Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων Πληροφορικης

Ανάπτυξη Διαδικτυακής Εφαρμογής Δημιουργίας και Επεξεργασίας Δομών, Σχεδιασμένες για Περιγραφή και Ενορχήστρωση Μικρο-υπηρεσιών

Διπλωματικη Εργασια

του

ΜΗΝΑ Ν. ΑΛΒΕΡΤΗ

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια

Αθήνα, Μάρτιος 2021

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων Πληροφορικης

Ανάπτυξη Διαδικτυακής Εφαρμογής Δημιουργίας και Επεξεργασίας Δομών, Σχεδιασμένες για Περιγραφή και Ενορχήστρωση Μικρο-υπηρεσιών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΜΗΝΑ Ν. ΑΛΒΕΡΤΗ

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Μαρτίου 2021.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Θεοδώρα Βαρβαρίγου	Εμμανουήλ Βαρβαρίγος	Συμεών Παπαβασιλείου
Καθηγήτρια	Καθηγητής	Καθηγητής

Αθήνα, Μάρτιος 2021

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων Πληροφορικης

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Μηνάς Αλβέρτης

9 Μαρτίου 2021

Περίληψη

Ένα Ψηφιακό Κιβώτιο Δεδομένων (VDC) είναι ένας αναπτυσσόμενος τρόπος οργάνωσης και επεξεργασίας πληροφοριών. Ένας διαχειριστής δεδομένων που έχει οργανώσει το προϊόν του σε ένα VDC επιθυμεί να το "διαφημίσει" σε πιθανούς αγοραστές/χρήστες. Ο τρόπος με τον οποίο γίνεται αυτό είναι η δημοσίευση στο δίκτυο των VDC ενός κατάλληλου αρχείο που ονομάζεται "Artifact" ή "Blueprint" το οποίο περιγράφει την υπηρεσία που προσφέρει. Η συγγραφή ενός Artifact ή Blueprint γίνεται με συγκεκριμένο τρόπο και συνήθως βασίζεται σε ένα JSON-Schema ώστε να όλα να είναι άμεσα συγκρίσιμα μεταξύ τους. Με τον τρόπο αυτό οι πιθανοί αγοραστές/χρήστες μπορούν να κάνουν μία έρευνα αγοράς ώστε να καταλήξουν στην υπηρεσία που ταιριάζει καλύτερα τις ανάγκες τους.

Στόχος αυτής της διπλωματικής εργασίας είναι η ανάπτυξη μίας διαδικτυακής εφαρμογής που θα επιτρέπει στον χρήστη να εισάγει το JSON-Schema και με εύκολο, γρήγορο και ορθό τρόπο να παράγει το Artifact/Blueprint που επιθυμεί.

Λέξεις Κλειδιά

Ψηφιακό Κιβώτιο Δεδομένων, VDC, Διαδικτυακή εφαρμογή, Δημιουργία JSON βασισμένο σε JSON-Schema, Artifact, Blueprint, JSON, JSON Schema, JSON Editor, JSON Validator

Abstract

A Virtual Data Container (VDC) is a developing way o organizing and processing information. A data administrator that has organized their product in a VDC desires to advertise it to potential buyers/users. The way in which this is accomplished is by providing a specific file named "Artifact" or "Blueprint" to the VDC network describing the service. An Artifact of Blueprint is created in a specific way and is usually based on a JSON-Schema in order for all the files to be directly comparable. This way possible buyers/users can do some market research in order to find the service that best fulfills their needs.

This diploma thesis aims to the development of a web application that allows the user to provide a JSON-Schema as input and construct the Artifact/Blueprint they desire in a fast, easy and consistent manner

Keywords

Virtual Data Container, VDC, Web application, JSON generation based on JSON-Schema, Artifact, Blueprint, JSON, JSON Schema, JSON Editor, JSON Validator

στους γονείς μου

Ευχαριστίες

Θα ήθελα καταρχάς να ευχαριστήσω την καθηγήτρια κ.Θεοδώρα Βαρβαρίγου για την επίβλεψη αυτής της διπλωματικής εργασίας και για την ευκαιρία που μου έδωσε να αναπτύξω μία διαδικτυακή εφαρμογή αξιοποιώντας τις πλέον σύγχρονες προγραμματιστικές μεθόδους. Επίσης ευχαριστώ ιδιαίτερα το μεταπτυχιακό σπουδαστή Αναστάσιο Νικολακόπουλο για την καθοδήγησή του και την εξαιρετική συνεργασία που είχαμε. Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου για την υλική και ψυχολογική στήριξη που όλα αυτά τα χρόνια μου παρείχε.

Αθήνα, Μάρτιος 2021

Μηνάς Αλβέρτης

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Πρόλογος

Η εκπόνηση της παρούσας διπλωματικής εργασίας πραγματοποιήθηκε στον Τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ΣΗΜΜΥ) του Εθνικού Μετσόβιου Πολυτεχνείου (ΕΜΠ).

Αντικείμενο της εργασίας ήταν η ανάπτυξη μίας διαδικτυακής εφαρμογής για την δημιουργία και επεξεργασία δομών σχεδιασμένων για περιγραφή και ενορχήστρωση μικρο-υπηρεσιών.

Μέρος Ι

Θεωρητικό Μέρος

Κεφάλαιο 1

Εισαγωγή

Η τεράστια αξία των δεδομένων στη σημερινή εποχή της πληροφορίας έχει αναγνωριστεί τόσο από τον επιστημονικό όσο και από τον επιχειρησιακό κλάδο. Αυτό σε συνδυασμό με την εκθετική αύξηση της παραγωγής τους καθημερινά, καθιστά αναγκαία την ανάπτυξη νέων μεθόδων διαχείρισης τους. Θεμέλιο σε αυτή τη διαχείριση είναι η σωστή και αποδοτική οργάνωσή τους.

Στα πλαίσια σχεδιασμού και ανάπτυξης μία εφαρμογής, καθοριστικό ρόλο παίζει το ποια δεδομένα θα χρησιμοποιηθούν. Συνεπώς, ο διαχειριστής των δεδομένων αποτελεί τον πρώτο κρίκο σε αυτή αλυσίδα. Μέλημά του είναι η αποδοτική οργάνωση των δεδομένων αυτών, όπως επίσης και η επικοινωνία του τρόπου με τον οποίο αυτά οργανώθηκαν σε όσους επιθυμούν να τα αξιοποιήσουν.

1.1 Εισαγωγή

1.1.1 Το Vitrual Data Container (VDC)

Ένα Ψηφιακό Κιβώτιο Δεδομένων - Virtual Data Container (VDC) στοχεύει στην επίλυση του προβλήματος της οργάνωσης των δεδομένων καθώς επίσης προσφέρει εύκολη πρόσβαση σε αυτά. Ένα VDC μπορεί να αποτελεί ένα σύνολο δεδομένων, μια διαδικτυακή υπηρεσία, ένα λειτουργικό σύστημα, μία εφαρμογή κτλ. οργανωμένα σε μία ψηφιακή απεικόνιση ώστε να παρουσιάζονται με ευκολία σε δυνητικούς αγοραστές ή χρήστες. Η χρήση ενός Virtual Data Container αποτελεί ένα μέσο εύκολης, γρήγορης και ασφαλούς παροχής δεδομένων όσων αφορά στην δομή και στην προέλευση τους. Λειτουργεί ως ένα μεσάζον λογισμικό (middleware) το οποίο:

- Δίνει την δυνατότητα στους προγραμματιστές να καθορίζουν απλά τις ανάγκες τους όσων αφορά στα δεδομένα
- Αναλαμβάνει την παροχή των δεδομένων αυτών γρήγορα, σωστά και με ασφάλεια.

- Αποκρύπτει την πολυπλοκότητα της δομής δεδομένων εξαλείφοντας την ανάγκη για την πλήρη κατανόηση ενός τέτοιου συστήματος από τον προγραμματιστή
- Φέρει ένα σύνολο τεχνικών επεξεργασίας και μεταποίησης δεδομένων όπως π.χ. η κωδικοποίηση (encryption), η συμπίεση (compression).
- Μπορεί να οργανωθεί με βάση το προγραμματιστικό μοντέλο node-RED (Το node-RED αποτελεί ένα προγραμματιστικό εργαλείο που δίνει τη δυνατότητα στους χρήστες να ενώνουν μεταξύ τους κομμάτια κώδικα nodes δημιουργώντας "ροές" (flows) με σκοπό την ανάπτυξη πολύπλοκων διαδικασιών απαιτώντας ελάχιστες γνώσεις προγραμματισμού[1]).
- Δίνει πρόσβαση σε δεδομένα ανεξάρτητα από το αν βρίσκονται στο "νέφος" (cloud) ή σε συσκευές απομακρυσμένης πρόσβασης (edge divices).[2]

Ένας διαχειριστής δεδομένων καλείται να δημιουργήσει μια ψηφιακή απεικόνιση VDC ή αλλιώς ένα VDC JSON ("Artifact" ή "Blueprint") το οποίο θα καθορίζει τα χαρακτηριστικά του VDC όπως:

- Ποιες είναι οι διαθέσιμες πηγές δεδομένων (data sources).
- Τον τρόπο πρόσβασης στα δεδομένα.
- Ποιες είναι τα διαθέσιμες Διασυνδέσεις Προγραμματισμού Εφαρμογών (APIs).
- Πώς τα δεδομένα από τις προαναφερθείσες πηγές θα πρέπει να επεξεργασθούν ώστε να είναι διαθέσιμα μέσω των APIs.
- Μη λειτουργικά χαρακτηριστικά που καθορίζουν την ποιότητα των δεδομένων και της υπηρεσίας.
- Το components cookbook: ένα script το οποίο καθορίζει τα τμήματα (modules) που απαρτίζουν το VDC καθώς και τον τρόπο ανάπτυξης τους.

Το παραπάνω γίνεται με σκοπό τη θέσπιση ενός ενιαίου τρόπου παρουσίασης των διαθέσιμων δεδομένων σε δυνητικούς χρήστες ή αγοραστές. Μέσω των VDC Artifacts ο πιθανός χρήστης ή αγοραστής θα μπορεί να πραγματοποιήσει μια "έρευνα αγοράς" συγκρίνοντας διαφορετικούς προμηθευτές VDCs ώστε να καταλήξει στην υπηρεσία η οποία ικανοποιεί τις προϋποθέσεις που έχει ορίσει ο ίδιος για την ανάπτυξη της εφαρμογής του.

Συμπεραίνουμε, λοιπόν, ότι είναι επιτακτική η ανάγκη ένας διαχειριστής δεδομένων να έχει τη δυνατότητα να δημιουργεί τα προαναφερθέντα VDC Artifacts εύκολα, γρήγορα και σωστά ώστε να διαφημίσει αποτελεσματικά το προϊόν του στο δίκτυο των VDCs.

1.1.2 Το VDC Artifact-Blueprint

Ένα VDC Artifact επιβάλλεται να πληροί ορισμένες προϋποθέσεις. Αρχική προϋπόθεση είναι να ακολουθεί τη μορφή JSON (JavaScript Object Notation)[3] που αποτελεί έναν απλό και εύκολο στην κατανόηση τρόπο αποθήκευσης και ομαδοποίησης δεδομένων που κατεξοχήν χρησιμοποιείται για την αποστολή δεδομένων από έναν διακομιστή σε μία ιστοσελίδα.[4, 5] Βασικότερη, όμως, προϋπόθεση είναι να επικυρώνεται να με βάση ένα JSON-Schema.

Ένα JSON-Schema είναι μία γραμματική γλώσσα που καθορίζει τη δομή, το περιεχόμενο και τη σημασιολογία ενός JSON αντικειμένου. Προσφέρει τη δυνατότητα προσδιορισμού μεταδεδομένων σχετικά με τη σημασία των "ιδιοτήτων" (properties) ενός "αντικειμένου" (object) καθώς και το σύνολο τιμών οι οποίες είναι δεκτές για κάθε μία από αυτές τις "ιδιότητες". Η εφαρμογή αυτής της γραμματικής σε ένα αρχείο JSON έχει ως αποτέλεσμα το Schema να περιγράφει ένα σύνολο JSON αντικειμένων τα οποία είναι "δεκτά" σύμφωνα με το Schema αυτό. [6, 7]

Στα πλαίσια του VDC μπορούν να αναπτυχθούν διάφορα JSON-Schemas τα οποία να καθορίζουν πώς ένα Artifact-Blueprint θα πρέπει να συνταχθεί. Ένα Artifact, στη γενική του μορφή, μπορεί να αποτελείται από τέσσερα βασικά μέρη (στο εξής sections):

- Το πρώτο από αυτά περιλαμβάνει γενικές πληροφορίες που αφορούν στη σύνθεση του VDC. Σκοπός της ύπαρξής του είναι να παρουσιάσει την προσφερόμενη υπηρεσία στον αναγνώστη/πιθανό αγοραστή αναλύοντας τα βασικά της στοιχεία. Αφορά κάποιο ανώτερο στέλεχος μίας εταιρίας το οποίο είναι υπεύθυνο για τη λήψη βασικών αποφάσεων όπως την απόκτηση/αγορά του VDC.

- Το δεύτερο section περιγράφει την τιμολόγηση του VDC συνολικά καθώς και όλα τα οικονομικά θέματα που το αφορούν. Μία τέτοιου είδους ανάλυση είναι απαραίτητη για να καλύψει όλα τα ζητήματα τιμολόγησης που προκύπτουν από μία αρχιτεκτονική πολλών μικρο-υπηρεσιών και αφορά το εταιρικό στέλεχος που είναι υπεύθυνο για οικονομικά θέματα.

