χρειάζεται ένα συγκεκριμένο μοντέλο δεδομένων που αποτυπώνει τον τρόπο διαχείρισης μεταδεδομένων, με σκοπό την παροχή πληροφοριών σε εφαρμογές και τη δυνατότητα κατανόησης των περιεχομένων τους.

Προκειμένου ο Σημασιολογικός Ιστός να είναι προσαρμόσιμος και διαλειτουργικός στις εφαρμογές του, όπως οι μηχανές αναζήτησης, οι διαδικτυακές υπηρεσίες, τα web portals και οι intelligent agents (IA), η πληροφορία που αποτυπώνει πρέπει να παραμένει σωστά οργανωμένη όταν χρησιμοποιείται από κάθε τέτοια εφαρμογή. Για την αποτύπωση αυτής της πληροφορίας, χρησιμοποιείται μια πληθώρα γλωσσών, με τις πιο διαδεδομένες να είναι η XML (eXtensible Markup Language - Επεκτάσιμη Γλώσσα Σήμανσης), το XML Schema, το RDF (Resource Description Framework - Πλαίσιο Περιγραφής Πόρων), το RDF Schema, η OWL (Web Ontology Language - Γλώσσα Οντολογιών Διαδικτύου), η OIL (Ontology Inference Layer) και η DAML (DARPA Agent Markup Language). Τέτοιες γλώσσες βοηθούν στην ευκολότερη επεξεργασία εγγράφων από τους πράκτορες.

2.1. XML

Η XML (eXtensive Markup Language ή Επεκτάσιμη Γλώσσα Σήμανσης) είναι ένα πρότυπο που χρησιμοποιείται συχνά στη σύνθεση δεδομένων του Παγκόσμιου Ιστού. Είναι μια γλώσσα σήμανσης η οποία, με τη χρήση συγκεκριμένων κανόνων, περιγράφει δεδομένα των σελίδων του Παγκόσμιου Ιστού, προσθέτοντας ετικέτες μεταδεδομένων (metatags), αναγνώσιμες τόσο από έναν άνθρωπο, όσο και από έναν υπολογιστή, για τον προσδιορισμό των εννοιών που περιλαμβάνουν. Οι ετικέτες αυτές δεν είναι προκαθορισμένες από την XML, δηλαδή δεν υπάρχει ένα ορισμένο στατικό σύνολο ετικετών, και ο προγραμματιστής μπορεί να ορίσει όποιες θέλει χωρίς περιορισμούς. Αναλυτικότερα για τους κανόνες των ετικετών μεταδεδομένων, αυτοί ορίζονται από το XSD (XML Schema Definition), ένα καθορισμένο σύνολο τύπων δεδομένων (datatypes) που μπορούν να πάρουν οι ετικέτες.

Η XML εμφανίστηκε για πρώτη φορά το 1996, ενώ εντάχθηκε επισήμως ως πρότυπο του W3C (World Wide Web Consortium) το 1998. Από τότε, έγινε ευρέως γνωστή στον τομέα του web development και χρησιμοποιείται ακόμα και σήμερα παράλληλα με τη συγγραφή σελίδων σε HTML και JavaScript. Η XML παρουσιάζει πολλά πλεονεκτήματα όπως η ευελιξία της, ο σαφής τρόπος μεταφοράς των δεδομένων που περιγράφει, και το εύκολο συντακτικό της. Επιπλέον, χάρη στην ανάπτυξη της ως βασικό πρότυπο του W3C, μπορεί να ενσωματωθεί σε οποιοδήποτε web framework ή πλατφόρμα, ενώ η έκδοση Unicode που υποστηρίζει επιτρέπει κάθε μέρος του συντακτικού της να μπορεί να αποτυπωθεί με οποιοδήποτε σύστημα γραφής, κάνοντάς το προσιτό στην ανάπτυξη παγκοσμιοποιημένων πληροφοριών.

Η βασική σύνταξη της XML αποτελείται από τρία στοιχεία, γνωστά ως λογικές δομές (logical structures): elements, attributes και tags. Κάθε ένα στοιχείο ορίζεται σε μια δομή XML με δική της δήλωση (declaration), τις οποίες θα περιγράψουμε παρακάτω.

Τα *elements* είναι το πιο θεμελιώδες στοιχείο της XML και περιλαμβάνει κάθε δομή που συντάσει ένα XML έγγραφο. Μια δομή element αποτελείται από tags, attributes αν υπάρχουν, και τα περιεχόμενα του tag μόνο στην περίπτωση που αυτά είναι ένα ζεύγος αρχικού-τελικού tag. Για παράδειγμα, ένα element μπορεί να είναι το `<greeting>Hello, world!</greeting>`, το οποίο περιλαμβάνει τα tags `<greeting>` και `</greeting>` και το περιεχόμενό τους *Hello, world!*.

Τα *attributes* είναι οι ιδιότητες που έχει ένα στοιχείο element, οι οποίες είναι προαιρετικές στη δομή ενός XML εγγράφου και περιλαμβάνουν πληροφορίες οι οποίες μπορεί να είναι σημαντικές στην αποτύπωση του περιεχομένου του element. Η δομή τους περιλαμβάνει το όνομα του attribute σε αλφαριθμητική μορφή, το σύμβολο =, και την τιμή που έχει το attribute αυτό μέσα σε εισαγωγικά είτε “” ή “. Ένα παράδειγμα στο οποίο χρησιμεύουν τα attributes σ’ ένα element είναι το παρακάτω: το element `<person gender="male">John Doe</person>` που αποτυπώνει μια οντότητα ενός άντρα, έχει την ιδιότητα `gender="male"`. Τα attributes μπορούν να γραφούν με τη μορφή εμφωλευμένων element.

Τα *tags* είναι τα βασικά στοιχεία που αποτελούν ένα *element* και μπορεί να είναι ένα από τους δύο τύπους:

- Ζεύγη αρχικών-τελικών *tags* (*start and end tags*), όπου το αρχικό *tag* (*start-tag*) αρχίζει με το σύμβολο `<`, ακολουθεί το όνομα του *tag* αλφαριθμητικού τύπου με οποιοδήποτε σύστημα γραφής και τελειώνει με το σύμβολο `>`. Το τελικό *tag* (*end-tag*) αρχίζει με τη συμβολοσειρά `</`, ακολουθεί το όνομα του *tag*, το ίδιο με αυτό του αντίστοιχού του *start-tag*, και τελειώνει με το σύμβολο `>`. Ανάμεσα στα δύο αυτά *tags* υπάρχει το περιεχόμενό τους σε αλφαριθμητική μορφή. Ένα *start-tag* δεν μπορεί να υπάρχει χωρίς το αντίστοιχο *end-tag* του (δηλαδή με ίδια ονομασία) και αντιστρόφως, κάθε ζεύγος αρχικού-τελικού *tag* πρέπει να κλείνει κάπου γι' αυτό πρέπει να δίνεται προσοχή αν υπάρχει εμφώλευση τέτοιων *tags*.
- Ένα *tag* χωρίς περιεχόμενο (*empty-element-tag*), το οποίο αρχίζει με το σύμβολο `<`, ακολουθεί το όνομα του *tag* αλφαριθμητικού τύπου με οποιοδήποτε σύστημα γραφής και τελειώνει με τη συμβολοσειρά `/>`. Όπως το όνομά του, το *tag* αυτό δεν έχει περιεχόμενο και συχνά χρησιμοποιείται για *elements* που ορίζουν απλά ένα αντικείμενο που δεν εμφανίζει κάποια επεξήγηση, αλλά διαθέτουν ως περιγραφή κάποια *attributes*. Παράδειγμα *empty-element-tag* είναι το *element* μιας εικόνας το οποίο ορίζεται ως ``.

Και για τους δύο παραπάνω τύπους *tag*, η ονομασία που περικλείεται σε κάθε ένα απ' τα *start-tag*, *end-tag* και *empty-element-tag* δεν επιτρέπεται να ξεκινά με τη συμβολοσειρά "XML", είτε με πεζά ή με κεφαλαία γράμματα, καθώς αυτή είναι μια δεσμευμένη λέξη της XML, παρόμοια με άλλες γλώσσες προγραμματισμού.

Εκτός απ' τις λογικές δομές, ένα XML έγγραφο μπορεί να περιέχει και φυσικές δομές (*physical structures*), με την πιο διαδεδομένη να είναι η δομή *entity*. Τα *entities* λειτουργούν όπως οι σταθερές τιμές άλλων γλωσσών προγραμματισμού και αποτυπώνονται ως συντομογραφίες δεδομένων, οι οποίες χρησιμοποιούνται στα περιεχόμενα ενός *element*. Χωρίζονται σε εσωτερικά (*internal*), και σε εξωτερικά (*external*). Τα *internal* αρχίζουν με τη συμβολοσειρά `<!ENTITY`, ακολουθεί το όνομα του *entity* και η τιμή αυτού του *entity* σε αλφαριθμητική μορφή, και τελειώνει με το σύμβολο `>`, ενώ τα *external* αρχίζουν με τη συμβολοσειρά `<!ENTITY`, ακολουθεί το όνομα του *entity* σε αλφαριθμητική μορφή, η λέξη *SYSTEM*, και η τιμή αυτού του *entity* η οποία είναι ένα URL ή ένα URI, και τελειώνει με το σύμβολο `>`.

Ακόμη, ένα XML έγγραφο μπορεί να περιλαμβάνει και σχόλια (*comments*) τα οποία ορίζονται με τον ίδιο τρόπο όπως στην HTML, όπου το σχόλιο περικλείεται από `<!--` και `-->`.

Η XML είναι ένα εργαλείο το οποίο επεκτείνει την HTML, την πρωταρχική γλώσσα σήμανσης περιεχομένου που χρησιμοποιεί όλος ο Παγκόσμιος Ιστός. Σκοπός της XML είναι να περιγράφει δομημένες πληροφορίες, οι οποίες επεξεργάζονται και παρουσιάζονται στη συνέχεια σε αναγνώσιμη μορφή απ' την HTML.

Παρόλο που η XML έχει την ικανότητα να περιγράφει οργανωμένες οντότητες στον Παγκόσμιο Ιστό, αδυνατεί να αποτυπώσει τις συσχετίσεις που υπάρχουν μεταξύ των οντοτήτων, κάτι που την καθιστά δύσκολη για την δικτυοποίηση και διασύνδεση πολλών οντοτήτων μεταξύ τους.

2.2. XML Schema

Το *XML Schema*, και η εφαρμοσμένη μορφή του *XML Schema Definition (XSD)*, αποτελεί ένα τυποποιημένο σύνολο περιορισμών για τα δεδομένα που μπορεί να περιγράψει ένα XML έγγραφο, ως προς τον τύπο δεδομένων που μπορεί να περιέχει κάθε element ενός εγγράφου.

Το XML Schema δημοσιεύθηκε για πρώτη φορά από το W3C το 2001, με την έκδοση 1.0 Part 1 ως απλά XML Schema, ενώ 3 χρόνια αργότερα το 2004 δημοσιεύτηκε η δεύτερη έκδοσή του, XML Schema 1.0 Part 2. Αργότερα το 2012, το W3C δημοσίευσε μια καινούρια έκδοση 1.1 για τη διόρθωση δομικών σφαλμάτων, όπου και απέκτησε τη επίσημη ονομασία του XML Schema Definition ή αλλιώς XSD. Το XSD, κατά την πρώτη του δημοσίευση, είχε ως στόχο την ενοποίηση όλων των λειτουργιών που είχαν άλλες υπάρχουσες γλώσσες σχήματος που σχετίζονταν με την XML, ώστε η χρήση σχημάτων για τη δομή XML εγγράφων να είναι πιο τυποποιημένη και κατ' επέκταση πιο εύκολη στη χρήση.

Σημαντικότερο χαρακτηριστικό της XSD είναι ότι επιτρέπει στη μηχανή να χρησιμοποιεί XML namespaces με μοναδικές ονομασίες element και attribute, καθώς και να αναγνωρίζει συγκεκριμένους τύπους δεδομένων για τα περιεχόμενα αυτών των δύο στοιχείων, όπως ακέραιοι, ημερομηνίες, ακόμη και bytes, αντί για απλό αλφαριθμητικό κείμενο.

Το XML Schema έχει ως στόχο την περαιτέρω επέκταση του XML και να προσφέρει μεγαλύτερη ευκολία στην ανάγνωση XML εγγράφων σε ό,τι έχει να κάνει με τον ορισμό των τύπων δεδομένων του. Ακόμη και με το ήδη πλούσιο σύνολο τύπων δεδομένων, το XSD επιτρέπει στον προγραμματιστή του να συντάσσει νέους σύνθετους τύπους, κάτι που έχει ως αποτέλεσμα την μείωση του χρόνου, αλλά και του κόστους ανάπτυξης XML σχημάτων.

2.3. RDF

Η *RDF* (Resource Description Framework ή Πλαίσιο Περιγραφής Πόρων) είναι ένα μοντέλο αναπαράστασης δεδομένων του Ιστού, με τη χρήση απλών δηλώσεων που περιγράφουν τις ιδιότητές τους. Αν και ακολουθεί ένα ορισμένο συντακτικό όμοιο με γλώσσας προγραμματισμού, στην πραγματικότητα είναι ένα *μοντέλο δεδομένων* (data model). Παρόμοια με την XML, σκοπός της είναι να ορίσει οντότητες με τέτοιο τρόπο ώστε να είναι αναγνώσιμες και από έναν άνθρωπο και από έναν υπολογιστή, αλλά προσθέτοντας την ικανότητα που έλλειπε απ' την XML να συσχετίζει αυτές τις οντότητες μεταξύ τους με σημασιολογικό τρόπο.

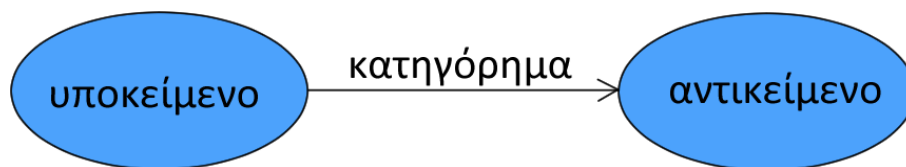
Αποτελεί τη βάση του σημερινού Σημασιολογικού Ιστού χάρη στο απλό αλλά ευέλικτο συντακτικό της. Από την αρχική της ενσωμάτωση στις συστάσεις προτύπων του W3C έχει ενσωματωθεί σε ολοένα και περισσότερες εφαρμογές του Ιστού. Το πρώτο στοιχείο του RDF είναι το Resource και είναι οτιδήποτε που μπορεί να έχει ένα μοναδικό προσδιοριστικό (URI) π.χ. σελίδες, τοποθεσίες, άνθρωποι, εικόνες. Το δεύτερο συστατικό είναι το Description και είναι τα αντικείμενα, τα χαρακτηριστικά και οι σχέσεις των πηγών. Τέλος το Framework είναι μοντέλα, γλώσσες και συντάξεις για αυτές τις περιγραφές.

Πρωτότυπα σχέδια και τεκμηριώσεις της RDF δημοσιεύτηκαν στο κοινό το 1997, με την επίσημη ενσωμάτωσή του στα πρότυπα του W3C το 1999.[2] Στα πρώτα χρόνια της ύπαρξής της η χρήση της ήταν περιορισμένη, κάτι που οφειλόταν στην εσφαλμένη αντίληψη που είχε δημιουργηθεί στην παγκόσμια κοινότητα του web development ότι σκοπεύει στην επέκταση της αποτύπωσης μεταδεδομένων, ή ακόμα ότι είναι μια βιβλιοθήκη της XML. Όμως, η χρήση της σε γνωστά εργαλεία και υπηρεσίες, όπως η υπηρεσία online ροής ειδήσεων RSS και η οντολογία FOAF, καθώς και αργότερα η ολική αναθεώρησή της το 2004, έδωσε το έναυσμα για τη μελέτη της πρακτικής χρήσης του Σημασιολογικού Ιστού με τη χρήση εφαρμογών RDF, και την μετέπειτα εφαρμογή του σε πολλούς άλλους κλάδους.[3]

Για τη σύνθεσή του ως προς τις έννοιες που περιγράφει, το πρότυπο RDF βασίστηκε ακόμη σε διάφορα άλλα πεδία στον κλάδο ανάπτυξης συστημάτων, όπως η αναπαράσταση γνώσης (knowledge representation), η διαχείριση δεδομένων (data management), οι σχεσιακές βάσεις (relational databases) και οι εννοιολογικοί γράφοι (conceptual graphs), και η τεχνητή νοημοσύνη (artificial intelligence).

Η RDF στην πραγματικότητα, είναι απλά ένα πρότυπο μοντέλο δεδομένων με συγκεκριμένους κανόνες, πάνω στην οποία βασίζονται διάφορες άλλες μορφές *σειριοποίησης* (serialization formats), όπως η Turtle, η N-Triples, RDF/JSON, RDF/XML. Η κάθε μία μορφή σειριοποίησης διαχειρίζεται τους κανόνες που ορίζει η RDF διαφορετικά, και παράγουν διαφορετικά αποτελέσματα ως προς τον τρόπο της δόμησης των δεδομένων τους. Με άλλα λόγια, η RDF είναι κάτι ανάλογο του ψευδοκώδικα, που χρησιμοποιείται ως βάση για την παραγωγή νέων και εξελιγμένων γλωσσών προγραμματισμού. Υπάρχουν πολλά στοιχεία που συγκροτούν το πρότυπο RDF, τα οποία ονομάζονται συλλογικά όροι RDF (RDF terms). Αυτοί οι όροι, που περιγράφονται αναλυτικά παρακάτω, είναι: οι τριάδες ή τριπλέτες RDF (triples), τα URIs, οι απλές τιμές (literals), οι γράφοι RDF (RDF graphs) και οι κενοί ή ανώνυμοι κόμβοι (blank nodes).

Βασικό στοιχείο της σύνταξης στο μοντέλο της RDF για την περιγραφή πληροφορίας, είναι η *πρόταση* (statement), συχνά αποκαλούμενη και *τριάδα* ή *τριπλέτα* (triple). Ο όρος τριάδα ή τριπλέτα προέρχεται από τον τρόπο με τον οποίο δομείται μια πρόταση RDF, η οποία χρησιμοποιεί παρόμοια στοιχεία με αυτά της βασικής δομής προτάσεων της ανθρώπινης γλώσσας, υποκείμενο (subject), κατηγορημα (predicate) και αντικείμενο (object). Η χρήση των τριάδων RDF δίνει τη δυνατότητα στον προγραμματιστή να περιγράψει με λεπτομέρεια τις οντότητες ενός χώρου καθώς και τις συσχετίσεις αυτών.[4]



Μια δήλωση RDF

Αυτά τα τρία μέρη αποκαλούνται επίσης και πόροι (resources), δηλώσεις (statements) και ιδιότητες (properties) αντίστοιχα, όπου οι δηλώσεις συνδέουν πόρους με τις ιδιότητες (πχ. η οντότητα ενός ανθρώπου συσχετίζεται με το κατηγορήμα *έχει όνομα* (*hasName*) για τη διατύπωση της ιδιότητας του ονόματός του).

Το *υποκείμενο* ή *πόρος* είναι η βασική οντότητα που περιγράφεται απ' την τριπλέτα και αναπαρίσταται απ' το URI του πόρου του. Ένα υποκείμενο μπορεί να είναι οποιαδήποτε οντότητα, αρκεί να έχει ένα αντίστοιχο URI που την αντιπροσωπεύει στον Ιστό.

Το *κατηγορήμα* ή *δήλωση* είναι ο τύπος της ιδιότητας που έχει το υποκείμενο, που προδιαγράφει τον τρόπο με τον οποίο συνδέεται το υποκείμενο με κάποιο αντικείμενο ως ιδιότητά του. Όπως τα υποκείμενα, αναπαρίσταται από ένα δικό του URI, το οποίο όμως προέρχεται από κάποιο ορισμένο λεξιλόγιο (vocabulary) και αντιπροσωπεύει μια συσχέτιση παρά μια μοναδική οντότητα (πχ. το κατηγορήμα *hasName* δεν αντιπροσωπεύει κάποια συγκεκριμένη οντότητα). Τα λεξιλόγια είναι καθορισμένα σύνολα URIs που συμπεριλαμβάνονται σε υπάρχουσες βιβλιοθήκες, και χρησιμοποιούνται για την ανάθεση νοήματος στις συσχετίσεις τριάδων. Παραδείγματα κατηγορημάτων είναι ποια είναι η ημερομηνία δημοσίευσης ενός εγγράφου ή ποια σχέση έχει ένας άνθρωπος με έναν άλλο άνθρωπο (πχ. ένα άτομο να είναι ο εργοδότης ενός άλλου).

Το *αντικείμενο* ή *ιδιότητα* είναι μια οντότητα με την οποία συσχετίζεται το υποκείμενο και του ορίζει ένα γνώρισμα μέσω ενός κατηγορήματος, δηλαδή είναι η τιμή του κατηγορήματος. Μπορεί να είναι ενός από δύο είδη, είτε μία τιμή οποιουδήποτε τύπου (literal), όπως συμβολοσειρά (string), αριθμός (int, double, ...), ημερομηνία (date, dateTime, ...) κλπ, είτε μια άλλη οντότητα ή πόρος αντιπροσωπευόμενη από το δικό της URI, σαν τα υποκείμενα.

Με τη διασύνδεση όλο και περισσότερων διαφορετικών οντοτήτων μεταξύ τους, χρησιμοποιώντας την δομή των τριάδων που περιγράψαμε παραπάνω, σχηματίζεται ένα δίκτυο κατευθυνόμενων γράφων, το οποίο ονομάζεται *γράφος RDF* (RDF graph). Σ' έναν τέτοιο γράφο, οι οντότητες υποκειμένων και αντικειμένων αναπαριστώνται από κόμβους του γράφου, ενώ τα κατηγορήματα παίρνουν το ρόλο των κατευθυνόμενων συνδέσμων που αναπαρίστανται από τις ακμές που συνδέουν τους κόμβους. Στην ουσία τους, οι γράφοι RDF είναι ομάδες από τριπλέτες συνδεδεμένες μεταξύ τους, όπου είτε ένα υποκείμενο έχει παραπάνω από ένα αντικείμενα και αντίστροφα, είτε το αντικείμενο μιας τριάδας γίνεται υποκείμενο μιας άλλης. Οι τριάδες ενός γράφου RDF αποτελούν ένα ξεχωριστό σύνολο το οποίο επεξεργάζεται ενοποιημένα, δηλαδή όλες οι τριάδες επεξεργάζονται ταυτόχρονα, και όχι σειριακά. Οι RDF γράφοι μπορούν να συνδυάσουν δεδομένα από πολλαπλούς γράφους και πηγές. Μ' αυτόν τον

τρόπο, ο γράφος διευκολύνει την αναζήτηση δεδομένων και οντοτήτων μέσω της περιήγησης στους κόμβους και τις ακμές του, καθώς η διασύνδεση πολλαπλών μεμονωμένων γράφων, αποτελεί έναν δυναμικό καθολικό χώρο δεδομένων (global data space).

Κενοί ή ανώνυμοι κόμβοι (blank nodes) είναι οι οντότητες, είτε υποκείμενα ή αντικείμενα, ενός γράφου RDF οι οποίες δεν αντιπροσωπεύονται από κάποιο URI ή τιμή literal μέσα στην τριάδα που ανήκουν. Ο λόγος που ονομάζονται ανώνυμοι είναι ότι δεν κατέχουν κάποια “ονομασία” URI, και η μόνη ονομασία που τους αποδίδεται σε μια τριάδα είναι μόνο τοπική, και δεν μπορεί να χρησιμοποιηθεί πέρα από τον γράφο (ή το αρχείο) στον οποίο ορίστηκαν. Το περιεχόμενό τους είναι αυθαίρετο και μπορεί να χρησιμοποιηθεί στις περιπτώσεις όπου μια τριπλέτα δεν περιγράφει μια συγκεκριμένη οντότητα. Στις περισσότερες μορφές σειριοποίησης της RDF, τέτοιοι κόμβοι ορίζονται με τη συμβολοσειρά _: πριν το τοπικό όνομα του κόμβου (πχ. _:bnode).

Στο πρότυπο RDF απουσιάζει η έννοια του χρόνου. Αυτό έχει ως αποτέλεσμα ότι οι τριάδες RDF δεν έχουν κανέναν χρονικό προσδιορισμό, αλλά και μόνο ισχύουν τη στιγμή της δήλωσής τους. Ως εκ τούτου λοιπόν θεωρούμε έναν γράφο RDF ως ένα χρονικό στιγμιότυπο (snapshot) δεδομένων RDF. Αν είναι απαραίτητο να γίνει προσθήκη χρονικής πληροφορίας, μπορεί να χρησιμοποιηθεί το κατάλληλο λεξιλόγιο καθώς και πρόσθετες τριάδες.

Ο λόγος που δεν μπορούμε να αναπαραστήσουμε τις πληροφορίες χρησιμοποιώντας την XML είναι γιατί αυτή δεν δεσμεύει ποιες λέξεις θα πρέπει να χρησιμοποιηθούν για να περιγράψουν ένα δεδομένο σύνολο εννοιών. Η XML είναι ενδεδειγμένη για την αναπαράσταση κλειστών και σταθερών πεδίων, και όχι για δυναμικούς πόρους διαδικτυακού περιεχομένου.

Πλεονεκτήματα RDF

- Η RDF έδειξε ότι μπορεί να υπερισχύσει στη σύνθεση του Σημασιολογικού Ιστού, και ενώ στα επόμενα χρόνια εμφανίστηκαν άλλα παρόμοια πρότυπα με περισσότερες λειτουργίες και χαρακτηριστικά όπως η OWL, χρησιμοποιείται πολύ συχνά ακόμα και σήμερα. Αυτό οφείλεται σε διάφορους παράγοντες:
- Η σύνταξη του μοντέλου είναι απλή με μόνο τρεις έννοιες, αλλά διαθέτει πολλές δυνατότητες ως προς τις πληροφορίες που μπορεί να αποτυπώσει και τους τρόπους που τις συσχετίζει.
- Τα δεδομένα του Ιστού αποκτούν μια νέα σημασιολογική έννοια, συνεπώς προσφέρουν διαλειτουργικότητα και ευελιξία για τη χρήση σε διάφορες εφαρμογές.
- Οι πόροι που αποτελούν ένα σχήμα RDF είναι ευέλικτοι, καθώς οι τροποποιήσεις που γίνονται σ' αυτό γίνονται με απλές προσθήκες ή αφαιρέσεις τριάδων RDF.
- Το σύνολο των πόρων που υπάρχουν σ' ένα σχήμα RDF, ακολουθεί έναν ορισμένο χώρο ονομάτων (namespace) που είναι διαφορετικό από γράφο σε γράφο. Η χρήση ξεχωριστών τέτοιων namespaces καθιστά κάθε σχήμα RDF δημόσιο και

επαναχρησιμοποιήσιμο, δηλαδή οποιοσδήποτε νέος γράφος μπορεί να το ενσωματώσει παρόμοια με μια βιβλιοθήκη.

- Χρησιμοποιώντας δεδομένα από διαφορετικές πηγές ισχυροποιούν τη δημιουργία νέων δεδομένων, κάτι που είναι πιο εύκολα εφαρμόσιμο καθώς δημόσια αποθετήρια δεδομένων που έχουν ήδη περιγραφεί με τη χρήση RDF μπορούν να συμπεριληφθούν σε νέους γράφους.
- Χρησιμοποιώντας URIs σαν μοναδικά αναγνωριστικά πόρων, τόσο για τα δεδομένα όσο και για όρους των λεξιλογίων, επιτρέπει σε οποιοδήποτε πόρο να συσχετιστεί με έναν άλλο πόρο χωρίς περιορισμούς. Διαφορετικές πηγές δεδομένων μπορούν να συσχετιστούν και συνδυαστούν μεταξύ τους για να σχηματίσουν έναν κοινό γράφο.
- Ένας RDF γράφος που βρίσκεται στον Ιστό, μπορεί να αποτελέσει αφετηρία για την εξερεύνηση του συγκεκριμένου πεδίου δεδομένων, αναζητώντας τα συνδεδεμένα URI.
- Η ενσωμάτωση διαφορετικών σχημάτων και λεξιλογίων σε έναν γράφο RDF δίνει τη δυνατότητα να αυξηθεί η εκφραστικότητα στην περιγραφή των πόρων του.
- Ένα σχήμα RDF μπορεί να μετατραπεί πολύ απλά σε ένα γράφο συνδεδεμένων πόρων, αλλά και το αντίστροφο.[5]

2.4. RDF Schema

Μετά τη δημιουργία της RDF, διαπιστώθηκε ότι προκειμένου ο υπολογιστής να αναγνωρίζει καλύτερα και να βγάζει πιο ορθά αποτελέσματα σχετικά με τις οντότητες και τις δηλώσεις που επεξεργάζεται, χρειάζεται να έχει μια πιο σαφή περιγραφή τους. Αυτό έδωσε το κίνητρο στη δημιουργία του προτύπου *RDF Schema* (Resource Description Framework Schema ή *RDFS*), μια προκαθορισμένη δομή από τύπους, ιδιότητες και κλάσεις για χρήση της παράλληλα με την RDF, και έχει σκοπό την επέκταση των δυνατοτήτων της ως προς την πιο συγκεκριμένη αποτύπωση πόρων.

Το RDFS είναι ένα πρότυπο που δημοσιεύτηκε από το W3C το 2002, τρία χρόνια μετά την αρχική ενσωμάτωση του RDF στα πρότυπα του W3C. Η ενημερωμένη και τελειοποιημένη έκδοσή του, 1.1, εκδόθηκε το 2014, η οποία ορίστηκε ως τελική της έκδοση.[6]

Όπως αναφέρθηκε παραπάνω, το πρότυπο RDFS αποτελείται από ένα σύνολο ειδικών κλάσεων και ιδιοτήτων, για την περιγραφή οντοτήτων που έχουν παρόμοιες λειτουργίες συλλογικά με τη μορφή κλάσεων, και γενικευμένων δηλώσεων σχετικά με τις ιδιότητες που έχει κάθε μέλος μιας τέτοιας κλάσης. Οντότητες που ανήκουν σε κάποια κλάση αναφέρονται ως *στιγμιότυπα* αυτής της κλάσης. Οι κλάσεις οντοτήτων αλλά και οι ιδιότητες μπορούν επίσης να οργανωθούν και να διαβαθμιστούν περαιτέρω με ιεραρχικό τρόπο, με τη χρήση ειδικών συσχετίσεων. Με τη χρήση της υπάρχουσας δομής αναπαράστασης πληροφορίας που περιγράφει η RDF, δίνεται ένα πιο σύνθετο και πιο συγκεκριμένο νόημα σε κάθε στοιχείο μιας

τριπλέτας RDF. Διάφοροι τρόποι στους οποίους συμβάλει η RDFS στο σχεδιασμό γράφων RDF είναι η περιγραφή συγκεκριμένου τύπου ιδιότητας, η ιεράρχηση οντοτήτων και η περαιτέρω μοντελοποίηση των στοιχείων που περιβάλλει μια οντότητα, πέρα από τη συσχέτισή της με άλλες οντότητες.