- Το τρίτο section περιλαμβάνει μία γενική εικόνα για τεχνικά θέματα όπως οι εκδόσεις (versioning), οι βιβλιοθήκες (libraries), η επεκτασιμότητα (scalability), οι περιορισμοί (limitations) και τις υποστηριζόμενες πηγές δεδομένων μέσα στο VDC. Επίσης, περιγράφει την πλήρη διαδικασία ανάπτυξης(deployment) του VDC καθώς και τις εξαρτήσεις (dependencies) μεταξύ των μικρο-υπηρεσιών. Αφορά, το στέλεχος υπεύθυνο για τα επιστημονικά και τεχνολογικά θέματα μέσα σε έναν οργανισμό.

- Τέλος, το τέταρτο section απευθύνεται στους τεχνικούς λογισμικού. Έχει ως στόχο την πλήρη κατανόηση της δομής και της λειτουργίας του VDC από τους

παραπάνω ώστε να είναι σε θέση ανά πάσα στιγμή να εγγυηθούν την ομαλή λειτουργία του λογισμικού της εταιρίας.

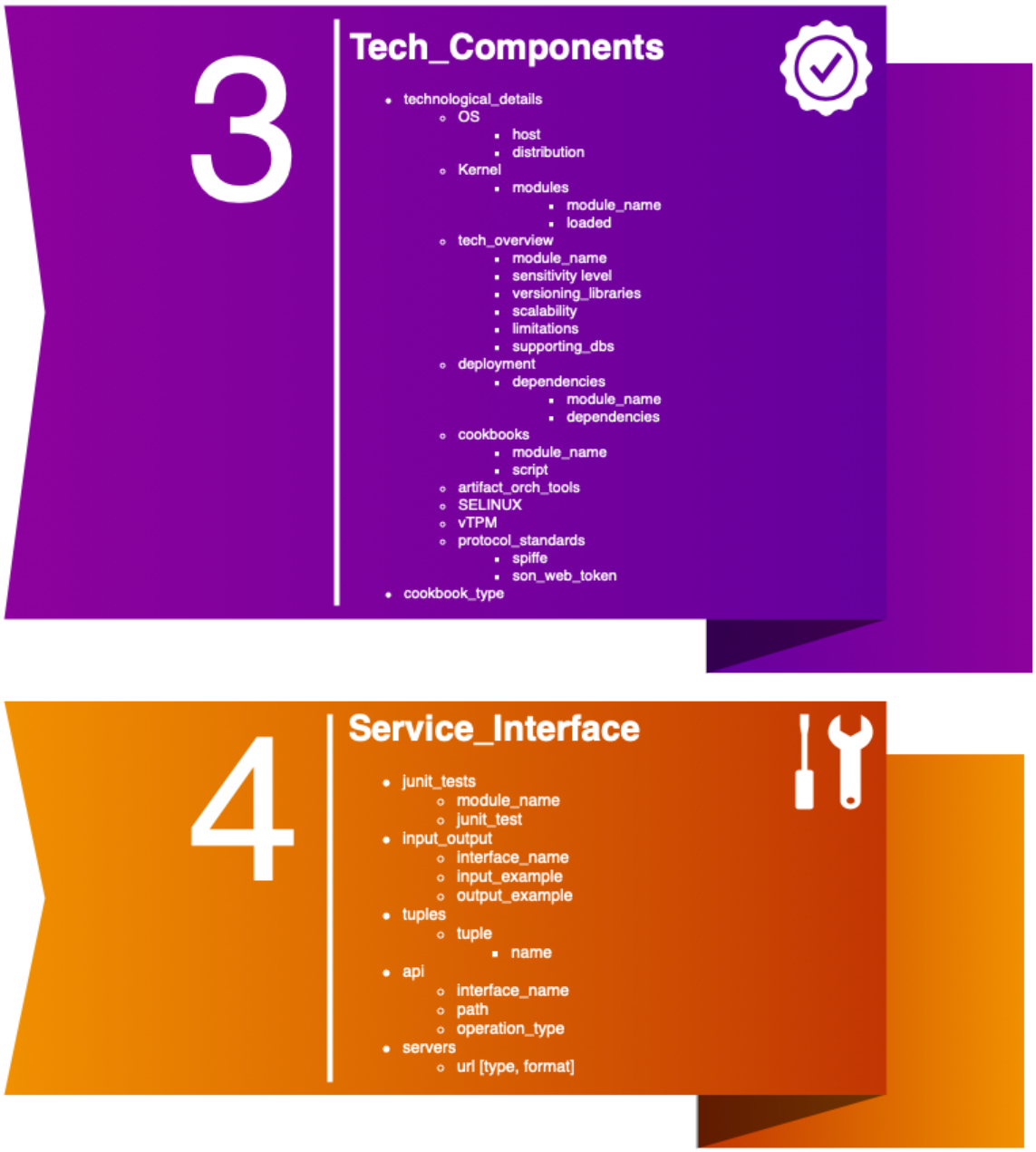
Τα σχήματα 1.1 και 1.2 παρουσιάζουν ένα γενικό παράδειγμα[8] του προηγούμενου σε μία ψευδοδομή JSON. Τονίζεται, παρ' όλα αυτά ότι τα παραπάνω δεν αποτελούν περιοριστικούς κανόνες, αλλά μία καλή πρακτική προσέγγιση. Ο εκάστοτε δημιουργός ενός Artifact μπορεί να είτε αυξήσει είτε να μειώσει τα sections εμπλουτίζοντας ή περιορίζοντάς τα πεδία τους ώστε να καταλήξει στο αποτέλεσμα που καλύτερα εκφράζει/περιγράφει το VDC του.

Τέλος, το παραπάνω μπορούν να περιγραφούν με τυπικό τρόπο από ένα JSON-Schema. Όπως αναφέρθηκε ήδη αυτό το Schema θα περιγράφει ένα σύνολο JSONs - Artifacts τα οποία θα είναι "σύμφωνα" με αυτό το Schema. Συνεπώς, ο συντάκτης ενός Artifact έχοντας στην κατοχή του ένα JSON-Schema δεν έχει παρά να δημιουργήσει ένα JSON το οποίο να επικυρώνεται (validated) με βάση το Schema.

1.2 Σκοπός της εργασίας

Αναγνωρίζοντας την παραπάνω ανάγκη για δημιουργία VDC JSONs - Artifacts η εκπόνηση αυτής της εργασίας στοχεύει στη δημιουργία μιας εφαρμογής που θα δίνει την δυνατότητα στον χρήστη - διαχειριστή δεδομένων να παράξει εύκολα και γρήγορα ένα VDC Artifact. Η λειτουργία της εφαρμογής αυτής θα περιλαμβάνει την εισαγωγή ως είσοδο ενός JSON-Schema και θα δημιουργεί μία εύχρηστη διεπαφή χρήστη (User Inteface - UI) μέσω της οποίας θα είναι δυνατή η εύκολη και γρήγορη δημιουργία ενός JSON αντικειμένου για την περιγραφή ενός VDC το οποίο θα είναι "δεκτό"/επικυρωμένο με βάση το δοθέν JSON-Schema.

Σχήμα 1.1: *Παράδειγμα section 1 και 2 ενός Artifact*



Σχήμα 1.2: *Παράδειγμα section 3 και 4 ενός Artifact*

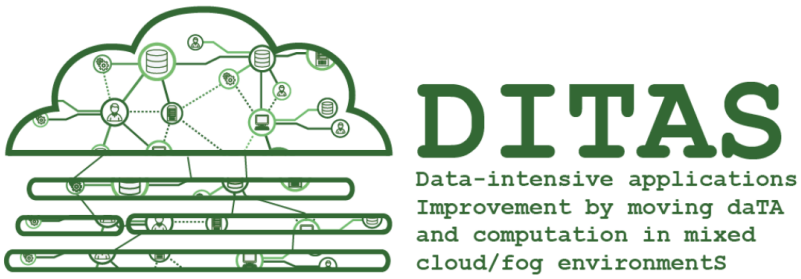
Κεφάλαιο 2

Σχετικές Εργασίες

Σε αυτό κεφάλαιο αυτό γίνεται αναφορά σε μία σχετική εφαρμογή βασισμένη στο Blueprint JSON Schema από το Ευρωπαϊκό Πρόγραμμα DITAS[9] η οποία γέννησε την ιδέα για την εκπόνηση της παρούσας εργασίας.

2.1 DITAS VDC Blueprint editor

2.1.1 DITAS JSON Schema



Εικόνα 2.1: *Λογότυπο του Ευρωπαικού προγράμματος DITAS.*

Πηγή: https://www.ditas-project.eu/

Το DITAS VDC Blueprint's standard JSON schema αποτελείται από τα εξής πέντε μέρη:

1. Internal Structure, όπου αναφέρονται γενικές πληροφορίες σχετικά με το VDC Blueprint

2. Data Managemen, που αποτελεί έναν json πίνακα (array) με που περιλαμβάνει διαφορετικές μεθόδους.

3. Abstract Properties, το οποίο περιλαμβάνει όλες τις ιδιότητες του VDC που δεν ακολουθούν κάποια συγκεκριμένη δομή.

4. Cookbook Appendix, που περιγράφει το Cookbook Appendix του VDC.

5. Exposed API, αποτελεί το CAF RESTful API του VDC σύμφωνα με την έκδοση (3.0.2) του OpenAPI Specification (OAS)[10]

2.1.2 Ο DITAS JSON Schema Editor

Ο DITAS JSON Schema Editor (το διαδικτυακό αποθετήριο (Git repository) που περιλαμβάνει τον κώδικα για τον Editor μπορεί να βρεθεί στη βιβλιογραφία[11]) μπορεί να χρησιμοποιηθεί για τη δημιουργία ενός Blueprint σύμφωνα με τις προδιαγραφές που ορίζει το DITAS JSON Schema. Ο χρήστης που χρησιμοποιεί την εφαρμογή μπορεί να περιηγηθεί σε μία οπτική απεικόνιση ενός Blueprint του οποίου τα πεδία μπορεί να συμπληρώσει. Πραγματοποιούνται οι απαραίτητοι έλεγχοι για την συμπλήρωση των πεδίων που χαρακτηρίζονται ως απαραίτητα (required) καθώς και κάθε άλλος έλεγχος που οδηγεί στην παραγωγή ενός αρχείου json (Blueprint) σύμφωνο με το DITAS Schema. Η χρήση του είναι απλή για τον χρήστη και τον οδηγεί να συμπληρώσει τα πεδία του json με τρόπο που μειώνει σε πολύ μεγάλο βαθμό τα λάθη που θα προέκυπταν ο χρήστης προσπαθούσε να δημιουργήσει το αποτέλεσμα γράφοντάς το μόνος του.

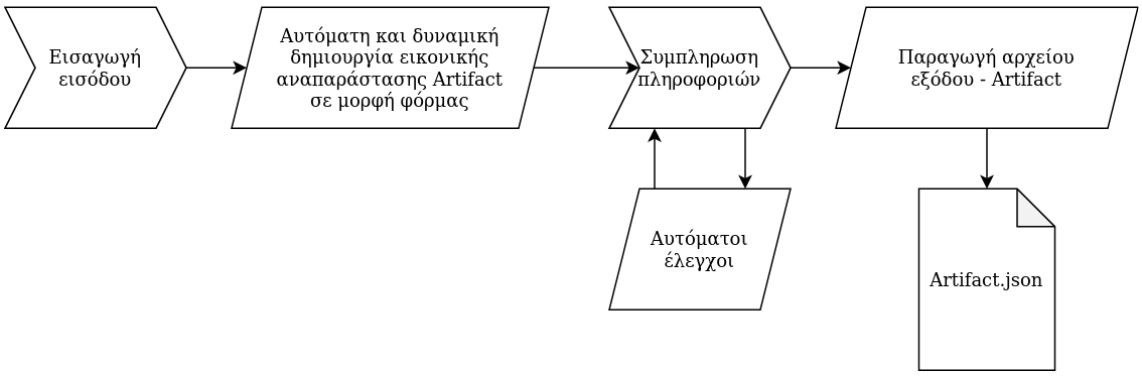
Τα παραπάνω αποτέλεσαν έμπνευση για την υλοποίηση του VDC Artifact Editor καθώς η εφαρμογή που αποτελεί αντικείμενο της εργασίας αυτής έχει ως σκοπό να δημιουργήσει μία γενικότερη εκδοχή του προηγούμενου. Ουσιαστικά, ο VDC Artifact Editor θα μπορεί να δημιουργήσει δυναμικά παρόμοια αποτελέσματα ανάλογα με την είσοδο που λαμβάνει. Συνεπώς, αν η είσοδός του είναι το DITAS Schema θα προκύψει κάτι λειτουργικά παρόμοιο.

Κεφάλαιο 3

Προσέγγιση της λύσης

Στο κεφάλαιο αυτό γίνεται μία γενική και abstract περιγραφή του τρόπου με τον οποίο προσεγγίζεται η λύση. Περιγράφεται η γενική ιδέα, οι στόχοι που τέθηκαν για τις δυνατότητες που θα προσφέρει η εφαρμογή και επεξηγούνται κάθε φορά οι λόγοι για τους οποίους επιλέχθηκε αυτός ο τρόπος προσέγγισης. Τέλος, αναλύεται ο τρόπος/φιλοσοφία αποθήκευσης των δεδομένων.

3.1 Γενική ιδέα



Σχήμα 3.1: *Γενική ιδέα της εφαρμογής*

Όπως αναφέρθηκε και σε προηγούμενα κεφάλαια, σκοπός για την εκπόνηση της εργασίας αυτής αποτελεί η δημιουργία μίας διαδικτυακής εφαρμογής η οποία θα επιτρέπει στο χρήστη να δημιουργεί εύκολα και γρήγορα ένα Artifact επικυρωμένο από ένα JSON-Schema. Το JSON-Schema αυτό θα δίνεται ως είσοδος(input) μέσω της διεπαφής χρήστη στην εφαρμογή και αυτόματα θα δημιουργείται μία οπτική αναπαράσταση του προς δημιουργία Artifact σε μορφή φόρμας συμπλήρωσης. Εν συνεχεία, ο χρήστης θα συμπληρώνει κατάλληλα όλες τις απαραίτητες πληροφορίες στα πεδία της φόρμας δημιουργώντας έτσι το ζητούμενο αποτέλεσμα σύντομα και με ευκολία. Σημαντικό πλεονέκτημα θα είναι τη γνώση πως το αποτέλεσμα θα συμφωνεί πλήρως με τους κανόνες του JSON-Schema που έχει δοθεί ως είσοδος, καθώς θα

έχουν γίνει αυτόματα οι απαραίτητοι έλεγχοι, ενώ ο χρήστης θα έχει οδηγηθεί από την εφαρμογή προς την κατεύθυνση της σωστής συμπλήρωσης ενός Artifact.

3.2 Δυνατότητες της εφαρμογής

3.2.1 Εισαγωγή δεδομένων εισόδου

Βασική και προφανής δυνατότητα για την εφαρμογή αυτή θα είναι η εισαγωγή του προαναφερθέντος JSON-Schema. Το παραπάνω θα μπορεί να επιτευχθεί με δύο τρόπους. Στην αρχική οθόνη (Home) θα υπάρχει ένα ευρύ πεδίο στο οποίο θα επιτρέπεται η εισαγωγή του input από τον χρήστη. Ο προηγούμενος θα μπορεί είτε να πληκτρολογήσει εξολοκλήρου την είσοδο μέσα στο πεδίο είτε να την εισάγει αυτόματα από ένα αρχείο με κατάληξη .json κάνοντας χρήση του κατάλληλου πλήκτρου ("Upload") που θα βρίσκεται στην οθόνη και θα επιτρέπει αναζήτηση εντός της συσκευής.

3.2.2 Εμφάνιση κανόνων και βοηθειών χρήσης

Μία επίσης δυνατότητα που θα προσφέρει η εφαρμογή από την αρχική οθόνη θα είναι η εμφάνιση των κανόνων που έχουν θεσπιστεί για την εισαγωγή του input και βοηθειών χρήσης. Το παραπάνω γίνεται με σκοπό ο χρήστης να γνωρίζει εκ των προτέρων ποιες είναι οι αναγκαίες συμβάσεις που έχουν σχετικά με τον τρόπο γραφής του JSON-Schema εισόδου ώστε να αποφεύγονται μελλοντικά μηνύματα λαθών καθώς και η ανάγκη εκ των υστέρων αλλαγής του input. Από την άλλη, οι βοήθειες θα επεξηγούν ορισμένες περιπτώσεις κατά τη χρήση της εφαρμογής οι οποίες ενδεχομένως να μην είναι απολύτως ξεκάθαρες στον καθένα με την πρώτη ματιά.

3.2.3 Μηνύματα και αντιμετώπιση σφαλμάτων

Το input θα γίνεται δεκτό από την εφαρμογή μόνο αν πληροί τους προαναφερθέντες κανόνες (π.χ. αν χρησιμοποιεί τους βασικούς τύπους ενός JSON-Schema όπως αυτοί ορίζονται [12]. Αναλυτική εξήγηση των κανόνων πραγματοποιείται στην υποενότητα 4.1.2 του επόμενου κεφαλαίου) και εφόσον αποτελεί ένα σωστά συνταγμένο JSON. Σε περίπτωση που αυτό δεν συμβαίνει ο χρήστης θα ενημερώνεται με σχετικό μήνυμα ώστε να κάνει τις απαραίτητες διορθώσεις. Στην σπάνια περίπτωση που εμφανιστεί κάποιο σφάλμα κατά την εκτέλεση της εφαρμογής, αφού ο χρήστης έχει ήδη αρχίσει να προσθέτει τις πληροφορίες, και όχι στην αρχή κατά την "παραγωγή" της δυναμικής φόρμας, θα υπάρχει η εξής αντιμετώπιση:

- Αρχικά ο χρήστης ενημερώνεται με μήνυμα για τον τύπο του σφάλματος

- Του δίνεται η δυνατότητα να συνεχίσει την εργασία του χωρίς να χάσει την πρόοδο που έχει σημειώσει, παρά μόνο ό,τι αφορά στο "προβληματικό" σημείο του input.

- Αντίθετα, μπορεί, εφόσον το επιθυμεί, να επιστρέψει στην αρχική οθόνη (χάνοντας την πρόοδο που έχει σημειώσει) και να διορθώσει την είσοδο/input ξεκινώντας από την αρχή.

Όπως σημειώνεται και στο κεφάλαιο ;;;;; που αφορά στα αποτελέσματα των ελέγχων, η παραπάνω κατάσταση αν και ιδιαίτερα δυσμενής, καθότι μπορεί να χαθεί ένα μέρος της καταβληθείσας προσπάθειας του χρήστη, κρίνεται ιδιαίτερα σπάνια. Έχουν τοποθετηθεί πολυάριθμοι έλεγχοι κατά τη δημιουργία της δυναμικής φόρμας (πριν δηλαδή ο χρήστης αρχίσει την εργασία του) ώστε η πιθανότητα του να συμβεί το παραπάνω σχεδόν να εξαλείφεται.

3.2.4 Πλοήγηση

Κατά τη χρήση του Editor ο χρήστης θα μπορεί να πλοηγηθεί στην εικονική αναπαράσταση του Artifact με όποιον τρόπο αυτός ή αυτή επιθυμεί. Θα υπάρχει πάντα στην κορυφή της σελίδας ένα menu το οποίο θα επιτρέπει την άμεση μετάβαση σε κάθε διαφορετικό section. Επιπλέον, θα υπάρχουν κατάλληλα πλήκτρα "Επόμενο" και "Προηγούμενο" (next and previous buttons) για σειριακή μετάβαση σε διαφορετικό section.

Κάθε ένα από τα παραπάνω sections θα εμφανίζει στο αριστερό μέρος της οθόνης ένα υπο-μενού το οποίο θα εμφανίζει τα "Κυρίως properties" (στο εξής head properties) του section. Με αυτόν τον τρόπο ο χρήστης θα μπορεί να πλοηγείται μέσα στο Artifact και να το συμπληρώνει με όποια σειρά αυτός επιθυμεί.

Δίνεται, λοιπόν, η δυνατότητα της μη σειριακής συμπλήρωσης των πληροφοριών ώστε να η χρήση του Editor να είναι εύκολη και ευχάριστη επιτρέποντας την επιστροφή σε προηγούμενο σημείο για πραγματοποίηση αλλαγών χωρίς κανένα πρόβλημα.

3.2.5 Συμπλήρωση πεδίων και πινάκων

Κατά τη χρήση της εφαρμογής, η συμπλήρωση των πεδίων (fields) γίνεται με προφανή τρόπο. Π.χ. ένα πεδίο τύπου συμβολοσειράς (string) δέχεται όλες τις τιμές από το πληκτρολόγιο, ένα πεδίο αριθμητικού τύπου (number) δέχεται αριθμητικές τιμές με δυνατότητα αυξομείωσης μέσω κατάλληλων πλήκτρων με μορφή βέλους (arrow buttons), ενώ ένα πεδίο δυαδικής τιμής (boolean) παρουσιάζεται με τη μορφή ενός πλαισίου ελέγχου (checkbox) το οποίο μπορεί να είναι ενεργοποιημένο ή απενεργοποιημένο (τιμές true/false αντίστοιχα) κοκ.

Τα παραπάνω πεδία, εφόσον δεν ορίζονται από την είσοδο ως required έχουν τη δυνατότητα, μέσο κατάλληλου πλήκτρου αφαίρεσης πεδίου (remove button) με μορ-

φή κάδου απορριμάτων, να αφαιρεθούν από το αποτέλεσμα. Παραμένουν, πάραυτα, στη φόρμα έχοντας μια πιο αχνή εμφάνιση από το κανονικό και χωρίς τη δυνατότητα συμπλήρωσης τους. Αυτό γίνεται, ώστε ο χρήστης να μπορεί να θυμηθεί ότι στο συγκεκριμένο σημείο είχε αρχικά οριστεί αυτό το πεδίο και αν αλλάξει γνώμη να το επαναφέρει μέσω του κατάλληλου πλήκτρου επαναφοράς πεδίου (restore button).

Σε ότι αφορά τους πίνακες (arrays), κατά τη δημιουργία της δυναμικής φόρμας περιέχουν ένα αντικείμενο με "άδειες"/μη-συμπληρωμένες τιμές. Στο τέλος του πίνακα υπάρχει κατάλληλο πλήκτρο προσθήκης αντικειμένου (add item button) καθώς, επίσης, υπάρχει και πλήκτρο διαγραφής αντικειμένου (delete item button) (με μορφή κάδου απορριμάτων με κόκκινο φόντο, ώστε να μην συγχέεται με το restore button) δίπλα σε κάθε αντικείμενο.

Τέλος, σε περίπτωση που γίνει χρήση της ιδιότητας "oneOf" του JSON-Schema υπάρχουν κατάλληλα πλήκτρα για την επιλογή από το χρήστη της επιθυμητής περίπτωσης για κάθε αντικείμενο.

3.2.6 Αυτόματη αποθήκευση πληροφοριών

Για την διευκόλυνση του χρήστη, όλα όσα αναφέρονται στην υποενότητα 3.2.5 αποθηκεύονται αυτόματα αμέσως μόλις γίνει κάποια αλλαγή. Μοναδική εξαίρεση στο αυτό αποτελεί το πεδίο τύπου αντικειμένου (object type field) το οποίο για λόγους που θα εξηγηθούν στο επόμενο κεφάλαιο επιβάλει τη χρήση πλήκτρου ανάλυσης (parse button). Το παραπάνω γεγονός εξαλείφει την ανάγκη για επαναλαμβανόμενη χρήση κατάλληλου πλήκτρου ώστε τα δεδομένα να αποθηκευτούν στην εφαρμογή. Για την αποφυγή δημιουργίας μεγάλου και επαναλαμβανόμενου επεξεργαστικού φορτίου από το παραπάνω (π.χ. κατά τη συμπλήρωση ενός πεδίου string) έχουν υλοποιηθεί κατάλληλες μέθοδοι μείωσης αυτού του φορτίου χρησιμοποιώντας τεχνικές επαναχρησιμοποίησης δεδομένων.

3.2.7 Παραγωγή αρχείου εξόδου - Artifact

Τελευταία, αλλά ίσως η βασικότερη, δυνατότητα που θα προσφέρει ο Editor είναι η παραγωγή του αρχείου εξόδου - Artifact. Στο κάτω μέρος της σελίδας θα υπάρχει πάντα ένα πλήκτρο δημιουργίας αρχείου εξόδου (output file button). Το παραπάνω πλήκτρο αρχικά θα εκκινεί τον έλεγχο συμπλήρωσης των απαραίτητων πεδίων σύμφωνα με το input. Ανάλογα με το αποτέλεσμα του ελέγχου αυτού υπάρχουν οι εξής δύο εκδοχές:

1. Ο έλεγχος ήταν επιτυχής: δημιουργείται το αρχείο εξόδου και δίνεται στον χρήση η δυνατότητα λήψης του (download)

2. Ο έλεγχος ήταν ανεπιτυχής: αυτό σημαίνει ότι υπάρχουν στο Artifact κάποια πεδία τα οποία έχουν τεθεί από την είσοδο ως απαραίτητα αλλά παρ' όλα αυτά

δεν έχουν συμπληρωθεί. Στην περίπτωση αυτή ο χρήστης ενημερώνεται για αυτό με μία λίστα που περιέχει τα προαναφερθέντα μη συμπληρωμένα πεδία και του δίνεται η επιλογή είτε να κάνει λήψη του ημιτελούς αρχείου είτε να επιστρέψει στη φόρμα για τη συμπλήρωσή του.

3.3 Η προσέγγιση Single Source of Truth

Ο τρόπος αποθήκευσης των δεδομένων/πληροφοριών είναι ένα από τα σημαντικότερα σημεία του σχεδιασμού μίας εφαρμογής. Μία διαδικτυακή εφαρμογή, όπως αυτή που αναπτύχθηκε στα πλαίσια της εργασίας, δομείται εκ φύσεως από πολλά μικρότερα και μεγαλύτερα μέρη (components). Αυτά τα compontents έχουν συνήθως τη δυνατότητα να αποθηκεύουν πληροφορίες. Θα ήταν δηλαδή λογικό να γίνει η σκέψη τα δεδομένα της εφαρμογής να είναι μοιρασμένα σε πολλά και διαφορετικά μέρη και στο τέλος να συλλέγονται, ώστε να γραφούν σε ένα αρχείο το οποίο να παράγεται ως έξοδος.

Η παραπάνω λογική, αν και θα μπορούσε να αποτελέσει μια λύση, δημιουργεί αρκετά προβλήματα. Όταν η πληροφορία διαμοιράζεται μεταξύ των components αναπόφευκτα θα υπάρξουν περιπτώσεις όπου κάποια στιγμή θα πρέπει να αποθηκευτεί η τιμή για ένα πεδίο σε δύο ή περισσότερα μέρη. Έπειτα από αυτό, αν υπάρξει απόπειρα αλλαγής της τιμής αυτής θα πρέπει να γίνει πολύ προσεκτικά ώστε να αλλάξει σε όλα τα σημεία όπου βρίσκεται. Αν αυτό δε γίνει σωστά, θα προκύψουν προβλήματα καθώς η τιμή δε θα είναι παντού η ίδια. Για το λόγο αυτό, θα περιγράψουμε την πρακτική της Μοναδικής Πηγής Αληθείας.