Παρακάτω, περιγράφονται αναλυτικότερα οι ιδιότητες και οι συσχετίσεις του λεξιλογίου της RDFS:

- *rdf:type*: μια ιδιότητα που κατευθύνει μια οντότητα ή ένα αντικείμενο στη συγκεκριμένη κλάση του. Κάποιες φορές χρησιμοποιείται και η συντομογραφία του “a”, ιδιαίτερα όταν αυτή η οντότητα βρίσκεται στο ίδιο namespace γράφου μαζί με την κλάση
- *rdfs:domain*: ορίζει το υποκείμενο ως πεδίο ορισμού για την απόδοση κάποιου αντικειμένου ως ιδιότητα
- *rdfs:range*: ορίζει ποια κλάση ή τύπος δεδομένων μπορεί να έχει ως εύρος αντικειμένων του ένα υποκείμενο
- *rdfs:subClassOf*: ορισμός της ιεράρχησης μεταξύ κλάσεων, όπου μια υποκλάση υιοθετεί πληροφορίες της μητρικής της κλάσης
- *rdfs:subPropertyOf*: ορισμός της ιεράρχησης μεταξύ ιδιοτήτων, όπου μια υπο-ιδιότητα υιοθετεί πληροφορίες της μητρικής της ιδιότητας
- *rdfs:label*: μια συμβολοσειρά η οποία προσφέρει την ονομασία μιας οντότητας με τρόπο αναγνώσιμο προς τον άνθρωπο. Ιδιαίτερα χρήσιμη σε περιπτώσεις όπου ένας πόρος αναφέρεται με κάποια ασαφή ονομασία στον RDF γράφο
- *rdfs:comment*: χρησιμοποιείται για τη διασαφήνιση ή περιγραφή οποιασδήποτε οντότητας μιας τριπλέτας, η οποία είναι αναγνώσιμη μόνο από ανθρώπους

Παρακάτω, περιγράφονται αναλυτικότερα οι κλάσεις που παρέχει η RDFS, και πώς σχετίζονται ιεραρχικά μεταξύ τους:

- *rdfs:Resource*: η πρωταρχική κλάση στη δομή της RDFS, η οποία αντιπροσωπεύει έναν οποιονδήποτε πόρο
- *rdfs:Class*: η υπερκλάση μέσα στην οποία περιλαμβάνεται κάθε κλάση των πόρων
- *rdf:Property*: ένα στιγμιότυπο της Class, μια κλάση που αποτυπώνει ολόκληρο το σύνολο των ιδιοτήτων ενός πόρου
- *rdfs:Literal*: αντιπροσωπεύει μια προκαθορισμένη τιμή οποιουδήποτε τύπου, ανεξάρτητη από τους πόρους με URI ταυτοποίηση. Ο τύπος που παίρνει μια τιμή Literal περιγράφεται απ’ την κλάση Datatype

- *rdfs:Datatype*: ένα στιγμιότυπο και υποκλάση της *Literal*, που περιγράφει τον τύπο που έχει μια τιμή *Literal*. Τέτοιος τύπος μπορεί να είναι συμβολοσειρά, αριθμός, ημερομηνία κλπ
- *rdfs:langString*: ένα στιγμιότυπο της κλάσης *Datatype*, το οποίο ορίζει τη γλώσσα στην οποία είναι γραμμένη ένα *Literal* τύπου συμβολοσειράς
- *rdfs:HTML*: ένα στιγμιότυπο της κλάσης *Datatype*, το οποίο ορίζει *Literals* τύπου HTML, δηλαδή έγγραφα ιστοσελίδων
- *rdfs:XMLLiteral*: ένα στιγμιότυπο της κλάσης *Datatype*, το οποίο ορίζει *Literals* τύπου XML, δηλαδή έγγραφα που μπορεί να περιλαμβάνουν metadata ή άλλη δομημένη πληροφορία

Ακόμη, ένα παράδειγμα εφαρμογής της RDFS σε έναν απλό γράφο RDF είναι το παρακάτω:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix example: <http://example.org/Example#> .

example:Animal rdf:type rdfs:Class .
example:Canine rdf:type rdfs:Class ;
                rdfs:subClassOf example:Animal .
example:Dog rdf:type rdfs:Class .
example:hasGenus rdf:type rdf:Property ;
                rdfs:domain example:Canine ;
                rdfs:range example:Dog .
```

2.5. Μορφές σειριοποίησης RDF

Όπως έχει ήδη αναφερθεί ένας RDF γράφος είναι μια εικονική αναπαράσταση ενός συνόλου RDF τριπλετών συνδεδεμένων μεταξύ τους. Για να γίνει δημοσίευση ενός τέτοιου γράφου στον Παγκόσμιο Ιστό πρέπει να προηγηθεί η σειριοποίησή του, δηλαδή να μεταφέρουμε τις τριπλέτες σε κάποιο αρχείο, γραμμένο με μια συγκεκριμένη σύνταξη (RDF syntax). Το RDF, ακόμη, δεν είναι μια γλώσσα αλλά ένα γενικό μοντέλο δεδομένων με κανόνες για την περιγραφή προτάσεων RDF και ως εκ τούτου δεν υπάρχει μια μοναδική μορφή σειριοποίησής του. Το W3C έχει αναγνωρίσει ως πρότυπα τρεις τρόπους σειριοποίησης, το RDF/XML, το RDFa (RDF annotation) και το N-Triples. Μαζί με αυτά υπάρχουν και άλλες μη προτυποποιημένες μορφές που χρησιμοποιούνται συχνά, λόγω του ότι καλύπτουν συγκεκριμένες ανάγκες, όπως η Turtle και η N3 (Notation 3).

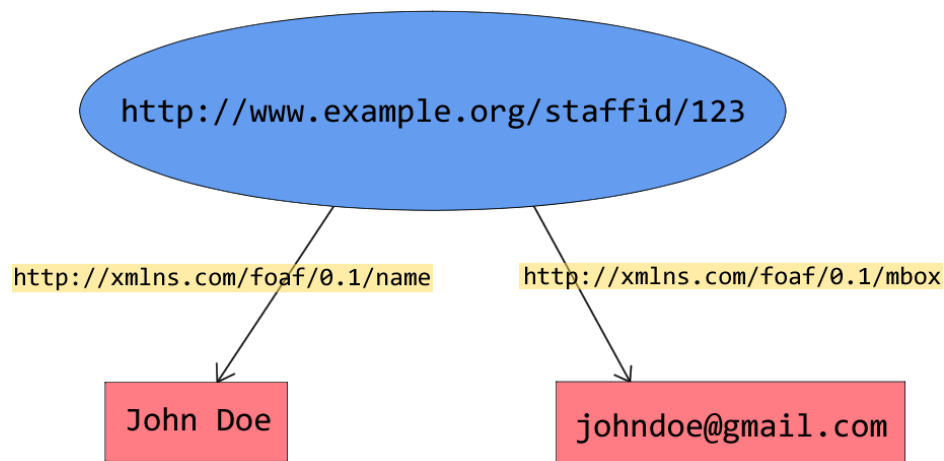
- *RDF/XML* (σύνταξη βασισμένη στην XML)
- *RDFa* (για ενσωμάτωση στην HTML και XML)
- *JSON-LD* (σύνταξη βασισμένη στη JSON)

- *Οικογένεια Turtle*: N-Triples, N-Quads, Turtle, N3, TriG

RDF/XML

Η *RDF/XML* αποτέλεσε την αρχική σύσταση του W3C για την σειριοποίηση των RDF γράφων, καθώς ήταν η μόνη διαθέσιμη σύνταξη την εποχή που προτάθηκε το πρότυπο RDF. Τον πρώτο καιρό, ήταν η ιδανική λύση καθώς μπορούσε να αξιοποιήσει όλες τις δυνατότητες της XML. Ωστόσο αργότερα περιορίστηκε η χρήση της, λόγω του ότι η συγκεκριμένη σύνταξη θεωρείται δύσκολη στην εκμάθηση και κατανόηση από τον άνθρωπο και ταυτόχρονα εμφανίστηκαν νέες μορφές σειριοποίησης με παρόμοιες και πολλές φορές περισσότερες δυνατότητες.

Π.χ. έχουμε τον παρακάτω γράφο:



με τη μορφή RDF/XML ο γράφος του παραπάνω σχήματος θα περιγραφεί ως εξής:

```
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rdf:Description
rdf:about="http://www.example.org/staffid/123">
    <foaf:name>John Doe</foaf:name>
    <foaf:mbox
      rdf:resource="mailto:johndoe@gmail.com"/>
  </rdf:Description>
</rdf:RDF>
```

RDFa

Το *RDFa* είναι ένας τρόπος σειριοποίησης, ειδικότερα σχεδιασμένος για την ενσωμάτωση τριάδων RDF μέσα σε HTML έγγραφα. Συγκεκριμένα, οι τριάδες ενσωματώνονται μέσα σε κόμβους DOM (Document Object Model), και αυτό επιτρέπει οι

μηχανές αναζήτησης να μπορούν να συγκεντρώνουν τα δεδομένα των τριάδων στον Ιστό και να τα χρησιμοποιούν με τέτοιο τρόπο ώστε να εμπλουτίζουν τα αποτελέσματα των ερωτημάτων που τίθενται. Ακόμη, ήδη υπάρχοντα δεδομένα μπορούν να οριστούν ως RDFa δεδομένα με αλλαγές στον HTML κώδικα, ώστε να παρουσιάζονται ως δομημένα δεδομένα στον Ιστό.

JSON-LD

Η σύνταξη *JSON-LD* είναι μια μορφή σειριοποίησης της RDF η οποία παρουσιάζει τις τριάδες RDF σε μορφή αντικειμένων JSON. Η JSON υποστηρίζεται από πολλές σύγχρονες γλώσσες προγραμματισμού και frameworks, και αντικείμενα που δημιουργούνται με αυτήν τη σύνταξη είναι εύκολα αναγνώσιμα τόσο απ' τους ανθρώπους όσο και απ' τους υπολογιστές. Ακόμη, χάρη στην ολοένα και πιο διαδεδομένη χρήση της JSON σε διάφορους τρόπους εφαρμογής, η JSON-LD ενσωματώθηκε στις συστάσεις της W3C το 2020. Η σύνταξη της JSON-LD φέρνει έναν εύχρηστο αλλά και μοντέρνο τρόπο σειριοποίησης μιας ιδιαίτερα παλιάς σύνταξης τριπλετών όπως η RDF, και μια εφαρμογή μπορεί εύκολα να εντάξει σημασιολογικά δεδομένα στη δομή της, ακόμη και αν αυτά συντίθενται με τη χρήση μιας αυτοματοποιημένης διαδικασίας.[7]

N-Triples

Η *N-Triples* αποτελεί την πιο απλή μορφή σειριοποίησης της RDF, με μια απλή σύνταξη, εύκολη στην ανάλυση και στην ανάπτυξη σχημάτων. Κάθε γραμμή σε ένα N-Triples αρχείο αντιπροσωπεύει μία ενιαία δήλωση που χρησιμοποιεί τη σύνταξη της RDF στην πιο απλή της μορφή. Ξεκινά με ένα υποκείμενο (subject), ένα κατηγορημα (predicate) και ένα αντικείμενο (object), που εκφράζονται με απόλυτα URI και περικλείονται σε εισαγωγικά < και >, και τερματίζεται με μια τελεία. Ακολουθώντας τη σύνταξη RDF, οι κενοί κόμβοι (blank nodes) και οι απλές τιμές (literals) δεν εκφράζονται με κάποιο URI. Τα υποκείμενα και τα αντικείμενα που είναι κενοί κόμβοι αναπαριστώνται σαν `_:name`. Οι απλές τιμές είδους string μπορούν προαιρετικά να καθορίσουν και τη γλώσσα τους με τη δήλωση `@lang` μετά την τιμή literal, όπου *lang* είναι ένας κωδικός που ακολουθεί το μοντέλο ISO 639 language. Ακόμη τα literals έχουν τη δυνατότητα να προσφέρουν συγκεκριμένες πληροφορίες ως προς τον τύπο δεδομένων που έχουν με τη δήλωση `^^type`, όπου *type* είναι ένας τύπος του προτύπου XSD. Ο λόγος που απαιτείται η χρήση απόλυτων URIs για τις οντότητες, είναι ότι η N-Triples δεν υποστηρίζει την αναφορά σε prefixes άλλων οντολογιών, ώστε να παραμένει όσο το δυνατόν πιο απλή μορφή της σύνταξής της.[8]

Π.χ. ο γράφος του σχήματος που είδαμε στο κεφάλαιο RDF/XML σε μορφή N-Triples θα αναπαρασταθεί ως εξής:

```

<http://www.example.org/staffid/123>
  <http://xmlns.com/foaf/0.1/name> "John Doe" .

<http://www.example.org/staffid/123>
  <http://xmlns.com/foaf/0.1/mbox> <mailto:johndoe@gmail.com>
.

```

Notation3

Η *Notation 3 (N3)* είναι μια μορφή σειριοποίησης της RDF σχεδιασμένη με γνώμονα την ανθρώπινη αναγνωσιμότητα. [9] Προσθέτει μια φόρμα μεταβλητών, λογικών υλοποιήσεων και συναρτησιακών προθεμάτων, καθώς προμηθεύει σύμφωνα με τη σύνταξη των εκφράσεων ένα εναλλακτικό RDF/XML. Η N3 λύνει το μεγαλύτερο πρόβλημα της N-Triples, την επανάληψη των ίδιων URIs πολλές φορές, τόσο με τη χρήση προθεμάτων *@prefix* για τον ορισμό οντολογιών που χρησιμοποιούνται στο γράφο, όσο και με τη συγχώνευση τριπλετών ανάλογα με το ποιο στοιχείο της επαναλαμβάνεται. Αυτά τα συντακτικά μέρη αναλύονται παρακάτω.

Για τη δήλωση οντολογιών που χρησιμοποιούνται ως βιβλιοθήκες, χρησιμοποιείται το πρόθεμα *@prefix* με μια συντομογραφία της οντολογίας αυτής ως χώρο ονόματος (namespace), ακολουθούμενη απ' το URI της οντολογίας. Υποκείμενα και κατηγορήματα τριπλετών μπορούν να επαναχρησιμοποιηθούν με τη χρήση των συμβόλων ; και , αντίστοιχα στο τέλος κάθε δήλωσης. Ειδικότερα, ένα υποκείμενο με παραπάνω από ένα κατηγορήματα αποτυπώνεται με τη δήλωσή του μόνο μια φορά, ακολουθούμενο από κάθε κατηγορήμα του (με το αντίστοιχο αντικείμενο) σε νέα γραμμή, χωρισμένα με το σύμβολο “;” για κάθε τριάδα. Με παρόμοιο τρόπο αποτυπώνεται και ένα κατηγορήμα με παραπάνω από ένα αντικείμενα, όπου δηλώνεται το υποκείμενο και κατηγορήμα μόνο μια φορά, ακολουθούμενο από κάθε αντικείμενο χωρισμένο με το σύμβολο “,” για κάθε τριάδα. Παραδείγματα και των δύο μορφοποιήσεων ακολουθούν παρακάτω:

```

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://www.example.org/staffid/123> foaf:name "John Doe" .
<http://www.example.org/staffid/123> foaf:mbox
mailto:johndoe@gmail.com .

```

ή αλλιώς

```

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://www.example.org/staffid/123>
                                foaf:name "John
                                Doe" ;
                                foaf:mbox
                                mailto:johndoe@gmail.com .

```

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://www.example.org/staffid/123> foaf:mbox
mailto:johndoe@gmail.com .
<http://www.example.org/staffid/123> foaf:mbox
mailto:johndoe1@gmail.com .
```

ή αλλιώς

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://www.example.org/staffid/123> foaf:mbox

                                amailto:johndoe@gmail.com ,
                                amailto:johndoe1@gmail.com .
```

Turtle

Η *Turtle* (Terse RDF Triple Language ή *TTL*) είναι μια απ' τις πιο διαδεδομένες και πιο ευρέως χρησιμοποιούμενες μορφές σειριοποίησης της RDF. Καθώς η *Turtle* αποτελεί μια βελτίωση παρόμοιων μορφών σύνταξης όπως οι N3 και η N-Triples, όλες αυτές συχνά συμπεριλαμβάνονται σε ένα σύνολο που ονομάζεται ομάδα συντάξεων της οικογένειας *Turtle* (*Turtle family syntaxes group*).

Στην αρχή εγγράφων *Turtle*, δηλώνονται σε συντομογραφίες τα namespaces των οντολογιών που χρησιμοποιούνται, με τη χρήση του προθέματος *@prefix* στην αρχή, τη συντομογραφία που θα χρησιμοποιείται στις τριάδες, και το URI της οντολογίας, όπως στην N3. Άλλη μια λειτουργία που υιοθέτησε απ' την N3, είναι η συγχώνευση των τριπλετών με τη χρήση ";" όταν ένα υποκείμενο έχει παραπάνω από ένα κατηγορήμα ή ",", όταν ένα κατηγορήμα έχει παραπάνω από ένα αντικείμενο. Ακόμη, μια λειτουργία που υιοθέτησε απ' την N-Triples είναι η χρήση όρων *@lang* για τη δήλωση της γλώσσας ενός string literal και *^^type* για τη δήλωση ενός συγκεκριμένου τύπου literal απ' τους τύπους της XSD. Μια νέα προσθήκη της *Turtle* είναι η δήλωση σχολίων παρόμοια με κοινές γλώσσες προγραμματισμού, με τη χρήση του συμβόλου #, τα οποία μπορούν να δώσουν διευκρινήσεις σ' έναν ανθρώπινο χρήστη.

Χάρη στα παραπάνω χαρακτηριστικά, τα οποία είναι επιλεγμένα ώστε να συγκεντρώνονται όλα τα δυνατά σημεία διαφόρων μορφών σειριοποίησης RDF σε μία, αποφασίσαμε να επιλέξουμε την *Turtle* ως γλώσσα συγγραφής της οντολογίας μας, η οποία περιγράφεται σε επόμενο κεφάλαιο.

2.6. OWL

Η OWL (Web Ontology Language ή Γλώσσα Οντολογίας Ιστού) είναι μια γλώσσα δημιουργίας οντολογιών. Είναι επέκταση των προτύπων RDF και RDFS, στα οποία παρέχει τη δυνατότητα ορισμού νέων κλάσεων οι οποίες δημιουργούνται από την ένωση, την τομή ή την άρνηση μιας ή και περισσότερων άλλων κλάσεων, παρόμοια με τον ορισμό μαθηματικών συνόλων. Βασίζεται στην περιγραφική λογική και έχει σχεδιαστεί για την αναπαράσταση σύνθετης γνώσης σχετικά με οντότητες ή ομάδες οντοτήτων και των σχέσεων μεταξύ τους. Σκοπός της είναι να διευκολύνει την ερμηνεία μεταξύ του περιεχομένου του ιστού, χρησιμοποιώντας λεξιλόγιο και μορφοποίηση που επιτρέπει την αυτόματη επεξεργασία της μηχανής παρόμοια με τον ανθρώπινο συλλογισμό. Το πρότυπο OWL περιγράφει την ιεραρχική οργάνωση των ιδεών σε έναν τομέα (domain), με τρόπο που μπορεί να αναλυθεί και να κατανοηθεί από το λογισμικό. Η OWL έχει περισσότερες δυνατότητες για την έκφραση εννοιών και σημασιολογίας από την XML, το RDF και το RDFS, γι αυτό το λόγο μπορεί να σχηματίσει πιο πολύπλοκες σχέσεις και ως εκ τούτου να δημιουργηθούν εφαρμογές που έχουν την ικανότητα να χρησιμοποιούν τη λογική. Παράλληλα δίνει τη δυνατότητα ορισμού πιο εκφραστικών κλάσεων (τομή, ένωση, συμπλήρωση, δήλωση ξένων κλάσεων) καθώς και ορισμού εκφραστικότερων ιδιοτήτων (μεταβατικές, λειτουργικές, συμμετρικές ιδιότητες, περιορισμούς τιμής). Η OWL αποτελεί μέρος των συστάσεων του W3C σχετικά με τον Σημασιολογικό Ιστό.

Κατά τα επόμενα χρόνια μετά τη δημιουργία της RDF στα τέλη της δεκαετίας του '90, εξετάστηκε η εύρεση μιας πιο εκτεταμένης και πιο εκφραστικής μορφής αναπαράστασης οντολογίας, η οποία ακόμη θα ήταν δομημένη από ένα πιο ενοποιημένο συντακτικό, αντίθετα με το απλό και γενικό framework που διέθετε η RDF. Έτσι, σε συνδυασμό με τις προσθήκες της RDFS, δημιουργήθηκε το σύστημα περιγραφικής λογικής DAML+OIL το 2001, το οποίο, μόλις αναπτύχθηκε στην τελική του μορφή, ενσωματώθηκε στα πρότυπα του οργανισμού W3C το 2004 και έλαβε την ονομασία OWL. Αργότερα το 2009, η W3C δημοσίευσε τη δεύτερη έκδοση της OWL, με όνομα OWL 2, η οποία κυρίως βελτίωσε την επεκτασιμότητα της σύνταξης OWL σε διάφορες εφαρμογές της.[10]

Η OWL, για την αναπαράσταση γνώσης σε μορφές οντολογιών, χρησιμοποιεί κλάσεις και ιδιότητες, ενώ μπορεί ακόμη να τις ορίσει ιεραρχικά, με μία κλάση ή ιδιότητα να περιέχεται στο σύνολο μιας άλλης. Επίσης, μπορεί να ορίσει στιγμιότυπα, τα οποία αποτελούν κανόνες για την οδήγηση ως προς την παραγωγή ονοματισμένων οντοτήτων, μέσω της γνώσης από υπάρχοντα δεδομένα. Αν και η σύνταξη της OWL έχει ως θεμέλιό της τα πρότυπα RDF και RDFS και τη δομή της σειριοποίησης RDF/XML, είναι πιο εκφραστική, καθώς σε συνδυασμό με δικές της κλάσεις, ιδιότητες και στιγμιότυπα, διαθέτει κάποιες σύνθετες ιδιότητες, οι οποίες καθορίζουν την ισότητα ή μη μεταξύ κλάσεων (οι ιδιότητες `sameAs`, `differentFrom`, `equivalentClass`).

Στην αρχή κάθε εγγράφου OWL για την περιγραφή μιας οντολογίας, υπάρχει η *κεφαλίδα* (header) του εγγράφου, μια περιοχή δήλωσης των βιβλιοθηκών που χρησιμοποιούνται, με τις ονομασίες των namespaces τους, και το αντίστοιχό τους URI. Αυτή η

περιοχή ορίζεται ως attributes ενός tag ρίζας με όνομα `rdf:RDF`, όπως το παρακάτω παράδειγμα:

```
<rdf: RDF
  xmlns: owl ="http: //www.w3.org/2002/07/owl#"
  xmlns: rdf ="http: //www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns: rdfs="http: //www.w3.org/2000/01/rdf-schema#"
  xmlns: xsd ="http: //www.w3.org/2001/XMLSchema#">
```

Ένα έγγραφο OWL μπορεί να αρχίσει με μια σειρά από statements, τα οποία αναφέρουν διάφορες πληροφορίες σχετικά με την οντολογία που περιγράφεται, και ομαδοποιούνται μέσα σ' ένα tag με όνομα `owl:Ontology`. Τέτοια statements μπορεί να είναι attributes τέτοιων tags (πχ. `owl:imports`), ή να περιέχονται ανάμεσα σ' ένα ζεύγος τέτοιων tags (πχ. `rdfs:comment`). Παρακάτω ένα παράδειγμα χρήσης του `owl:Ontology`:

```
<owl: Ontology rdf: about="">
  <rdfs: comment>An example OWL ontology</rdfs: comment>
  <owl: priorVersion
    rdf: resource="http: //www.mydomain.org/uni-ns-old"/>
  <owl: imports
    rdf: resource="http: //www.mydomain.org/persons"/>
  <rdfs: label>University Ontology</rdfs: label>
</owl: Ontology>
```

Μετά τα headers, ακολουθεί το βασικό σώμα του εγγράφου, όπου περιγράφονται οι κλάσεις (classes), οι ιδιότητες (properties) και τα στιγμιότυπα (instances). Οι κλάσεις της OWL ορίζονται με την δήλωση `owl:Class`. Αυτές μπορούν να οργανωθούν ιεραρχικά με τη χρήση της δήλωσης `rdfs:subClass`, η οποία ορίζει ότι μία κλάση είναι υπο-κλάση μιας άλλης. Όλες όμως οι OWL κλάσεις έχουν μια κοινή υπερ-κλάση, την `owl:Thing`, που εκφράζει μια γενική οντότητα, ένα πράγμα (thing). Με άλλα λόγια, κάθε κλάση που ορίζεται σε μια οντολογία είναι υποκλάση της. Το αντίστροφο της `owl:Thing` είναι η κλάση `owl:Nothing`, η οποία εκφράζει μια κενή οντότητα, το τίποτα (nothing). Σε αντίθεση όμως με την `owl:Thing`, η `owl:Nothing` δεν είναι απαραίτητο να οριστεί σε μια οντολογία. Τέλος, η δήλωση `owl:equivalentClass` ορίζει την ισοδυναμία μεταξύ δύο κλάσεων. Οι ισοδύναμες κλάσεις έχουν τις ίδιες οντότητες και χρησιμεύουν για την συγχώνευση οντολογιών, όπου κλάσεις που έχουν ισοδύναμες έννοιες εκφράζονται με διαφορετική ονομασία.

Στην OWL, παρατηρούνται δύο είδη ιδιοτήτων, οι *ιδιότητες αντικειμένων* (object properties) και οι *ιδιότητες τύπων δεδομένων* (datatype properties). Επειδή όμως δεν διαθέτει προκαθορισμένους τύπους δεδομένων όπως η επέκταση RDFS, είναι απαραίτητη η ενσωμάτωση των τύπων δεδομένων της XML. Οι ιδιότητες αντικειμένων χρησιμοποιούνται για τη σύνδεση δύο οντοτήτων μεταξύ τους, έτσι για παράδειγμα η ιδιότητα `hasParent` συνδέει μια οντότητα με μια άλλη που αντιπροσωπεύει τον πατέρα της. Αντίθετα οι ιδιότητες τύπων δεδομένων συνδέουν μια οντότητα με μια τιμή τύπου δεδομένων, όπως για παράδειγμα η ιδιότητα `hasAge` η οποία συνδέει μια οντότητα με μια τιμή που αντιπροσωπεύει την ηλικία της. Στη συνέχεια, παρατίθενται ένα σύνολο δηλώσεων σχετικών με τις ιδιότητες:

- *owl:ObjectProperty*: Δηλώνει μία ιδιότητα αντικειμένων
- *owl:DatatypeProperty*: Δηλώνει μία ιδιότητα τύπων δεδομένων
- *rdfs:subPropertyOf*: Δηλώνει ότι μία ιδιότητα είναι υποπερίπτωση μιας άλλης ιδιότητας
- *rdfs:domain*: Δηλώνει το πεδίο ορισμού μιας ιδιότητας
- *rdfs:range*: Δηλώνει το πεδίο τιμών μιας ιδιότητας
- *owl:equivalentProperty*: Δύο ιδιότητες μπορούν να δηλωθούν ως ισοδύναμες
- *owl:inverseOf*: Μία ιδιότητα μπορεί να δηλωθεί ως αντίστροφη μιας άλλης. Για παράδειγμα, αν η ιδιότητα P1 δηλώνεται ως αντίστροφη της ιδιότητας P2, και η οντότητα X σχετίζεται με μία άλλη οντότητα Y μέσω της P1, τότε και η οντότητα Y σχετίζεται με την οντότητα X μέσω της ιδιότητας P2
- *owl:TransitiveProperty*: Μία ιδιότητα που δηλώνεται ως μεταβατική. Για παράδειγμα, αν η ιδιότητα P δηλωθεί μεταβατική, και ισχύουν οι τριπλέτες X P Y και Y P Z, τότε μία μηχανή συμπερασμού (reasoner) μπορεί να συμπεράνει ότι X P Z
- *owl:SymmetricProperty*: Μία ιδιότητα που δηλώνεται ως συμμετρική. Για παράδειγμα, αν η ιδιότητα P δηλωθεί συμμετρική, και ισχύει οι τριπλέτες X P Y, τότε μία μηχανή συμπερασμού μπορεί να συμπεράνει ότι Y P X
- *owl:FunctionalProperty*: Μία ιδιότητα που δηλώνεται ως συναρτησιακή. Μια τέτοια ιδιότητα έχει το πολύ μία τιμή για κάθε οντότητα του πεδίου ορισμού της. Η δήλωση αυτή αποτελεί μία συντόμευση των περιορισμών *minCardinality 0* με *maxCardinality 1* αυτής της ιδιότητας
- *owl:inverseFunctionalProperty*: Μία ιδιότητα για την οποία δύο διαφορετικά αντικείμενα δεν μπορούν να έχουν την ίδια τιμή.

Με τη δήλωση *owl:sameAs*, δύο στιγμιότυπα μπορούν να δηλώσουν ότι είναι τα ίδια. Έτσι μπορούν να χρησιμοποιηθούν για να δημιουργήσουν διαφορετικά ονόματα που αναφέρονται στο ίδιο στιγμιότυπο. Αντιθέτως, με τη δήλωση *owl:differentFrom* δηλώνεται ότι ένα στιγμιότυπο είναι διαφορετικό από άλλα στιγμιότυπα. Η δήλωση ότι δύο στιγμιότυπα είναι διαφορετικά είναι σημαντική στη γλώσσα OWL, καθώς μία μηχανή συμπερασμού (reasoner) μπορεί να καταλήξει σε εσφαλμένα συμπεράσματα. εφόσον δεν απαιτείται κάθε στιγμιότυπο να έχει ένα μόνο όνομα.

Η δήλωση *owl:AllDifferent* ορίζει ότι ένας αριθμός στιγμιότυπων μπορούν να δηλωθούν ως διαφορετικά σε μία δήλωση κάτι που είναι ιδιαίτερα χρήσιμο όταν υπάρχουν σύνολα από ευδιάκριτα αντικείμενα και όταν θέλουμε να εισάγουμε την υπόθεση της μοναδικότητας των ονομάτων μεταξύ των αντικειμένων του συνόλου. Πάντοτε χρησιμοποιείται μαζί με τη δήλωση *owl:distinctMembers* για να δηλώσει ότι όλα τα μέλη μιας λίστας είναι ξεχωριστά και ευδιάκριτα μεταξύ τους.

2.7. SPARQL

Μετά την αρχική έκδοση του προτύπου RDF, δημιουργήθηκε και η ανάγκη για την ανάκτηση πληροφοριών μέσα στους RDF γράφους. Έτσι, λίγους μήνες μετά τη δημοσίευση της RDF, το 1998 δημιουργήθηκε το πρώτο προσχέδιο της SPARQL (Simple Protocol and RDF Query Language), και αργότερα το 2004 έγινε η επίσημη δημοσίευσή του στις συστάσεις της W3C μαζί με το πρότυπο RDF.[11]

Η SPARQL είναι μία γλώσσα που επιτρέπει την σύνταξη και την εκτέλεση ερωτημάτων για την ταυτοποίηση δεδομένων από γράφους RDF. Πολύ συχνά συγκρίνεται με τη δημοφιλέστερη γλώσσα ερωτημάτων σχεσιακών βάσεων δεδομένων, την SQL, καθώς οι δύο αυτές γλώσσες έχουν πολλά κοινά χαρακτηριστικά, από τη λογική με την οποία γίνονται ερωτήματα μέχρι το ίδιο το συντακτικό. Με άλλα λόγια, η SPARQL είναι το ακριβές ανάλογο της SQL, με τη σημαντικότερη διαφορά ότι πεδίο ορισμού της είναι οι γράφοι RDF, αντί οι σχεσιακές βάσεις δεδομένων, με μόνη διαφορά ότι δεν είναι γνωστά τα ακριβή χαρακτηριστικά που αναζητούνται. Σκοπός της είναι να επεξεργαστεί τα δεδομένα όπως είναι, παρά να εξάγει λογικά συμπεράσματα όπως μια γενική γλώσσα προγραμματισμού.

Ένα ερώτημα στη γλώσσα SPARQL συνήθως αρχίζει με τον ορισμό των προθεμάτων κάθε οντολογίας πάνω στην οποία θέλουμε να κάνουμε το ερώτημα με τη λέξη PREFIX και μετά ακολουθεί το όνομα της οντολογίας ως συντομία και το URI που της αντιστοιχεί.[12]

Αφού καθοριστούν οι οντολογίες στα προθέματα, αρχίζει το κυρίως μέρος του ερωτήματος. Υπάρχουν τέσσερις τύποι ερωτήματος στην SPARQL, οι SELECT, CONSTRUCT, ASK και DESCRIBE. Η πιο συνηθισμένη απ' αυτές η SELECT, χρησιμοποιείται για την απλή εξαγωγή δεδομένων και ακολουθείται από τη σειρά από μεταβλητές, ορισμένες με ένα ερωτηματικό "?" και το όνομα της μεταβλητής προς ανάκτηση. Η CONSTRUCT, με τη χρήση μιας τεχνητής τριπλέτας με μεταβλητές, εξάγει δεδομένα από μια οντολογία για την κατασκευή, όπως λέγεται και ο όρος, ενός νέου γράφου. Η ASK εμφανίζει μια απάντηση Αληθές ή Ψευδές σε σχέση με τις ορίζουσες μεταβλητές. Τέλος, η DESCRIBE επιστρέφει έναν γράφο με οποιαδήποτε τριπλέτα κρίνει το σύστημα της SPARQL ότι αντιστοιχεί με τις ορίζουσες μεταβλητές. Ακόμη, σ' αυτό το σημείο ορίζεται ποιες μεταβλητές πρέπει να εμφανιστούν μόνο μια φορά, με τη χρήση του όρου DISTINCT.

Μετά τη δήλωση του τύπου ερωτήματος, ακολουθεί προαιρετικά ένα όρισμα FROM σε περίπτωση που το ερώτημα έχει ως πεδίο αναζήτησης έναν συγκεκριμένο υπογράφο, και μετά η λίστα από τριπλέτες που αναζητούνται σαν μοτίβα μέσα σε αγκύλες ορισμένη απ' την λέξη WHERE. Οποιοδήποτε μέρος κάθε τριπλέτας μπορεί να αντικατασταθεί από μια μεταβλητή που ορίστηκε κατά την επιλογή τύπου ερωτήματος (πχ SELECT, DESCRIBE, κλπ), και το τελικό αποτέλεσμα χρησιμοποιεί τις ανάλογες τιμές που αντιστοιχούν σ' αυτές τις μεταβλητές. Ο διαχωρισμός των τριπλετών αυτών, επίσης, ακολουθεί τη δομή της RDF, ως προς την χρήση τελείας "." για τριπλέτες με ίδιο υποκείμενο και κόμματος "," για τριπλέτες με μια σειρά από αντικείμενα. Κάθε τριπλέτα της λίστας θεωρείται ότι σχετίζεται με το λογικό AND, όπου κάθε μία απ' αυτές πρέπει να ισχύει ταυτόχρονα προκειμένου να αποτυπωθεί ένα αποτέλεσμα. Το λογικό OR σε ερωτήματα SPARQL αντιπροσωπεύεται από την εμφώλευση

τριπλετών μέσα σε όρισμα OPTIONAL, οι οποίες, όπως εννοείται απ' την ονομασία, είναι προαιρετικές στο τελικό αποτέλεσμα και φέρουν αποτέλεσμα ανεξάρτητα των υπολοίπων τριπλετών. Ένα ακόμη όρισμα που μπορεί να εμφωλευτεί μέσα στη λίστα τριπλετών είναι το FILTER, το οποίο επιτρέπει τον περιορισμό τιμών που εμφανίζουν τα αποτελέσματα του ερωτήματος με τη χρήση κανονικών εκφράσεων (regular expressions).

Το τελικό μέρος ενός ερωτήματος SPARQL είναι προαιρετικό, και απλά έχει ως ρόλο την περαιτέρω μορφοποίηση των έτοιμων αποτελεσμάτων. Μερικά ορίσματα του τελικού μέρους είναι το LIMIT για τον περιορισμό του αριθμού των εμφανιζόμενων αποτελεσμάτων, το ORDER BY για την κατάταξη των αποτελεσμάτων με βάση την αλφαβητική σειρά μιας μεταβλητής και το GROUP BY για την ομαδοποίηση αποτελεσμάτων με βάση μια μεταβλητή.

PREFIX ex: <http://example.org/schema>	Πρόθεμα
SELECT ?x	Τύπος ερωτήματος & μεταβλητές
FROM <http://example.org/graph.rdf>	Ορισμός ερωτούμενων τριπλέτων
WHERE { ?y ex:pred ?x }	
ORDER BY ?x	Μορφοποίηση αποτελεσμάτων

Ένα παράδειγμα χρήσης ερωτημάτων SPARQL γίνεται με τη χρήση του παρακάτω RDF γράφου όπου περιγράφονται οντότητες σε μια λίστα υπαλλήλων μαζί με μερικά χαρακτηριστικά που έχουν, όπως όνομα, γένος και e-mail:

```
@prefix foaf <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/schema#> .

ex:E5627 foaf:name "John Doe" ;
        foaf:gender "Male" ;
        foaf:homepage <http://example.org/jdoe> ;
        foaf:mbox "jdoe5267@example.org" .
ex:E9361 foaf:name "Harry Mann" ;
        foaf:gender "Male" ;
        foaf:mbox "hmann9361@example.org" .
ex:E0923 foaf:name "Mary Smith" ;
        foaf:gender "Female" ;
        foaf:birthday "07-21" ;
        foaf:mbox "msmith0923@example.org" .
```

Για την εύρεση της διεύθυνσης e-mail ενός υπαλλήλου με το όνομα Mary Smith, θέτουμε το παρακάτω ερώτημα στο σύστημα της SPARQL:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
PREFIX ex: <http://example.org/schema#> .

SELECT ?email
```

```
WHERE {
    ?person foaf:name "Mary Smith" ;
           foaf:mbox ?email .
}
```

όπου ?email είναι η μεταβλητή για το e-mail που αναζητείται, και τα μοτίβα τριπλετών ορίζονται ώστε να αναζητηθεί πρώτα το URI με το foaf:name όνομα “Mary Smith”, και μετά το αντίστοιχο αντικείμενο foaf:mbox που είναι και η μεταβλητή που αναζητείται.