Η πρακτική της Μοναδικής Πηγής Αληθείας (Single Source of Truth - SSoT) ορίζει ότι τα δεδομένα δομούνται με τέτοιον τρόπο ώστε να αποθηκεύονται σε ένα μόνο σημείο. Κάθε φορά που απαιτείται η χρήση τους, αυτή θα πρέπει να γίνεται αυστηρά και μόνο με αναφορά. Αυτό εξασφαλίζει ότι η αλλαγή/επεξεργασία αυτών δεν θα δημιουργήσει προβλήματα στο σύστημα καθώς η αλλαγή θα "φανεί" παντού (αφού όλες οι αναφορές θα αφορούν την ίδια θέση αποθήκευσης). Η παραπάνω προσέγγιση επιλέχθηκε κατά τον σχεδιασμό της παρούσας εφαρμογής. Όλα τα δεδομένα αποθηκεύονται στο ίδιο μοναδικό σημείο και δέχονται επεξεργασία μόνο εκεί από κατάλληλες συναρτήσεις.

Μέρος II

Πρακτικό Μέρος

Κεφάλαιο 4

Υλοποίηση της Εφαρμογής

Η ανάπτυξη της εφαρμογής στα πλαίσια της συγκεκριμένης διπλωματικής εργασίας έχει πραγματοποιηθεί χρησιμοποιώντας τη γλώσσα ReactJS[13]. Ουσιαστικά η ReactJS είναι μια βιβλιοθήκη για τη γλώσσα JavaScript. Πρόκειται δηλαδή για μία γλώσσα υπερσύνολο της JavaScript η οποία χρησιμοποιείται για την ανάπτυξη διαδραστικών και δυναμικών διεπαφών χρήστη σε διαδικτυακές εφαρμογές μοναδικής σελίδας (single-page applications).

Οι Single-page applications είναι οι εφαρμογές που κατά τη διάδραση με το χρήστη επαναπροσδιορίζουν την ίδια μοναδική σελίδα σε αντίθεση με τον παραδοσιακό τρόπο όπου φορτώνεται διαφορετική σελίδα κάθε φορά όπως σε εφαρμογές πολλαπλών σελίδων (multi-page applications).

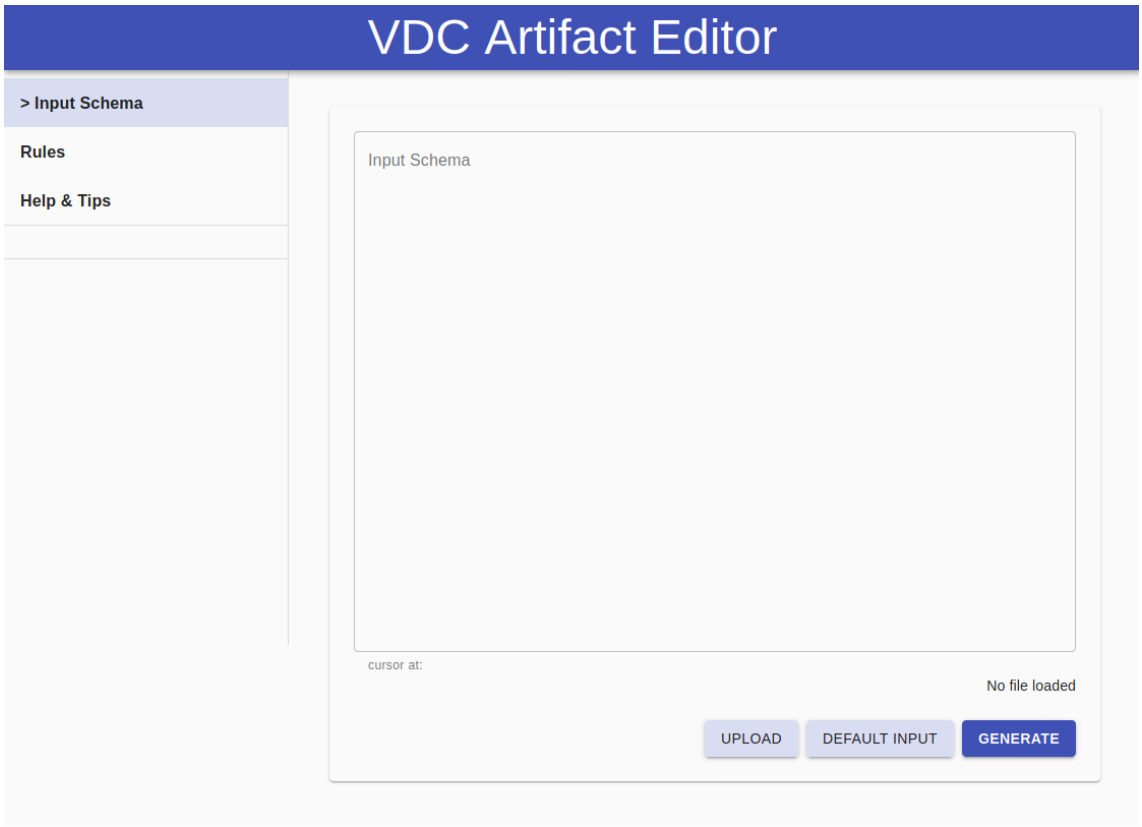
Επίσης, έχει χρησιμοποιηθεί η βιβλιοθήκη Material-UI[14] η οποία περιέχει χρήσιμα "συστατικά" (στο εξής components) για τη React ώστε να διευκολύνεται η ανάπτυξη της εφαρμογής στο κομμάτι της παρουσίασης της σελίδας σε έναν περιηγητή ιστού (web browser). Η υποστήριξη των δύο παραπάνω αποτελεί απαραίτητη προϋπόθεση για την εκτέλεση της εφαρμογής σε οποιοδήποτε σύστημα.

Στο κεφάλαιο αυτό γίνεται η ανάλυση των σημαντικότερων σημείων της εφαρμογής από πλευράς δομής και αναλύονται τα σημαντικότερα σημεία του κώδικα. Σημειώνεται πως ο κορμός την εφαρμογής, από προγραμματιστική σκοπιά, αποτελείται από τρία μεγάλα αρχεία: Home.js, Section.js, ItemMasterComponent.js τα οποία αλληλεπιδρούν με διάφορους τρόπους με πολλά μικρότερα αρχεία javascript. Ο πλήρης κώδικας της εφαρμογής βρίσκεται στη διαδικτυακό αποθετήριο GitHub στη διεύθυνση `https://github.com/AlvertisMinas/VDC-Artifact-Editor`

4.1 Αρχική σελίδα - Home Page

Στο Home.js δημιουργείται η αρχική σελίδα της εφαρμογής. Όπως φαίνεται και στην εικόνα 4.1, στο αριστερό μέρος της οθόνης υπάρχει το υπομενού που επιτρέπει τη μετάβαση σε μία από τις τρεις σελίδες/καρτέλες της αρχικής οθόνης:

1. Input Schema: Καρτέλα εισαγωγής του JSON-schema εισόδου
2. Rules: Όπου απαριθμούνται οι κανόνες που έχουν θεσπιστεί και πρέπει να διέπουν την είσοδο.
3. Help & Tips: Βοήθειες σχετικά με τη χρήση της εφαρμογής.



Εικόνα 4.1: *Αρχική Οθόνη/Home Page του VDC Artifact Editor*

4.1.1 Input Schema

Στην τρέχουσα καρτέλα γίνεται η εισαγωγή του input. Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, ο χρήστης μπορεί να πληκτρολογήσει στο μεγάλο πλαίσιο κειμένου (text box) το JSON-schema που επιθυμεί να εισάγει στην εφαρμογή. Τονίζεται ότι έχει υπάρξει μέριμνα για την εισαγωγή του χαρακτήρα tab (’\t ’) από το πληκτρολόγιο[15]. Σε περίπτωση που το παραπάνω δεν είχε γίνει, η πληκτρολόγηση του tab από το πληκτρολόγιο θα είχε ως αποτέλεσμα τη μετακίνηση της επιλογής στο επόμενο πεδίο ή πλήκτρο και θα ήταν αδύνατη η εισαγωγή του συγκεκριμένου χαρακτήρα ο οποίος είναι απαραίτητος για τη στοίχιση ενός JSON.

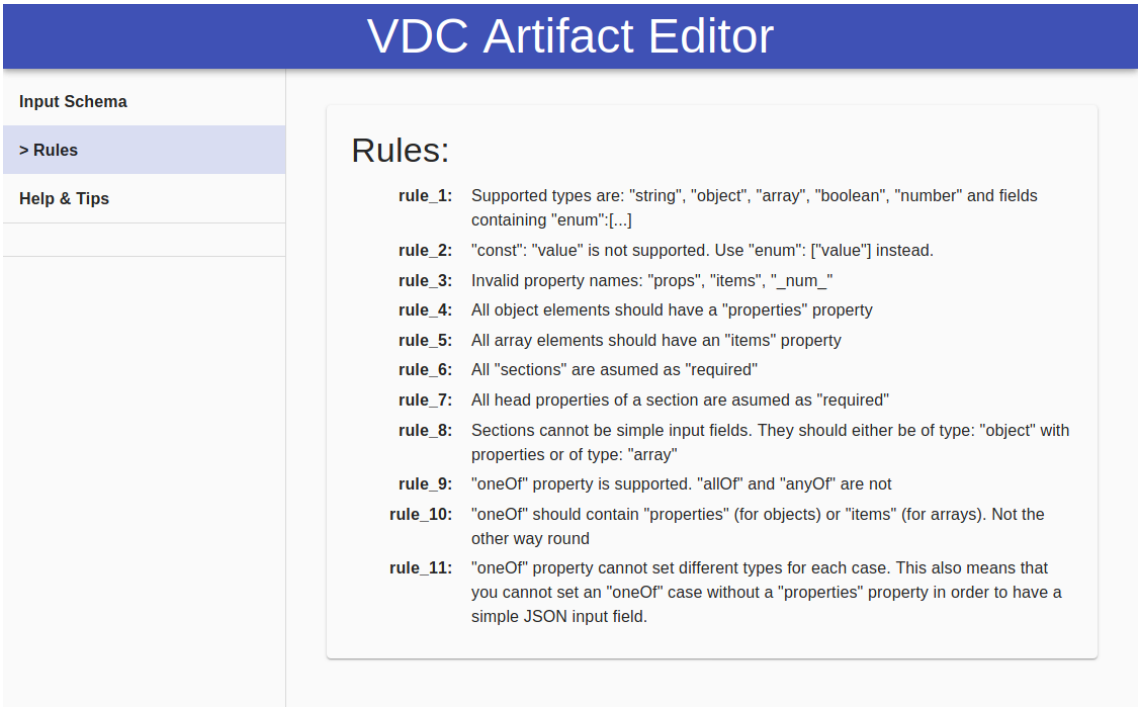
Έχουμε, όμως, ήδη αναφέρει ότι η παραπάνω διαδικασία δεν είναι πρακτική. Το πιθανότερο είναι πως ένας χρήστης θα έχει ήδη στην κατοχή του ένα JSON-schema. Συνεπώς, του δίνεται η δυνατότητα, μέσω του πλήκτρου ’Upload’, να αναζητήσει μέσα

στο σύστημα του και να μεταφορτώσει το αρχείο αυτό στην εφαρμογή ως είσοδο. Τα περιεχόμενα του αρχείου θα εμφανιστούν πλαίσιο κειμένου και σε αυτό το σημείο θα μπορεί, εφόσον το επιθυμεί, να κάνει αλλαγές.

Τέλος, σε περίπτωση που δεν έχει στην κατοχή του κάποιο Schema και δεν επιθυμεί να πληκτρολογήσει ένα ο ίδιος, του δίνεται η δυνατότητα να εισάγει αυτόματα το genericSchema.json που περιγράφηκε σε προηγούμενο κεφάλαιο μέσω του πλήκτρου 'Default Input'.

Όταν τελικά, με οποιονδήποτε τρόπο η είσοδος έχει δοθεί στην εφαρμογή, ο χρήστης πατά το πλήκτρο 'Generate' ώστε η εφαρμογή να παράξει την οπτική αναπαράσταση του Artifact.

4.1.2 Rules



Εικόνα 4.2: *Rules: Καρτέλα απαρίθμησης κανόνων που πρέπει να διέπουν το input schema*

Στην ενότητα αυτή θα γίνει η επεξήγηση των κανόνων που φαίνονται στην εικόνα 4.2. Παράλληλα με αυτό δίνεται η ευκαιρία με ορισμένους κανόνες να τεθούν οι βάσεις για την κατανόηση του τρόπου με τον οποίο γίνεται η δυναμική (dynamic) παραγωγή της φόρμας η οποία θα αναλυθεί στις επόμενες ενότητες.

Κανόνας #1

Οι υποστηριζόμενοι τύποι είναι οι εξής: "string", "object", "array", "boolean", "number" καθώς και πεδία που περιέχουν "enum":[...].

Με βάση το JSON-schema, ο τύπος ενός property μπορεί να δηλωθεί ως εξής:

```
{
  "type":  "number"
}
```

Στη θέση του "number" μπορεί να μπει οποιοσδήποτε άλλος τύπος από αυτούς που αναφέρθηκαν προηγουμένως, με εξαίρεση το "enum". Έχουμε:

- "string": για συμβολοσειρές
- "object": για αντικείμενα που περιέχουν δικά τους properties ή πεδίο εισαγωγής ενός JSON.
- "array": για πίνακες/λίστες
- "boolean": για δυαδικές τιμές true/false
- "number": για αριθμητικές τιμές

Η περίπτωση "enum" ορίζει τις τιμές που είναι δεκτές για το πεδίο και οι οποίες μπορεί να έχουν διαφορετικούς τύπους. Π.χ. η επιλογή μεταξύ των τιμών: "one", "two", 3 και false ορίζεται ως εξής:

```
{
  "enum":  ["one", "two", 3, false]
}
```

Κανόνας #2

Ο τύπος "const": "value" δεν υποστηρίζεται. Αντί του προηγούμενου μπορεί να χρησιμοποιηθεί το "enum": ["value"].

Το παραπάνω αναφέρεται στον τύπο "const" του JSON-schema. Πρόκειται για τύπο σταθερής τιμής (π.χ. "value"). Ο συγκεκριμένος τύπος δεν υποστηρίζεται από την εφαρμογή άλλα το ίδιο αποτέλεσμα μπορεί να υπάρξει αν χρησιμοποιηθεί το "enum" με μία μόνο δυνατή τιμή. Αυτή η τιμή θα προεπιλεγεί αυτόματα και ουσιαστικά θα λειτουργήσει σαν σταθερά.

Κανόνας #3

Μη έγκυρα ονόματα property: "props", "items", "_num_" .

Οι παραπάνω συμβολοσειρές έχουν δεσμευτεί από την εφαρμογή για εσωτερική χρήση και δεν είναι δυνατή η χρήση τους ως ονόματα property. Ένα παράδειγμα σωστού τρόπου ορισμού ενός πεδίου/αντικειμένου είναι το παρακάτω:

```
"property_name":{
    "type":  "string"
}
```

όπου στη θέση του "property_name" μπορεί να μπει οποιαδήποτε μη δεσμευμένη συμβολοσειρά.

Κανόνας #4

Όλα τα αντικείμενα τύπου "object" πρέπει να περιλαμβάνουν ένα property "properties".

Στο παρακάτω παράδειγμα βλέπουμε έναν τυπικό τρόπο ορισμού ενός αντικειμένου τύπου "object":

```
"one_object":{
    "type":  "object",
    "properties":  {
        "property_A":  {
            "type":  "string"
        },
        "property_B":  {
            "type":  "boolean"
        }
    }
}
```

πρόκειται δηλαδή για ένα αντικείμενο με όνομα "one_object", τύπου "object", που περιέχει δύο πεδία: το "property_A" τύπου "string" και το "property_B" τύπου "boolean". Ένα παράδειγμα JSON το οποίο θα ήταν δεκτό με βάση το παραπάνω Schema είναι το εξής:

```
"one_object":{
    "property_A":  "some_string",
    "property_B":  false
}
```

Αντιθέτως, σε περίπτωση που δεν υπάρχει το property "properties" τότε το παραπάνω θα θεωρηθεί ως πεδίο εισαγωγής JSON. Το ίδιο θα συμβεί και σε περίπτωση που ένα property είναι "άδειο", δηλαδή ορίζεται ως: **{ }**

Κανόνας #5

Όλα τα αντικείμενα τύπου "array" πρέπει να περιλαμβάνουν ένα property "items".

Ο κανόνας αυτός αναφέρεται στους πίνακες/λίστες οι οποίοι στον ορισμό τους πρέπει να περιλαμβάνουν το property "items" το οποίο θα καθορίσει τα αντικείμενα που μπορούν να περιέχουν. Ακολουθούν δύο παραδείγματα:

- Παράδειγμα Α

```
"array_name":{
    "items":{
        "type":  "object",
        "properties":  {
            "property_A":  {
               "type":  "string"
            },
            "property_B":  {
               "type":  "boolean"
            }
        }
    }
}
```

Στο παράδειγμα Α βλέπουμε τον ορισμό ενός πίνακα με όνομα "array_name". Μέσα στο "items" ορίζονται τα αντικείμενα τα οποία θα περιέχει. Εν προκειμένω έχουμε ένα αντικείμενο ίδιο με αυτό του παραδείγματος του κανόνα #4. Ένα JSON το οποίο θα ήταν δεκτό με βάση το παραπάνω Schema είναι το εξής:

```
"array_name":  [
    {
      "property_A":  "some_string",
      "property_B":  false
    },
    {
      "property_A":  "some_other_string",
      "property_B":  true
    }
]
```

- Παράδειγμα Β

```
"array_name":{
    "items":{
        "type":  "string"
    }
}
```

Στο παράδειγμα Β ορίζεται ο πίνακας "array_name" ο ποίος μπορεί να περιέχει strings. Π.χ.

```
"array_name":  [
    "first_string",
    "second_string
]
```

Στη θέση του τύπου "string" θα μπορούσε να είναι κάθε άλλος βασικός τύπος. Στο εξής, αυτοί οι πίνακες/λίστες θα αναφέρονται ως πίνακες απλών πεδίων (simple field arrays).

Κανόνας #6

Όλα τα sections θεωρούνται ως απαραίτητα.

Όταν σε ένα JSON-schema πρέπει να οριστεί ένα αντικείμενο ως απαραίτητο, αυτό γίνεται προσθέτοντας το όνομά του μέσα στη λίστα "required" του αντικειμένου γονέα (parent element). Π.χ.

```
"one_object":{
    "type":  "object",
    "properties":  {
        "property_A":  {
            "type":  "string"
        },
        "property_B":  {
            "type":  "boolean"
        }
    }
    "required":  [
        "property_B"
    ]
}
```

Στο παραπάνω παράδειγμα, για το αντικείμενο "one_object", είναι απαραίτητο το πεδίο "property_B" ενώ το πεδίο "property_A" είναι προαιρετικό.

Sections θεωρούμε τα αντικείμενα που βρίσκονται στο αρχικό "properties" του input. Για πρακτικούς λόγους, όλα τα sections θεωρούνται απαραίτητα. Σε περίπτωση που ο χρήστης δεν επιθυμεί να συμπεριλαμβάνονται στο τελικό αποτέλεσμα δεν έχει παρά να τα αφαιρέσει από το JSON-schema που δίνεται ως είσοδος.

Κανόνας #7

Όλα τα Head Properties ενός section θεωρούνται ως απαραίτητα.

Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, Head Properties θεωρούμε τα αντικείμενα που βρίσκονται στο αρχικό "properties" ενός section. Ανά πάσα στιγμή κατά την εκτέλεση της εφαρμογής, μπορούν να φαίνονται το πολύ τα περιεχόμενα ενός μόνο Head Property. Υπάρχουν άλλωστε και στο αριστερό μέρος της οθόνης ως μέρη του υπο-μενού πλοήγησης σε κάθε section. Όπως και για τον προηγούμενο κανόνα, είναι αρκετά απίθανο να μην χρειάζονται στο αποτέλεσμα. Συνεπώς, για πρακτικούς λόγους θεωρούνται ως απαραίτητα από την εφαρμογή.

Κανόνας #8

Τα sections δεν μπορούν να είναι απλά πεδία. Πρέπει να έχουν είτε τύπο "object" με properties είτε τύπο "array".

Παρά το γεγονός ότι ένα section θα μπορούσε θεωρητικά να είναι ένα απλό πεδίο (π.χ. τύπου string), αυτό δεν έχει ιδιαίτερο νόημα καθώς ένα Artifact περιέχει τα

sections για να χωριστεί σε διαφορετικά μέρη. Για τον λόγο αυτό θεωρούμε ότι κάθε section μπορεί να είναι είτε αντικείμενο με properties είτε ένας πίνακας/λίστα.

Κανόνας #9

Υποστηρίζεται ο κανόνας "oneOf" ως property. Οι κανόνες. "allOf και "anyOf δεν υποστηρίζονται.

Στο παρακάτω παράδειγμα βλέπουμε έναν τυπικό τρόπο ορισμού ενός κανόνα "oneOf":

```
"one_object":{
    "type":  "object",
    "oneOf":  [
        {
            "properties":  {
                "property_A":  {
                    "type":  "string"
                }
            }
        },
        {
            "properties":  {
                "property_B":  {
                    "type":  "boolean"
                }
            }
        }
    ]
}
```

Στο παράδειγμα αυτό ορίζεται το αντικείμενο με όνομα "one_object" το οποίο μπορεί να έχει **μόνο μία** από τις δύο δομές που ορίζονται μέσα στον πινακα "oneOf". Π.χ. και τα δύο παρακάτω είναι δεκτά με βάση το προηγούμενο σχήμα.

```
1.
"one_object":{
    "property_A":  "some_string",
}
```

```
2.
"one_object":{
    "property_B":  false
}
```

Από τους τρεις κανόνες συνδυασμού σχημάτων ("oneOf", "allOf", "anyOf") επιλέχθηκε να υποστηρίζεται ο πρώτος. Αυτό έγινε καθότι θεωρήθηκε πιο χρήσιμος από τους υπόλοιπους. Τα JSONs που οι άλλοι δύο κανόνες ορίζουν μπορούν να περιγραφούν και με άλλο τρόπο χωρίς τη χρήση αυτών των κανόνων. Από την άλλη τα JSONs που ορίζει ένας κανόνας "oneOf" είναι αδύνατο να περιγραφούν διαφορετικά. Συνεπώς, απο τη στιγμή που κρίθηκε αναγκαίο να υποστηρίζεται τουλάχιστον ένας κανόνας συνδυασμού σχημάτων, επιλέχθηκε το "oneOf".

Κανόνας #10

Ο κανόνας "oneOf" πρέπει να περιέχει το "properties" (αν πρόκειται για αντικείμενα) ή το "items" (αν πρόκειται για πίνακες/λίστες) και όχι το ανάποδο.

Σύμφωνα με τον ορισμό του JSON-Schema, αντί για το παράδειγμα του κανόνα #9 θα μπορούσαμε να γράψουμε το εξής:

```
"one_object":{
    "type":  "object",
    "properties":  {
        "oneOf":[
            {
                "property_A":  {
                    "type":  "string"
                }
            },
            {
                "property_B":  {
                    "type":  "boolean"
                }
            }
        ]
    }
}
```

Το παραπάνω θα όριζε το ίδιο σύνολο JSONs με το παράδειγμα του κανόνα #9. Για πρακτικούς λόγους, όμως, λόγους επιλέχθηκε, χωρίς βλάβη της γενικότητας, να υποστηρίζεται ο πρώτος τρόπος (παράδειγμα κανόνα #9).

Κανόνας #11

'Ενα "oneOf" property δεν μπορεί να ορίζει διαφορετικούς τύπους για κάθε περίπτωσή του. Αυτό, επίσης, σημαίνει ότι δεν μπορεί να οριστεί μια περίπτωση ενός "oneOf" χωρίς να περιλαμβάνει ένα "properties" property με σκοπό να υπάρξει ένα απλό πεδίο εισαγωγής JSON.

Με τον παραπάνω κανόνα τονίζεται στον χρήστη ότι δεν μπορεί να ορίζει διαφορετικούς τύπους μέσα στον ίδιο κανόνα "oneOf". Για πρακτικούς λόγους, ο τύπος (αντικείμενο ή πίνακας/λίστα) ορίζεται στο ίδιο σημείο με τον κανόνα "oneOf" και μόνο μία φορά. Σε πρώτη ανάγνωση, το παραπάνω περιορίζει τις περιπτώσεις που μπορούν να οριστούν για τα JSONs. Αυτό, όμως δεν ισχύει καθώς θα μπορούσε να οριστεί ένα αντικείμενο με "oneOf" ένα βήμα “ψηλότερα” στην ιεραρχία του JSON-Schema το οποίο να περιλαμβάνει περιπτώσεις οι οποίες στο αμέσως επόμενο επίπεδο να ορίζουν το ίδιο πεδίο με διαφορετικό τύπο.

Το δεύτερο σκέλος του κανόνα αναφέρεται στην περίπτωση που ορίζεται ένα "oneOf" τύπου "object" αλλά σε μία από τις περιπτώσεις συνειδητά δεν υπάρχει "properties" property με σκοπό τον ορισμό ενός απλού πεδίου εισαγωγής JSON. Το προηγούμενο, για τους ίδιους λόγους που αναφέρθηκαν και προηγουμένως, επιλέχθηκε να μην υποστηρίζεται. Παρ' όλα αυτά μπορεί να χρησιμοποιηθεί και εδώ η λύση που προτείνεται στην προηγούμενη παράγραφο, ώστε να μην “αποκλείονται” ορισμένες περιπτώσεις για το αποτέλεσμα.