Το SPARQL τέλος θα επιστρέψει την παρακάτω αντιστοιχία μεταβλητής με την τιμή (ή τιμές) που βρήκε:

email
msmith0923@example.org

3. Μεταδεδομένα

Τα *μεταδεδομένα (metadata)*, είναι ένας τύπος δεδομένων που περιγράφει ή αλλιώς εξηγεί κάποια άλλα δεδομένα, δίνοντας έτσι τη δυνατότητα να γίνει πιο εύκολη η ανάκτηση, η επεξεργασία καθώς και η διαχείριση της πληροφορίας που περιέχουν. Ο τρόπος με τον οποίο περιγράφονται αυτά τα δεδομένα, βασίζεται στην ομαδοποίηση σχετικά με τα κοινά χαρακτηριστικά που μπορεί να έχουν μεταξύ τους και τον διαχωρισμό εκείνων με ασυσχέτιστες πληροφορίες. Παρέχουν επίσης ξεχωριστά αναγνωριστικά σε μορφή πρόσθετων “ετικετών” για κάθε κομμάτι πληροφορίας και εξυπηρετούν στην αποτελεσματικότερη πρόσβαση στους πόρους που διαθέτουν, επιτρέποντας να βρεθούν σχετικά γνωρίσματα μέσω μιας απλής αναζήτησης.[12] Ένα παράδειγμα ενσωμάτωσης μεταδεδομένων, είναι αυτά που έχει ένα έγγραφο στον υπολογιστή, με διαθέσιμες πληροφορίες για τον συντάκτη του, την ημερομηνία δημιουργίας, την ημερομηνία τελευταίας τροποποίησης και το μέγεθος που κατέχει το αρχείο στο δίσκο. Εκτός από αρχεία εγγράφων σε τοπικό μέσο αποθήκευσης, μεταδεδομένα μπορούν να χρησιμοποιηθούν σε:

- Εικόνες (πχ. τόπος λήψης, άτομα που εμφανίζονται)
- Βίντεο (πχ διάρκεια)
- Υπολογιστικά Φύλλα (πχ. αριθμός σειρών και στηλών)
- Ιστοσελίδες (πχ. URL, ενσωματωμένοι σύνδεσμοι)

Τα μεταδεδομένα συχνά διακρίνονται σε υποκατηγορίες, ώστε να περιγράφεται πιο αναλυτικά η χρήση ή τα περιεχόμενά τους. Αυτές είναι:

- *Περιγραφικά μεταδεδομένα (Descriptive metadata)*, περιγράφουν ένα αντικείμενο με τέτοιο τρόπο ώστε να μπορεί να αναγνωριστεί και στη συνέχεια να ανακτηθεί. Περιλαμβάνουν στοιχεία όπως ο τίτλος, η περίληψη, ο συγγραφέας και κάποιες λέξεις-κλειδιά.

- *Δομικά μεταδεδομένα* (Structural metadata), αποτυπώνουν πιο σύνθετα αντικείμενα σε σχέση με τα συστατικά της πληροφορίας, όπως ένα κεφάλαιο που αποτελείται από ένα σύνολο ταξινομημένων σελίδων.
- *Επιχειρησιακά ή Διαχειριστικά μεταδεδομένα* (Administrative metadata) περιλαμβάνουν πληροφορίες σχετικές με τη διαχείριση των αρχείων, οι οποίες συχνά χρησιμοποιούνται σε δομές αρχείων, όπως πότε δημιουργήθηκε ένα αντικείμενο, πώς ψηφιοποιήθηκε το αρχείο, ο τύπος αρχείου του, ποιος έχει δικαιώματα προσπέλασης, και άλλα. Μεταδεδομένα αυτής της κατηγορίας συνήθως αντιπροσωπεύονται περαιτέρω με τους παρακάτω δύο τύπους:
 - *Μεταδεδομένα διαχείρισης πνευματικών δικαιωμάτων* (Rights management metadata) αφορούν τα πνευματικά δικαιώματα και τα δικαιώματα χρήσης του.
 - *Μεταδεδομένα αρχειοθέτησης* (Preservation metadata), τα οποία περιέχουν πληροφορίες για την διευκόλυνση της αρχειοθέτησης.

Καθώς ο αριθμός των πόρων που φιλοξενούνται στον Ιστό αυξάνεται συνεχώς, η οργάνωση αυτών με βάση κοινά χαρακτηριστικά που έχουν και η ανάλογη διασύνδεση αυτών σε μεμονωμένες σελίδες καθίσταται ιδιαίτερα χρήσιμη. Κατά αυτόν το τρόπο δημιουργούνται στατικές σελίδες, οι οποίες περιέχουν τα ονόματα και τις τοποθεσίες των πόρων τα οποία ενσωματώνονται στην HTML. Εντούτοις, συνηθίζεται να δημιουργούνται πλέον δυναμικές σελίδες αποτελούμενες από μεταδεδομένα που αποθηκεύονται σε βάσεις δεδομένων. Για να γίνει δε αυτόματη εξαγωγή καθώς και διαμόρφωση των πληροφοριών που περιέχουν χρησιμοποιούνται τα κατάλληλα εργαλεία λογισμικού.

Η χρήση μεταδεδομένων στις ιστοσελίδες μπορεί να είναι πολύ σημαντική. Πιθανά μεταδεδομένα που μπορούν να χρησιμοποιηθούν στον Ιστό είναι τα περιεχόμενα μιας σελίδας, όπως επίσης και λέξεις-κλειδιά οι οποίες συνδέονται με το περιεχόμενό της. Στον σύγχρονο Παγκόσμιο Ιστό η αποστολή ερωτημάτων στις μηχανές αναζήτησης χρησιμοποιεί αντιστοίχιση μεταδεδομένων με τις μετα-ετικέτες (meta-tags) που διαθέτουν τα αρχεία ιστοσελίδων στην κεφαλίδα (header) τους, επιστρέφοντάς τους ως αποτελέσματα. Κάτι τέτοιο επιτρέπει στον χρήστη να ανατρέξει όποια αποτελέσματα έχει στη διάθεσή του, και να συλλέξει όσες πληροφορίες χρειάζεται.

Τα δεδομένα διαφέρουν απ' τα μεταδεδομένα στο ότι τα πρώτα αναφέρονται στα αντικείμενα που αναζητούνται και χωρίς αυτά δεν μπορούν να υπάρξουν τα μεταδεδομένα, ενώ τα δεύτερα αποτελούν στοιχεία τα οποία περιγράφουν αυτά τα δεδομένα.

4. Διασυνδεδεμένα Δεδομένα

Ως *διασυνδεδεμένα δεδομένα* (Linked Data) ορίζεται η μέθοδος δημοσιοποίησης δομημένων δεδομένων που μπορούν να αλληλοσυνδέονται και να γίνουν πιο χρήσιμα μέσω των σημασιολογικών επερωτήσεων. Κατά αυτόν τον τρόπο δημιουργούν ένα σύνολο

διασυνδεδεμένων πληροφοριών που η μια συμπληρώνει, επεξηγεί ή επεκτείνει την άλλη. Αφορούν το πώς το διαδίκτυο μπορεί να δημιουργεί δεσμούς μεταξύ πόρων διαφορετικών πηγών. Οι πηγές αυτές μπορεί να είναι είτε πολλαπλές βάσεις δεδομένων πολλαπλών οργανισμών σε διαφορετικά μέρη του κόσμου, είτε απλών ετερογενών συστημάτων ενός οργανισμού που σε επίπεδο δεδομένων δεν διασυνδέονται με ευκολία. [13]

Το διαδίκτυο είναι δομημένο από έγγραφα (documents) HTML τα οποία είναι συνδεδεμένα με υπερσυνδέσμους (hyperlinks), αντιθέτως τα συνδεδεμένα δεδομένα χρησιμοποιούν έγγραφα τα οποία περιέχουν δεδομένα σε μορφή RDF. Τα διασυνδεδεμένα δεδομένα αναφέρονται στα δεδομένα που δημοσιεύονται στον ιστό έτσι ώστε να είναι αναγνώσιμα από υπολογιστές, να έχουν σαφές νόημα, να είναι συνδεδεμένα με άλλες συλλογές δεδομένων και να είναι δυνατή η διασύνδεσή τους με εξωτερικές συλλογές.

Το 2006 ο Tim Berners-Lee πρότεινε ένα σύνολο κανόνων που θα έπρεπε να πληρούν τα δεδομένα ώστε ο Παγκόσμιος Ιστός να μπορεί να μετατραπεί από Ιστός των Εγγράφων σε Ιστό των Συνδεδεμένων Δεδομένων:

1. *Η χρήση των URIs σαν ονόματα των οντοτήτων.* Σαν οντότητα ορίζεται οτιδήποτε υπαρκτό, (πρόσωπο, αντικείμενο, τοποθεσία κλπ). Δηλαδή οτιδήποτε έχει κάποια αξία σαν ύπαρξη. Σύμφωνα με τον ίδιο τον δημιουργό, Berners – Lee, αν δεν υπάρχουν URI τότε δεν υφίσταται Σημασιολογικός Ιστός και κατ' επέκταση Συνδεδεμένα Δεδομένα.
2. *Η χρήση HTTP URIs, έτσι ώστε να αποτελούν επισκέψιμες διευθύνσεις στον παγκόσμιο ιστό.* Αυτό σημαίνει ότι κάποιος χρήστης θα μπορεί να στείλει ένα αίτημα HTTP σε κάποιον εξυπηρετητή και εκείνος να αποκριθεί με τον ίδιο τρόπο. Ο μηχανισμός διαπραγμάτευσης του HTTP πρωτοκόλλου επικοινωνίας είναι αρκετά εξελιγμένος δίνοντας πολλές δυνατότητες στον πελάτη για την αναζήτηση περιγραφών, όπως για παράδειγμα κάποιο βάρος προτίμησης.
3. *Όταν κάποιος άνθρωπος αναζητά ένα URI πρέπει να του παρέχονται και χρήσιμες πληροφορίες με χρήση προτύπων RDF, SPARQL.* Γίνεται κατανοητή η ανάγκη δημιουργίας ενός νέου προτύπου για τη διάθεση των Δομημένων Δεδομένων (αντίστοιχο με το HTML). Ένα τέτοιο πρότυπο είναι το RDF, καθώς είναι ένα απλό μοντέλο δεδομένων που βασίζεται στη δημιουργία και επεξεργασία γράφων. Όπως και να έχει, οποιοδήποτε μοντέλο μπορεί να χρησιμοποιηθεί, με αυστηρή προϋπόθεση ότι μπορεί να αναγνωρίζεται και να επεξεργάζεται από μηχανές. Με την έννοια αναγνώριση και επεξεργασία, νοείται ότι γίνεται ευκρινή διάκριση των οντοτήτων.
4. *Να περιέχονται σύνδεσμοι σε άλλα URI's ώστε να μπορούν να εξερευνήσουν περισσότερους πόρους.* Ο συγκεκριμένος κανόνας δημιουργεί στον χρήστη ένα γράφο με όλους τους διαθέσιμους πόρους, αρχής γενομένης από τον πόρο που αναζήτησε. Δηλαδή αναφέρεται στην πραγματική έννοια του Σημασιολογικού Ιστού, που είναι η συσχέτιση των οντοτήτων. Με αυτόν τον κανόνα δίνεται αξία στις έννοιες των σχέσεων μεταξύ των οντοτήτων και η γνώση περνάει από ανθρώπινο επίπεδο σε επίπεδο κατανόησης από τις μηχανές.

5. Πόρος (Resource) & URI

Πόρος είναι οτιδήποτε μπορεί να φιλοξενηθεί ή να χρησιμοποιηθεί στον Παγκόσμιο Ιστό, το οποίο μπορεί αλληλεπιδράσει με άλλους παρόμοιους πόρους. Προκειμένου να χρησιμοποιηθεί ένας πόρος στον Ιστό, πρέπει πρώτα να είναι σε θέση να εντοπιστεί από οποιονδήποτε χρήστη, είτε είναι άνθρωπος ή υπολογιστής, και επίσης να υπάρχει κάποιο μέσο το οποίο επιτρέπει τη διαχείρισή του. Με βάση αυτές τις προϋποθέσεις δημιουργήθηκαν ειδικά πρότυπα για τη διαχείριση πόρων, με το πιο διαδεδομένο απ' αυτά, το *πρότυπο URI* (Uniform Resource Identifiers ή Ενιαίο αναγνωριστικό πόρων), το οποίο περιλαμβάνεται στις συστάσεις του W3C.[14]

Σκοπός του URI είναι ταυτοποιεί έναν πόρο και ταυτόχρονα να του αναθέτει μια μοναδική διεύθυνση ή αλλιώς ένα δικό του όνομα, ώστε να είναι δυνατή η προσπέλαση και η διαχείρισή του σε πιο σύνθετο βαθμό, με τη χρήση κάποιου διαδικτυακού πρωτοκόλλου, όπως το HTTP. Το URI ενός πόρου τον διαχωρίζει από οποιονδήποτε άλλο, ακόμη κι αν τυχαίνει να έχει ίδιο όνομα, ενώ μέσω αυτού είναι δυνατή οποιαδήποτε αλληλεπίδραση με τη θέση και το περιεχόμενό του. Για την καλύτερη ομαδοποίησή τους, κάποιοι πόροι μπορεί να αναφέρονται στην ίδια διεύθυνση URI αντί σε ξεχωριστές, κάτι που επιτρέπει, σε περιπτώσεις όπου ένας πόρος έχει επανεκδοθεί με άλλο όνομα, να συγχωνεύονται και να μην προκαλούν σύγχυση. Όμως, το αντίστροφο, δηλαδή ένας πόρος να αναφέρεται με δύο URIs, δεν είναι δυνατό.[15]

Η χρήση του URI μπορεί να επεκταθεί και σε οντότητες που αφορούν τον πραγματικό κόσμο, όπως ανθρώπους, τοποθεσίες, αντικείμενα, αόριστες έννοιες και τις σχέσεις που συνδέουν όλα τα παραπάνω μεταξύ τους. Μια τέτοια αποτύπωση φυσικών οντοτήτων συμβάλει στην επέκταση του Ιστού από το κλειστό περιβάλλον του με ψηφιακούς πόρους, στον πραγματικό κόσμο και τις οντότητες που ένας άνθρωπος έρχεται καθημερινά σε επαφή.

Σύμφωνα με τον ορισμό της W3C, η σύνταξη του URI αποτελείται από μια συμβολοσειρά (string), η οποία είναι όμοια με τη σύνταξη ενός URL του Ιστού. Η συνηθέστερη σύνταξη ενός URI αποτελείται από:

- Το όνομα του σχήματος (scheme) ακολουθούμενο από τα σύμβολα `://`, μία λέξη που αναφέρεται στην προδιαγραφή για τον ορισμό αναγνωριστικών αυτού του σχήματος. Τέτοια σχήματα που χρησιμοποιούνται πολύ συχνά είναι τα `http`, `https`, `mailto` κλπ.
- Το προαιρετικό αλλά πολύ συχνά χρησιμοποιούμενο όνομα του φιλοξενητή, ο οποίος μπορεί να είναι ένας καταχωρημένος κόμβος (hostname) ή μια διεύθυνση IP.
- Τη διαδρομή (path), μια ακολουθία διαφορετικών τμημάτων που χωρίζονται με το σύμβολο `/` και οδηγούν στο στοιχείο που περιγράφει το URI.
- Προαιρετικά ένα ερώτημα (query), που αρχίζει με το ερωτηματικό `?`, ή/και ένα απόσπασμα (fragment), που αρχίζει με τη δέση `#`.

6. Οντολογίες

Η υλοποίηση του Σημασιολογικού Ιστού συμβάλει στον εμπλουτισμό της πληροφορίας με σημασιολογικό τρόπο, μέσα από τη χρήση των οντολογιών. Ο όρος οντολογία χρησιμοποιήθηκε αρχικά από τους αρχαίους Έλληνες φιλοσόφους καθώς προσπαθούσαν να δώσουν απαντήσεις σε κάποια φιλοσοφικά ερωτήματα που σχετίζονταν με την συστηματική μελέτη της ύπαρξης των όντων.

Σύμφωνα με τον Gruber (1993): «Μια οντολογία είναι μια ρητή προδιαγραφή μιας εννοιολογικής αναπαράστασης» (An ontology is an explicit specification of a conceptualization). Στη συνέχεια (1997) ο Boorst προσπάθησε να τροποποιήσει τον αρχικό ορισμό, δίνοντας ένα πιο ξεκάθαρο και συγκεκριμένο νόημα, ανέφερε ότι «Μια οντολογία είναι μια τυπική, κατηγορηματική προδιαγραφή μιας διαμοιραζόμενης εννοιολογικής αναπαράστασης» (An ontology is a formal, explicit specification of a shared conceptualization). Αναλυτικότερα:

- Ο όρος “*τυπική*” (formal) δηλώνει ότι η οντολογία πρέπει να είναι μηχαναγνώσιμη, να μπορεί δηλαδή να τη διαβάσει ένα υπολογιστικό σύστημα.
- Ο όρος “*κατηγορηματική*” (explicit) αναφέρεται στις κατηγορίες των εννοιών που χρησιμοποιούνται, και τους κανόνες που αφορούν την χρήση αυτών των εννοιών οι οποίοι πρέπει να προσδιορίζονται με σαφήνεια.
- Ο όρος “*διαμοιραζόμενη*” (shared) δηλώνει ότι η οντολογία πρέπει να εκφράζει γνώση που είναι κοινά αποδεκτή στο πλαίσιο μιας κοινότητας.
- Ο όρος “*εννοιολογική αναπαράσταση*” (conceptualization) αναφέρεται σε οτιδήποτε ανήκει στο πεδίο ενδιαφέροντος μιας οντολογίας, όπως αντικείμενα, ιδέες και άλλες οντότητες, και στους τρόπους με τους οποίους συσχετίζονται.

Αργότερα το 2008 ο Gruber έδωσε έναν πιο συγκεκριμένο ορισμό της έννοιας οντολογίας, στα πλαίσια της επιστήμης των υπολογιστών και της πληροφορικής, μια οντολογία ορίζει ένα σύνολο από αρχέτυπα αναπαράστασης (representational primitives) με τα οποία μπορεί να μοντελοποιήσει ένα πεδίο γνώσης. Τα αρχέτυπα αναπαράστασης είναι συνήθως κλάσεις (ή σύνολα), χαρακτηριστικά (ή ιδιότητες), και σχέσεις (ή σχέσεις μεταξύ των μελών της κλάσης). Οι ορισμοί των αναπαραστατικών αρχέτυπων περιλαμβάνουν πληροφορίες σχετικά με τη σημασία τους και τους περιορισμούς στην λογικά συνεπή εφαρμογή τους.

Μια οντολογία αποτελείται από τα εξής συστατικά:

- *Κλάσεις (classes)*: Αναπαριστούν έννοιες που σχετίζονται με ένα πεδίο ορισμού, οι οποίες είναι συνήθως ιεραρχημένες σε κάποιο ταξινομικό σύστημα.
- *Σχέσεις (relations)*: Αναπαριστούν συσχετίσεις μεταξύ εννοιών μιας κλάσης. Οι σχέσεις μπορεί να έχουν ένα πεδίο ορισμού (domain) και ένα πεδίο τιμών (range). Το πεδίο τιμών μπορεί να είναι είτε μια κλάση είτε ένας τύπος δεδομένων.
- *Συναρτήσεις (functions)*: Μια ειδική περίπτωση σχέσης στην οποία το ν-οστό στοιχείο της σχέσης προσδιορίζεται μοναδικά από τα ν-1 προηγούμενα στοιχεία.
- *Αξιώματα (axioms)*: Αναπαριστούν προτάσεις οι οποίες είναι πάντα αληθείς.

- *Στιγμιότυπα (instances)*: Αναπαριστούν συγκεκριμένα στοιχεία.

Ανάλογα με το βαθμό τυπικότητάς της, μια οντολογία μπορεί να πάρει μια από τις παρακάτω μορφές, οι οποίες πρέπει να περιλαμβάνουν ένα λεξιλόγιο όρων και κάποιες προδιαγραφές για τη σημασία τους.

- *Άτυπη (informal)*: εκφρασμένη με φυσική γλώσσα.
- *Ημι-άτυπη (semi-informal)*: διατυπωμένη με ένα περιορισμένο και δομημένο υποσύνολο κάποιας φυσικής γλώσσας.
- *Ημι-τυπική (semi-formal)*: διατυπωμένη σε μια τεχνητή γλώσσα, με συγκεκριμένους όρους.
- *Αυστηρά τυπική (rigorously formal)*: διατυπωμένη με αυστηρά σημασιολογικό τρόπο.

Υπάρχουν διάφορες μεθοδολογίες ανάπτυξης οντολογιών. Σύμφωνα με τους Noy & McGuinness (2001), η μεθοδολογία που ακολουθείται συνήθως για την ανάπτυξη μιας οντολογίας, αποτελείται από τα παρακάτω βήματα:

- Καθορισμός του τομέα και του πεδίου της οντολογίας
- Επαναχρησιμοποίηση υπαρχουσών οντολογιών
- Απαρίθμηση των σημαντικών όρων της οντολογίας
- Ορισμός και ιεράρχηση κλάσεων
- Ορισμός των κλάσεων και ιδιοτήτων (slots)
- Ορισμός των απόψεων (facets) των ιδιοτήτων
- Δημιουργία οντοτήτων

Ωστόσο μέχρι σήμερα, πολλές ερευνητικές ομάδες έχουν προτείνει διάφορες μεθοδολογίες και περιβάλλοντα ανάπτυξης οντολογιών. Παρατηρώντας διάφορες μεθοδολογίες συμπεραίνουμε ότι, παρόλο που έχουν κοινά σημεία, δεν υπάρχει κάποια που να ταιριάζει και να ανταποκρίνεται σε όλες τις περιπτώσεις. Επειδή κάθε περίπτωση είναι διαφορετική, η επιλογή της κατάλληλης μεθοδολογίας εξαρτάται από διάφορους παράγοντες (π.χ. ο αριθμός των εμπλεκόμενων στην ανάπτυξη της οντολογίας, το είδος της οντολογίας, το πεδίο ενδιαφέροντος, η τελική εφαρμογή που θα χρησιμοποιήσει την οντολογία, η εμπειρία, το επίπεδο της γνώσης των εμπλεκόμενων, η διαθεσιμότητα των απαραίτητων πόρων). Παρακάτω παρουσιάζονται οι σημαντικότερες μεθοδολογίες ανάπτυξης οντολογιών:

- Μεθοδολογίες που σχεδιάζουν μια οντολογία «εκ του μηδενός» (from scratch)
- Μεθοδολογίες που σχεδιάζουν μια οντολογία χρησιμοποιώντας υπάρχουσες οντολογίες (reengineering ontologies)
- Μεθοδολογίες Συνεργατικής Ανάπτυξης Οντολογιών (cooperative ontologies)
- Μεθοδολογίες Συγχώνευσης Οντολογιών (merging ontologies).

3

Τεχνολογικό Υπόβαθρο

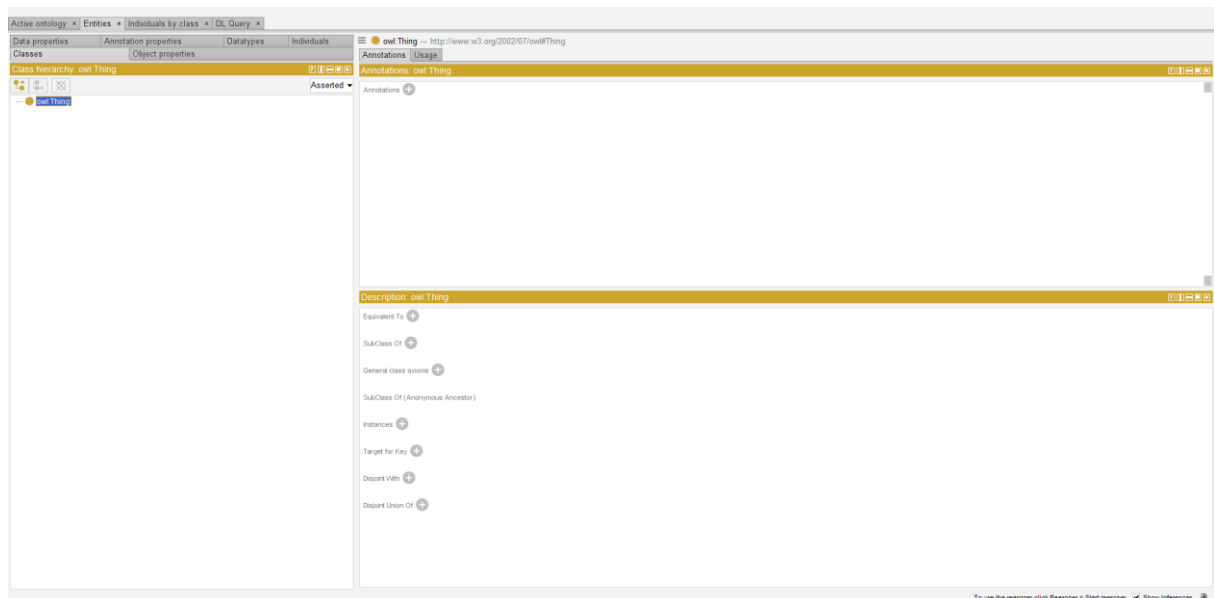
Στο προηγούμενο κεφάλαιο μελετήθηκε ο όρος του Σημασιολογικού Ιστού, καθώς και τα θέματα του ευρύτερου πεδίου του, όπως τα πρότυπα για την εφαρμογή και τη χρήση του, τα διασυνδεδεμένα δεδομένα και οι οντολογίες.

Το κεφάλαιο αυτό προσφέρει περιγραφές των εργαλείων που χρησιμοποιήθηκαν για την δημιουργία της οντολογίας, των frameworks για την κατασκευή της συνοδεύουσας εφαρμογής, και τους τρόπους με τους οποίους διαμορφώθηκαν για τη δημιουργία του τελικού συστήματος.

1. Protégé

Το Protégé είναι μια εφαρμογή ανοικτού κώδικα η οποία είναι σχεδιασμένη για τη δημιουργία σχεσιακών μοντέλων με τη χρήση οντολογιών και γενικότερων εφαρμογών γνώσης. Αναπτύχθηκε το 1999 από το Πανεπιστήμιο του Stanford με τη χρήση της γλώσσας προγραμματισμού Java, και συντηρείται στην 5η του έκδοση μέχρι σήμερα με τη συνεργασία άλλων σχολών και οργανισμών. Ακόμη, όντας λογισμικό ανοικτού κώδικα, έχουν δημιουργηθεί διάφορα πρόσθετα εργαλεία και επεκτάσεις που χειρίζονται ειδικότερες διαδικασίες ή προσφέρουν καινούργιες λειτουργίες στο Protégé, οι οποίες αναπτύσσονται και συντηρούνται από μεμονωμένους προγραμματιστές, ακόμα και ακαδημαϊκούς οργανισμούς. Πολλά απ' αυτά τα πρόσθετα εργαλεία είναι ελεύθερα στο κοινό, και φιλοξενούνται στην ιστοσελίδα του Protégé.

Βασικός σκοπός του Protégé είναι η υποστήριξη διαφόρων δομών σχεσιακών μοντέλων για τη δημιουργία, την αποτύπωση και τη διαχείριση τέτοιων δομών, κατασκευάζοντας κάθε είδους οντολογία με τα εργαλεία που διαθέτει, με το τελικό μοντέλο που παράγεται να ακολουθεί σωστά το συντακτικό κάθε προτύπου οντολογίας που υποστηρίζει, όπως RDF(s), OWL και XML. Ουσιαστικά, είναι ένα λογισμικό κατασκευής οντολογιών σ' ένα γραφικό περιβάλλον, αντί τη σύνταξη μιας οντολογίας με το χέρι σε προγράμματα επεξεργασίας κειμένου. Μαζί με τη σύνταξη οντολογιών, το Protégé υποστηρίζει και την εισαγωγή δεδομένων dataset σε μια δομή, είτε συμπληρώνοντας τα απαραίτητα πεδία που εμφανίζει για κάθε κλάση και ιδιότητα με τιμές, είτε εισάγοντας ένα αρχείο αυτόματα.



Η σελίδα της λίστας όλων των οντοτήτων στο Protégé 5, κατά τη δημιουργία νέου αρχείου

Η αρχιτεκτονική που χρησιμοποιεί το Protégé είναι χωρισμένη στα τμήματα του Μοντέλου και της Απεικόνισης. Το τμήμα του Μοντέλου αποτελεί τον μηχανισμό εσωτερικής αναπαράστασης μιας οντολογίας, ενώ το τμήμα της Απεικόνισης παρέχει εργαλεία για την μορφοποίηση ενός περιβάλλοντος γραφικής αναπαράστασης για το Μοντέλο, και επιτρέπει την επεξεργασία του.

Ένα Μοντέλο αναπαριστά μια οντολογία και τα επιμέρους χαρακτηριστικά του ως οντότητες (Entities), όπως οι κλάσεις (Classes), οι ιδιότητες (Object and Data Properties), και τα στιγμιότυπα (Named Individuals). Η Απεικόνιση αποτελεί ένα σύνολο εργαλείων του Protégé, με το οποίο μπορεί ένας χρήστης να δημιουργήσει κλάσεις, να αναθέσει ιδιότητες στις κλάσεις αυτές και να ορίσει τα αντικείμενα αυτών των ιδιοτήτων.

Κατά τη συγγραφή μιας οντολογίας ή κατά φόρτωση μιας υπάρχουσας στο περιβάλλον του Protégé, δημιουργείται μια επιφάνεια με το γραφικό περιβάλλον διεπαφής, με τμήματα για προβολή των στοιχείων που περιγράφουν κάθε οντότητα και τις συσχετίσεις που έχουν με άλλες οντότητες. Τα τμήματα αυτά έχουν τη δυνατότητα να προσαρμοστούν απ' τον χρήστη, ως προς το τι προβάλλουν και με ποια σειρά. Ακόμα, διαθέτουν τις φόρμες για την επεξεργασία των χαρακτηριστικών κάθε οντότητας, οι οποίες έχουν τη μορφή πεδίων συμπλήρωσης ή προσθήκης τιμών. Ένα απ' τα απαραίτητα τμήματα στο γραφικό περιβάλλον του Protégé περιέχει ένα τερματικό για την εκτέλεση ερωτημάτων (DL Query) στην τρέχουσα οντολογία, για την πρόσβαση στα δεδομένα της και μια γραφική απεικόνισή της.

2. Apache Jena

Το *Apache Jena* είναι μια πλατφόρμα εργαλείων ανοιχτού κώδικα, γραμμένη με τη γλώσσα προγραμματισμού Java, για την αποθήκευση και διαχείριση διασυνδεδεμένων δεδομένων RDF και τη δημιουργία εφαρμογών με βάση τον Σημασιολογικό Ιστό. Αρχικά αναπτύχθηκε από τον κλάδο ανάπτυξης λογισμικού της HP το 2000, και αργότερα ενσωματώθηκε στο σύνολο των frameworks της Apache Software Foundation το 2010. Το Apache Jena και το API του χρησιμοποιούνται συχνά στην ανάπτυξη του Σημασιολογικού Ιστού, ειδικότερα για την κατασκευή εφαρμογών. Είναι μια ευέλικτη πλατφόρμα ανάπτυξης με χρήσιμες βιβλιοθήκες για την καλύτερη υποστήριξη προγραμματιστών στη διαχείριση των προτύπων RDF, OWL και SPARQL.[16]

Βασικό χαρακτηριστικό του Apache Jena κατά την ανάπτυξη εφαρμογών είναι η εισαγωγή RDF γράφων στο σύστημά της, στους οποίους έχει πλήρη έλεγχο και μπορεί να εκτελέσει διάφορες λειτουργίες, όπως η ανάγνωσή του και η εγγραφή μέσα σ' αυτόν, η ανάθεση ερωτημάτων (queries), και η συγχώνευση δύο γράφων μεταξύ τους. Τα ερωτήματα που υποστηρίζονται είναι της γλώσσας SPARQL, και μπορούν ακόμα να εκτελούνται κατά τη διάρκεια της επεξεργασίας ενός RDF γράφου. Τα συγκεκριμένα ερωτήματα που υποστηρίζονται από το Apache Jena καλύπτουν όλο το εύρος των ερωτημάτων SPARQL που είναι διαθέσιμο, όπως η ανάκτηση τριάδων, η ενημέρωσή τους, η προσθήκη νέων και η διαγραφή τους. Ως αποθηκευτικό μέσο, χρησιμοποιεί τη μέθοδο Triplestore, μια βάση δεδομένων ειδική για την αποθήκευση δεδομένων σε τριπλέτες.

Για κάθε ακμή σε έναν RDF γράφο, το Apache Jena αποδίδει έναν όρο Statement, στον οποίο ορίζεται κάθε τριάδα του γράφου και έχει μια δομή παρόμοια με αυτή των RDF τριάδων που έχει ως εξής:

- Υποκείμενο (Subject) που είναι ένας Πόρος (Resource) από τον οποίο ξεκινάει μία ακμή.
- Κατηγορημα (Predicate) που είναι η Ιδιότητα (Property) που χαρακτηρίζει την ακμή.
- Αντικείμενο (Object) που είναι είτε ένας Πόρος (Resource) είτε μία Τιμή (Literal) όπου καταλήγει η ακμή.

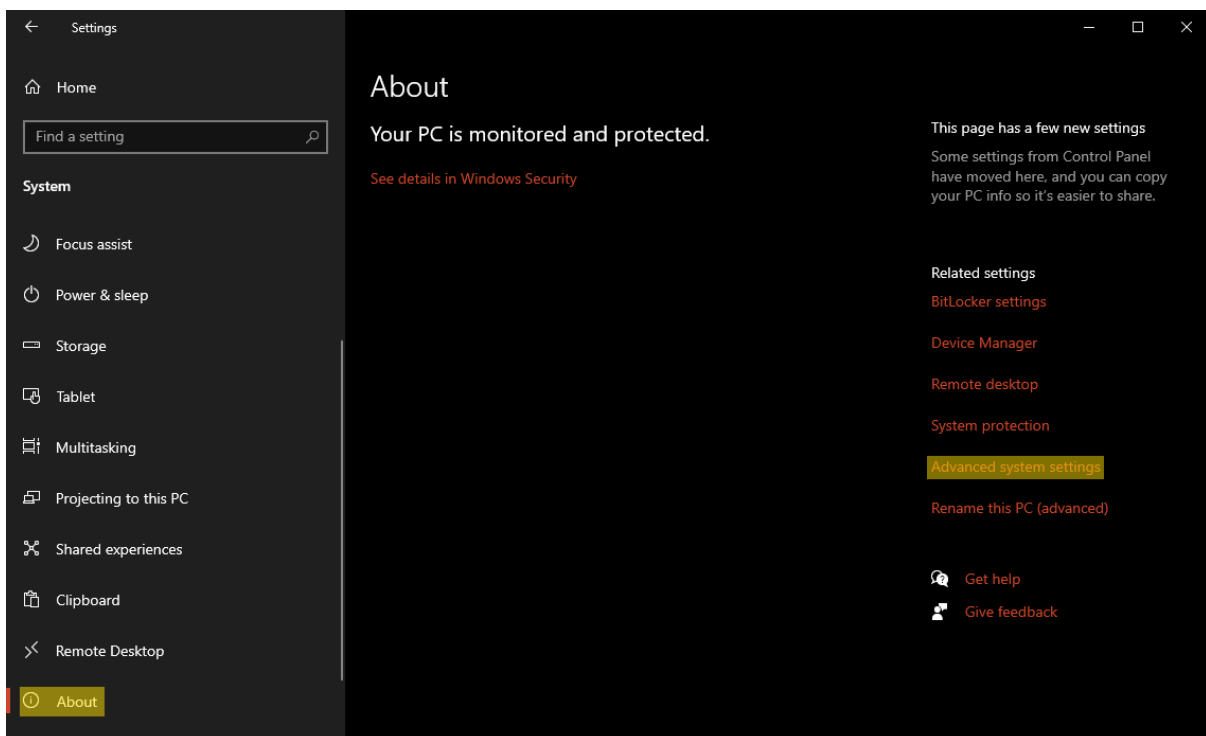
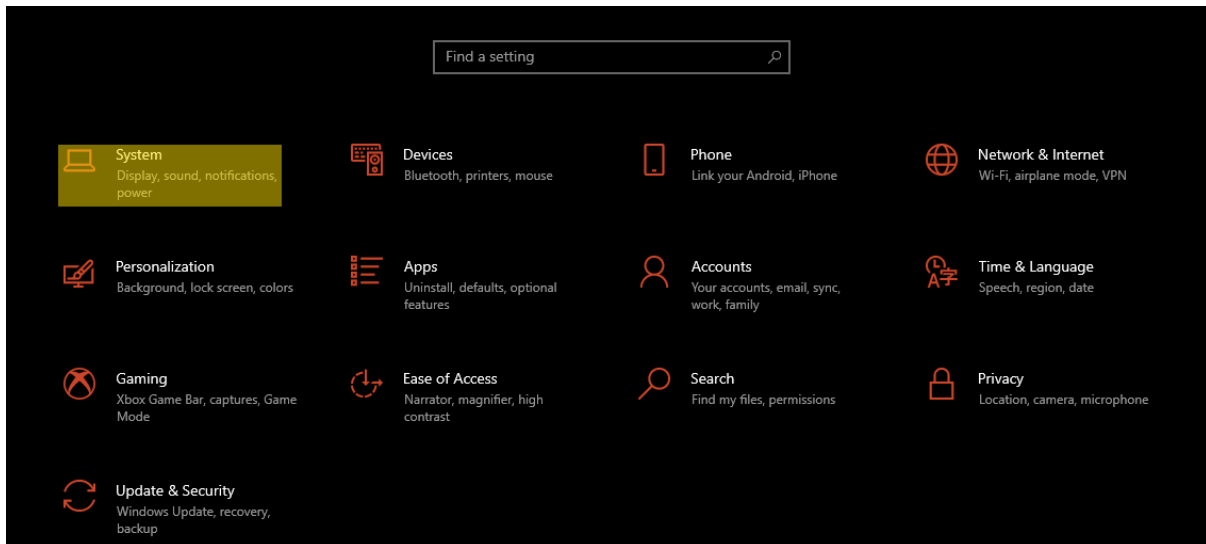
2.1. Εγκατάσταση Apache Jena

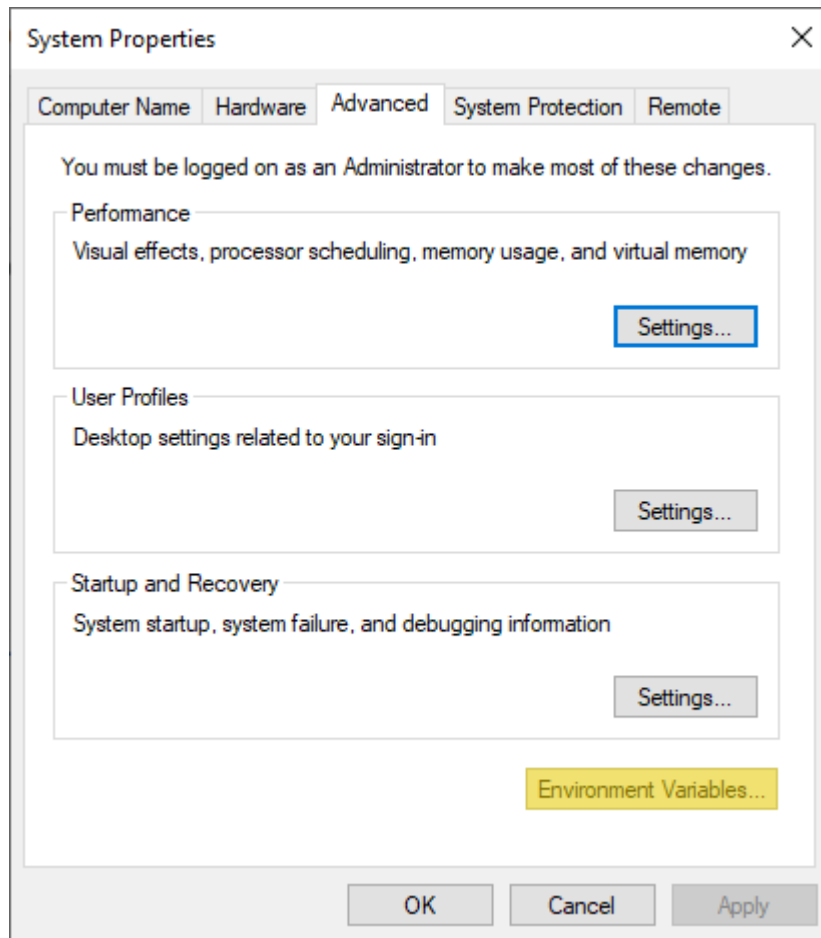
Αρχικά, για τη δημιουργία της οντολογίας, και της τελικής εφαρμογής, χρειάστηκε να εγκαταστήσουμε το [Apache Jena](#), το οποίο προσφέρει το περιβάλλον και όλα τα απαραίτητα components για τη δόμησή της.

Η εγκατάστασή του απαιτεί λίγα περισσότερα βήματα, επειδή το framework αυτό δεν περιλαμβάνεται σε κάποιο εκτελέσιμο αρχείο.

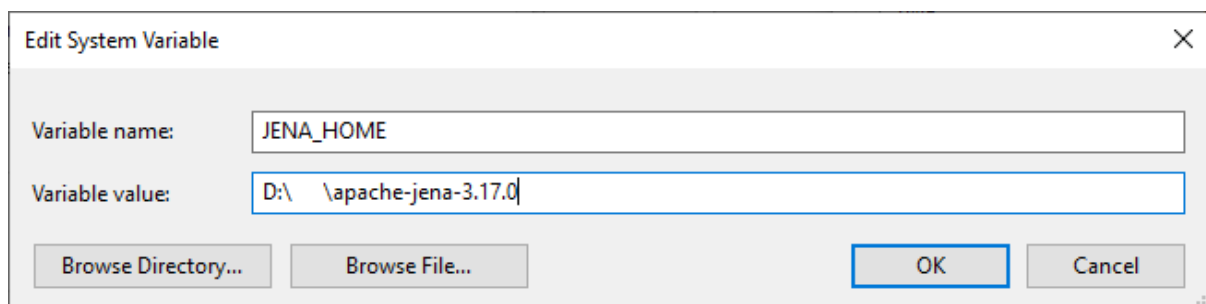
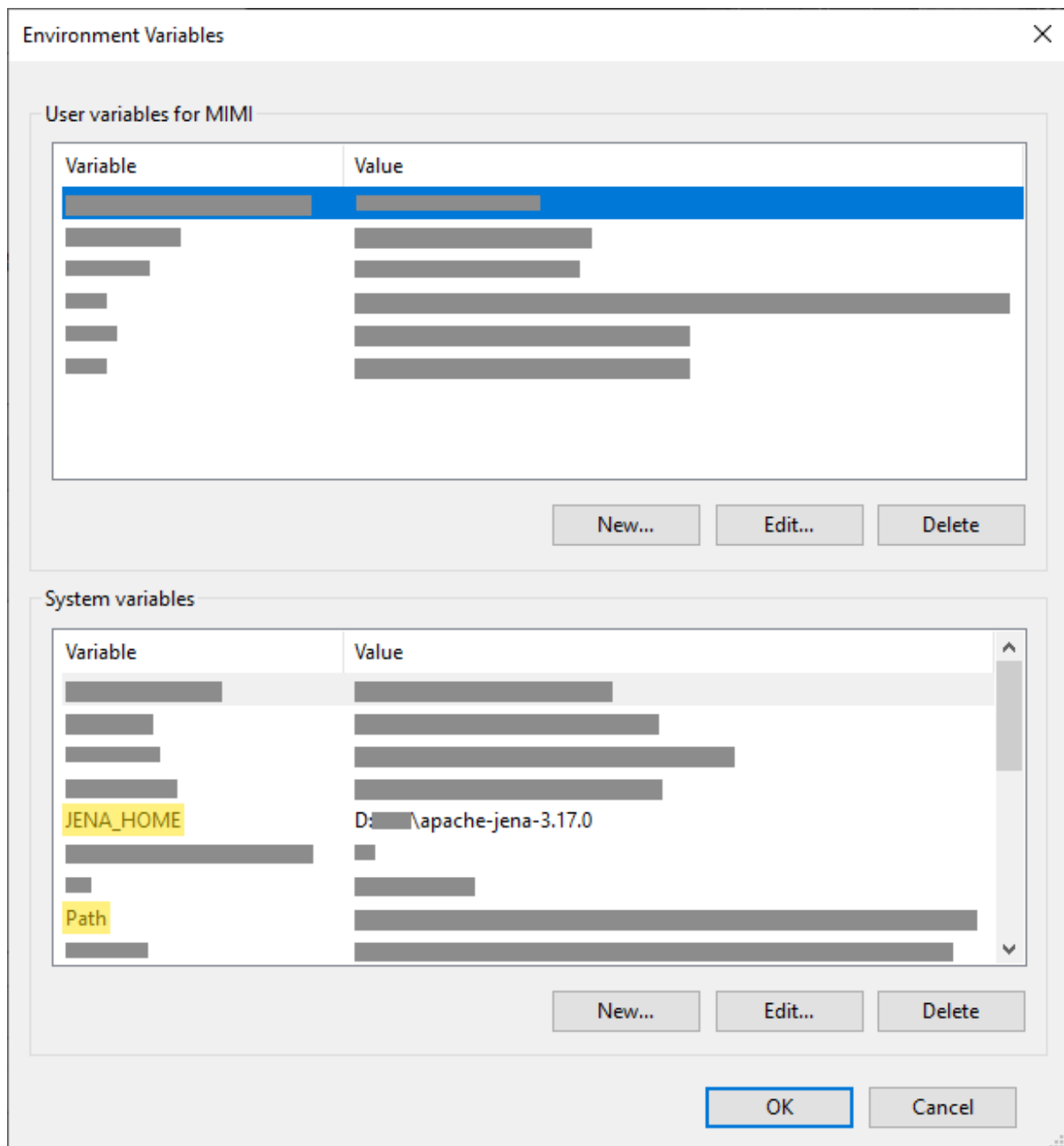
Το Apache Jena λειτουργεί με τη χρήση μεταβλητών περιβάλλοντος του συστήματος, οι οποίες δείχνουν σε ποιο φάκελο βρίσκονται τα αρχεία του framework, όπως τα εργαλεία του και τα προγράμματά του. Αυτές οι μεταβλητές μπορούν να οριστούν με δύο τρόπους:

- Μέσω των προχωρημένων ρυθμίσεων του συστήματος:
- Μεταβαίνουμε στις *Ρυθμίσεις* -> *Σύστημα* -> *Πληροφορίες* -> *Ρυθμίσεις συστήματος για προχωρημένους*, και στο παράθυρο *Ιδιότητες Συστήματος*, επιλέγουμε *Μεταβλητές Περιβάλλοντος*.

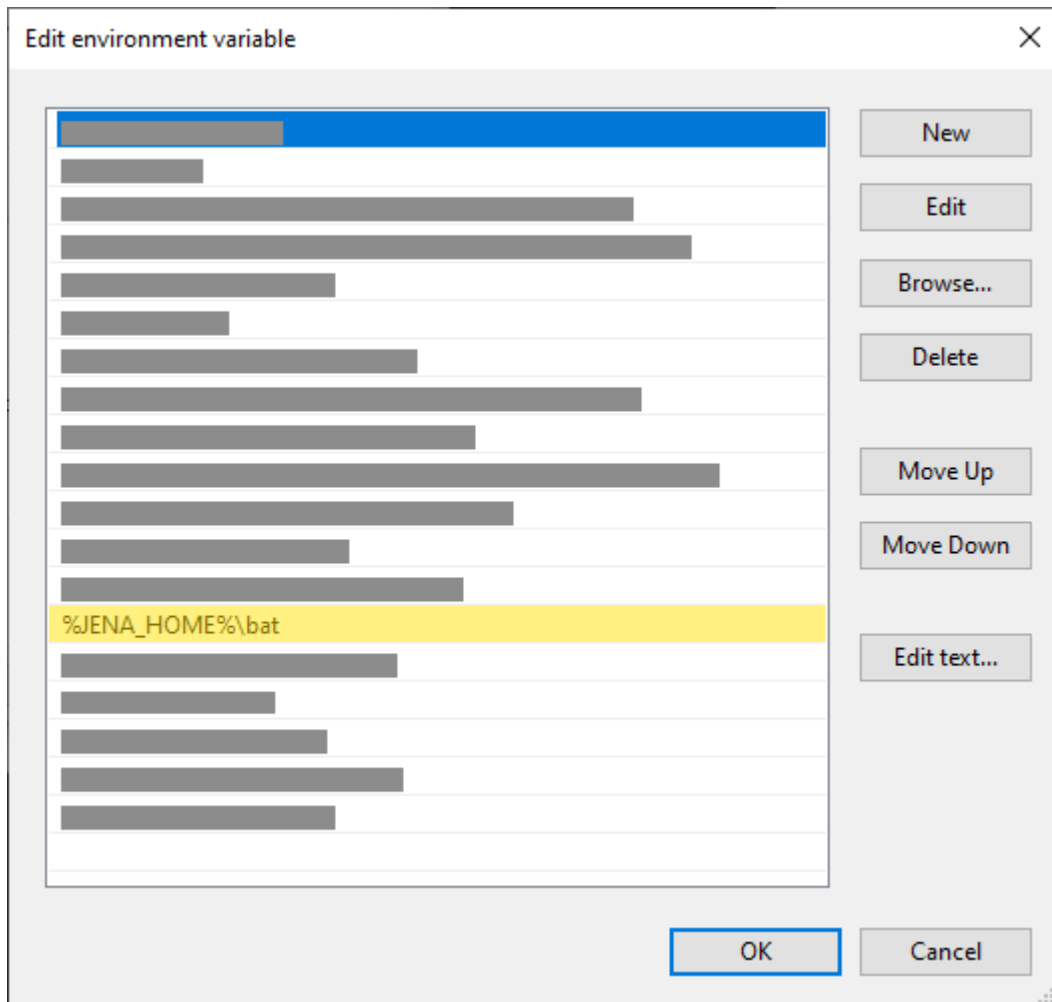




- Ύστερα, στο νέο παράθυρο, δημιουργούμε την πρώτη *Μεταβλητή Συστήματος* με όνομα *JENA_HOME* και τιμή τον φάκελο στον οποίο έγινε αποσυμπίεση του ZIP που κατεβάσαμε (περιέχει τους φακέλους bat, bin, lib, κλπ).



- Τέλος, για τη δεύτερη *Μεταβλητή Συστήματος*, ανοίγουμε την μεταβλητή *Path*, και στο παράθυρό της, προσθέτουμε την τιμή `%JENA_HOME%\bat`, όπου `%JENA_HOME%` είναι η πρώτη μεταβλητή που φτιάξαμε.



➤ Μέσω της γραμμής εντολών:

- Μέσω της γραμμής εντολών, εκτελούμε την πρώτη εντολή (προσοχή στην σύνταξη, χωρίς κενά):

```
setx JENA_HOME=[ο φάκελος των αρχείων του Apache Jena]
```

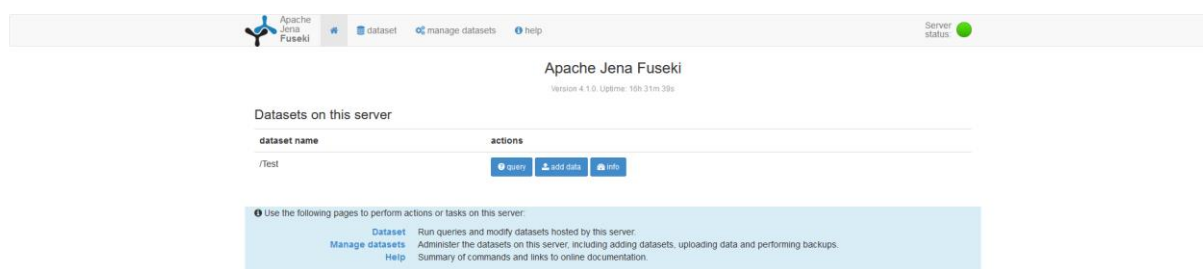
- Μετά, εκτελούμε τη δεύτερη εντολή, όπου %JENA_HOME% είναι η μεταβλητή που φτιάξαμε παραπάνω:

```
setx PATH=%PATH%;%JENA_HOME%\bat
```

- Σημείωση: Χρησιμοποιούμε setx αντί για set, ώστε οι μεταβλητές να παραμένουν στο σύστημα, και αν τις ξαναχρειαστούμε να μην είναι ανάγκη να εκτελέσουμε τις παραπάνω δύο εντολές ξανά.

3. Apache Fuseki

Μαζί με το Apache Jena, αναπτύχθηκε η επέκτασή του Apache Jena Fuseki (ή αλλιώς Apache Fuseki), μία RESTful διαδικτυακή διεπαφή RDF δεδομένων που προσφέρει έναν δικό του εξυπηρετητή SPARQL για ερωτήματα και ενημερώσεις γράφων, βασιζόμενος στο γνωστό framework εξυπηρετητή Apache Tomcat. Μπορεί να λειτουργήσει επίσης ως δική του υπηρεσία, ως διαδικτυακή εφαρμογή, ή ως αυτόνομος εξυπηρετητής.[17] Παρέχει ασφάλεια χρησιμοποιώντας το Apache Shiro (framework της Java για αυθεντικοποίηση και διαχείριση συνεδρίας) και διαθέτει μια διεπαφή χρήστη για την παρακολούθηση και διαχείριση της κατάστασης του εξυπηρετητή. Ακόμη, διαθέτει εγγενώς το σύστημα Triplestore TDB, επίσης μέλος της οικογένειας Apache Jena, για την αποθήκευση RDF γράφων. Το TDB έχει ακόμη τη δυνατότητα να αποθηκεύει μόνιμα (persistently) τους γράφους και τα στιγμιότυπά του, που εισάγονται στο σύστημα του Fuseki, ώστε να είναι διαθέσιμοι ακόμη και μετά τον τερματισμό του server σαν μια βάση δεδομένων.



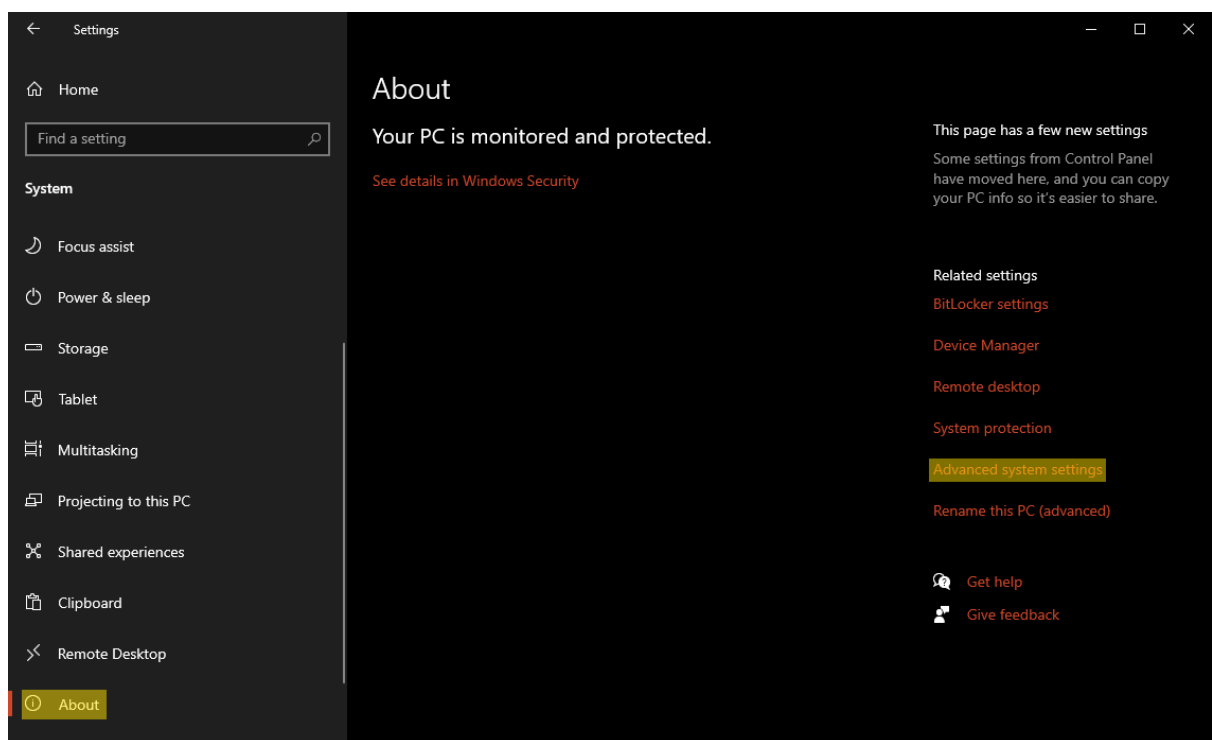
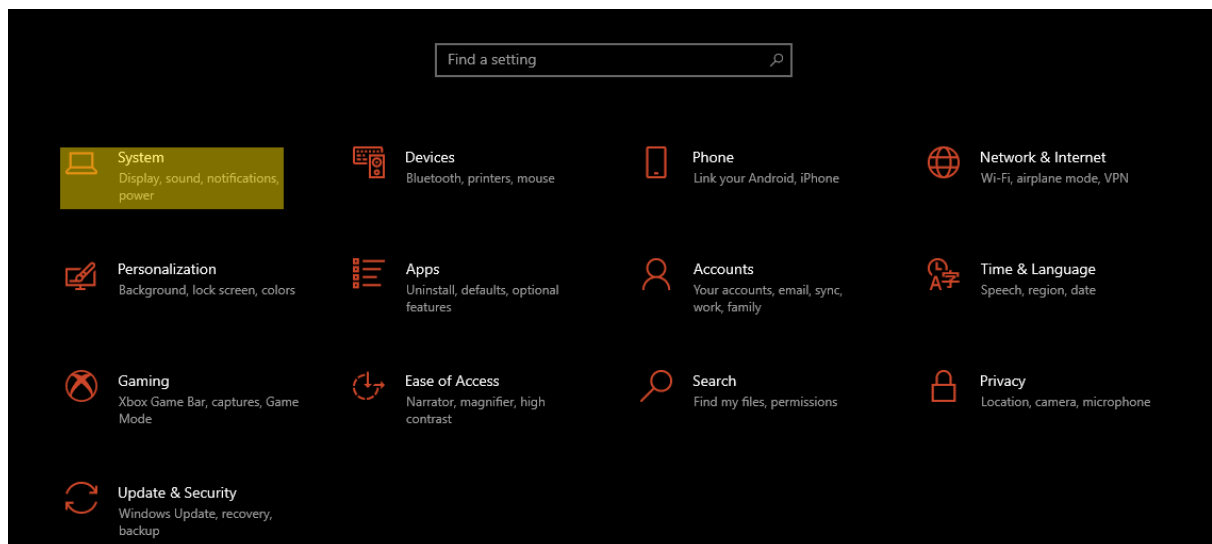
Η αρχική σελίδα της διεπαφής χρήστη του Apache Fuseki

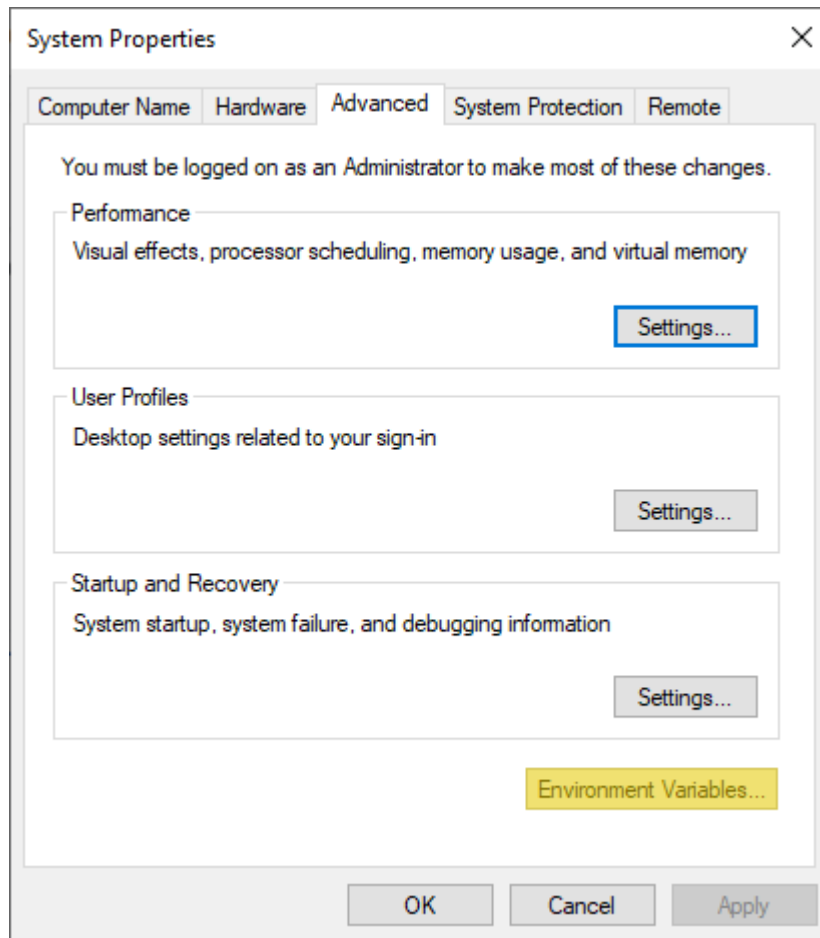
3.1. Εγκατάσταση του Apache Fuseki

Κατεβάζουμε το [Apache Jena Fuseki](#).

Το Apache Jena λειτουργεί με τη χρήση μεταβλητών περιβάλλοντος του συστήματος, οι οποίες δείχνουν σε ποιο φάκελο βρίσκονται τα αρχεία του framework, όπως τα εργαλεία του και τα προγράμματά του. Αυτές οι μεταβλητές μπορούν να οριστούν με δύο τρόπους:

- Μέσω των προχωρημένων ρυθμίσεων του συστήματος:
- Μεταβαίνουμε στις *Ρυθμίσεις* -> *Σύστημα* -> *Πληροφορίες* -> *Ρυθμίσεις συστήματος για προχωρημένους*, και στο παράθυρο *Ιδιότητες Συστήματος*, επιλέγουμε *Μεταβλητές Περιβάλλοντος*.





- Ανοίγουμε την μεταβλητή *Path*, και στο παράθυρό της, προσθέτουμε ως τιμή τον φάκελο στον οποίο έγινε αποσυμπίεση του ZIP που κατεβάσαμε (περιέχει αρχεία όπως *fuseki-server*, *fuseki.war*, κλπ).

➤ Μέσω της γραμμής εντολών:

- Μέσω της γραμμής εντολών, εκτελούμε την πρώτη εντολή (προσοχή στην σύνταξη, χωρίς κενά):
- Μετά, εκτελούμε τη δεύτερη εντολή, όπου `%JENA_HOME%` είναι η μεταβλητή που φτιάξαμε παραπάνω:

```
setx PATH=%PATH%;[ο φάκελος των αρχείων του Apache Jena]
```

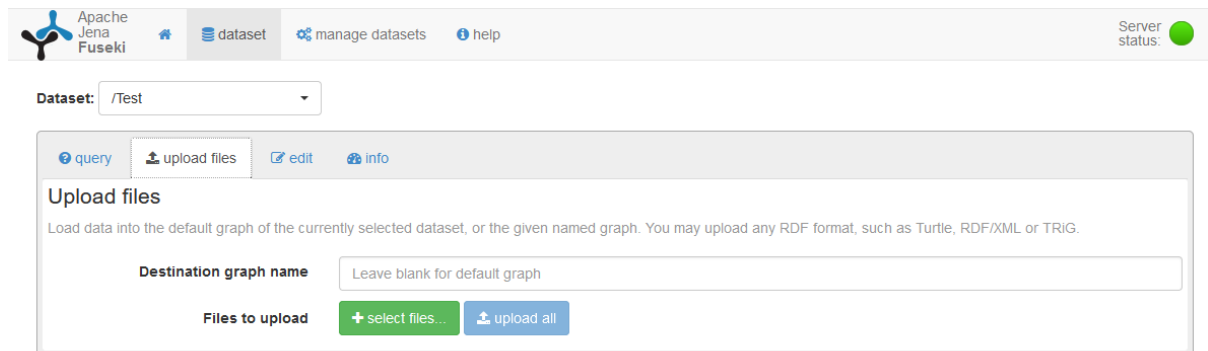
Καθώς τώρα ο φάκελος του Fuseki αναγνωρίζεται απ' τον υπολογιστή, μπορούμε να τρέξουμε τον server. Πρώτα, μετακινούμαστε στον φάκελο του Fuseki, και τρέχουμε την εντολή:

```
fuseki-server
```

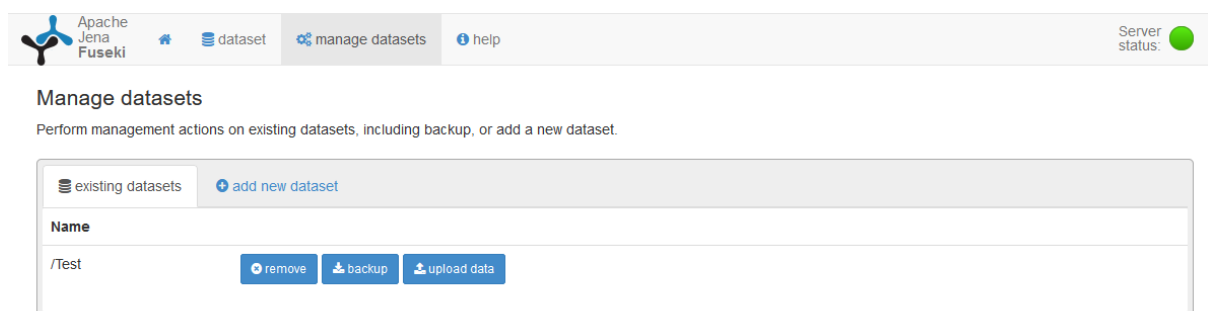
Ο server του Fuseki τώρα τρέχει στη σελίδα `localhost:3030` (εκτός κι αν ορίσουμε μια ρύθμιση με την εντολή `fuseki-server` η οποία θα τον ανοίγει σε άλλη θύρα). Ανατρέχοντας σ' αυτόν τον

σύνδεσμο μέσω ενός browser, μας εμφανίζεται το γραφικό περιβάλλον του Fuseki, όπου μπορούμε να εισάγουμε δεδομένα, να εξετάσουμε στατιστικά και υπηρεσίες σχετικά με την βάση οντολογίας, και βέβαια να κάνουμε ερωτήσεις (queries).