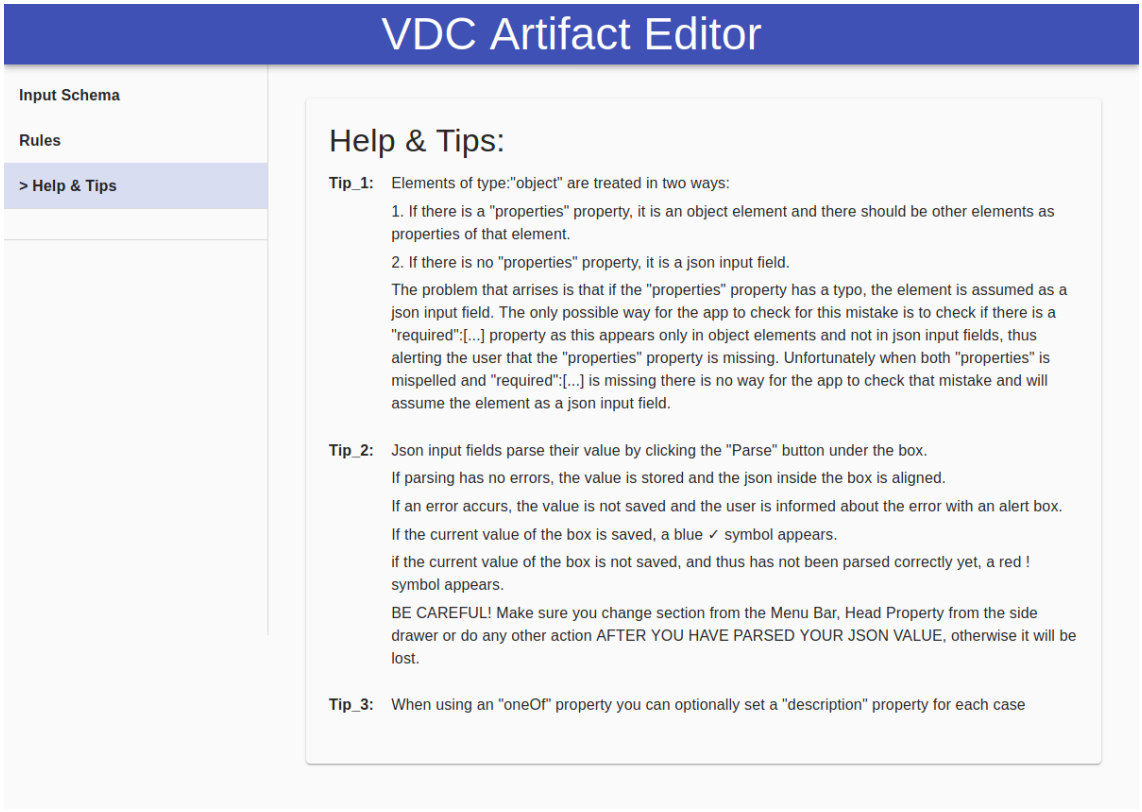
4.1.3 Help & Tips

Τρίτη και τελευταία καρτέλα της Αρχικής Οθόνης είναι το Help & Tips. Όπως βλέπουμε και στην εικόνα 4.3, εδώ περιλαμβάνονται ορισμένες συμβουλές για τη χρήση της εφαρμογής τις οποίες ο χρήστης θα ήταν καλό να γνωρίζει. Η μελέτη αυτών δίνει την ευκαιρία για περαιτέρω κατανόηση των μηχανισμών της εφαρμογής.

Συμβουλή #1

Στην παρούσα συμβουλή περιλαμβάνεται μία εξήγηση του τρόπου με τον οποίο η εφαρμογή θα προσπαθήσει να ειδοποιήσει το χρήστη σε περίπτωση που έχει κάνει ένα ορθογραφικό στον ορισμό του "properties" για ένα αντικείμενο τύπου "object". Σημειώνεται πως ο τρόπος διαχωρισμού από την εφαρμογή ενός αντικειμένου "object" με επιμέρους properties από ένα απλό πεδίο εισαγωγής JSON γίνεται

ελέγχοντας την ύπαρξη του παραπάνω property καθώς ο τύπος είναι και στις δύο περιπτώσεις "object". Σε περίπτωση όμως που ο χρήστης, ενώ θα επιθυμούσε να ορίσει ένα πλήρες αντικείμενο, έχει κάνει κάποιο ορθογραφικό λάθος στη λέξη "properties" τότε αυτό θα οδηγούσε στην δημιουργία ενός απλού πεδίου εισαγωγής JSON αγνοώντας όλα τα υπόλοιπα. Για το λόγο αυτό υπάρχει αυτόματα και ο επιπλέον έλεγχος για την ύπαρξη του property "required". Το παραπάνω property εμφανίζεται προαιρετικά μόνο στην περίπτωση ενός "object" αντικειμένου. Αυτό σημαίνει πως αν το παραπάνω έχει οριστεί, τότε πιθανότατα ο χρήστης έχει κάνει κάποιο ορθογραφικό λάθος στο "properties" ή το έχει παραλείψει εντελώς από λάθος. Στην περίπτωση αυτή ειδοποιείται για το παραπάνω με μήνυμα ώστε να κάνει τις απαραίτητες διορθώσεις.



Εικόνα 4.3: *Help & Tips: καρτέλα συμβουλών για την καλύτερη χρήση της εφαρμογής*

Συμβουλή #2

Όπως έχει ήδη αναφερθεί, κάθε αλλαγή που γίνεται στη φόρμα αποθηκεύεται αυτόματα ακόμα και αν πρόκειται για το πάτημα ενός πλήκτρου. Εξαίρεση σε αυτό αποτελούν τα πεδία εισαγωγής JSON. Τα προηγούμενα, όπως είναι λογικό, για να αποθηκεύσουν μία τιμή πρέπει πρώτα να την ¨αναλύσουν¨ μέσω της εντολής .parse( ). Αν το περιεχόμενο είναι ένα ορθώς συνταγμένο JSON τότε αυτό αποθηκεύεται, αλλιως εμφανίζονται μηνύματα που περιγράφουν στον χρήστη ποιο είναι το σφάλμα.

Σε περίπτωση που γινόταν απόπειρα αποθήκευσης για κάθε πλήκτρο που ο χρήστης πατούσε, αυτό θα κατέληγε σε αμέτρητα μηνύματα σφάλματος καθώς η φύση της σύνταξης ενός JSON προβλέπει ότι κατά την πληκτρολόγηση των πεδίων του δεν θα είναι απαραίτητα ορθό από άποψη σύνταξης καθώς δεν θα έχουν "κλείσει" όλες οι παρενθέσεις, αγκύλες κλπ. Συνεπώς, για τον παραπάνω λόγο τα πεδία αυτά διαθέτουν ένα πλήκτρο Parse (ΝΑ ΚΑΝΩ ΡΕΦ ΤΗΝ ΕΙΚΟΝΑ) το οποίο θα εκτελέσει την παραπάνω διαδικασία μόνο όταν πατηθεί από τον χρήστη, δηλαδή τη στιγμή που αυτός θεωρεί ότι το περιεχόμενο του πεδίου είναι "σωστό" από άποψη σύνταξης. Τέλος, στα δεξιά του πεδίου υπάρχει κατάλληλο σύμβολο που δείχνει κατά πόσον η τρέχουσα τιμή είναι αποθηκευμένη ή όχι. Χρειάζεται ιδιαίτερη προσοχή από τον χρήστη ώστε να μην αλλάξει καρτέλα ή section πριν αποθηκεύσει το περιεχόμενο του πεδίου γιατί αλλιώς αυτό θα χαθεί.

Συμβουλή #3

Η τελευταία συμβουλή υπενθυμίζει στον χρήστη πως μπορεί προαιρετικά κατά τη χρήση του κανόνα "oneOf" να ορίσει ένα "description" property για κάθε περίπτωση. Η εφαρμογή θα φροντίσει αυτό να εμφανίζεται κατά την επιλογή κάθε περίπτωσης. Ακολουθεί παράδειγμα ορισμού "description" για κάθε περίπτωση ενός "oneOf":

```
"one_object":{
    "type":  "object",
    "oneOf":  [
        {
            "properties":  {
                "property_A":  {
                    "type":  "string"
                },
            "description":  "The description of the first case"
            }
        },
        {
            "properties":  {
                "property_B":  {
                    "type":  "boolean"
                }
            },
            "description":  "The description of the second case"
        }
    ]
}
```

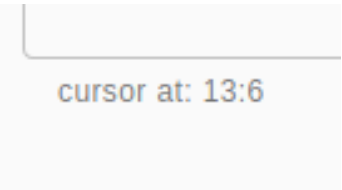
4.2 Έλεγχος εισόδου

Όταν η εισαγωγή του JSON-Schema εισόδου έχει ολοκληρωθεί, ο χρήστης πατάει το πλήκτρο "Generate" για την δυναμική παραγωγή της εικονικής αναπαράστασης του προς παραγωγή Artifact. Αυτόματα ξεκινά η διαδικασία ελέγχου της εισόδου. Αυτή μπορεί να χωριστεί σε δύο μέρη:

1. Έλεγχος κανόνων ορθής σύνταξης ενός JSON.
2. Έλεγχος για την ικανοποίηση των κανόνων που θεσπίστηκαν για την εφαρμογή αυτή και οι οποίοι περιγράφηκαν στην υποενότητα 4.1.2

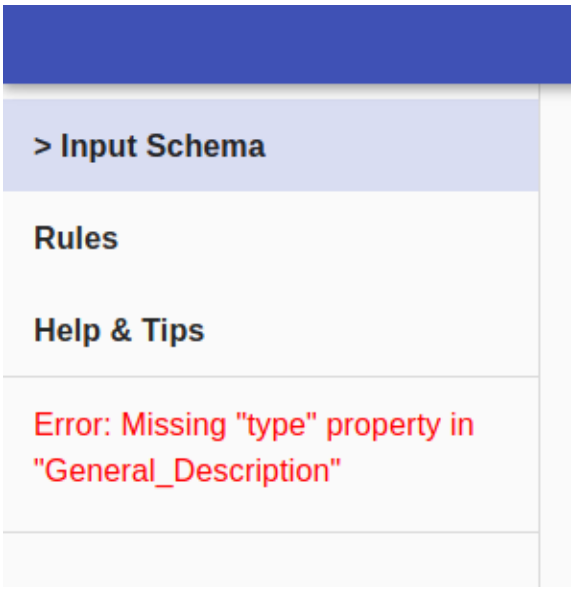
1. Έλεγχος κανόνων ορθής σύνταξης ενός JSON

Με χρήση της εντολής .parse( ) γίνεται η μετατροπή της συμβολοσειράς εισόδου σε ένα αντικείμενο JSON διαχειρίσιμο από την εφαρμογή. Σε περίπτωση ύπαρξης σφαλμάτων, αυτόματα εμφανίζονται κατάλληλα μηνύματα τα οποία ενημερώνουν τον χρήστη για συντακτικά λάθη στο JSON όπως π.χ. “ανοικτές” αγκύλες “ { ”, έλλειψη του χαρακτήρα κόμμα “,” μεταξύ των property ενός αντικειμένου κλπ. Στο μήνυμα υπάρχουν συνήθως οι συντεταγμένες όπου βρίσκεται το λάθος. Στο κάτω-αριστερό μέρος του πεδίου εισόδου υπάρχει ανά πάσα στιγμή η θέση του κέρσορα σε συντεταγμένες ώστε να είναι εύκολο για τον χρήστη βρει και να διορθώσει το λάθος.



Εικόνα 4.4: *Συντεταγμένες θέσης κέρσορα*

Όπως βλέπουμε στην εικόνα 4.5 το μήνυμα εξακολουθεί να φαίνεται στο αριστερό μέρος της οθόνης μέχρι να πατηθεί ξανά το πλήκτρο "Generate" και αυτό για να μπορεί ο χρήστης να ανατρέξει σε αυτό σε περίπτωση που δεν το συγκράτησε όταν εμφανίστηκε ως μήνυμα παραθύρου.



Εικόνα 4.5: *Μήνυμα σφάλματος*

2. Έλεγχος κανόνων εφαρμογής

Αν η είσοδος είναι ένα σωστό συντακτικά JSON η εφαρμογή περνάει στην επόμενη διαδικασία. Ξεκινά τη δυναμική παραγωγή του Artifact (η οποία αναλύεται σε βάθος στην επόμενη υποενότητα) και ταυτόχρονα σε κάθε βήμα της ελέγχει του κανόνες που έχουν περιγραφεί προηγουμένως. Αν εντοπιστεί κάποιο σφάλμα, ο χρήστης ενημερώνεται ακριβώς όπως και πριν ώστε να κάνει τις απαραίτητες διορθώσεις.

Τονίζεται για ακόμη μία φορά ότι οι έλεγχοι είναι εξονυχιστικοί ώστε τα πιθανά σφάλματα της εισόδου να εντοπιστούν κατά την παραγωγή και όχι κατά τη διάρκεια της εισαγωγής των πληροφοριών. Αυτό εξασφαλίζει σε μεγάλο βαθμό την αποφυγή απώλειας πληροφοριών από σοβαρό σφάλμα που ενδεχομένως θα γινόταν αντιληπτό αργότερα. Όπως έχει ήδη γίνει γνωστό, ακόμα και στην ιδιαίτερα σπάνια περίπτωση που θα συνέβαινε κάτι τέτοιο, δηλαδή να υπάρξει σφάλμα για το οποίο δεν υπάρχει πρόβλεψη, επιστρατεύεται η λύση της αυτόματης αφαίρεσης μόνο του "προβληματικού" μέρους.

4.3 Δυναμική δημιουργία του Artifact

Στην ενότητα αυτή περιγράφεται αναλυτικά ο τρόπος με τον οποίο δημιουργείται το Artifact στα δεδομένα της εφαρμογής καθώς επίσης και ο τρόπος με τον οποίο αυτό αναπαρίσταται οπτικά στον χρήστη. Σημειώνεται αρχικά ότι τα αντικείμενα της React χρησιμοποιούν το state για να αποθηκεύουν δεδομένα τα οποία αλλάζουν μέσω της συνάρτησης setState( ). Αυτό συμβαίνει ώστε να μπορεί η εφαρμογή να κρίνει κάθε στιγμή τον τρόπο με τον οποίο θα αλλάξει τα δεδομένα και να αυξήσει

την απόδοσή της. Το state ενός αντικειμένου μπορεί να μελετηθεί σαν ένα JSON το οποίο εν προκειμένω είναι ιδιαίτερα βοηθητικό για την εξέλιξη της εφαρμογής.