Για τη δημιουργία μιας βάσης οντολογίας στο Fuseki, επιλέγουμε την καρτέλα *dataset*, και μετά *upload files*. Γράφουμε το όνομα της βάσης στο *Destination graph name*, και επιλέγουμε το αρχείο της οντολογίας πατώντας το κουμπί *select files...*



Η βάση ακόμα, μπορεί να επεκταθεί ή να διορθωθεί, εισάγοντας ένα άλλο αρχείο πάνω στην υπάρχουσα βάση. Αυτή η λειτουργία βρίσκεται στην καρτέλα “manage datasets”, απ’ την οποία επιλέγοντας την επιθυμητή βάση και μετά επιλέγοντας “upload data”, μας εμφανίζει την προηγούμενη εικόνα για την εισαγωγή αρχείου.



4. ReactJS

Το ReactJS (ή απλώς React) είναι μια βιβλιοθήκη της γλώσσας προγραμματισμού JavaScript, ειδική για τη δημιουργία διεπαφών χρήστη και τη δόμηση γραφικών περιβαλλόντων σε διαδικτυακούς τόπους και κινητές εφαρμογές. Το έργο αυτό διατηρείται από την ομάδα ανάπτυξης της δημοφιλής πλατφόρμας κοινωνικής δικτύωσης Facebook και από μια ευρεία κοινότητα ανεξάρτητων προγραμματιστών και εταιρειών.[18]

Ωστόσο, το React ασχολείται μόνο με τη διαχείριση της κατάστασης και την απόδοση αυτής της κατάστασης στο DOM, οπότε η δημιουργία εφαρμογών React απαιτεί συνήθως τη χρήση πρόσθετων βιβλιοθηκών για δρομολόγηση, καθώς και ορισμένων λειτουργιών από την πλευρά του πελάτη.

Η ανάπτυξη του ReactJS ξεκίνησε το 2011 από τον μηχανικό λογισμικού της Facebook, Jordan Walke. Στα πρώτα του στάδια, όταν ακόμη δεν είχε λάβει δική του ονομασία, χρησιμοποιήθηκε σε διάφορες εφαρμογές που ανήκαν στην εταιρεία Facebook, όπως το Instagram. Αργότερα το 2013 δημοσιεύτηκε για πρώτη φορά για δημόσια χρήση, με το τελικό του όνομα ReactJS, και ταυτόχρονα ορίστηκε ως βιβλιοθήκη ανοικτού κώδικα. Στα επόμενα χρόνια, δημιουργήθηκαν διάφορες ενημερώσεις και άλλες εκδόσεις του React, όπως το React Native για την ανάπτυξη εφαρμογών ειδικά σχεδιασμένες για κινητές συσκευές Android και iOS.[19]

Βασική δομή για την ανάπτυξη εφαρμογών στο περιβάλλον του React είναι τα components (συστατικά στοιχεία), κομμάτια ερμηνευμένου (rendered) κώδικα HTML, τα οποία ενθαρρύνουν την επαναχρησιμοποίηση στοιχείων μέσα σε μια εφαρμογή. Αυτά τα components μπορούν να εμφωλεύονται μέσα σε άλλα και να αποθηκεύουν δεδομένα μοναδικά σε κάθε στιγμιότυπο του component σε μορφή states, τα οποία μπορεί μετά να εισάγει σε εμφωλευμένα components κατά την κλήση τους, σε μορφή props. Αυτά τα χαρακτηριστικά διευκολύνουν την ανάπτυξη διαδικτυακών εφαρμογών, καθώς η δομή των components είναι παρόμοια με απλά blocks κώδικα, τα οποία χτίζουν την εφαρμογή.[19]

Ένα άλλο μοναδικό χαρακτηριστικό του React είναι το Virtual DOM (Virtual Document Object Model ή Εικονικό Μοντέλο Αντικειμένου Εγγράφου), μια επέκταση της υπάρχουσας δομής DOM για την οργάνωση αντικειμένων ενός HTML εγγράφου. Κατά το compile μιας εφαρμογής React, κάθε component αποθηκεύεται στην προσωρινή μνήμη του συστήματος, ώστε σε περίπτωση αλλαγών του περιεχομένου της εφαρμογής που τυπώνεται, να υπολογίζονται οι αλλαγές που έγιναν και να ενημερώνονται αναλόγως μόνο τα στοιχεία εκείνα τα οποία πρέπει να αλλάξουν, παρά να φορτωθεί ολόκληρη η εφαρμογή απ' την αρχή. Αυτό βελτιώνει σημαντικά τον χρόνο εκτέλεσης της εφαρμογής, με μόνη θυσία ένα μικρό μέρος της κρυφής μνήμης του συστήματος για την αποθήκευση των components.

Ένα μεγάλο μέρος του συντακτικού του React βασίζεται στη γλώσσα JSX (JavaScript XML), μια επέκταση της JavaScript που, όπως έχει και το όνομά της, χρησιμοποιεί τη δομή της XML, ενσωματώνοντας μέσα του μέρη του συντακτικού της JavaScript. Αυτό επιτρέπει μεταξύ άλλων τη χρήση μεταβλητών και συναρτήσεων από την JavaScript, μέσα σε αντικείμενα μορφής XML ή HTML, κάτι που βοηθά στην απλοποίηση της σύνταξης React components, δομώντας πιο ευέλικτα τελικά έγγραφα HTML. Αν και η μορφή JSX είναι ο πιο συνηθισμένος και πιο δημοφιλής τρόπος σύνθεσης κώδικα React, υποστηρίζεται παράλληλα και το βασικό συντακτικό της JavaScript.

4.1. Εγκατάσταση του ReactJS

Για την ανάπτυξη της εφαρμογής που θα επικοινωνεί με το Fuseki και κατ' επέκταση και με την οντολογία μας, θα χρησιμοποιήσουμε το [ReactJS](#), μια βιβλιοθήκη JavaScript για την ανάπτυξη οποιουδήποτε είδους web εφαρμογής διεπαφής με το χρήστη.

Το ReactJS επιτρέπει τη δημιουργία εφαρμογών ακόμα και με έναν online IDE, αλλά για την καλύτερη διαχείριση της διαδικασίας ανάπτυξης, θα εργαστούμε τοπικά κι έτσι θα εγκαταστήσουμε το ReactJS στον υπολογιστή μας. Για την εγκατάσταση του ReactJS, υπάρχουν κάποια προαπαιτούμενα εργαλεία που πρέπει να εγκαταστήσουμε, με κάποια να είναι προτεινόμενα απ' τους ίδιους τους δημιουργούς του ReactJS. Τα εργαλεία που διαλέξαμε περιγράφονται παρακάτω:

NodeJS

Πρώτο προαπαιτούμενο εργαλείο είναι το [NodeJS](#), πάνω στο οποίο έχει βασιστεί το ReactJS και είναι υποχρεωτικό για την ανάπτυξη εφαρμογών. Η ιστοσελίδα του NodeJS διαθέτει ένα πρόγραμμα εγκατάστασης, το οποίο απλά εκτελούμε κατευθείαν. Πέρα απ' την εγκατάστασή του, το NodeJS δεν θα χρησιμοποιηθεί απευθείας περαιτέρω.

Create React App

Σημαντικό βήμα στη δημιουργία μιας React εφαρμογής, είναι η χρήση ενός toolchain το οποίο, μεταξύ άλλων, κάνει compile τον κώδικα της εφαρμογής. Το ReactJS υποστηρίζει πολλά toolchains, το καθένα απ' αυτά με τις δικές του ειδικότητες, αλλά για τη δημιουργία μιας απλής αλλά ευέλικτης εφαρμογής όπως η δική μας, οι δημιουργοί του React προτείνουν το [Create React App](#).

Για να δημιουργήσουμε ένα project με τη χρήση του Create React App, εκτελούμε την παρακάτω εντολή η οποία δημιουργεί το file system της εφαρμογής, με το βασικό φάκελο *my-app*:

```
npx create-react-app my-app
```

Μετά, μεταβαίνουμε σ' αυτόν τον φάκελο, και αδειάζουμε τα υπάρχοντα περιεχόμενα του υποφακέλου *src* (Σημείωση: Δεν διαγράφουμε τον φάκελο όμως), ώστε να τον γεμίσουμε ύστερα με δικά μας αρχεία:

```
cd my-app
cd src
del *
```

Δημιουργούμε δύο αρχεία *index.js* και *index.css* μέσα στον φάκελο *src*, και προσθέτουμε τις παρακάτω γραμμές στο πρώτο αρχείο:

```
import React from 'react';
import './index.css';
```

Τέλος, ξεκινάμε τον server με την παρακάτω εντολή:

```
npm start
```

και στη διεύθυνση <http://localhost:3000> θα έχει ξεκινήσει η νέα ReactJS εφαρμογή.

4

Η Οντολογία

Το παρόν κεφάλαιο περιγράφει τον τρόπο με τον οποίο χρησιμοποιήθηκαν τα εργαλεία ώστε να δημιουργηθεί η οντολογία που θα χρησιμοποιηθεί σαν βάση δεδομένων στην εφαρμογή που θα υλοποιηθεί.

Η οντολογία Game Documentation Ontology έχει ως στόχο την αποτύπωση πληροφοριών που αφορούν τα video games, όπως ποιοι δημιούργησαν το παιχνίδι, τι είδος είναι, σε ποιο σύστημα ή πλατφόρμα κυκλοφόρησε, το έτος κυκλοφορίας του κ.α.

Παρακάτω περιγράφουμε τον τρόπο με τον οποίο σχεδιάστηκε αυτή η οντολογία, το υπόβαθρο της, και τις αναλυτικές λειτουργίες κάθε κλάσης και συσχέτισής της.

Αυτή η οντολογία χρησιμοποιήθηκε αργότερα ως βάση της ανάπτυξης της διαδικτυακής εφαρμογής, απ' την οποία αντλούνται δεδομένα ανάλογα με την κλάση τους, τις ιδιότητές τους, και άλλες σχετικές τους διασυνδέσεις που μπορεί να έχουν.

1. Ανάπτυξη της Οντολογίας

1.1. Συλλογή και χρήση πηγών

Video Game Ontology

Η [Video Game Ontology \(VGO\)](#) των Janne Parkkila, Filip Radulovic, Daniel Garijo και María Poveda ήταν το πρώτο σημείο αναφοράς που χρησιμοποιήσαμε, όντας κι εκείνη μια οντολογία με θέμα την αποτύπωση βιντεοπαιχνιδιών.

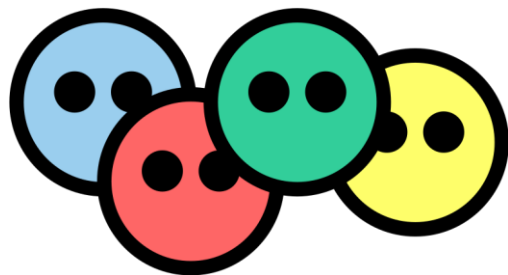
Αρχικά, χρησιμοποιήσαμε την VGO για την κατανόηση της σύνταξης Turtle (TTL) και για τον σωστό προσανατολισμό του περιεχομένου των κλάσεών μας. Ακόμη, οι κλάσεις Game και Genre και η ιδιότητα releaseDate της VGO συνδέθηκαν με κλάσεις της δικής μας οντολογίας κατά τις πρώτες εκδοχές της οντολογίας μας.

Στην πορεία, όμως, και καθώς αναπτυσσόταν περισσότερο η οντολογία μας, εμφανίστηκαν πολλές διαφορές μεταξύ των δύο οντολογιών, με τη μεγαλύτερη διαφορά να βρίσκεται στον τρόπο που προσεγγίζεται το θέμα το βιντεοπαιχνιδιών. Πιο αναλυτικά, το αντικείμενο που περιγράφει η VGO είναι κυρίως τις οντότητες που υπάρχουν μέσα σ' ένα παιχνίδι, όπως ο χαρακτήρας που ελέγχει ο παίκτης, τα αντικείμενα που μπορεί να συλλέξει, και τα συμβάντα που λαμβάνουν χώρα σ' αυτό, όπως μια συγκεκριμένη κίνηση του παίκτη και η πρόοδος της ιστορίας. Η δική μας οντολογία έχει ως στόχο την αποτύπωση των πληροφοριών που περιβάλλουν αυτό το παιχνίδι στον φυσικό κόσμο, όπως ποιοι δημιούργησαν το παιχνίδι, τι είδος είναι, σε ποιο σύστημα ή πλατφόρμα κυκλοφόρησε κλπ.

Ακόμα και με τις διαφορές τους, οι κλάσεις Game και Genre και η ιδιότητα releaseDate της VGO, χρησιμοποιήθηκαν ως βάσεις των αντίστοιχων της Game Documentation Ontology και έχουν όμοιες λειτουργίες. Για λόγους ευκολίας και συνοχής, όμως, αντί πια να εντάξουμε την VGO στην οντολογία μας και να τη συσχετίσουμε με τις υπόλοιπες πρωτότυπες κλάσεις μας, φτιάξαμε αυτά τα στοιχεία απ' την αρχή, χρησιμοποιώντας το δικό μας namespace.

FOAF

Η [FOAF](#) είναι μια γνωστή οντολογία σχεδιασμένη για την αποτύπωση ανθρώπινων οντοτήτων, καθώς και ό,τι τα περιβάλλει, όπως τα χαρακτηριστικά τους, τα στοιχεία τους, τις δραστηριότητές τους, πως συσχετίζονται με άλλες οντότητες, και άλλα. Η FOAF ήταν πολύ χρήσιμη στην αποτύπωση ανθρώπινων οντοτήτων της οντολογίας μας, γι' αυτό και είναι μια υπάρχουσα οντολογία που ενσωματώσαμε στη δική μας. Ο



Το λογότυπο της FOAF

τρόπος με τον οποίο χρησιμοποιήσαμε την FOAF περιγράφεται παρακάτω σ' αυτό το κεφάλαιο.

Θεμελιώδες στοιχείο της FOAF είναι η κλάση *foaf:Agent*, η οποία, σύμφωνα με την τεκμηρίωση της ίδιας της οντολογίας, εκφράζει “αντικείμενα που κάνουν πράγματα” (*things that do stuff*), ή πιο συγκεκριμένα, μια οντότητα η οποία έχει την ικανότητα να πράξει. Αυτή η βασική κλάση μπορεί να χρησιμοποιηθεί ειδικότερα στην αποτύπωση ανώνυμων ή και απρόσωπων οντοτήτων, αρκεί απλά αυτές να είναι σε θέση να έχουν έναν συγκεκριμένο ρόλο. Παράδειγμα απρόσωπης οντότητας μπορεί να είναι ένα λογισμικό που δημιουργεί και διαχειρίζεται τον δικό του χρήστη σε ένα υπολογιστικό σύστημα, χωρίς αυτό να επιβλέπεται από κάποιο ανθρώπινο πρόσωπο.

Η *foaf:Agent* κατά κύριο λόγο είναι μια υπερκλάση που επεκτείνεται σε τρεις ειδικότερες υποκλάσεις, οι οποίες περιγράφουν πιο συγκεκριμένες οντότητες, και είναι αυτές που χρησιμοποιούνται πιο συχνά όταν μια άλλη οντολογία θέλει να ενσωματώσει την FOAF. Αυτές είναι οι *foaf:Person*, *foaf:Group*, και *foaf:Organization*. Λειτουργίες και παραδείγματα αναφέρονται παρακάτω:

Η κλάση *foaf:Person* περιγράφει ένα ανθρώπινο πρόσωπο και έχει τις περισσότερες ιδιότητες οποιασδήποτε υποκλάσης της *foaf:Agent*, όπως το ονοματεπώνυμό του, η ηλικία και η ημερομηνία γέννησής του, έργα που έχει δημοσιεύσει, και τις ιστοσελίδες του χώρου εργασίας ή εκπαίδευσής του.

Η κλάση *foaf:Group* είναι μια πιο σύνθετη υποκλάση, η οποία περιγράφει μια ομάδα προσώπων, η οποία μπορεί να λειτουργήσει και η ίδια ως μία οντότητα. Τέτοιες ομάδες μπορεί να είναι κλειστά συνδεδεμένες όπως ένας συγκεκριμένος τομέας μιας επιχείρησης, έως ευρύτερες όπως η διαδικτυακή κοινότητα ενός forum. Όσο τα άτομα μέσα σ' αυτήν την ομάδα μοιράζονται κάποιο κοινό χαρακτηριστικό ή κάποιο κοινό σκοπό μεταξύ τους, μπορούν να θεωρηθούν ως μέλη μιας *foaf:Group*. Ο τρόπος που η FOAF διαχειρίζεται το ποιες οντότητες είναι μέλη ενός *foaf:Group*, βασίζεται στη χρήση της ιδιότητάς της *foaf:member*, η οποία έχει ως τιμή μια οντότητα κλάσης *foaf:Agent* ή *foaf:Person*. Με παρόμοιο τρόπο, με τη χρήση της ιδιότητας *foaf:membershipClass*, μπορεί ακόμα να δημιουργηθεί και μια υποκλάση της *foaf:Agent* ή *foaf:Person* προσαρμοσμένη απ' τον χρήστη, η οποία περιγράφει μια οντότητα που είναι μέλος μιας οντότητας *foaf:Group* (πχ: Για την περιγραφή ενός μέλους της ομάδας έρευνας και ανάπτυξης μιας εταιρείας, αρχικά δημιουργούμε μια ομάδα *foaf:Group* με όνομα “ResearchAndDevelopment”. Ύστερα, ορίζουμε μια υποκλάση της *foaf:Person* με όνομα “RDMember”, και θα είναι η κλάση που θα έχει το παραπάνω μέλος. Τέλος, θέτουμε την RDMember ως *membershipClass* της ResearchAndDevelopment, και τώρα οποιαδήποτε οντότητα της κλάσης RDMember, θα είναι μέλος της ResearchAndDevelopment.

Η κλάση *foaf:Organization* περιγράφει ένα οργανωμένο σώμα προσώπων ως μια ενιαία οντότητα. Λειτουργεί παρόμοια με την *foaf:Group* στο ότι και οι δύο κλάσεις περιγράφουν ένα σύνολο ατόμων, αλλά η μεγαλύτερη διαφορά είναι ότι η *foaf:Organization* δεν συγκροτείται από μεμονωμένες οντότητες *foaf:Agent* ή *foaf:Person*, και αντιπροσωπεύει ολόκληρη την ομάδα ως μια οντότητα.

Σε αντίθεση με την Video Game Ontology, η FOAF διατηρήθηκε ως πηγή της δικής μας οντολογίας, καθώς παρουσιάζει πολλές διαφορετικές χρήσεις, χάρη στην κεντρική κλάση της foaf:Agent. Παρακάτω περιγράφουμε τους λόγους που χρησιμοποιήσαμε την FOAF ως οντολογία-πηγή, καθώς και τους τρόπους που τη συνδέσαμε.

Μια πληροφορία των βιντεοπαιχνιδιών που αποτυπώνει η οντολογία μας είναι ποιοι ήταν οι δημιουργοί ενός παιχνιδιού, δηλαδή ποια εταιρεία ή ποιο πρόσωπο δημιούργησε ένα παιχνίδι. Καθώς η FOAF είναι αρκετά διαδεδομένη ως προς την περιγραφή ανθρώπινων οντοτήτων και την σαφήνεια της χρήσης της γι' αυτόν το σκοπό, αποφασίσαμε να την εντάξουμε στη δική μας οντολογία, ώστε να διαχειρίζεται εκείνη τον τομέα των δημιουργών παιχνιδιού. Ένα μέρος στο οποίο βοήθησε σημαντικά η FOAF ήταν το εξής: Όταν μια εταιρεία ανάπτυξης παιχνιδιών είναι ιδιαίτερα μεγάλη με εκατοντάδες υπαλλήλους, υπάρχει η πιθανότητα ότι οι ίδιοι υπάλληλοι δεν δημιούργησαν όλα τα παιχνίδια της εταιρείας αυτής, ή ότι κάποιοι υπάλληλοι δεν είναι καν σχεδιαστές παιχνιδιών, αλλά έχουν άλλους ρόλους μέσα στην εταιρεία, όπως επιχειρησιακά στελέχη. Σ' αυτήν την περίπτωση, για την ευκολότερη αντιστοίχιση παιχνιδιών με τους δημιουργούς τους, χρειάζεται ένας οργανωμένος τρόπος για την αποτύπωση κάθε μεμονωμένου ατόμου που δημιούργησε ένα παιχνίδι. Γι' αυτό, χρησιμοποιώντας τις τρεις υποκλάσεις της foaf:Agent, δημιουργήσαμε τρεις νέες υποκλάσεις με δικές τους ονομασίες και δικές τους μοναδικές ιδιότητες, επιπρόσθετες των αρχικών κλάσεων. Αυτές είναι:

- Η κλάση Individual είναι υποκλάση της foaf:Person, και περιγράφει ένα μεμονωμένο άτομο, το οποίο μπορεί να είναι υπάλληλος μιας εταιρείας που δημιούργησε ένα παιχνίδι, κάτω από το όνομα αυτής (τα άτομα αυτά εμφανίζονται τυπικά στα credits ενός παιχνιδιού) ή ένας ανεξάρτητος δημιουργός (indie developer).
- Η κλάση Team είναι υποκλάση της foaf:Group, και περιγράφει μια ομάδα ατόμων που αποτελείται από στοιχεία της κλάσης Individuals
- Η κλάση Company είναι υποκλάση της foaf:Organization, και περιγράφει μια εταιρεία ως δική της οντότητα

Wikidata

Το Wikidata, είναι η γνωσιακή βάση δεδομένων που χρησιμοποιεί το Wikipedia από το 2012. Σκοπός του Wikidata είναι η καλύτερη συσχέτιση και εύρεση όλων των λημμάτων που φιλοξενούνται στο Wikipedia, αλλά και για την βελτιστοποίηση της σύνδεσης των δεδομένων του με τις μηχανές αναζήτησης του διαδικτύου. Αν και λειτουργεί κατά κύριο λόγο ως το κεντρικό backend του Wikipedia αναγνώσιμο από μηχανή (machine-readable), το Wikidata φιλοξενείται σε δική του ιστοσελίδα, όπου κάθε στοιχείο της βάσης του μπορεί να διαβαστεί και από έναν ανθρώπινο χρήστη (human-readable).

Το Wikidata αποτελείται από ένα σύνολο χιλιάδων κλάσεων, ή αντικειμένων (items), οι οποίες αντιστοιχούν σε κάθε λήμμα του Wikipedia. Οι κλάσεις αυτές συνδέονται μεταξύ

τους με σημασιολογικό τρόπο, χρησιμοποιώντας δηλώσεις (statements), οι οποίες περιγράφουν το με ποιον τρόπο συνδέονται ή πώς συσχετίζονται.

Κάθε κλάση στη βάση του Wikidata αντιπροσωπεύεται από μια δική του ονομασία, έναν μοναδικό κωδικό που αρχίζει με το γράμμα Q και ακολουθείται από έναν ακέραιο αριθμό. Αυτός ο κωδικός είναι σε “γλώσσα” που μπορεί να διαβαστεί μόνο από μια μηχανή, αλλά ένας άνθρωπος δεν μπορεί να γνωρίζει τι αντιπροσωπεύει αυτός ο κωδικός. Γι’ αυτό, όλες οι κλάσεις του Wikidata συνοδεύονται από μια ετικέτα που μπορεί να διαβαστεί από άνθρωπο, αλλιώς human-readable label. Ακόμα, αυτές οι ετικέτες είναι μεταφρασμένες σε κάθε γλώσσα στην οποία είναι διαθέσιμο το αντίστοιχο λήμμα στο Wikipedia, πχ. η κλάση Q144 που αντιπροσωπεύει την κλάση του σκύλου, έχει την Αγγλική ετικέτα “Dog”, την αντίστοιχη Ελληνική “Σκύλος”, την αντίστοιχη Γαλλική “Chien” κ.ο.κ. Αυτή η κωδικοποίηση βοηθάει στη διατήρηση μοναδικών αναγνωριστικών, τα οποία ενώ μπορεί να έχουν την ίδια ονομασία, αναφέρονται σε διαφορετικά πράγματα.



Το λογότυπο του Wikidata

dog (Q144)

domestic animal

domestic dog | *Canis lupus familiaris* | *Canis familiaris* | dogs | 🐶 | 🐕

[In more languages](#)

[Configure](#)

Language	Label	Description	Also known as
English	dog	domestic animal	domestic dog <i>Canis lupus familiaris</i> <i>Canis familiaris</i> dogs 🐶 🐕
Greek	σκύλος	No description defined	σκυλί κύων <i>canis lupus familiaris</i>
French	chien	sous-espèce domestique de <i>Canis lupus</i>	<i>canis lupus familiaris</i> <i>canis familiaris</i> <i>canis canis</i> <i>canis familiaris domesticus</i> 🐶 🐕
German	Haushund	als Heim- und Nutztier gehaltenes Haustier	Hund 🐶 🐕

Η κλάση του σκύλου στο Wikidata, μαζί με τις human-readable ετικέτες του

Με παρόμοιο τρόπο ορίζονται και οι δηλώσεις, όπου κάθε είδος δήλωσης έχει δικό του κωδικό, απλώς αντί για το γράμμα Q, ξεκινά με το γράμμα P, και ακολουθείται από έναν ακέραιο αριθμό. Όπως οι κλάσεις που περιγράψαμε παραπάνω, οι δηλώσεις χρησιμοποιούν

αυτόν τον κωδικό για να είναι αναγνώσιμες από τη μηχανή, και έχουν ετικέτες μεταφρασμένες σε κάθε υποστηριζόμενη γλώσσα.

Πολύ χρήσιμο εργαλείο στη συλλογή δεδομένων για τη δημιουργία του dataset της οντολογίας είναι το εργαλείο Wikidata Query Service το οποίο είναι διαθέσιμο στη σελίδα του Wikidata. Όπως λέει και το όνομά του, βασική του λειτουργία είναι η ανάθεση ερωτημάτων στην εσωτερική οντολογία του Wikidata με τη χρήση SPARQL ερωτημάτων. Το Wikidata Query Service ακόμη υποστηρίζει την αποθήκευση αποτελεσμάτων που προκύπτουν από ερωτήματα που γίνονται σ' αυτό σε μορφή JSON, CSV και άλλα. Ακόμη μια χρήσιμη δυνατότητα του Wikidata Query Service είναι η αυτόματη δημιουργία κώδικα σε διάφορες γλώσσες και πρότυπα, όπως JavaScript, Python, Ruby και PHP, για τη χρήση του επιλεγμένου ερωτήματος από οποιαδήποτε εφαρμογή.

1.2. Συγγραφή της οντολογίας

Με τα εργαλεία ανάπτυξης έτοιμα, η συγγραφή της οντολογίας είναι πιο εύκολη. Αρχικά, χρησιμοποιήσαμε ως βάση την VGO, η οποία είναι γραμμένη σε σύνταξη OWL/XML, την οποία έπρεπε πρώτα να μετατρέψουμε σε σύνταξη TTL. Ο λόγος που απαιτείται αυτή η μετατροπή είναι ότι η χρήση ευκολότερης στην κατανόηση σύνταξης τριπλετών RDF οδηγεί σε μια απλούστερη οντολογία, κάτι που βοηθά στη συντήρηση και μελέτη της.

Για τη μετατροπή του αρχείου, μέσω της γραμμής εντολών, μεταφερόμαστε στο φάκελο που αποθηκεύσαμε το OWL αρχείο της VGO, και χρησιμοποιούμε το εργαλείο του Apache Jena, riot, με την εντολή:

```
riot --out=turtle GameOntologyv3.owl > GameOntologyv3.ttl
```

Ύστερα, σ' έναν επεξεργαστή κειμένου, γράψαμε τις πρώτες κλάσεις της οντολογίας με το χέρι, αλλάζοντας τις ιδιότητες των υπαρχουσών κλάσεων ώστε να ταιριάζουν με το αρχικό πλάνο, ακολουθώντας ταυτόχρονα τη σωστή σύνταξη της TTL. Σε επόμενα στάδια, η πλήρης ανάπτυξη της οντολογίας έγινε μέσα στο περιβάλλον του *Protégé*, το οποίο βοήθησε στον καλύτερο έλεγχο των ιδιοτήτων των κλάσεων και των συσχετίσεών τους, αλλά και για την περαιτέρω βελτίωση της δομής της οντολογίας.

File Edit View Reasoner Tools Refactor Window Help

OntologyID(Anonymous-6) Search...

Game

Active ontology x Entities x Individuals by class x DL Query x

Individuals

Annotation properties Datatypes

Data properties

Classes Object properties

Class hierarchy: Game

owl:Thing

- foaf:Agent
- foaf:Group
 - Team
- foaf:Organization
 - Company
- foaf:Person
 - Individual
- Game
 - Game Developer
 - Company
 - Individual
 - Team
 - Game Publisher
 - Company
 - Individual
 - Team
 - Genre
 - Platform
 - Series

Game — http://purl.org/net/MyOntology#Game

Annotations Usage

Usage: Game

Show: ☒ this ☒ disjoints ☒ named sub/superclasses

Found 20 uses of Game

- Game
 - Game rdfs:label "Game"@en
 - Class: Game
 - Game rdfs:comment "A Game"@en
- has game genre
 - 'has game genre' Domain Game
- is developed by
 - 'is developed by' Domain Game
- is part of series
 - 'is part of series' Domain Game
- is published by
 - 'is published by' Domain Game
- is related to
 - 'is related to' Range Game
 - 'is related to' Domain Game
- is released on platform
 - 'is released on platform' Domain Game
- release date
 - 'release date' Domain Game

Description: Game

Equivalent To +

SubClass Of +

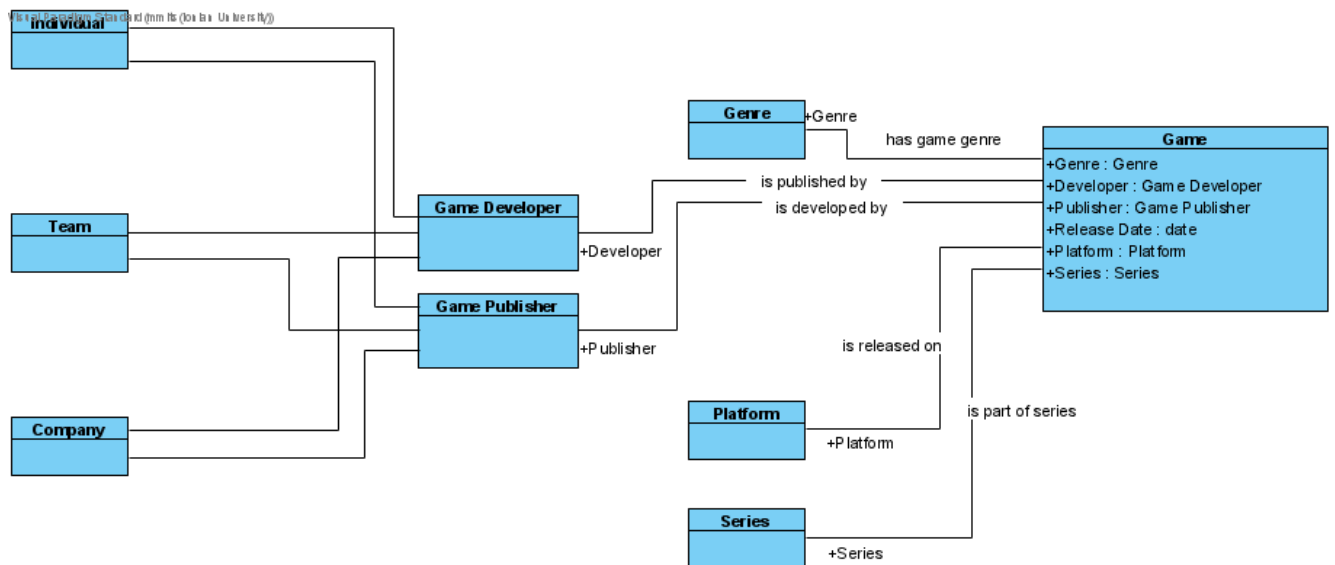
General class axioms +

SubClass Of (Anonymous Ancestor)

To use the reasoner click Reasoner > Start reasoner ☒ Show Inferences

2. Δομή της Οντολογίας

Βασική νοοτροπία για την δόμηση της οντολογίας ήταν η επιλογή των στοιχείων εκείνων που είναι βασικότερα στην αναγνώριση μιας μοναδικής οντότητας παιχνιδιού. Με βάση και την προσωπική γνώση πάνω στο θέμα των ηλεκτρονικών παιχνιδιών, η επιλογή αυτή κατέληξε σε 6 βασικά χαρακτηριστικά: ένα παιχνίδι, η πλατφόρμα κυκλοφορίας του, το είδος του, ο δημιουργός του, ο εκδότης του και η σειρά στην οποία ανήκει, αν υπάρχει. Ακόμα, σχεδιάσαμε μια γραφική αναπαράσταση αυτών των κλάσεων, και πώς θα πρέπει να συνδέεται η κάθε μία μέσα στην οντολογία:



2.1. Κλάσεις

Με βάση αυτήν την εισαγωγική νοοτροπία, και το παραπάνω σχέδιο, αναπτύξαμε την οντολογία με τις παρακάτω κλάσεις:

- Game
- Genre
- GameDeveloper
- GamePublisher
- Platform
- Series

απ' τις οποίες, οι GameDeveloper και GamePublisher έχουν 3 υποκλάσεις που μοιράζονται μεταξύ τους:

- Individual
- Team
- Company

Ακολουθεί μια περιγραφική επεξήγηση κάθε μιας κλάσης που αναφέραμε:

Game

Ένα παιχνίδι το οποίο λειτουργεί σ' έναν ψηφιακό χώρο, και αλληλεπιδρά με τον χρήστη χρησιμοποιώντας μια μορφή απεικόνισης.

Αυτή η κλάση είναι το ιδιαίτερο αντικείμενο που μελετά η οντολογία και το κεντρικό μέρος της. Πολλές απ' τις υπόλοιπες κλάσεις της οντολογίας συσχετίζονται με την κλάση Game ως περιγραφές των ιδιοτήτων της, οι οποίες είναι οι παρακάτω:

- Genre (είδος παιχνιδιού)
- GameDeveloper (σχεδιαστής παιχνιδιού)
- GamePublisher (εκδότης παιχνιδιού)
- Platform (πλατφόρμα κυκλοφορίας)
- Series (σειρά παιχνιδιού)

και μια ιδιότητα, η releaseDate (ημερομηνία κυκλοφορίας).

Genre

Το είδος ενός παιχνιδιού στο οποίο κατηγοριοποιείται, και περιγράφει το περιεχόμενο, τον σκοπό ή τον τρόπο που παίζεται. Παραδείγματα οντοτήτων Genre είναι Action (δράσης), Roleplaying (παιχνίδι ρόλων), Puzzle (επίλυση γρίφων), Shooter (σκοποβολής), και άλλα.

Οι οντότητες της Genre μπορεί έχουν συγκεκριμένο αριθμό, καθώς η κατηγοριοποίηση των παιχνιδιών με βάση το είδος τους δεν ευνοεί την προσθήκη ολοένα και περισσότερων κατηγοριών.

GameDeveloper

Σχεδιαστής και δημιουργός ενός παιχνιδιού, ο οποίος συνεισφέρει στην ανάπτυξή του με διάφορους τρόπους.

Έχει τρεις υποκλάσεις τις οποίες μοιράζεται με την GamePublisher, τις:

- Individual
- Team
- Company

Σε περίπτωση όπου ο σχεδιαστής του παιχνιδιού είναι μια εταιρεία, ταυτόχρονα με την οντότητα Company της εταιρείας, GameDeveloper ενός Game θα είναι και οντότητες ξεχωριστών υπαλλήλων της, δηλαδή οντότητες Individual.

GamePublisher

Ο εκδότης ενός παιχνιδιού, ο οποίος διαχειρίζεται την προώθηση ενός παιχνιδιού στην αγορά. Περιγράφεται στην οντολογία λόγω της συμβολής που μπορεί να έχει στην ύπαρξη ενός παιχνιδιού, καθώς χωρίς την έκδοση ενός παιχνιδιού, αυτό ουσιαστικά δεν υπάρχει.

Έχει τρεις υποκλάσεις τις οποίες μοιράζεται με την GameDeveloper, τις:

- Individual
- Team
- Company

Οι κλάσεις GameDeveloper και GamePublisher είναι διακριτές σε δύο κλάσεις αντί ο σχεδιαστής και ο εκδότης ενός παιχνιδιού να αποτυπώνεται με μόνο τις κλάσεις Individual, Team και Company, ώστε να μην υπάρχει σύγχυση μεταξύ των δύο ρόλων, αλλά και για να υπάρχει ευκολία κατά την εύρεση των δύο ρόλων ξεχωριστά (δηλαδή, ως παράδειγμα ερώτησης πάνω στην οντολογία, να είναι πιο εύκολο κανείς να θέσει ένα query με μόνο τα παιχνίδια που έχει εκδώσει μια οντότητα χρησιμοποιώντας μόνο την κλάση GamePublisher, παρά τα παιχνίδια που έχει σχεδιάσει).

Individual

Ένα μοναδικό άτομο το οποίο μπορεί να είναι ένας υπάλληλος μιας εταιρείας ανάπτυξης παιχνιδιών, μέλος μιας ομάδας ανάπτυξης παιχνιδιών, ή ένας ανεξάρτητος δημιουργός. Η κλάση Individual συνήθως χρησιμοποιείται ως υποκλάση μόνο της GameDeveloper, αλλά σε περιπτώσεις όπου από την εταιρεία έκδοσης του παιχνιδιού έχει αναφερθεί κάποιο συγκεκριμένο άτομο στη λίστα συντελεστών το οποίο έχει συνεισφέρει με οποιονδήποτε τρόπο στη δημιουργία του παιχνιδιού, αυτό θα εμφανίζεται ως οντότητα Individual και υποκλάση της GamePublisher.

Είναι υποκλάση της foaf:Person, για την υιοθέτηση των ιδιοτήτων της, καθώς και των GameDeveloper και GamePublisher που αποτυπώνουν ποιο ρόλο έχει.

Team

Μια ομάδα ατόμων τα οποία έχουν κάποιο κοινό σκοπό, αλλά ορίζονται και σαν ξεχωριστά άτομα. Στην έκταση της οντολογίας μας, χρησιμοποιείται σε περιπτώσεις όπου ένα παιχνίδι αναπτύχθηκε από μεμονωμένα άτομα, αλλά αντί να οριστούν ως εταιρεία, ορίζονται συγκεντρωτικά ως μια ομάδα, καθώς σε πολλές περιπτώσεις, μια τέτοια ομάδα δεν έχει την ίδια δύναμη (ή χρηματοδότηση) όπως μια εταιρεία.

Η κλάση Team είναι υποκλάση της foaf:Group, και γι' αυτό το λόγο υιοθετεί τις ιδιότητές της ως σύνθεση άλλων οντοτήτων, στην περίπτωση μας οντότητες Individual.

Ακόμη, είναι υποκλάση των GameDeveloper και GamePublisher, οι οποίες αποτυπώνουν ποιο ρόλο έχει μια οντότητα Team.

Company

Μια εταιρεία ως ξεχωριστή οντότητα.

Η κλάση Company βοηθά στις περιπτώσεις όπου χρειάζεται η αποτύπωση μιας γενικευμένης οντότητας ως σχεδιαστής ή εκδότης ενός παιχνιδιού αν τα πρόσωπα είναι άγνωστα, ή για τη χρήση ενός πιο αναγνωρίσιμου ονόματος (πχ περισσότεροι γνωρίζουν την Atari ως δημιουργό του Pong, παρά τον Allan Alcorn ο οποίος ήταν ο σχεδιαστής και προγραμματιστής του).

Φυσικά, οντότητες της κλάσης Company μπορούν να συσχετιστούν με ένα παιχνίδι παράλληλα και με άλλες οντότητες της κλάσης Individual και Team όπου χρειάζεται.

Είναι υποκλάση της foaf:Organization, για την υιοθέτηση των ιδιοτήτων της, καθώς και των GameDeveloper και GamePublisher που αποτυπώνουν ποιο ρόλο έχει.

Platform

Η πλατφόρμα στην οποία κυκλοφόρησε ένα παιχνίδι, η οποία μπορεί να είναι σε φυσική μορφή όπως κάποια κονσόλα σαν το PlayStation, ή σε ψηφιακή μορφή όπως το online κατάστημα Steam.

Πολλά παιχνίδια συχνά επανακυκλοφορούν σε νέες πλατφόρμες ώστε να είναι πιο εύκολα διαθέσιμα. Αυτά τα παιχνίδια, αν δεν έχουν αλλάξει δραστικά ή συμπεριλαμβάνονται σε μια συλλογή, ορίζονται στην οντολογία με τη χρήση της αντίστοιχης οντότητας Platform, αντί για τη δημιουργία μιας ξεχωριστής οντότητας Game. Για παράδειγμα, το παιχνίδι Super Mario 64, που αρχικά κυκλοφόρησε στην πλατφόρμα Nintendo 64, συμπεριλαμβάνεται στη συλλογή Super Mario 3D All-Stars για το Nintendo Switch χωρίς αλλαγές, άρα αυτή η οντότητα Game θα έχει και ως ιδιότητα Platform το Nintendo Switch.

Series

Μια σειρά παιχνιδιών στην οποία μπορεί να ανήκει ένα παιχνίδι. Η κατηγοριοποίηση παιχνιδιών με βάση τη σειρά που ανήκουν, βοηθά τόσο στην εξερεύνηση παρόμοιων παιχνιδιών, όσο και στην ευκολότερη καταμέτρησή τους.

Κάποια παιχνίδια μπορεί ν' ανήκουν σε παραπάνω από μία σειρά, όπως το Makai Tōshi SaGa το οποίο, ενώ αρχικά δημιουργήθηκε σαν παιχνίδι της σειράς SaGa, όταν μεταφέρθηκε στη δύση ονομάστηκε The Final Fantasy Legend, κι έτσι ταυτόχρονα θεωρείται και παιχνίδι της σειράς Final Fantasy.

2.2. Συσχετίσεις / Ιδιότητες αντικειμένων

hasGameGenre

Η συσχέτιση που ορίζει ένα είδος ως ιδιότητα ενός παιχνιδιού στο οποίο κατηγοριοποιείται, όπου domain της είναι η κλάση Game και range της μια οντότητα Genre.

Κάποιες φορές ένα παιχνίδι μπορεί να ανήκει σε παραπάνω από μία κατηγορία είδους, γι' αυτό μπορεί να έχει παραπάνω από μία ιδιότητα hasGameGenre, αλλά σπάνια πάνω από δύο.

isDevelopedBy

Η συσχέτιση που ορίζει έναν σχεδιαστή ως ιδιότητα ενός παιχνιδιού, όπου domain της είναι η κλάση Game και range της μια οντότητα GameDeveloper. Ακόμα, η συσχέτιση isDevelopedBy έχει μια υποϊδιότητα, την hasRole.

Μια οντότητα Game μπορεί να έχει παραπάνω από μία ιδιότητα isDevelopedBy, καθώς ο σχεδιαστής του παιχνιδιού μπορεί να είναι παραπάνω από ένας.

isPublishedBy

Η συσχέτιση που ορίζει έναν εκδότη ως ιδιότητα ενός παιχνιδιού, όπου domain της είναι η κλάση Game και range της μια οντότητα GamePublisher.

Μια οντότητα Game μπορεί να έχει παραπάνω από μία ιδιότητα isPublishedBy, ειδικά όταν ένα παιχνίδι επανεκδίδεται σε άλλες πλατφόρμες ή σε άλλες περιοχές ή χώρες.

isReleasedOn

Η συσχέτιση που ορίζει μια πλατφόρμα ως ιδιότητα ενός παιχνιδιού στην οποία κυκλοφόρησε, όπου domain της είναι η κλάση Game και range της μια οντότητα Platform.

Πολλά παιχνίδια κυκλοφορούν σε παραπάνω από μία πλατφόρμα, γι' αυτό πολλές οντότητες Game έχουν περισσότερες από μία ιδιότητες isReleasedOn.

isPartOfSeries

Η συσχέτιση που ορίζει μια σειρά ως ιδιότητα ενός παιχνιδιού στην οποία ανήκει, όπου domain της είναι η κλάση Game και range της μια οντότητα Series.

Σε κάποιες περιπτώσεις, ένα παιχνίδι μπορεί ν' ανήκει σε παραπάνω από μία σειρά παιχνιδιών, γι' αυτό ορισμένες οντότητες Game έχουν περισσότερες από μία ιδιότητες isPartOfSeries.

isRelatedTo

Η συσχέτιση που συνδέει δύο παιχνίδια μεταξύ τους, όπου domain της είναι η κλάση Game και range της μια άλλη οντότητα Game. Ακόμα, η συσχέτιση isRelatedTo έχει μια υποιδιότητα, την hasRelation.

Η συσχέτιση μεταξύ δύο παιχνιδιών που ορίζει η isRelatedTo είναι πιο στενή απ' της isPartOfSeries, καθώς, για παράδειγμα, το Persona 4 και η εκτεταμένη επανέκδοσή του Person 4 Golden είναι παιχνίδια της σειράς Persona, αλλά είναι πιο στενά συνδεδεμένα μεταξύ τους, παρά με άλλα παιχνίδια της υπόλοιπης σειράς. Γι' αυτό, αυτές οι δύο οντότητες θα έχουν και την οντότητα Persona ως ιδιότητα isPartOfSeries, και θα είναι η μία ιδιότητα isRelatedTo της άλλης.

2.3. Ιδιότητες δεδομένων

releaseDate

Η ημερομηνία στην οποία κυκλοφόρησε ένα παιχνίδι.

Είναι μια ιδιότητα δεδομένων της κλάσης Game και παίρνει ως τιμή της μια ημερομηνία σε μορφή xsd:date.

Σε περίπτωση που οποιοδήποτε μέρος της ημερομηνίας είναι άγνωστο, θέτουμε το αντίστοιχο μέρος του format σε 1900 (έτος), 01 (μήνας) και 01 (ημέρα) αντίστοιχα.

5

Ανάπτυξη της Εφαρμογής

Η ανάπτυξη της εφαρμογής έγινε στο περιβάλλον του framework ReactJS, μια απ' τις δημοφιλέστερες βιβλιοθήκες της JavaScript για την ανάπτυξη διαδικτυακών εφαρμογών. Ειδικότητα της React είναι να επιτρέπει την δημιουργία στοιχείων, τα λεγόμενα components, τα οποία μπορούν να εμφωλευτούν μέσα σε άλλα ή να χρησιμοποιηθούν ξανά σε διάφορα μέρη της εφαρμογής, χωρίς να απαιτείται η επανάληψη κώδικα. Αυτή η ειδικότητα βοήθησε στην καλύτερη οργάνωση της εφαρμογής, όπως για παράδειγμα στις λίστες αντικειμένων μπορούμε απλά να επαναλάβουμε την κλήση του component που θέλουμε να ενσωματωθεί στη λίστα. Ακόμη, με την χρήση των components, η ανάπτυξη του κώδικα γίνεται σε μορφή εμφωλευμένων blocks, κάτι που επιτρέπει τον καλύτερο έλεγχο κάθε ξεχωριστής λειτουργίας της εφαρμογής. Αυτό βοηθάει στο να μπορούμε να εντοπίσουμε ποιες λειτουργίες χρειάζονται διόρθωση ή περισσότερη ανάπτυξη.

1. Δομή του Κώδικα

Παρακάτω εμφανίζεται η λίστα με τη δομή των αρχείων και components που χρησιμοποιήθηκαν και τον γενικό τρόπο με τον οποίο αυτά εμφωλεύονται. Τα υπογραμμισμένα στοιχεία είναι τα ονόματα των φακέλων που περιέχουν κάθε component:

```
| index.js
| -App
| --common
| ---Top
| ---SideNav
| ----CategoryLink
| ---SideBlank
| ---Footer
| --home
| ---HomeMain
| ----HomeDesc
| ----HomeRands
| -----RandGameLeft
| -----RandGameRight
```

```
|-----RandGameContent
|--search
|---SearchMain
|----SearchOptions
|-----SearchResults
|--lists
|---ListsMain
|----ListsHome
|--game
|---GameMain
|----GameData
|-----GameOther
|-----Suggestions
|--genre
|---GenreMain
|----GenreData
|----GenreGames
|--platform
|---PlatformMain
|----PlatformData
|----PlatformGames
|--static
|---Framework
|---About
```

Σε αυτό το σημείο, θα περιγράψουμε αναλυτικά κάθε αρχείο της παραπάνω λίστας, βασικές λειτουργίες του, τον σκοπό που έχει στην εφαρμογή, και μερικά αποσπάσματα του κώδικα που περιέχει. Το σύνολο του κώδικα είναι ανοιχτά διαθέσιμο σε αποθετήριο του GitHub με τον ακόλουθο σύνδεσμο: <https://github.com/mmits/Game-Ontology-Web-App>

index.js

Το index.js είναι το βασικό αρχείο που διαβάζει το σύστημα του React κατά το compile της εφαρμογής. Εκτός από την κλήση του component App, δεν έχει άλλο περιεχόμενο, ώστε τυχόν επεξεργασίες που γίνονται στην εφαρμογή να μπορούν να τεκμηριώνονται πιο εύκολα. Για παράδειγμα, είναι πιο εύκολο να μιλήσουμε για χαρακτηριστικά και αλλαγές στο component App, το οποίο θεωρείται ως η ρίζα της εφαρμογής, παρά στο index.js.

App

Σαν θεμέλιο και ταυτόχρονα κεντρικό σημείο της εφαρμογής, δημιουργήσαμε το component App, το οποίο είναι το μοναδικό component στο οποίο κάνει κλήση το index.js για την εξαγωγή της εφαρμογής. Με άλλα λόγια, αν μπορούμε να θεωρήσουμε το αρχείο index.js

ως ρίζα της εφαρμογής, τότε και το component App είναι ο κορμός της. Ως συγκεντρωτικό component, περιλαμβάνει όλα τα στοιχεία που εμφανίζονται πάντα στην οθόνη, όπως η γραμμή πλοήγησης με χρήσιμους συνδέσμους και η αριστερή λίστα γρήγορων αναζητήσεων, και φυσικά το βασικό περιεχόμενο κάθε σελίδας.

Τα περιεχόμενά του αλλάζουν σπάνια, με μόνο το κεντρικό component να αλλάζει ανά σελίδα, γι' αυτό γίνεται κλίση κάθε σταθερού component εδώ, ώστε ο χρόνος φόρτωσης της εφαρμογής να παραμένει ελάχιστος. Αυτά τα σταθερά components διαχειρίζονται σταθερούς συνδέσμους σε διάφορα άλλα μέρη της εφαρμογής, όπως η λειτουργία αναζήτησης και η εμφάνιση παιχνιδιών με τη χρήση συγκεκριμένων φίλτρων.

Το μεταβλητό μέρος του component App διαχειρίζεται από έναν Router, ο οποίος αποτυπώνει διαφορετικό περιεχόμενο ανάλογα με την τρέχουσα διεύθυνση. Ο Router καλεί έναν Switch container, ο οποίος θα διαχειρίζεται το σημείο εκείνο στο οποίο πρέπει ένα component να αντικαταστήσει ένα άλλο, το καθένα απ' αυτά ορισμένο με ένα Route. Τα Routes που επιλέγονται αντιστοιχούν σε κάποια διεύθυνση URL με το “μονοπάτι” (path) που θα οδηγεί σ' αυτό, το οποίο προσαρτάται στη βασική διεύθυνση της εφαρμογής. Παρακάτω εμφανίζουμε το απόσπασμα κώδικα το οποίο διαχειρίζεται το Switching των περιεχομένων:

```
<Switch>
  <Route exact path="/" component = {HomeMain} />
  <Route exact path="/game/:id" component={GameMain} />
  <Route exact path="/platform/:id" component={PlatformMain} />
  <Route exact path="/genre/:id" component={GenreMain} />
  <Route exact path="/search" component={SearchMain} />
  <Route exact path="/lists" component={ListsMain} />
  <Route exact path="/lists/:id" component={ListsMain} />
  <Route exact path="/framework" component={FrameworkPage} />
</Switch>
```

Κάθε ένα απ' αυτά τα Routes αντιστοιχεί σε μια διαφορετική λειτουργία ή σελίδα της εφαρμογής, ενώ οι σελίδες κάθε παιχνιδιού, είδους ή πλατφόρμας ακολουθούνται από μια μοναδική ονομασία που ορίζεται με την μεταβλητή id, ώστε να διαμορφώνεται κάθε σελίδα και τα δεδομένα της αυτόματα.

common

Ο φάκελος που περιέχει τα components που χρησιμοποιούνται καθολικά σε κάθε σελίδα της εφαρμογής, όποιο και να είναι το κεντρικό περιεχόμενο που βρίσκεται στο Switch container. Αποτελούν τα δομικά μέρη μιας διεπαφής χρήστη, που σκοπό έχουν να του προσφέρουν γρήγορους τρόπους πρόσβασης στις διάφορες λειτουργίες της εφαρμογής, με το απλό πάτημα ενός συνδέσμου.

Κάθε ένα από τα components που βρίσκονται σ' αυτόν το φάκελο περιγράφονται παρακάτω.

Top

Το component της γραμμής πλοήγησης της εφαρμογής. Όπως τα υπόλοιπα στοιχεία της διεπαφής χρήστη, χρησιμοποιείται παντού στην εφαρμογή και παραμένει στατική για να μειώνεται ο χρόνος φόρτωσής της. Η γραμμή πλοήγησης περιέχει διάφορους συνδέσμους για την πρόσβαση σε κάθε λειτουργία της εφαρμογής. Αυτοί οι σύνδεσμοι είναι:

- Το λογότυπο της εφαρμογής, που οδηγεί στην αρχική σελίδα
- Ο “Search” για τη σελίδα της λειτουργίας της αναζήτησης
- Ο “Lists” για τη σελίδα με όλες τις διαθέσιμες λίστες από αντικείμενα ανάλογα με την κατηγορία
- Ο “Recent” για τη σελίδα των νέων προσθηκών στην εφαρμογή που έγιναν απ’ τον διαχειριστή
- Ο “Framework” για τη σελίδα με την ενημέρωση και διάφορες πληροφορίες σχετικά τη δομή της εφαρμογής
- Ο “About” για τη σελίδα με το υπόβαθρο της εφαρμογής, όπως ιστορικά του στοιχεία.

SideNav

Το component της αριστερής γραμμής πλοήγησης της εφαρμογής. Όπως τα υπόλοιπα στοιχεία της διεπαφής χρήστη, χρησιμοποιείται παντού στην εφαρμογή και παραμένει στατική για να μειώνεται ο χρόνος φόρτωσής της. Η γραμμή αυτή περιέχει μια λίστα από διάφορους γρήγορους συνδέσμους που είναι λιγότερο σημαντικοί και δεν εμφανίζονται στην γραμμή πλοήγησης του component Top. Οι σύνδεσμοι αυτοί είναι κυρίως λίστες από γενικές εγγραφές της εφαρμογής, όπως η εμφάνιση μιας λίστας με όλες τις σελίδες πλατφορμών, και χρησιμεύουν στη γρηγορότερη πρόσβαση σε γενικές πληροφορίες, αντί την εφαρμογή μιας αναζήτησης. Το όνομα και το path κάθε συνδέσμου ορίζεται μέσα σ’ αυτό το component με μεταβλητές, και ύστερα πραγματοποιούνται σ’ ένα στιγμιότυπο του component `CategoryLink`, όπως παρακάτω:

```
<CategoryLink name = "All Games" link = "/lists/games"/>
```

όπου “All Games” είναι το εμφανιζόμενο όνομα του συνδέσμου, και “/lists/games” είναι το path του συνδέσμου.

CategoryLink

Το component που καλείται από το SideNav, και απλός του σκοπός είναι η παραμετροποίηση μεμονωμένων συνδέσμων που στέλνονται απ’ αυτό το component, αλλά και η καθαρότερη εμφάνιση του κώδικά του.

SideBlank

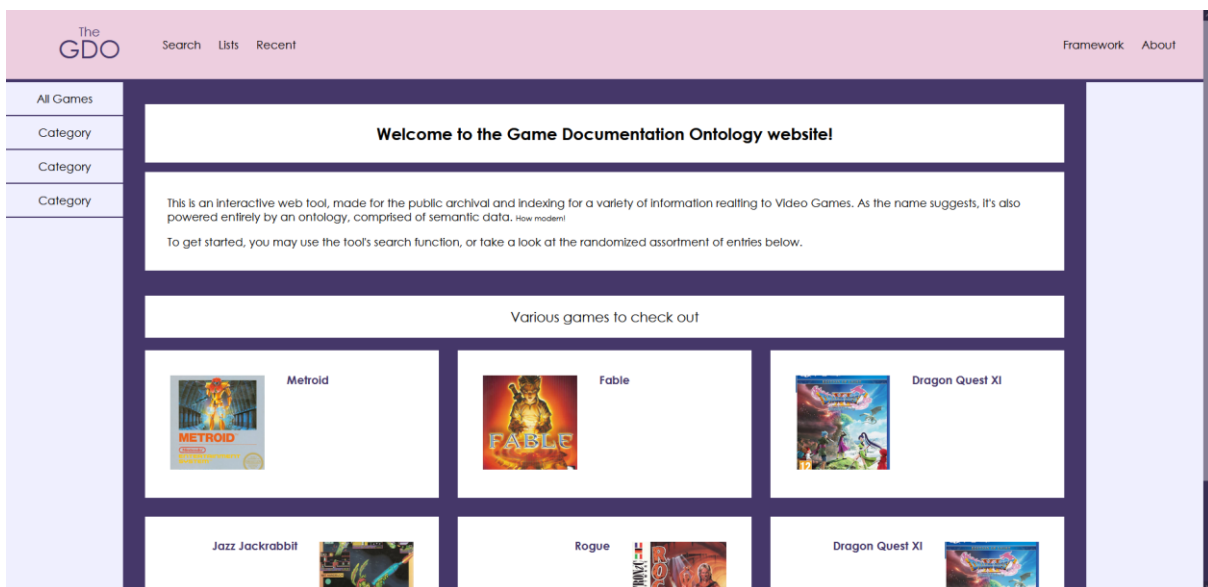
Ένα από τα components που χρησιμοποιούνται παντού στην εφαρμογή για τη διεπαφή χρήστη, το οποίο, ενώ είναι άδειο, βοηθά στην στοίχιση των περιεχομένων της σελίδας, ωθώντας το κεντρικό περιεχόμενο της εφαρμογής να παραμένει στοιχισμένο στο κέντρο.

Καθώς για το layout της εφαρμογής χρησιμοποιείται κυρίως η μέθοδος Flexbox, χωρίς αυτό, τα περιεχόμενα εκτείνονται μέχρι το δεξί άκρο του παραθύρου, και δεν φαίνονται ομοιόμορφα στην εμφάνιση. Έτσι, ορίζοντας την ίδια style class με το component SideNav, έχουμε ένα συμμετρικό layout, το οποίο χάρη και στο Flexbox προσαρμόζεται ανάλογα με το μέγεθος του παραθύρου.

Footer

Το component με μερικές πρόσθετες πληροφορίες σε μορφή κειμένου σχετικά με την εφαρμογή, κυρίως το όνομα του δημιουργού και το έτος δημιουργίας. Χρησιμοποιείται παντού στην εφαρμογή και παραμένει στατικό στο τέλος κάθε σελίδας για να μειώνεται ο χρόνος φόρτωσής της, αλλά δεν παρέχει κάποια διάδραση, καθώς είναι απλό κείμενο.

home



Ο φάκελος που περιέχει όλα τα components που εμφανίζονται στην αρχική σελίδα της εφαρμογής. Η αρχική σελίδα της εφαρμογής φέρει μια γρήγορη πρώτη ματιά για τον σκοπό της, δηλαδή τι μπορεί να βρει ένας επισκέπτης σ' αυτήν την εφαρμογή. Γι' αυτό, πρέπει με σύντομο τρόπο να αποτυπώσει τις βασικότερες λειτουργίες της, και να ωθήσει τον χρήστη να την εξερευνήσει περαιτέρω.

Η αρχική σελίδα αποτελείται από έναν χαιρετισμό, μια σύντομη περιγραφή του σκοπού της, και τέλος μια λίστα από τυχαίες σελίδες παιχνιδιών, για τη γρήγορη πρόσβαση στη βασική λειτουργία της εφαρμογής, αλλά και για την διαδραστικότητα σε μια στατική αρχική σελίδα.

Τα components που βρίσκονται σ' αυτόν το φάκελο, και κατ' επέκταση που χρησιμοποιούνται σ' αυτή τη σελίδα, περιγράφονται παρακάτω.

HomeMain

Το συγκεντρωτικό component της αρχικής σελίδας. Δεν έχει δικά του περιεχόμενα, αλλά καλεί τα παρακάτω components:

- HomeDesc, με μια εισαγωγική παράγραφο σχετικά με τον σκοπό της εφαρμογής και διάφορες άλλες πληροφορίες
- HomeRands, με μια λίστα από συνδέσμους για τυχαία επιλεγμένα παιχνίδια

HomeDesc

Το component με έναν χαιρετισμό και μια σύντομη εισαγωγή που περιγράφει τον σκοπό της εφαρμογής. Είναι το πρώτο component που καλεί το HomeMain για την αρχική σελίδα. Τα περιεχόμενα του HomeDesc βρίσκονται σε ξεχωριστό component αντί να είναι μέρος του HomeMain, ώστε να μπορούν να αλλάζονται πιο εύκολα απ' τον διαχειριστή της εφαρμογής.

HomeRands

Το component που εμφανίζει μια σειρά από τυχαία επιλεγμένα παιχνίδια στην αρχική σελίδα. Προκειμένου την καλύτερη εμφάνιση της σελίδας, τα τυχαία παιχνίδια εμφανίζονται σε δύο σειρές με εναλλασσόμενη στοίχιση, και γι' αυτό χρησιμοποιούνται δύο διαφορετικά components για την εκτύπωση των περιεχομένων, τα *RandGameLeft* και *RandGameRight*. Ακόμη, ο διαχειριστής μπορεί να ορίσει τον δικό του αριθμό τυχαίων παιχνιδιών που θα εμφανίζονται σε κάθε σειρά, που στην περίπτωση της δικής μας εφαρμογής είναι 3. Το παρακάτω απόσπασμα κώδικα δείχνει αυτή τη δυναμική εμφάνιση αντικειμένων, το οποίο χρησιμοποιείται και σε άλλα components της εφαρμογής:

```
var itemsLeft = [];  
var itemsRight = [];  
for (var i = 0; i < 3; i++) {  
    itemsLeft.push(<RandGameLeft />);  
    itemsRight.push(<RandGameRight />);  
}
```

RandGameLeft

Ένα απ' τα components εμφάνισης τυχαίων παιχνιδιών, όμοια με το *RandGameRight*, το οποίο στοιχίζει τα περιεχόμενα του *RandGameContent* στα αριστερά με τη χρήση παραμέτρων όπως παρακάτω:

```
<RandGameContent flexDirection = "row" />
```

Καλείται από το component *HomeRands*, μαζί με το *RandGameRight*.

RandGameRight

Ένα απ' τα components εμφάνισης τυχαίων παιχνιδιών, όμοια με το RandGameLeft, το οποίο στοιχίζει τα περιεχόμενα του *RandGameContent* στα δεξιά με τη χρήση παραμέτρων όπως παρακάτω:

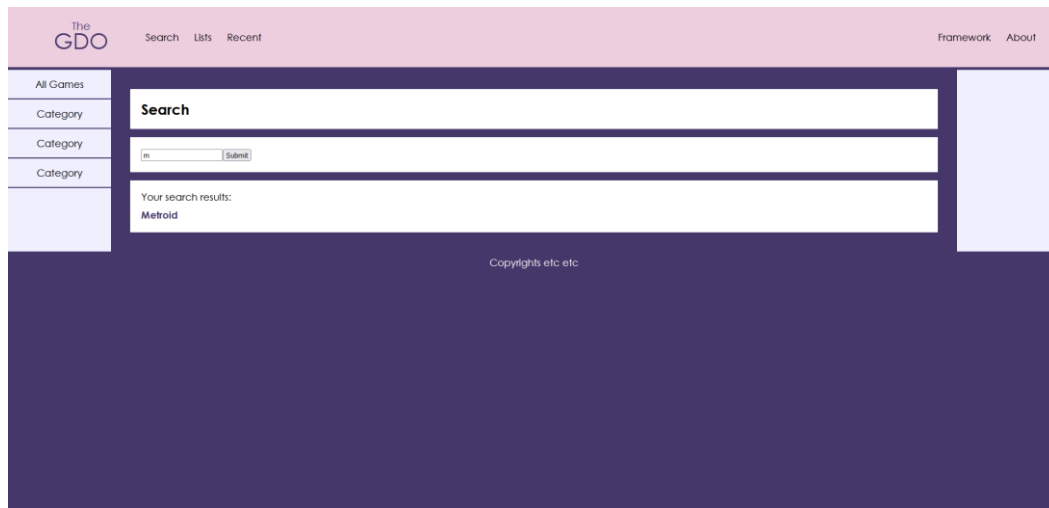
```
<RandGameContent flexDirection = "row-reverse" />
```

Καλείται από το component HomeRands, μαζί με το RandGameLeft.

RandGameContent

Το component για μεμονωμένα τυχαία παιχνίδια που εμφανίζονται στην αρχική σελίδα. Τα RandGameLeft και RandGameRight μπορούν να καλέσουν πολλά στιγμιότυπα αυτού το component για να εμφανίζονται σε μια σειρά.

search



Ο φάκελος που περιέχει όλα τα components που εμφανίζονται στην σελίδα αναζήτησης της εφαρμογής. Η λειτουργία αυτής της σελίδας αποτελεί μία απλή αναζήτηση ανάμεσα στις σελίδες παιχνιδιών που φιλοξενούνται στην εφαρμογή. Με την αναζήτηση μέρους ή ολόκληρου του ονόματος ενός παιχνιδιού, εμφανίζονται σαν αποτέλεσμα όλες οι σελίδες παιχνιδιών των οποίων ο τίτλος ταιριάζει με την αναζητούμενη λέξη.

Η σελίδα αναζήτησης αποτελείται από τον τίτλο της σελίδας ("Search"), το πεδίο συμπλήρωσης για την αναζητούμενη λέξη μαζί με ένα κουμπί για την εκκίνηση της αναζήτησης, και, αν υπάρχουν, μια λίστα με τα αποτελέσματα της αναζήτησης αυτής.

Τα components που βρίσκονται σ' αυτόν το φάκελο περιγράφονται παρακάτω.

SearchMain

Το συγκεντρωτικό component της σελίδας αναζήτησης. Δεν έχει δικά του περιεχόμενα, αλλά καλεί το component SearchOptions ώστε να εξασφαλιστεί η σωστή μορφοποίηση των περιεχομένων του.

SearchOptions

Το component με το πεδίο συμπλήρωσης της αναζητούμενης λέξης, το οποίο ταυτόχρονα επεξεργάζεται τα δεδομένα εισόδου για να διατυπωθεί το ερώτημα. Η διαδικασία διαχειρίζεται από μια HTML φόρμα με το πεδίο εισόδου, το οποίο αποθηκεύει την τιμή που παίρνει ως state μεταβλητή κατά την αποστολή της φόρμας. Η μεταβλητή αυτή ύστερα μεταφέρεται ως αναζητούμενος όρος για το component SearchResults, μέσα στο query που θα δούμε παρακάτω.

SearchResults

Το component που εφαρμόζει τον όρο και τον τύπο αναζήτησης που λαμβάνει απ' το component SearchOptions σε ένα template ερωτήματος SPARQL. Αυτό το component ενεργοποιείται μόλις αποσταλεί η φόρμα αναζήτησης, και στέλνει το ερώτημα στο σύστημα του Fuseki. Ύστερα, τα αντικείμενα της απάντησης που λαμβάνεται αποθηκεύονται σε μεταβλητές για τη μορφοποίησή τους κατά την εκτύπωση. Το κομμάτι κώδικα που διαχειρίζεται την ερώτηση στο Fuseki εμφανίζεται παρακάτω:

```
let searchInput = this.props.searchInput;
let searchOption = this.props.searchOption;

var query = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX schema:
<http://schema.org/> PREFIX MyOntology: <http://purl.org/net/MyOntology#>
SELECT DISTINCT ?searched ?label ?type WHERE{ ?searched rdf:type
MyOntology:"+searchOption+". ?searched rdfs:label ?label.
MyOntology:"+searchOption+" rdfs:label ?type. FILTER
regex(?label, '"+searchInput+"', 'i') }";
var url = "http://localhost:3030/Test/query";
var queryUrl = url + '?query=' + encodeURIComponent(query);
```

Στο απόσπασμα αυτό, πρώτον, αποθηκεύονται οι δύο τιμές state που πέρασε το component SearchOptions (searchInput και searchOption) σε μια ξεχωριστές μεταβλητές για ευκολότερη χρήση. Ύστερα, με τη μεταβλητή searchOption ορίζουμε την κλάση που αναζητούμε, και με τη δυνατότητα εύρεσης με κανονικές εκφράσεις (regular expressions ή απλώς regex) της SPARQL, ορίζουμε το μοτίβο εύρεσης από τη μεταβλητή searchInput, το οποίο θα εμφανίσει όλα τα ονόματα στιγμιοτύπων της κλάσης που ταιριάζουν.