4.3.1 Ο πίνακας masterStates[ ]

Μέσα στο state του αντικειμένου Home, που αποτελεί τη βάση της εφαρμογής, ορίζεται ο πίνακας masterStates. Κάθε στοιχείο του αντιστοιχεί στο state ενός section. Αρχικά, ανάλογα με τον αριθμό των sections της εισόδου, γεμίζει με τον ίδιο αριθμό από στοιχεία τα οποία αρχικά έχουν μία γενική προεπιλεγμένη τιμή όσων αφορά στα properties τους. Τα δύο σημαντικότερα από αυτά είναι το output και το info

4.3.2 "output" & "info"

Τα δύο αυτά properties περιέχουν την πλειοψηφία των πληροφορίων για ένα section. Ουσιαστικά, πρόκειται για δύο δομές JSON εκ των οποίο το output περιέχει το αποτέλεσμα που στο τέλος θα παραχθεί σαν έξοδος, ενώ το info είναι παράλληλο στο output αλλά αντί για τις τιμές εξόδου περιέχει πληροφορίες σχετικά με το κάθε πεδίο/αντικείμενο οι οποίες επιτρέπουν στον editor να το χειριστεί ανάλογα. Π.χ.

```
output:{
    "some_name":  "a_value"
}

info:{
    "some_name":{
        "type":  "string"
    }
}
```

Στο παραπάνω παράδειγμα παρατηρούμε ότι στο output υπάρχει το πεδίο με όνομα "some_name" και τιμή "a_value", ενώ στο info έχουμε την πληροφορία ότι το πεδίο αυτό είναι τύπου "string". Για κάθε αντικείμενο στο output υπάρχει πάντα το αντίστοιχο αντικείμενο στο info που περιέχει τις απαραίτητες πληροφορίες.

Μετά από το παραπάνω, εγείρεται η απορία του γιατί δεν γίνεται η αποθήκευση των πληροφοριών μέσα στο ίδιο το πεδίο. Μπορούμε, όμως, να δούμε με ευκολία ότι ο διαχωρισμός της εξόδου από τις πληροφορίες που τη διέπουν προσφέρει αρκετά πλεονεκτήματα. Βασικότερο αυτών είναι ότι ανά πάσα στιγμή υπάρχει έτοιμη η έξοδος και δεν χρειάζεται καμία επεξεργασία για να δοθεί στον χρήστη. Επιπλέον, γίνεται ξεκάθαρο για το σύστημα το πού θα βρίσκει τις τιμές των πεδίων και που θα βρίσκει τις πληροφορίες για αυτά όταν αναζητεί κάτι από τα δύο.

Μία ακόμα εύλογη απορία είναι το γιατί ακολουθήθηκε η παραπάνω προσέγγιση και δεν χρησιμοποιήθηκε ως info το ίδιο το input καθότι και αυτό στην ουσία περιγράφει το output. Αυτό εκ πρώτης όψεως θα ήταν μια καλή ιδέα. Παρόλα αυτά, όσο η εφαρμογή γίνεται πιο πολύπλοκη και εμπλουτίζονται οι δυνατότητές της, κάτι τέτοιο θα δυσκόλευε την υλοποίηση. Πολύπλοκες δομές απαιτούν πολλές πληροφορίες οι οποίες θα οριστούν από την εφαρμογή για την υποστήριξη πιο εξειδικευμένων περιπτώσεων από αυτή του παραδείγματος. Επιπλέον, η δημιουργία του info γίνεται με έναν γενικότερο τρόπο, από ότι ενδεχομένως το JSON-Schema, ώστε να μην χρειάζεται να ορίζονται αμέτρητες ειδικές περιπτώσεις για την υποστήριξη όλων των εισόδων. Συνεπώς, η ανάγκη αποθήκευσης περισσότερων πληροφοριών και η γενικότερος τρόπος δόμησης οδήγησαν στην επιλογή της παραπάνω προσέγγισης.

4.3.3 Δημιουργία ενός section

Όταν ο editor προσπαθεί να εμφανίσει ένα section (αρχείο Section.js) καλεί αρχικά τη συνάρτηση makeTheState( ). Αυτή πρώτα από όλα ελέγχει αν είναι η πρώτη φορά που ο χρήστης "άνοιξε" αυτό σε section μέσω του propeperty "isSet". Αν το παραπάνω έχει τιμή false τότε ξεκινάει η διαδικασία δημιουργίας.

Πρώτη ενέργεια είναι ο έλεγχος του τύπου του section. Όπως αναφέραμε και προηγουμένως στους κανόνες, αυτό μπορεί να είναι είτε "object" είτε "array". Θα ξεκινήσουμε με την περιγραφή της δημιουργίας για τύπο "object".

Object Section

Για κάθε ένα από τα Head Properties δημιουργείται μία εισαγωγή στο output και μία στο info. Αν αυτά είναι τύπου "array" τότε προστίθεται και το όνομα του Head Property στο "info_Arrays". Στη συνέχεια καλείται η αναδρομική συνάρτηση returnFields( ) με ορίσμα το "properties" αν πρόκειται για object ή το "items.properties" αν πρόκειται για array head property (εφόσον ορίζεται στο input, δίνεται ως όρισμα και η λίστα "required"). Η returnFields( ) επιστρέφει κάθε φορά ένα στοιχείο για το output και ένα για το info με βάση το όρισμα που της έχει δοθεί. Αυτό το κάνει ως εξής:

- Για κάθε ένα από τα properties που της δόθηκαν, ελέγχει τον τύπο του
- Αν ο τύπος είναι απλού πεδίου (string, number κλπ) τότε επιστέφει μία προεπιλεγμένη αρχική τιμή για το πεδίο και τις αντίστοιχες πληροφορίες για αυτό.
- Αν πρόκειται για object τότε καλεί αναδρομικά τον εαυτό της για κάθε ένα από τα properties του object, συλλέγει τα αποτελέσματα και τα επιστρέφει μαζί με κατάλληλες πληροφορίες για το object αυτό.

- Ομοίως, αν πρόκειται για array τότε καλεί αναδρομικά τον εαυτό της για κάθε ένα από τα items.properties του array, συλλέγει τα αποτελέσματα και τα επιστρέφει μαζί με κατάλληλες πληροφορίες για το array αυτό.

Απλά πεδία

Αναλυτικότερα τώρα, για τα απλά πεδία οι προεπιλεγμένες τιμές είναι οι εξής:

- string: "" (κενή συμβολοσειρά)
- number: 0
- object(πεδίο): { }
- boolean: false
- enum: (η πρώτη από τις) επιλογές

Για κάθε ένα από τα παραπάνω επιστρέφονται οι εξής πληροφορίες στο info:

- "type":(string/number/object/boolean/enum) ανάλογα με τον τύπο του πεδίου.
- "req":(true/false) από το required, δηλαδή αν το πεδίο είναι αναγκαίο.
- "desc": από το description, δηλαδή η περιγραφή του πεδίου (εφόσον υπάρχει).
- "pattern","format": κανόνας που πρέπει διέπει ένα string (εφόσον υπάρχει).
- "values": περιέχει τις δυνατές τιμές (για enum)

Αντικείμενα object

Όταν έχουμε ένα αντικείμενο object (και όχι πεδίο object) οι πληροφορίες για κάθε ένα από τα properties του περιλαμβάνονται μέσα στο "props". Προφανώς έχουμε "type":"object". Υπάρχει, επίσης, το "req" και μπορεί να επιστραφεί και "desc" για το ίδιο το αντικείμενο εφόσον υπάρχει.

Arrays

Εδώ οι πληροφορίες που επιστρέφονται είναι οι εξής:

- "items": περιέχει τις πληροφορίες για τα αντικείμενα που αυτός ο πίνακας/λίστα μπορεί να περιλαμβάνει.
- "items_template": ένα πρότυπο των αντικειμένων του πίνακα (χρησιμοποιείται για την εισαγωγή νέου αντικειμένου)

- "minItems": ελάχιστος επιτρεπτός αριθμός στοιχείων (εφόσον ορίζεται στο input)
- "req": (ομοίως με πριν)
- "desc": (ομοίως με πριν)

Σημειώνεται πως σε ό,τι αφορά το output οι πίνακες/λίστες ως προεπιλογή έχουν ένα στοιχείο με τα properties του οποίου έχουν τις προεπιλεγμένες τους τιμές. Επίσης, σημειώνεται ότι στην περίπτωση που το έχουμε array head property αποθηκεύεται και ένα πρότυπο αντικειμένου για το ίδιο το head property στο "head_Templates" του state. Τέλος, το property "isSet" τίθεται ως true.

Array Section

Στην περίπτωση που το section είναι τύπου "array" ακολουθούμε την ίδια διαδικασία για τα στοιχεία που θα περιέχει. Επιπλέον, κρατάμε όπως είναι λογικό ένα πρότυπο στο "template" του state.

4.3.4 Ο κανόνας "oneOf"

Στην προηγούμενη ενότητα, συνειδητά δεν έγινε αναφορά στον κανόνα "oneOf" ώστε να είναι ευκολότερη η κατανόηση της δημιουργίας ενός section. Τώρα, όμως, που η γενική περίπτωση έχει αναλυθεί, μπορεί να γίνει ανάλυση της δημιουργίας αντικειμένων που περιέχουν τον κανόνα αυτό.

Όπως τονίσθηκε σε προηγούμενες ενότητες, ο κανόνας "oneOf" επιτρέπει την επιλογή μίας εκδοχής για το αποτέλεσμα μεταξύ δύο ή περισσοτέρων περιπτώσεων. Γνωρίζουμε από τους κανόνες της ενότητας 4.1.2 ότι υποστηρίζεται η εφαρμογή του κανόνα αυτού μόνο σε αντικείμενα τύπου object και σε πίνακες/λίστες.

"oneOf" σε objects

Όταν η προαναφερθείσα συνάρτηση returnFields( ) αναγνωρίσει τον ορισμό ενός τέτοιου κανόνα κατά τη δημιουργία ενός object. Πραγματοποιεί τα ακόλουθα:

- Εκτελεί για κάθε μία περίπτωση τη διαδικασία που περιγράφηκε στην προηγούμενη ενότητα.
- Τοποθετεί τα αποτελέσματα για το output στο property/λίστα "casesOut"
- Τοποθετεί τα αποτελέσματα για το info στο property/λίστα "casesInfo"
- Σε περίπτωση που υπάρχουν descriptions για κάθε περίπτωση, τα τοποθετεί στο property/λίστα "casesDesc". Αν για κάποια περίπτωση δεν ορίζεται property "description" τότε στη λίστα τοποθετείται η συμβολοσειρά " " (ένας χαρακτήρας κενού).

- Δημιουργείται το property "_current_case" (τρέχουσα περίπτωση) με προεπιλεγμένη τιμή το 0. Το συγκεκριμένο θα χρησιμοποιηθεί στη συνέχεια για την αποθήκευση της επιλογής περίπτωσης που θα πραγματοποιήσει ο χρήστης.

- Ως επιστροφή για το output δίνεται η πρώτη περίπτωση, καθώς η προεπιλογή για το "_current_case" είναι 0 (η αρίθμηση των λιστών/πινάκων στη JavaScript ξεκινά από το 0).

- Τέλος, ορίζεται ο τύπος του αντικειμένου ως "type": "object_oneOf"

Μετά την παραπάνω διαδικασία έχουμε στην κατοχή μας ένα ευέλικτο αντικείμενο το οποίο μπορεί να προσφέρει την επιλογή για το ποιά θα είναι τα properties που θα περιλαμβάνει. Κάθε φορά, προεπιλέγεται η πρώτη περίπτωση αλλά στην πορεία ο χρήστης θα μπορεί μέσα από την εφαρμογή να την αλλάξει.

"oneOf" σε arrays

Αν προκύψει "oneOf" σε array τότε η συνάρτηση returnFields( ) ακολουθεί όπως και πριν την παραπάνω διαδικασία. Επιπλέον δημιουργεί το "casesTemplates", ένα property/λίστα που περιλαμβάνει τα πρότυπα των αντικειμένων του πίνακα για κάθε περίπτωση. Τέλος, ο τύπος του πίνακα ορίζεται ως "type": "array_oneOf"

Μετά την εκτέλεση των παραπάνω έχουμε και πάλι ένα ευέλικτο αντικείμενο πίνακα το οποίο έχει προεπιλεγμένη την πρώτη περίπτωση. Ο χρήστης μπορεί στη συνέχεια να αλλάξει αυτή την επιλογή.

4.3.5 Το ζήτημα του κανόνα "oneOf" μέσα σε πίνακες

Στην προηγούμενη ενότητα περιγράφηκε η δημιουργία αντικειμένων object και array τα οποία υποστηρίζουν την επιλογή μεταξύ διαφορετικών περιπτώσεων (κανόνας "oneOf"). Όταν, στη γενική περίπτωση, εμφανιστεί ένα τέτοιο αντικείμενο, είναι μοναδικό. Αυτό σημαίνει πως η επιλογή περίπτωσης για αυτό δεν επηρεάζει κάποιο άλλο αντικείμενο. Τι θα συνέβαινε, όμως, αν ένα τέτοιο αντικείμενο (είτε array_oneOf είτε object_oneOf εμφανιζόταν μέσα σε έναν πίνακα/λίστα; Ποια περίπτωση θα επιλεγόταν; Μία για όλα ή το κάθε αντικείμενο ξεχωριστά; Θεωρήθηκε ορθότερο να υποστηρίζεται η επιλογή σε κάθε αντικείμενο ξεχωριστά ώστε να προσφέρεται στον χρήστη ακόμα μεγαλύτερη ευελιξία. Η υλοποίηση, όμως, αυτού δεν είναι καθόλου τετριμμένη. Το πρόβλημα που υπάρχει μπορεί να μην είναι εμφανές εξαρχής. Για το λόγο αυτό θα γίνει μία προσπάθεια παρουσίασής του και στη συνέχεια θα αναλυθεί η λύση που επιλέχθηκε.