Σχετικά με το ίδιο το SPARQL ερώτημα, η μεταβλητή ?searched είναι το URI του αποτελέσματος, ?label είναι η ονομασία του URI διαβάσιμη από άνθρωπο, και ?type είναι η κλάση της οποίας είναι στιγμιότυπο το αποτέλεσμα.

Μετά την αποθήκευση του ερωτήματος, ορίζουμε τη μεταβλητή `url` με τον σύνδεσμο της υπηρεσίας ερωτημάτων του Fuseki, σ' αυτήν την περίπτωση έχει το `path /Test/query` (όπου `Test` είναι η βάση και `query` είναι η υπηρεσία), και τέλος τα δύο συνδυάζονται στη μεταβλητή `queryUrl`.

Για τη λήψη αποτελεσμάτων απ' το Fuseki, χρησιμοποιούμε μια μέθοδο `fetch` με τον σύνδεσμο απ' τη μεταβλητή `queryUrl`. Όμως, καθώς οι ετικέτες ονόματος που έχουν κάποιες οντότητες παιχνιδιών δεν αντιστοιχούν με τον κωδικό του URI, γι' αυτό, πρέπει να εξάγουμε τον κωδικό αυτό με τη χρήση της συνάρτησης `split()`. Ο parser του αντικειμένου απάντησης θα φαίνεται όπως παρακάτω:

```
var parsedObject;
const response = fetch(queryUrl, options)
  .then(response => {
    return response.json();
  })
  .then(data => {
    parsedObject = data['results']['bindings'];
    for (var i = 0; i < parsedObject.length; i++) {
      var URI = parsedObject[i]['game']['value'];
      var sub = URI.split("MyOntology#"); // Το "MyOntology#" είναι το
      σίγουρο μέρος στο οποίο θα διαχωριστεί το string
      id.push(sub[1]);
      name.push(parsedObject[i]['label']['value']);
      var type = parsedObject[i]['type']['value'];
    }
  })
```

Τέλος, οι τιμές που συλλέχθηκαν απ' το αντικείμενο `parsedObject` ανατίθενται σε ξεχωριστές μεταβλητές του `state` με το `setState`, οι οποίες θα χρησιμοποιηθούν για τον ορισμό κάθε συνδέσμου στην αντίστοιχη σελίδα του, με την URI ονομασία του, μια ετικέτα διαβάσιμη από ανθρώπους, και τον τύπο του αντικειμένου:

```
this.setState({
  id: id,
  name: name,
  type: type
});
```

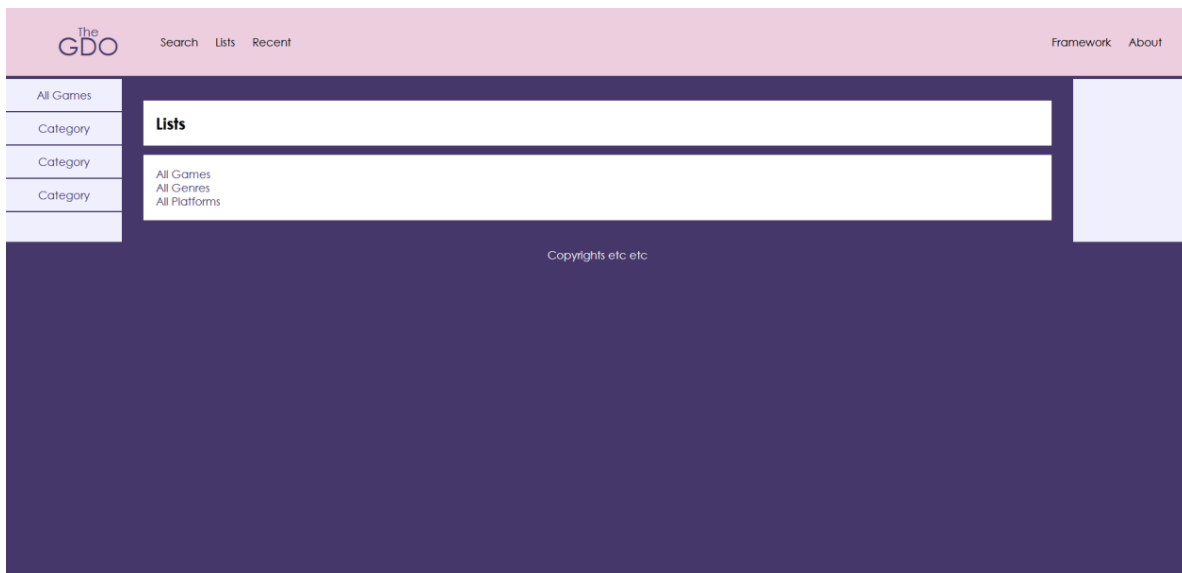
Η λίστα των αποτελεσμάτων αποτελείται από συνδέσμους, και ορίζονται μέσα σε επανάληψη `for` με το `container Link` και το `path` του συνδέσμου ως εξής:

```
for (var i = 0; i < this.state.id.length; i++) {
  listResults.push(
    <div>
      <Link to={`/${urlType}/${urlID[i]}`}><b>{urlName[i]}</b></Link>
    </div>
  )
}
```

Κάθε ένα από τα `urlType`, `urlID` και `urlName` είναι οι αποθηκευμένες τιμές κάθε αντικειμένου του τρέχοντος `state`, τα `type`, `id` και `name` αντίστοιχα. Για παράδειγμα, η δομή του `path` για το παιχνίδι “Rogue” θα οριστεί ως `/game/Rogue`, με το “game” να είναι η μεταβλητή `type` και το “Rogue” να είναι η μεταβλητή `urlID`.

Αν και είναι μέρος κυρίως της σελίδας αναζήτησης, αυτό το `component` χρησιμοποιείται και στη σελίδα συγκεντρωμένων λιστών, χάρη στη δυνατότητά του να παραμετροποιεί τη λειτουργία εκτύπωσης αποτελεσμάτων με μεταβλητές μέσα στο αποθηκευμένο `string` του ερωτήματος. Ο τρόπος που χρησιμοποιείται θα περιγραφεί στο ανάλογο τμήμα της περιγραφής της σελίδας συγκεντρωμένων λιστών.

lists



Ο φάκελος που περιέχει όλα τα `components` που εμφανίζονται στην σελίδα συγκεντρωτικών λιστών της εφαρμογής. Αντίθετα με την σελίδα αναζήτησης, η σελίδα λιστών προσφέρει έναν πιο εύκολο τρόπο γενικής εξερεύνησης των σελίδων της εφαρμογής ανάλογα με την κατηγορία που επιλέγεται.

Η σελίδα λιστών περιέχει το τίτλο της σελίδας “Lists” και μια σειρά από συνδέσμους. Όταν ο χρήστης επιλέξει έναν από αυτούς τους συνδέσμους, εμφανίζεται μια λίστα με στοιχεία ανάλογα με την επιλογή του. Ακόμη, κομμάτια του κώδικα μοιάζουν με αυτά της σελίδας αναζήτησης, καθώς οι λειτουργίες είναι παρόμοιες. Τα `components` αυτού του φακέλου περιγράφονται παρακάτω.

ListsMain

Το βασικό `component` για την εμφάνιση αυτής της σελίδας. Περιέχει τον τίτλο της σελίδας και, ανάλογα με την παρούσα διεύθυνση εμφανίζει είτε όλους τους συνδέσμους των λιστών, ή τα περιεχόμενα μιας επιλεγμένης λίστας.

Για την εμφάνιση των περιεχομένων κάθε λίστας, γίνεται χρήση του component `SearchResults`, καθώς η λειτουργία του `Lists` είναι πολύ παρόμοια με αυτή του φυσιολογικού `Search`, και απαιτείται η εμφάνιση αποτελεσμάτων μιας προεπιλεγμένης αναζήτησης, παρά μιας που εισάγει ο χρήστης. Για την εμφάνιση εναλλακτικά των συνδέσμων και μετά των περιεχομένων μιας λίστας, χρησιμοποιείται ένα container `Switch` όπως περιγράφεται παρακάτω:

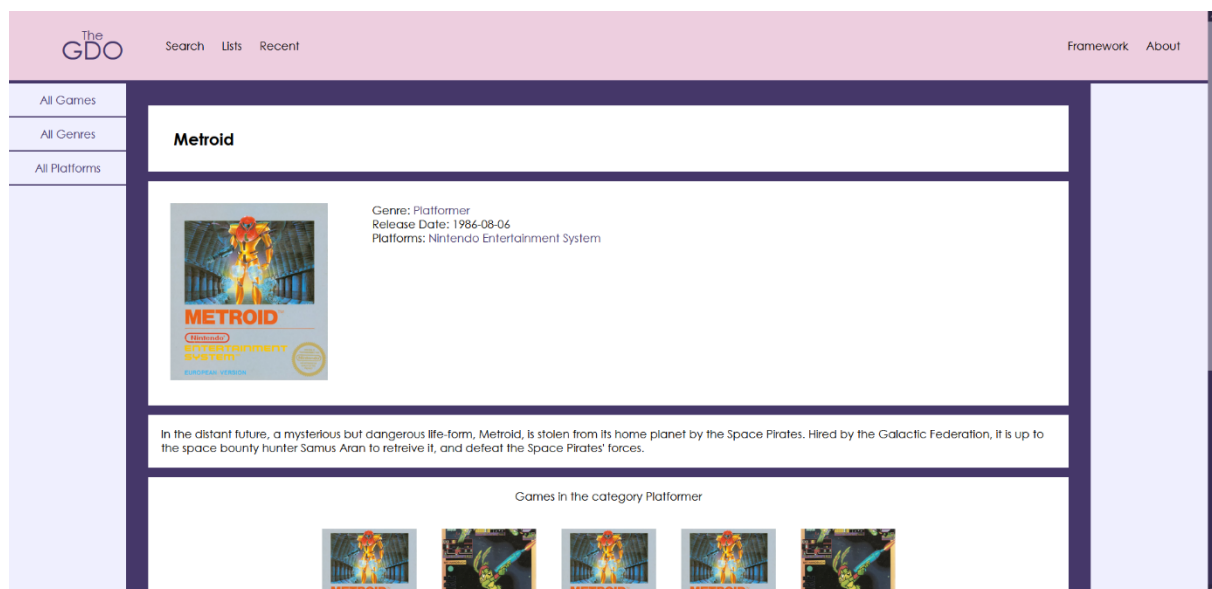
```
<Switch>
  <Route exact path="/lists">
    <ListsHome />
  </Route>
  <Route path="/lists/games">
    <SearchResults searchInput=".*" searchOption="Game" />
  </Route>
  <Route path="/lists/genres">
    <SearchResults searchInput=".*" searchOption="Genre" />
  </Route>
  <Route path="/lists/platforms">
    <SearchResults searchInput=".*" searchOption="Platform" />
  </Route>
</Switch>
```

Ουσιαστικά, όταν κάποιος ανατρέξει έναν σύνδεσμο για την εμφάνιση λίστας, η σειρά των συνδέσμων αντικαθίσταται με την επιλεγμένη λίστα.

ListsHome

Το component για την απλή διαχείριση της σειράς από συνδέσμους για κάθε μία λίστα που είναι διαθέσιμη, με τη χρήση συγκεκριμένων `paths`.

game



Ο φάκελος που περιέχει όλα τα components απ' τα οποία αποτελείται κάθε σελίδα περιγραφής των παιχνιδιών. Τα components που βρίσκονται σ' αυτόν το φάκελο περιγράφονται παρακάτω.

GameMain

Το βασικό component για κάθε σελίδα του παιχνιδιού. Στην πραγματικότητα δεν εμφανίζει δεδομένα από μόνο του, παραμόνο μεταφέρει το όνομα για το path της σελίδας προς το component GameData που καλεί. Αυτό επιτυγχάνεται με τη χρήση της συνάρτησης του React, useParams ως εξής:

```
const { id } = useParams();
return (
  <div className = "page-background">
    <GameData gameId = {id}/>
  </div>
);
```

GameData

Το component που εμφανίζει τις πληροφορίες ενός παιχνιδιού. Απ' όλα τα υπόλοιπα components που συνθελτουν την εφαρμογή, είναι το πιο άμεσα συνδεδεμένο με τον server του Fuseki, καθώς χρησιμοποιεί τη μέθοδο fetch για να επικοινωνήσει με τον server, στέλνοντας SPARQL queries.

Στο παρακάτω απόσπασμα κώδικα, χρησιμοποιείται η μεταβλητή query με τιμή το SPARQL ερώτημα για την εμφάνιση του παιχνιδιού (σε μία γραμμή όμως για να αναγνωρίζεται ως ενιαίο κείμενο απ' το React), τη μεταβλητή url για τον ορισμό του συνδέσμου της υπηρεσίας ερωτημάτων του Fuseki, και τέλος τα δύο συνδυάζονται στη μεταβλητή queryUrl:

```
var query='PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>PREFIX
rdfs: http://www.w3.org/2000/01/rdf-schema# prefix MyOntology:
http://purl.org/net/MyOntology# SELECT DISTINCT ?x ?y ?label ?type WHERE
{MyOntology:Metroid ?x ?y. OPTIONAL{?y rdfs:label ?label. ?y rdf:type
?type.}}';
var url="http://localhost:3030/Test/query";
var queryUrl = url + '?query=' + encodeURIComponent(query);
```

Με τη χρήση της μεταβλητής queryUrl, θα καλέσουμε μια μέθοδο fetch για να ανακτήσει τα αποτελέσματα του παραπάνω ερωτήματος σε JSON μορφή, όπως παρακάτω:

```
var parsedObject;
const response = fetch(queryUrl,options)
  .then(response => {
    return response.json();
  })
  .then(data => {
    parsedObject = data['results']['bindings'];
    for (var i = 0; i < parsedObject.length; i++) {
```

```
// Εξόρυξη τιμών απ' τα αντικείμενα JSON με τη χρήση case statements
})
```

Τέλος, οι τιμές που συλλέχθηκαν απ' το αντικείμενο `parsedObject` ανατίθενται σε ξεχωριστές μεταβλητές του state του component με το `setState`:

```
this.setState({
  name: name,
  platforms: platforms
  // κλπ...
});
```

GameOther

Εμφανίζει με δυναμικό τρόπο μια σειρά από παιχνίδια που έχουν κάποιο κοινό στοιχείο με το εμφανιζόμενο παιχνίδι, όπως είδος και πλατφόρμα. Τα παιχνίδια που εμφανίζονται είναι μεμονωμένα στιγμιότυπα του `Suggestions`. Ο αριθμός των παιχνιδιών μπορεί να οριστεί απ' τον διαχειριστή, αλλά για τη δική μας εφαρμογή επιλέξαμε να εμφανίζονται 5.

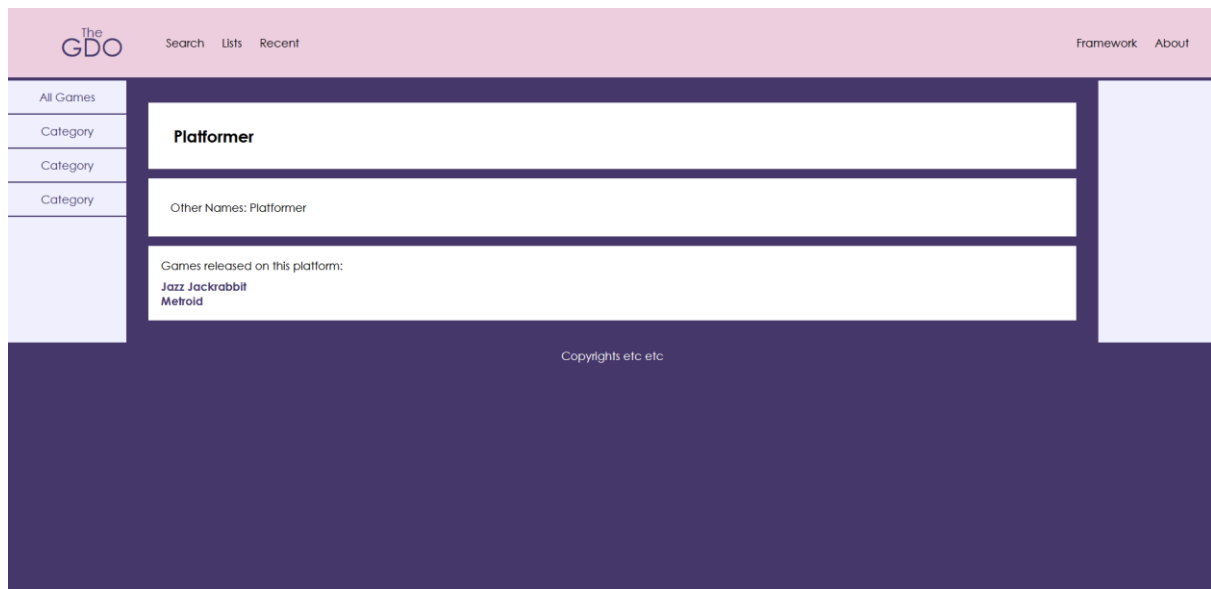
Suggestions

Το component για μεμονωμένα παιχνίδια που εμφανίζονται ως προτεινόμενα σε κάθε σελίδα παιχνιδιού. Το `GameOther` μπορεί να καλέσει πολλά στιγμιότυπα αυτού το component για να εμφανίζονται σε μια σειρά στο τέλος της σελίδας του παιχνιδιού.

Χρησιμοποιείται μια κλήση προς το Fuseki πολύ παρόμοια με αυτή του `RandGameContent`, μόνο που εδώ, χρησιμοποιούνται μεταβλητές για την παραμετροποίηση του SPARQL query με τη χρήση της ιδιότητας του παιχνιδιού (πχ. είδος ή πλατφόρμα) της τρέχουσας σελίδας, μαζί με την τιμή που έχει αυτή η ιδιότητα. Το query που προκύπτει έχει τη μορφή:

```
var query = 'PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns# PREFIX
rdfs: http://www.w3.org/2000/01/rdf-schema# PREFIX schema:
<http://schema.org/> PREFIX MyOntology: http://purl.org/net/MyOntology#
SELECT DISTINCT ?game ?label ?cover WHERE{?game MyOntology:'+type+'
MyOntology:'+categoryID+'. ?game rdf:type MyOntology:Game;
rdfs:label ?label; schema:image ?cover. BIND(RAND() AS ?randomGame)}ORDER
BY ?randomGame LIMIT 1'
var url="http://localhost:3030/Test/query";
var queryUrl = url + '?query=' + encodeURIComponent(query);
```

genre



Ο φάκελος που περιέχει όλα τα components απ' τα οποία αποτελείται κάθε σελίδα περιγραφής πλατφορμών κυκλοφορίας παιχνιδιών. Περιέχει το όνομα του είδους ως τίτλο και άλλες ονομασίες που μπορεί να έχει. Η λειτουργία του είναι παρόμοια με τις σελίδες παιχνιδιών, αλλά απαιτεί την εμφάνιση λιγότερων διαφορετικών πληροφοριών. Τα components που βρίσκονται σ' αυτόν το φάκελο περιγράφονται παρακάτω.

GenreMain

Καλεί τα GenreData και GenreGames με μια μεταβλητή, τον κωδικό ταυτοποίησης ενός είδους.

GenreData

Εμφανίζει τα δεδομένα ενός είδους. Χρησιμοποιεί τον κώδικα που έλαβε απ' το GenreMain και το ενσωματώνει στο ερώτημα SPARQL ως μεταβλητή. Η διαδικασία είναι πολύ παρόμοια με αυτή του component GameData. Για τον ορισμό του ερωτήματος έχουμε:

```
var query = 'PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>PREFIX
rdfs: <http://www.w3.org/2000/01/rdf-schema#>prefix MyOntology:
<http://purl.org/net/MyOntology#> PREFIX schema: <http://schema.org/>
SELECT DISTINCT ?x ?y ?label ?type WHERE {MyOntology:'+URI+' ?x ?y.
OPTIONAL{?y rdfs:label ?label. ?y rdf:type ?type.}}'
var url="http://localhost:3030/Test/query";
var queryUrl = url + '?query=' + encodeURIComponent(query);
```

Για την προσπέλαση και αποθήκευση των τιμών απ' το ερώτημα:

```
var parsedObject;
const response = fetch(queryUrl,options)
.then(response => {
  return response.json();
```

```

    })
    .then(data => {
      parsedObject = data['results']['bindings'];
      for (var i = 0; i < parsedObject.length; i++) {
        // Εξόρυξη τιμών απ' τα αντικείμενα JSON με τη χρήση case statements
      }
    })

```

Και τέλος, οι τιμές αποθηκεύονται στο state:

```

this.setState({
  mainName: mainName,
  names: names
});

```

GenreGames

Το component που θέτει τη διαφοροποίηση απ' τις σελίδες παιχνιδιών, το οποίο αποτυπώνει μια λίστα με παιχνίδια που ανήκουν σ' αυτό το είδος. Γίνεται κλήση του ερωτήματος:

```

var query = ' PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#>PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>prefix MyOntology:
<http://purl.org/net/MyOntology#>SELECT DISTINCT ?x ?y ?label ?type WHERE {
?x MyOntology:hasGameGenre MyOntology: '+URI+'. OPTIONAL{?x rdfs:label
?label. } }'
var url="http://localhost:3030/Test/query";
var queryUrl = url + '?query=' + encodeURIComponent(query);

```

Τα αποτελέσματα αποθηκεύονται σε μία σειρά όπως παρακάτω:

```

var parsedObject;
const response = fetch(queryUrl, options)
  .then(response => {
    return response.json();
  })
  .then(data => {
    parsedObject = data['results']['bindings'];
    for (var i = 0; i < parsedObject.length; i++) {
      var gameURI = parsedObject[i]['x']['value'];
      var sub = gameURI.split("MyOntology#");
      id.push(sub[1]);
      name.push(parsedObject[i]['label']['value']);
    }
  })

```

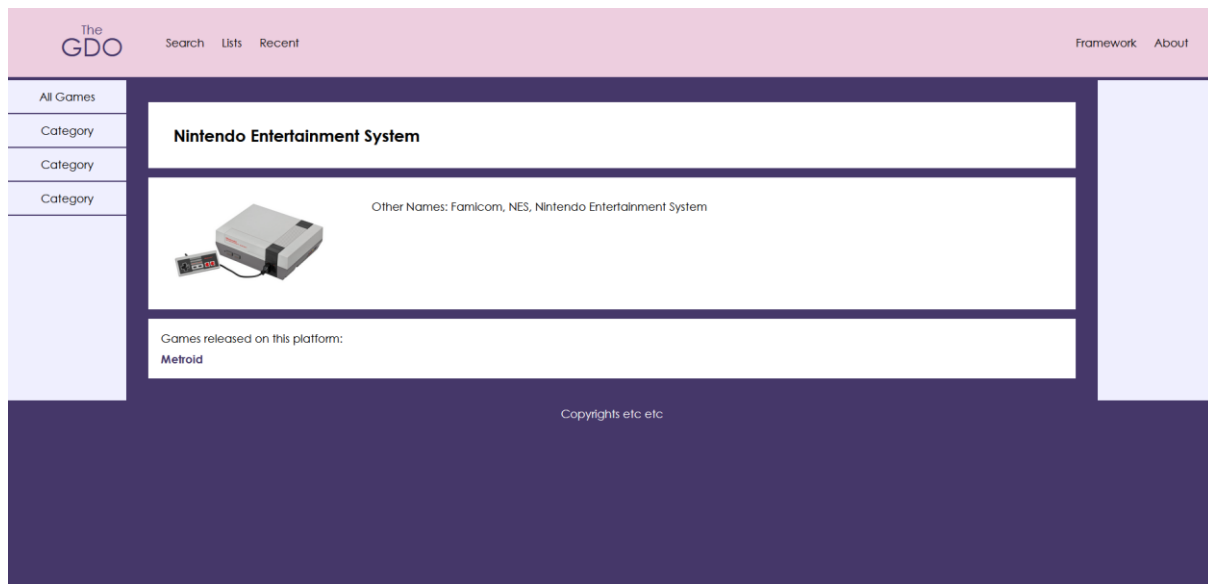
Και τέλος, οι τιμές αποθηκεύονται στο state:

```

this.setState ({
  id: id,
  name: name
});

```

platform



Ο φάκελος που περιέχει όλα τα components απ' τα οποία αποτελείται κάθε σελίδα περιγραφής πλατφορμών κυκλοφορίας παιχνιδιών. Περιέχει το όνομα της πλατφόρμας ως τίτλο και άλλες ονομασίες που μπορεί να έχει (πχ. Η γνωστή κονσόλα Nintendo Entertainment System κυκλοφόρησε αρχικά στην Ιαπωνία ως Famicom). Η λειτουργία του είναι παρόμοια με τις σελίδες παιχνιδιών, αλλά απαιτεί την εμφάνιση λιγότερων διαφορετικών πληροφοριών. Τα components που βρίσκονται σ' αυτόν το φάκελο περιγράφονται παρακάτω.

PlatformMain

Καλεί τα PlatformData και PlatformGames με μια μεταβλητή, τον κωδικό ταυτοποίησης μιας πλατφόρμας.

PlatformData

Εμφανίζει τα δεδομένα κάθε πλατφόρμας. Χρησιμοποιεί τον κώδικα που έλαβε απ' το PlatformMain και το ενσωματώνει στο ερώτημα SPARQL ως μεταβλητή. Η διαδικασία είναι πολύ παρόμοια με αυτή του component GameData. Για τον ορισμό του ερωτήματος έχουμε:

```
var query = 'PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>PREFIX
rdfs: <http://www.w3.org/2000/01/rdf-schema#>prefix MyOntology:
<http://purl.org/net/MyOntology#> PREFIX schema: <http://schema.org/>
SELECT DISTINCT ?x ?y ?label ?type WHERE {MyOntology:'+URI+' ?x ?y.
OPTIONAL{?y rdfs:label ?label. ?y rdf:type ?type.}}'
var url="http://localhost:3030/Test/query";
var queryUrl = url + '?query=' + encodeURIComponent(query);
```

Για την προσπέλαση και αποθήκευση των τιμών απ' το ερώτημα:

```
var parsedObject;
const response = fetch(queryUrl,options)
```

```

.then(response => {
  return response.json();
})
.then(data => {
  parsedObject = data['results']['bindings'];
  for (var i = 0; i < parsedObject.length; i++) {
    // Εξόρυξη τιμών απ' τα αντικείμενα JSON με τη χρήση case statements
  }
})

```

Και τέλος, οι τιμές αποθηκεύονται στο state:

```

this.setState({
  mainName: mainName,
  names: names,
  cover: cover
});

```

PlatformGames

Το component που θέτει τη διαφοροποίηση απ' τις σελίδες παιχνιδιών, το οποίο αποτυπώνει μια λίστα με παιχνίδια που κυκλοφόρησαν σ' αυτήν την πλατφόρμα. Γίνεται κλήση του ερωτήματος:

```

var query = PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>PREFIX
rdfs: <http://www.w3.org/2000/01/rdf-schema#>prefix MyOntology:
<http://purl.org/net/MyOntology#>SELECT DISTINCT ?x ?y ?label ?type WHERE {
  ?x MyOntology:isReleasedOn MyOntology:'+URI+'. OPTIONAL{?x rdfs:label
  ?label. }}'
var url="http://localhost:3030/Test/query";
var queryUrl = url + '?query=' + encodeURIComponent(query);

```

Τα αποτελέσματα αποθηκεύονται σε μία σειρά όπως παρακάτω:

```

var parsedObject;
const response = fetch(queryUrl,options)
.then(response => {
  return response.json();
})
.then(data => {
  parsedObject = data['results']['bindings'];
  for (var i = 0; i < parsedObject.length; i++) {
    var gameURI = parsedObject[i]['x']['value'];
    var sub = gameURI.split("MyOntology#");
    id.push(sub[1]);
    name.push(parsedObject[i]['label']['value']);
  }
}

```

Και τέλος, οι τιμές αποθηκεύονται στο state:

```

this.setState ({
  id: id,
  name: name
});

```

static

Ο φάκελος που περιέχει components στατικών σελίδων οι οποίες περιέχουν μόνο απλό κείμενο και είναι καθαρά πληροφοριακές. Κάθε αρχείο component σ' αυτό το φάκελο αντιπροσωπεύει μια μεμονωμένη σελίδα, η οποία δεν συνδέεται με άλλες.

Framework

Το component που περιέχει μια αναλυτική περιγραφή για το πώς δομείται αυτή η εφαρμογή, και μερικές επεξηγήσεις για τα δομικά του στοιχεία, όπως τι είναι η οντολογία με λίγα λόγια, και με ποιά εργαλεία και frameworks δημιουργήθηκε.

2. Αρχιτεκτονική

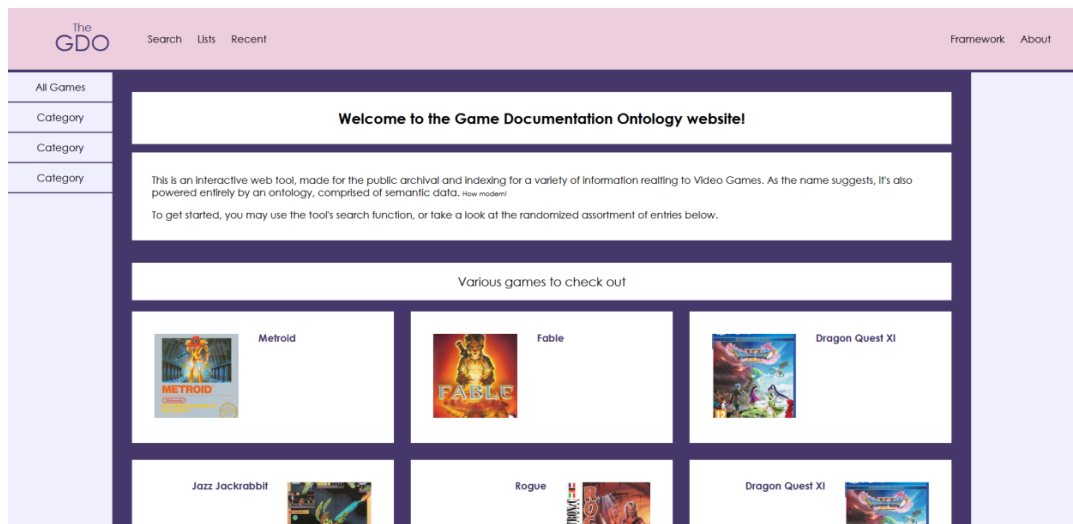
Η εφαρμογή δημιουργήθηκε με την ιδέα μιας δομής η οποία να συνδυάζει δύο ανεξάρτητες πλατφόρμες με τέτοιο τρόπο ώστε, όποτε η μία πλατφόρμα αναζητά κάποια πληροφορία, η άλλη να είναι σε θέση να της την προσφέρει. Αυτή η δομή είναι γνωστή και ως client-server system, ή σύστημα πελάτη-εξυπηρετητή. Ακόμη, το σύστημα αυτό παίρνει τη δευτερεύουσα δομή back-end front-end, καθώς το σύστημα του server δεν έχει σχεδιαστεί για προβολή απ' τον κοινό χρήστη, κι έτσι δεν είναι σε θέση να προσφέρει κάποια διεπαφή χρήστη ή αυτή είναι σε πολύ βασικό επίπεδο. Έτσι ο server αναφέρεται και ως back-end του συστήματος. Βασικός σκοπός του back-end server είναι η διαχείριση δεδομένων και η υποστήριξη της εφαρμογής, είτε πριν την εμφάνισή της στον χρήστη είτε ενώ λειτουργεί παράλληλα, ώστε να παρέχει δυναμικά ανακτώμενα δεδομένα και να επιτρέπει στην τελική εφαρμογή να είναι πιο ευέλικτη και πιο εμπλουτισμένη. Διάφορα τέτοια δεδομένα μπορεί να είναι δεδομένα μιας βάσης, CSS stylesheets με παλέτες χρωμάτων, γραμματοσειρές, μορφοποιήσεις τμημάτων κλπ, αρχεία εικόνων, και σενάρια JavaScript. Τα δεδομένα αυτά, όμως, πρέπει πρώτα να ζητηθούν με κάποιο τρόπο, αφού το back-end server δεν μπορεί να ξέρει από μόνο του πότε ή πού πρέπει να παρέχει τα δεδομένα του. Σ' αυτό το σημείο το σύστημα του client χρησιμοποιεί κλήσεις μέσω του δικτύου προς το back-end server με αιτήματα που περιλαμβάνουν οποιαδήποτε πληροφορία χρειάζεται, και αυτή αποθηκεύεται τοπικά στην έξοδο του χρήστη, και μπορεί από εκεί να την εκμεταλλευτεί με όποιον τρόπο χρειάζεται, όπως η απλή διαμόρφωση και εκτύπωση στην έξοδο, ή η χρήση του σε παραμέτρους, μεταβλητές και άλλα. Εδώ, καθώς ο client λειτουργεί στο περιβάλλον εξόδου με το οποίο αλληλεπιδρά ο χρήστης και είναι ουσιαστικά το “μπροστά” τμήμα της εφαρμογής, το μέρος του client ονομάζεται και front-end.

Στην περίπτωση μας, το back-end είναι το σύστημα του Apache Fuseki, στο οποίο έχουμε αποθηκευμένη την οντολογία μαζί με συνδεδεμένα στιγμιότυπα των κλάσεων της. Από μόνο του, το Fuseki δεν έχει παραπάνω δυνατότητες από την εκτύπωση αποτελεσμάτων σε SPARQL ερωτήματα, αλλά με ένα front-end, αυτά τα δεδομένα μπορούν να παρουσιαστούν με έναν πιο ομοιόμορφο τρόπο, ακόμη και να χρησιμοποιηθούν διαδραστικά. Το front-end σ' αυτήν την περίπτωση είναι η εφαρμογή που δημιουργήσαμε με το framework του React.

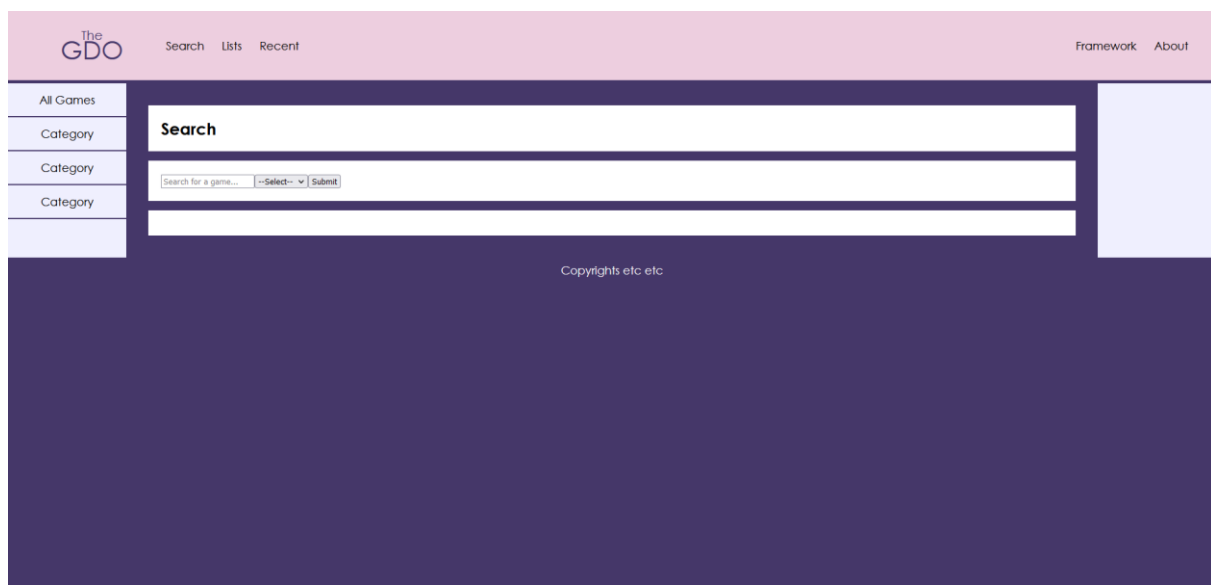
3. Σενάρια Χρήσης

3.1. Αναζήτηση ενός παιχνιδιού

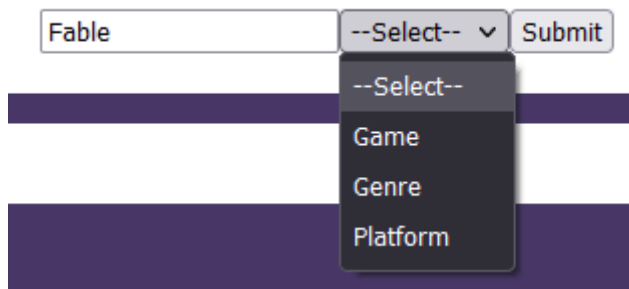
Μόλις εισέλθει ένας χρήστης στην εφαρμογή, θα εμφανιστεί πρώτα η αρχική της σελίδα.



Ο χρήστης αυτός αναζητά ένα παιχνίδι με όνομα “Fable”. Θα επιλέξει τον σύνδεσμο “Search” από τη γραμμή πλοήγησης, που θα τον ανακατευθύνει στη λειτουργία αναζήτησης.

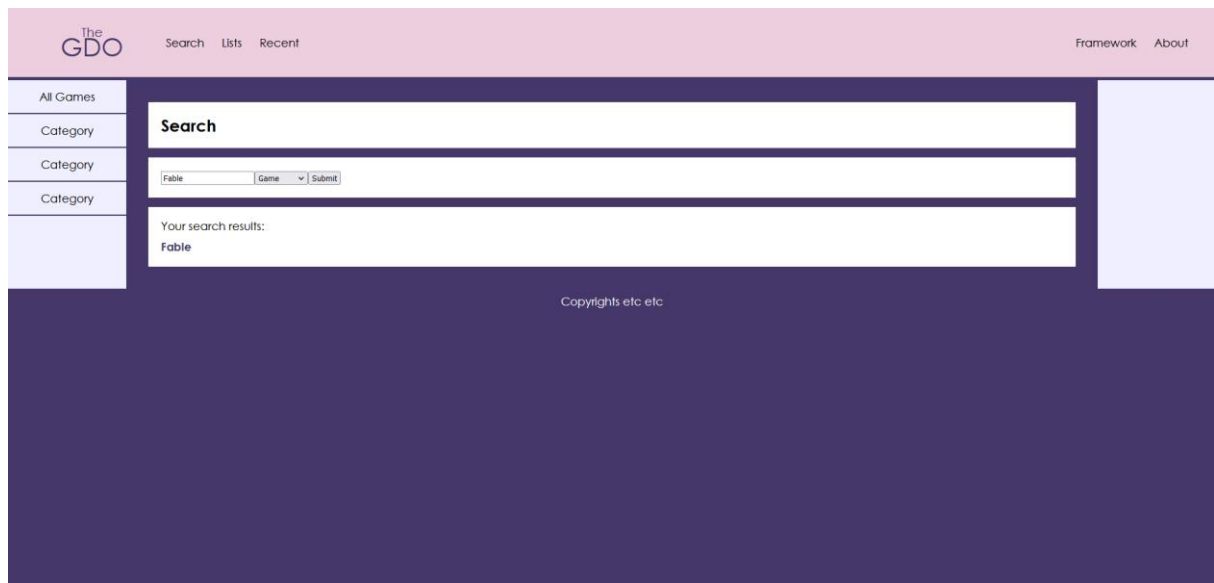


Από εκεί, θα συμπληρώσει το όνομα ή μέρος του ονόματος του παιχνιδιού στο πλαίσιο και θα επιλέξει τον τύπο εγγραφής που ψάχνει, σ' αυτήν την περίπτωση "Game" καθώς αναζητείται ένα παιχνίδι.



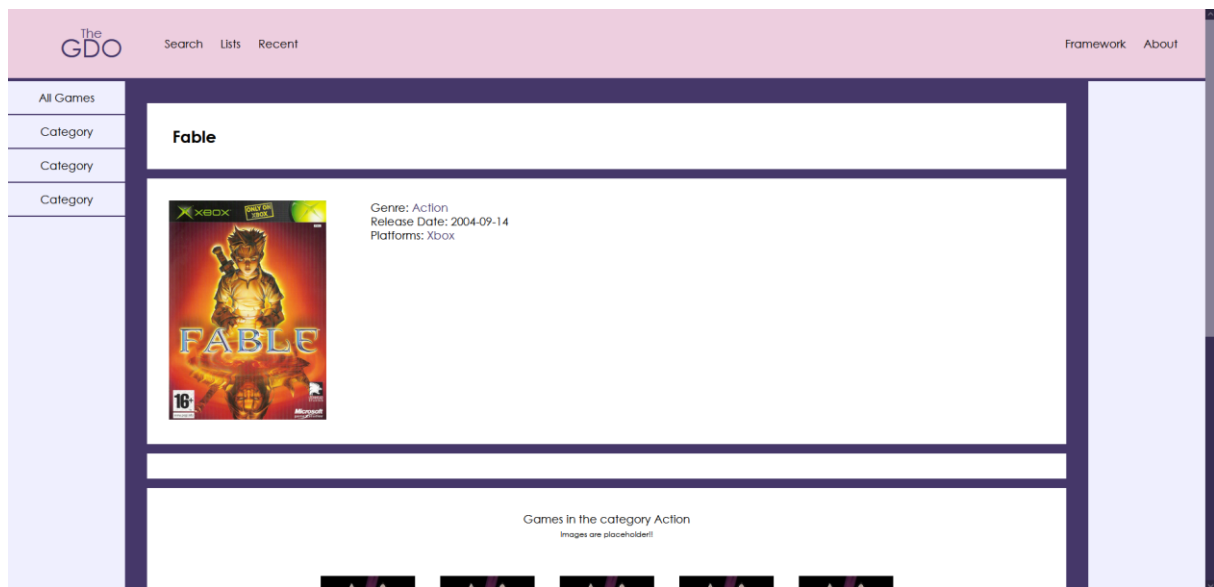
The image shows a search interface. It consists of a text input field containing the word "Fable". To the right of the input is a dropdown menu currently showing "--Select--". The dropdown menu is open, displaying three options: "Game", "Genre", and "Platform". To the right of the dropdown is a "Submit" button.

Πατώντας το κουμπί "Submit", θα εμφανιστούν τα αποτελέσματα που ταιριάζουν με την αναζήτηση σε μορφή συνδέσμων για κάθε ένα αποτέλεσμα.



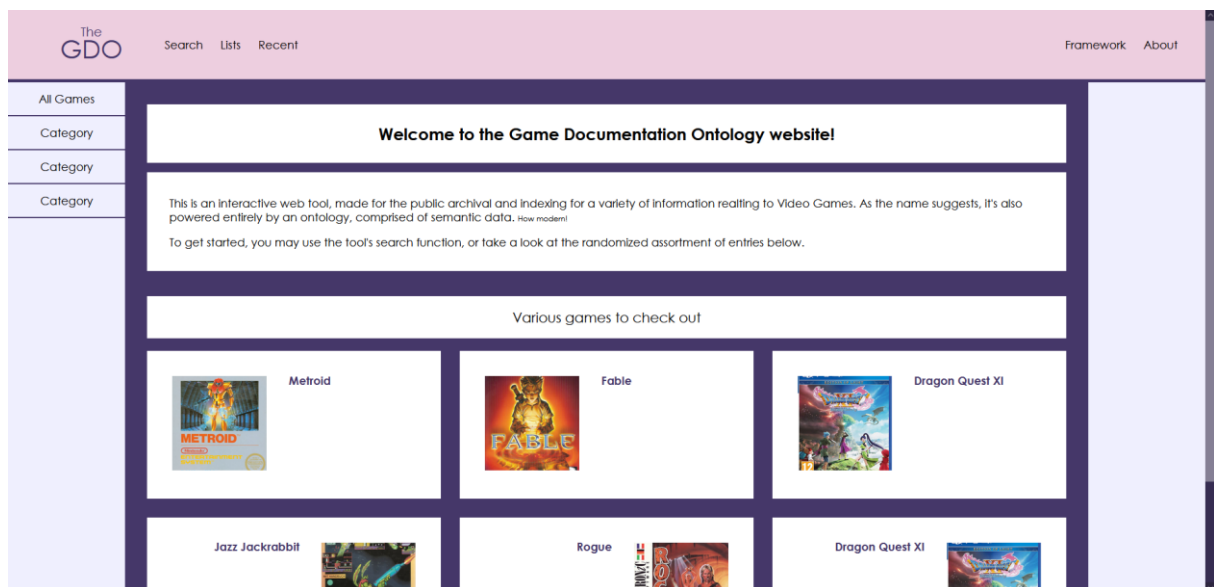
The image is a screenshot of a web application. At the top, there is a navigation bar with the logo "The GDO" and links for "Search", "Lists", "Recent", "Framework", and "About". Below the navigation bar, there is a sidebar on the left with links for "All Games", "Category", "Category", and "Category". The main content area has a "Search" section with a text input field containing "Fable", a dropdown menu showing "Game", and a "Submit" button. Below the search section, there is a section titled "Your search results:" which displays a single result: "Fable". At the bottom of the page, there is a footer with the text "Copyrights etc etc".

Το παιχνίδι που αναζητά ο χρήστης θα είναι το πρώτο αντικείμενο σ' αυτή τη λίστα. Πατώντας τον σύνδεσμο, θα κατευθυνθεί στη σελίδα του παιχνιδιού που αναζητά.

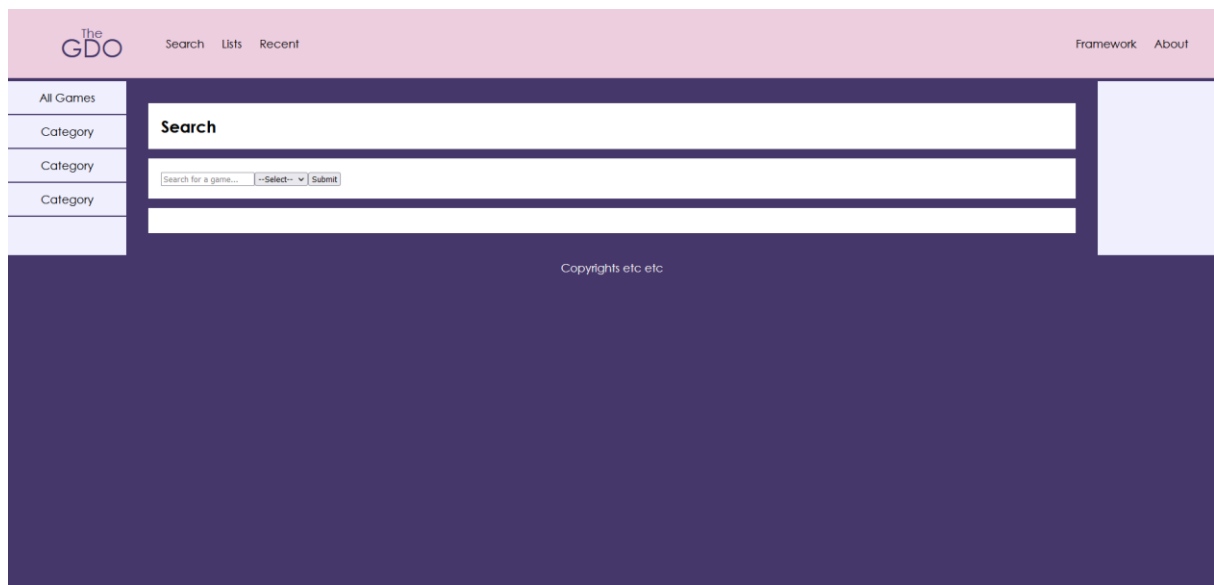


3.2. Αναζήτηση όλων των παιχνιδιών μιας πλατφόρμας

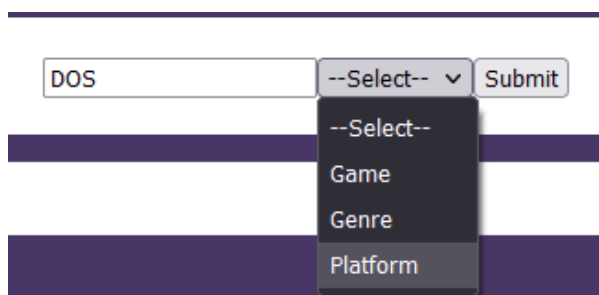
Μόλις εισέλθει ένας χρήστης στην εφαρμογή, θα εμφανιστεί πρώτα η αρχική της σελίδα.



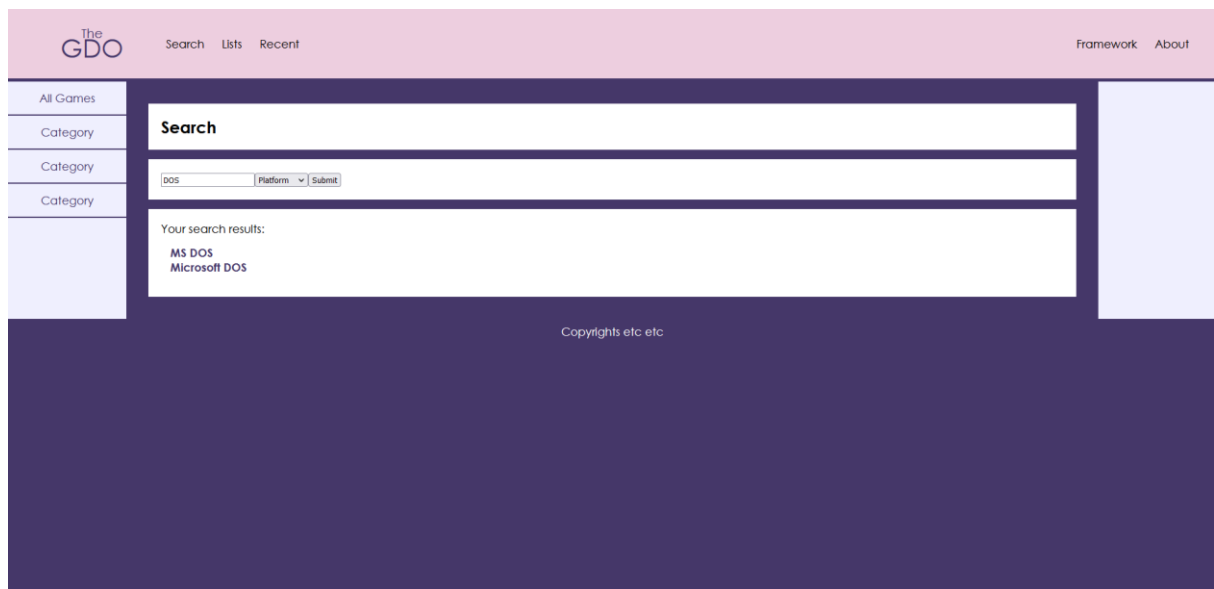
Ο χρήστης αυτός αναζητά ποια παιχνίδια κυκλοφόρησαν στην πλατφόρμα Microsoft DOS. Θα επιλέξει τον σύνδεσμο “Search” από τη γραμμή πλοήγησης, που θα τον ανακατευθύνει στη λειτουργία αναζήτησης.



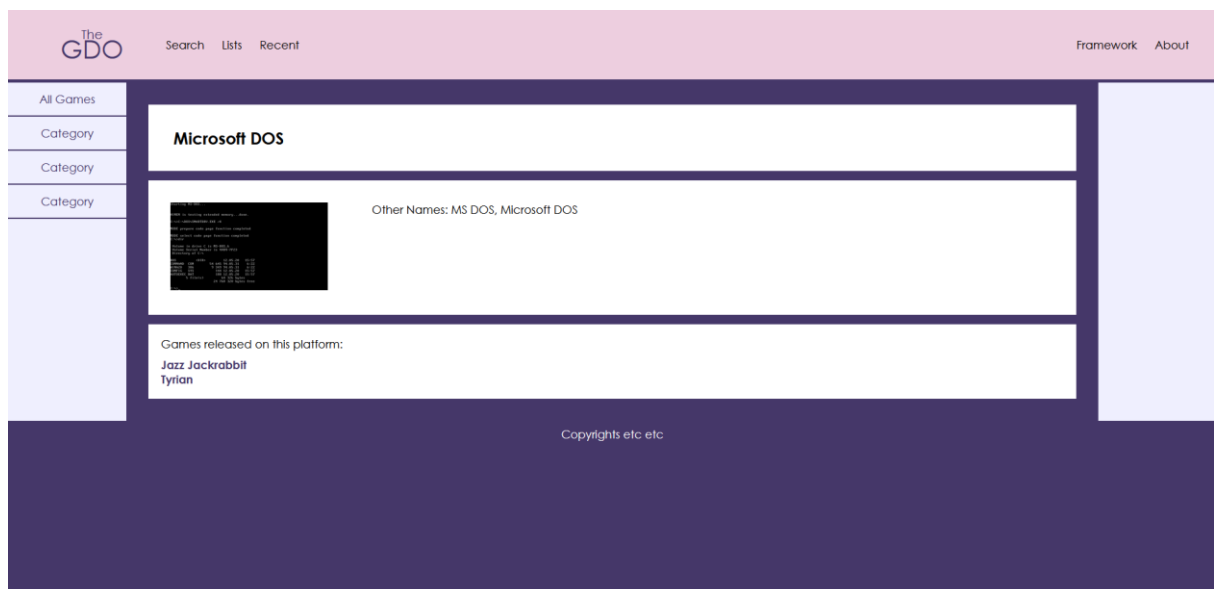
Από εκεί, θα συμπληρώσει το όνομα ή μέρος του ονόματος αυτής της πλατφόρμας στο πλαίσιο και θα επιλέξει τον τύπο εγγραφής που ψάχνει, σ' αυτήν την περίπτωση "Platform" καθώς αναζητείται ένα παιχνίδι.



Πατώντας το κουμπί "Submit", θα εμφανιστούν τα αποτελέσματα που ταιριάζουν με την αναζήτηση σε μορφή συνδέσμων για κάθε ένα αποτέλεσμα.

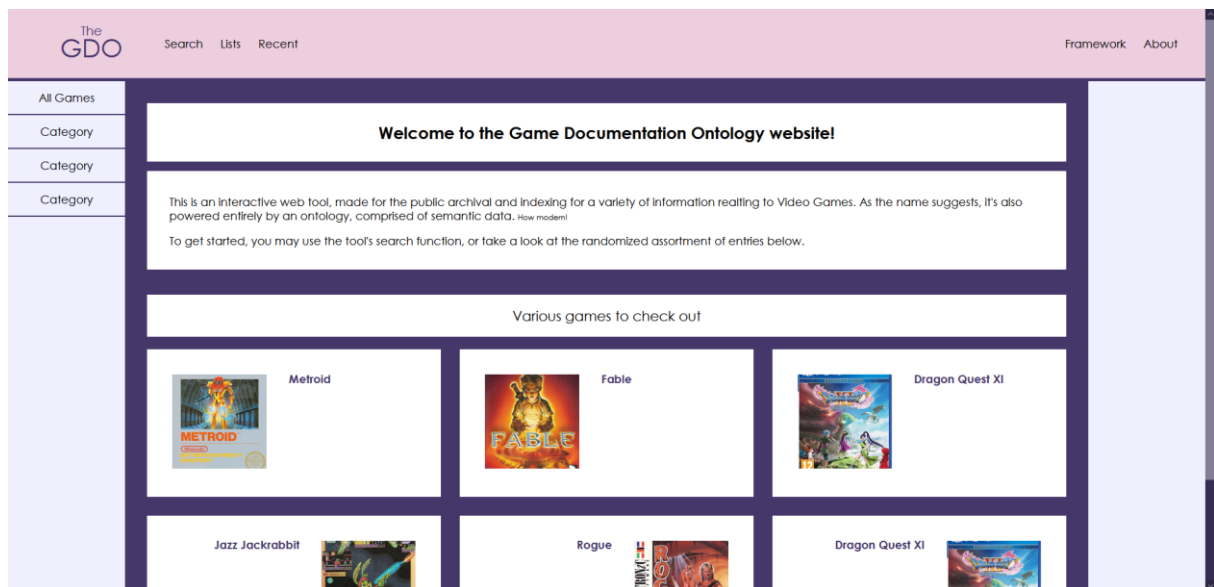


Το παιχνίδι που αναζητά ο χρήστης θα είναι το πρώτο αντικείμενο σ' αυτή τη λίστα. Πατώντας τον σύνδεσμο, θα κατευθυνθεί στη σελίδα της πλατφόρμας που αναζητά, μαζί με τα παιχνίδια που κυκλοφόρησαν σ' αυτή.

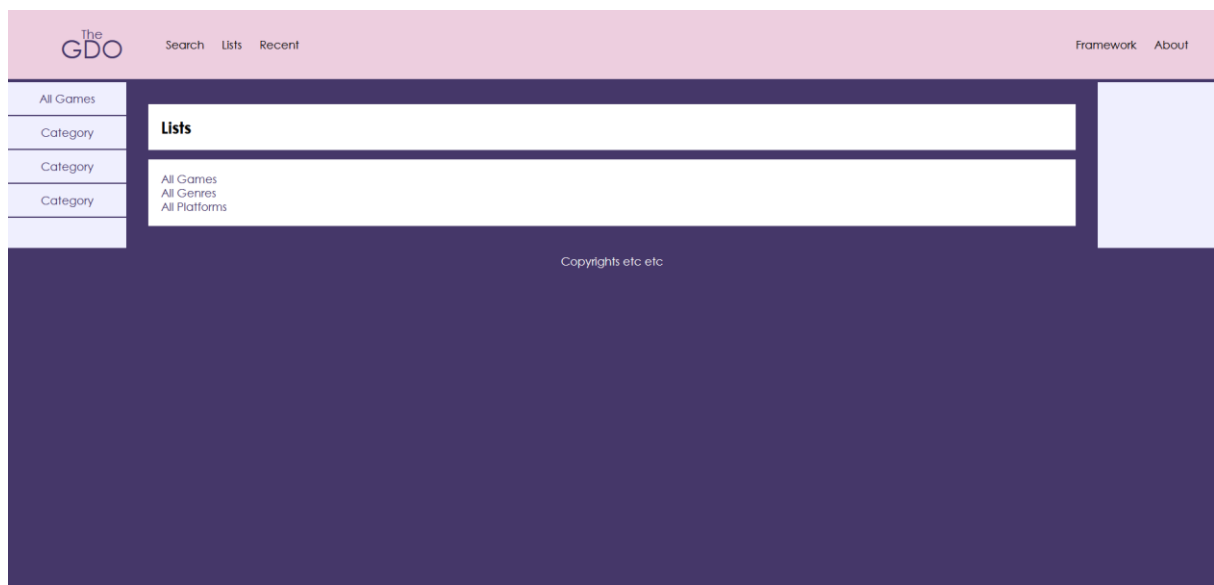


3.3. Εξερεύνηση όλων των παιχνιδιών που φιλοξενούνται

Μόλις εισέλθει ένας χρήστης στην εφαρμογή, θα εμφανιστεί πρώτα η αρχική της σελίδα.



Ο χρήστης αυτός θα ήθελε να κοιτάζει όλα τα παιχνίδια τα οποία μπορεί να βρει στην εφαρμογή, χωρίς να ψάχνει κάποιο συγκεκριμένο. Θα επιλέξει τον σύνδεσμο “Lists” από τη γραμμή πλοήγησης, που θα τον ανακατευθύνει στη σελίδα με όλες τις διαθέσιμες λίστες.



Από εκεί, θα επιλέξει τη λίστα που θέλει να δει, σ’ αυτήν την περίπτωση τη λίστα “All Games”, η οποία θα εμφανίσει όλους τους συνδέσμους για κάθε σελίδα παιχνιδιού που έχει η εφαρμογή, με αλφαβητική σειρά.

The GDO

SearchListsRecent

FrameworkAbout

All Games

Category

Category

Category

Lists

All Games

Dragon Quest 11

Dragon Quest XI

Dragon Quest XI: Echoes of an Elusive Age

Fable

Jazz Jackrabbit

Metroid

Rogue

Tyrian

Copyrights etc etc

6

Συμπεράσματα - Μελλοντικές Επεκτάσεις

Η ονομασία που έχει λάβει το διαδίκτυο και ο Παγκόσμιος Ιστός της σύγχρονης εποχής ως Διαδίκτυο των Πραγμάτων αποτελεί παράδειγμα του αντίκτυπου του αυξανόμενου όγκου των πληροφοριών που φιλοξενούνται σ' αυτό. Για την αποτελεσματικότερη διαχείριση αυτού, χρειάστηκαν πολλές προσπάθειες και καινοτομίες, με κάποιες να χρησιμοποιούνται αυτούσιες ακόμα και σήμερα, ή άλλες να έχουν συγχωνευτεί σε πιο συγκεντρωμένα πρότυπα. Από όλα αυτά τα πρότυπα, αυτό που παρέμεινε πιο διακεκριμένο απ' όλα ήταν ο Σημασιολογικός Ιστός του Tim Berners-Lee. Μέσα στα πάνω από 20 χρόνια ύπαρξής του, διαπιστώθηκε πως η χρήση σημασιολογικά ενωμένων οντοτήτων θα αποτελέσει το μέλλον των ψηφιακών δικτύων και τον τρόπο που διαχειρίζονται τα δεδομένα τους. Όμως, ακόμα και ως ένα από τα δημοφιλέστερα πεδία έρευνας του διαδικτύου, μόνο πρόσφατα έχει παρουσιάσει ευρύτερη χρήση, πέρα από μεμονωμένες πιλοτικές εφαρμογές.

Σκοπός λοιπόν της διαδικτυακής εφαρμογής που σχεδιάστηκε στα πλαίσια αυτής της εργασίας δεν ήταν μόνο μια απλή παρουσίαση επικοινωνίας μεταξύ οντολογίας και περιβάλλοντος χρήστη, αλλά και μια αρχική ματιά στο πώς θα μπορούσαν να χρησιμοποιηθούν τα σημασιολογικά δεδομένα από έναν κοινό χρήστη του διαδικτύου, με αντικείμενο την ενδεικτική χρήση ενός τόσο κοινού θέματος όπως τα βιντεοπαιχνίδια.

Σημαντικό στοιχείο στην ανάπτυξη αυτής της εφαρμογής, είναι ότι σε σχέση με άλλες παρόμοιες προσπάθειες δημιουργίας μιας διαδικτυακής πλατφόρμας για σημασιολογικά δεδομένα, έχει συγκριτικά μεγαλύτερη ευκολία στην κατανόηση ως προς τη δομή και τη λειτουργία της. Ο ίδιος ο κώδικας της εφαρμογής που δημιουργήσαμε είναι αρκετά απλός ώστε οποιοσδήποτε έχει διαθέσιμη αυτή τη βάση κώδικα, να μπορεί με λίγες αλλαγές να τον προσαρμόσει ανάλογα με τα δεδομένα μιας άλλης οντολογίας.

Αν και η εφαρμογή αυτή είναι στην τρέχουσα κατάστασή της πλήρης, υπάρχουν πολλοί τρόποι οι οποίοι θα μπορούσαν να την εμπλουτίσουν παραπάνω. Καθώς ο αριθμός των στιγμιotypών που βρίσκονται στην οντολογία είναι ιδιαίτερα μικρός, θα μπορούσε να προστεθεί ένα μεγαλύτερο dataset. Καθώς όμως το πεδίο των ηλεκτρονικών παιχνιδιών προβλέπει τη χρήση ενός dataset με χιλιάδες εγγραφές, η χειροκίνητη εισαγωγή τους φυσικά δεν είναι ρεαλιστικά εφικτή. Ένας τρόπος επίλυσης αυτού του ζητήματος είναι η μετατροπή υπάρχοντων σημασιολογικών δεδομένων από άλλες πηγές στη μορφή της δικής μας οντολογίας. Τέτοιες πηγές είναι η βάση γνώσης του Wikidata, το οποίο είχαμε περιγράψει σε προηγούμενο κεφάλαιο, το οποίο ακόμη διαθέτει ένα δικό του περιβάλλον ερωτημάτων SPARQL, κάτι που θα βοηθούσε σημαντικά στην γρηγορότερη συλλογή δεδομένων.

Ακόμα, σε ένα πιο τεχνικό επίπεδο, η λειτουργία αναζήτησης θα μπορούσε να επεκταθεί περαιτέρω με τη χρήση σύνθετων επιλογών αναζήτησης, όπως ο συνδυασμός δύο ή παραπάνω κατηγοριών αναζήτησης, η αναζήτηση παιχνιδιών με βάση ένα εύρος έτους κυκλοφορίας, και πολλά άλλα. Κάτι τέτοιο μπορεί να επιτευχθεί πιθανώς με τη χρήση περισσότερων μεταβλητών κατά τον ορισμό του ερωτήματος μέσα στον κώδικα, που αναφέρονται στην ιδιότητα ή στις ιδιότητες που αναζητούνται.

Βιβλιογραφία

1. Berners-Lee, T., J. Hendler, and O. Lassila, *The semantic web*. Scientific american, 2001. **284**(5): p. 34-43.
2. Consortium, W.W.W., *RDF 1.1 Primer*. 2014.
3. Lassila, O., *Resource description framework (RDF) model and syntax specification, W3C recommendation*. <http://www.w3.org/TR/PR-rdf-syntax>, 1999.
4. Consortium, W.W.W., *RDF 1.1 concepts and abstract syntax*. 2014.
5. Heath, T. and C. Bizer, *Linked data: Evolving the web into a global data space*. Synthesis lectures on the semantic web: theory and technology, 2011. **1**(1): p. 1-136.
6. Brickley, D., R.V. Guha, and B. McBride, *RDF Schema 1.1*. W3C recommendation, 2014. **25**: p. 2004-2014.
7. Sporny, M., et al., *JSON-LD 1.0*. W3C recommendation, 2014. **16**: p. 41.
8. Beckett, D., *Rdf 1.1 n-triples*. URL: <https://www.w3.org/TR/n-triples>, 2014.
9. Berners-Lee, T. and D. Connolly, *Notation3 (n3): a readable RDF syntax*. 2011. URL [www.w3.org/TeamSubmission](http://www.w3.org/TeamSubmission/n3), (3).
10. Consortium, W.W.W., *OWL 2 web ontology language document overview*. 2012.
11. Consortium, W.W.W., *SPARQL 1.1 overview*. 2013.
12. Stefanidakis, M., T. Andronikos, and I. Papadakis, *Open Linked Data and Related Applications*. 2015.
13. Riley, J., *Understanding metadata*. Washington DC, United States: National Information Standards Organization (<http://www.niso.org/publications/press/UnderstandingMetadata.pdf>), 2017. **23**.
14. Bizer, C., T. Heath, and T. Berners-Lee, *Linked data: The story so far*, in *Semantic services, interoperability and web applications: emerging concepts*. 2011, IGI global. p. 205-227.
15. Berners-Lee, T., *RFC1630: Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web*. 1994, RFC Editor.
16. Berners-Lee, T., *Universal resource identifiers in WWW: a unifying syntax for the expression of names and addresses of objects on the network as used in the world-wide web*. 1994.
17. Jena, A., *Getting started with Apache Jena*. 2015.
18. Jena, A., *Apache jena fuseki*. The Apache Software Foundation, 2014. **18**.

19. Aggarwal, S., *Modern web-development using reactjs*. International Journal of Recent Research Aspects, 2018. 5(1): p. 133-137.
20. bin Uzayr, S., N. Cloud, and T. Ambler, *React*, in *JavaScript Frameworks for Modern Web Development*. 2019, Springer. p. 507-521.