Όταν ένα αντικείμενο (object ή array) έχει oneOf τότε η εφαρμογή ελέγχει το property "_current_case" ώστε να επιλέξει την κατάλληλη περίπτωση από τις διάφορες λίστες ("casesOut", "casesInfo" κλπ). Το πρόβλημα που προκύπτει είναι όμως

ότι το info θα είναι κοινό για όλα τα αντικείμενα ενός πίνακα. Συνεπώς, δημιουργείται η εύλογη απορία: το "_current_case" σε ποιο αντικείμενο αναφέρεται; Είναι προφανές ότι θα πρέπει να αποθηκεύεται η περίπτωση για κάθε ένα αντικείμενο. Αν, όμως, στη "διαδρομή" (path) μεσολαβούν δύο ή περισσότεροι πίνακες;

Για τη λύση του παραπάνω προβλήματος επιστρατεύτηκε η παρακάτω μέθοδος. Κατά τη δημιουργία των αντικειμένων/πεδίων συγκρατείται ένας μετρητής πινάκων που μεσολαβούν από την αρχή της "διαδρομής" μέχρι το αντικείμενο. Αν προκύψει oneOf τότε αυτός ο μετρητής αποθηκεύεται ως το property "casesNOA" (προκύπτει από το cases Number Of Arrays). Επίσης δημιουργείται το property "casesArrayItems". Το τελευταίο πρόκειται για μία λίστα από λίστες με "βάθος" όσο η τιμή του "casesNOA". Ο τρόπος με τον οποίο αποθηκεύεται η περίπτωση ενός αντικειμένου μέσα σε αυτόν είναι ο εξής. Για κάθε πίνακα που μεσολαβεί στο path κρατάμε τον αύξοντα αριθμό αντικειμένου (index). Με βάση το πρώτο index επιλέγουμε λίστα στο πρώτο επίπεδο, με βάση το δεύτερο στο δεύτερο κ.ο.κ. Το τελευταίο από τα indexes που έχουμε θα πρέπει να αντιστοιχεί σε ένα στοιχείο της τελευταίας σε "ιεραρχία" λίστας. Ας δούμε ένα παράδειγμα:

```
"casesArrayItems":[
    [
        [
           1,       <--[0,0,0]
           2,       <--[0,0,1]
           3,       <--[0,0,2]
        ],
        [
           4        <--[0,1,0]
        ]
    ],
    [
        [
           5        <--[1,0,0]
        ],
        [],
        [
           6,       <--[1,2,0]
           7        <--[1,2,1]
        ]
    ]
]
```

Στο παραπάνω παράδειγμα φαίνονται επτά θέσεις αποθήκευσης όταν η τιμή του "casesNOA": είναι 3. Στα δεξιά τους ο συμβολισμός <--[x,y,z] δείχνει για κάθε θέση τα indexes που την προσδιορίζουν. Όταν υπάρξει η ανάγκη για ανάκτηση πληροφορίας θα αναζητηθεί με βάση τα indexes που υπάρχουν στο path του εκάστοτε αντικειμένου. Όπως και πριν, η προεπιλεγμένη τιμή είναι 0, δηλαδή η πρώτη περίπτωση.

4.3.6 "oneOf" στο section

Μία ακόμα δυνατότητα που υποστηρίζεται από τον Editor είναι η εφαρμογή του κανόνα "oneOf" σε ένα section. Η μέθοδος υλοποίησης είναι παρόμοια αυτές που αναφέρθηκαν προηγουμένως. Στην ουσία, δημιουργούνται όλα τα sections που ορίζονται στις περιπτώσεις του "oneOf" (output & info). Ορίζεται ως προεπιλογή η πρώτη περίπτωση και δημιουργείται το property "current_oneOf" το οποίο ορίζει ποια είναι η τρέχουσα.

4.4 Γραφική αναπαράσταση του Artifact

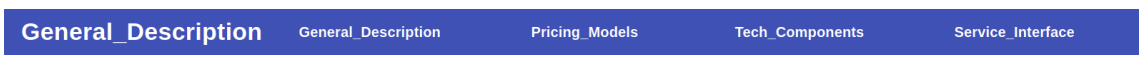
Η διαδικασία που περιγράφηκε προηγουμένως έθεσε όλες τις απαραίτητες βάσεις ώστε να είναι εφικτή η γραφική/εικονική αναπαράσταση του Artifact. Σε αυτή την ενότητα θα αναλυθεί ο τρόπος με τον οποίο το σύστημα εμφανίζει στον χρήστη όλα τα πεδία και τα αντικείμενα.

Έχουμε ήδη αναφέρει ότι ο χρήσης, ανά πάσα στιγμή, μπορεί να βλέπει στο κεντρικό μέρος της οθόνης τα περιεχόμενα ενός head property του τρέχοντος section. Στην κορυφή και στην αριστερή πλευρά υπάρχουν τα μενού πλοήγησης.

4.4.1 Μενού και υπο-μενού πλοήγησης

Κυρίως Μενού

Το Κυρίως Μενού (αρχείο MenuBar.js) εμφανίζεται με τη μορφή μίας μόνιμης οριζόντιας μπάρας στην κορυφή της οθόνης η οποία περιλαμβάνει όλα τα sections που έχουν οριστεί (εικόνα 4.6). Στο αριστερό της μέρος φαίνεται πιο είναι το τρέχον section του οποίου τα περιεχόμενα προβάλλονται αυτή τη στιγμή.



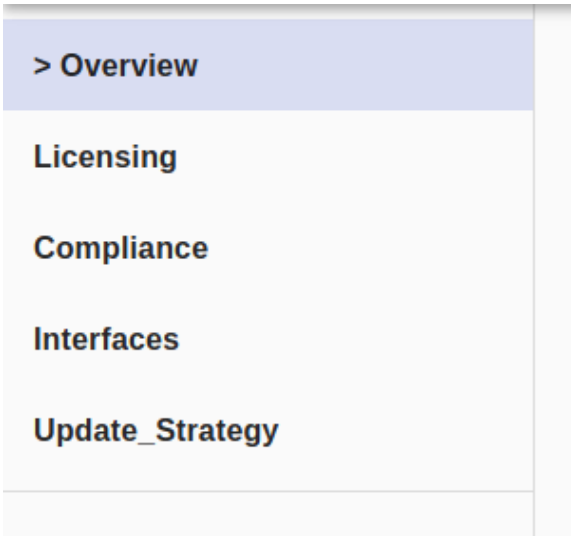
Εικόνα 4.6: *Παράδειγμα Κυρίως Μενού με τέσσερα sections*

Η παραπάνω μπάρα, όπως αναφέρθηκε, είναι σταθερή σε σχέση με την οθόνη, δηλαδή παραμένει παραμένει πάντα στην κορυφή της ακόμα και αν ο χρήστης έχει μετακινηθεί πάνω ή κάτω στη σελίδα (scroll). Ο χρήστης, κάνοντας κλικ μπορεί να

επιλέξει να εμφανιστεί ένα από τα άλλα sections. Μόλις μία επιλογή πατηθεί, αυτόματα αποθηκεύεται το τρέχον state του section στη γενική λίστα masterStates και λαμβάνεται από την ίδια λίστα το state του section που ζητήθηκε ώστε να εμφανιστεί.

Μενού Head Properties

Στο αριστερό μέρος της οθόνης, υπάρχει πάντα ένα υπο-μενού (αρχεία MyDrawer.js & MyDrawerUpper.js) το οποίο περιλαμβάνει τα head properties του τρέχοντος section (εικόνα 4.7. Κάθε φορά μπορεί να είναι επιλεγμένο μόνο ένα από αυτά, ώστε να εμφανίζονται τα περιεχόμενά του. Η επιλογή φαίνεται από ένα αχνό γαλάζιο χρώμα φόντου και τον χαρακτήρα ">" πριν από το όνομα του head property.



Εικόνα 4.7: *Παράδειγμα Μενού με πέντε Head Properties*

Όπως και με το κυρίως μενού, έτσι και εδώ το μενού αυτό έχει σταθερή θέση σε σχέση με την οθόνη. Ανάλογα με το ποιο head property είναι επιλεγμένο κάθε φορά δίνονται τα κατάλληλα δεδομένα που πρέπει να εμφανιστούν.

Σημειώνεται πως στην περίπτωση που έχουμε array section το υπο-μενού αυτό εμφανίζει τα αντικείμενα που περιλαμβάνει ο πίνακας (αρχείο MyArrayDrawer.js)

4.4.2 Εμφάνιση περιεχομένων ενός Head Property

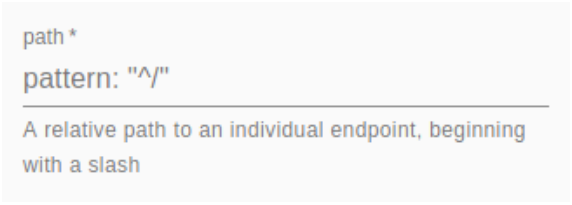
Υπεύθυνο για την εμφάνιση των περιεχομένων ενός head property είναι το ItemMasterComponent (Αρχείο ItemMasterComponent.js) Σε αυτό δίνονται ως παράμετροι τα output και info που αφορούν το συγκεκριμένο head property. Το component με τη σειρά του εφμανίζει αρχικά τον τίτλο του section και του head property και στη συνέχεια εξετάζει ένα-ένα τα πεδία/αντικείμενα που υπάρχουν στο output και σε συνδιασμό με τις πληροφορίες που υπάρχουν στο info και τα εμφανίζει κατάλληλα.

Εμφάνιση απλών πεδίων

Όταν πρόκειται για απλό πεδίο η εφαρμογή καλεί/δημιουργεί, ανάλογα με τον τύπο του, το κατάλληλο component και του δίνει ως παραμέτρους την τιμή που περιέχει και τις απαραίτητες πληροφορίες που το αφορούν ώστε αυτό να εμφανιστεί (όπως "req", "desc" κλπ). Παρακάτω βλέπουμε για κάθε τύπο πεδίου ποιο component καλείται/δημιουργείται.

- **string**: ItemString (εικόνα 4.8) ή ItemPrefixString εφόσον έχει οριστεί pattern ή format για το string (εικόνα 4.9)

Εικόνα 4.8: *Πεδίο τύπου "string" με περιγραφή*



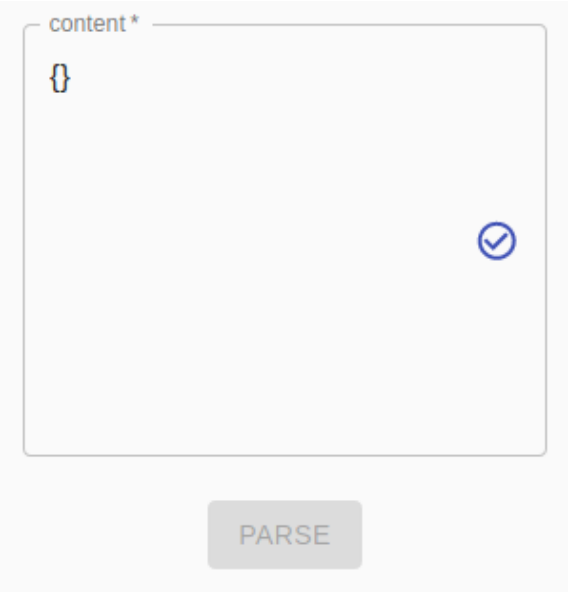
Εικόνα 4.9: *Πεδίο τύπου "string" με περιγραφή και ορισμένο pattern*

- **number**: ItemNumber (εικόνα 4.10)

Εικόνα 4.10: *Πεδίο τύπου "number"*

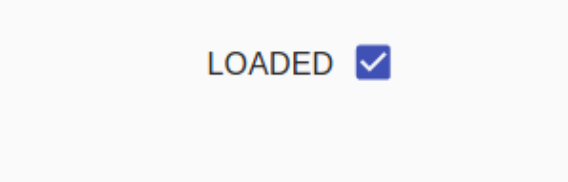
- **object**: ItemObject

  Στο παράδειγμα (εικόνα 4.11) το πλήκτρο "Parse" είναι απενεργοποιημένο καθώς η τρέχουσα τιμή (“{ }”) είναι είναι ήδη αποθηκευμένη, όπως φαίνεται άλλωστε και από το μπλε σύμβολο “τικ” στα δεξιά του πεδίου.



Εικόνα 4.11: *Πεδίο τύπου "object"*

- **boolean**: ItemBoolean (εικόνα 4.12)



Εικόνα 4.12: *Πεδίο τύπου "boolean" με τιμή true*

- **enum**: ItemEnum (εικόνα 4.13) ή ItemConst αν έχει δοθεί μόνο μία επιλογή, δηλαδή πρόκειται για σταθερά (εικόνα 4.14)

Εικόνα 4.13: *Πεδίο τύπου "enum"*

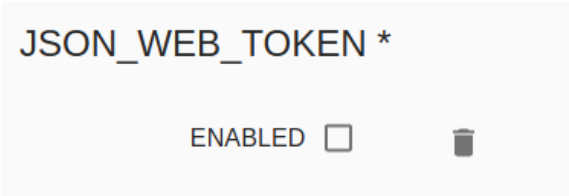
Εικόνα 4.14: *Πεδίο τύπου "const" με σταθερή τιμή false*

Εικόνα 4.15: *Πεδίο τύπου "enum" με επιλογή "other"*

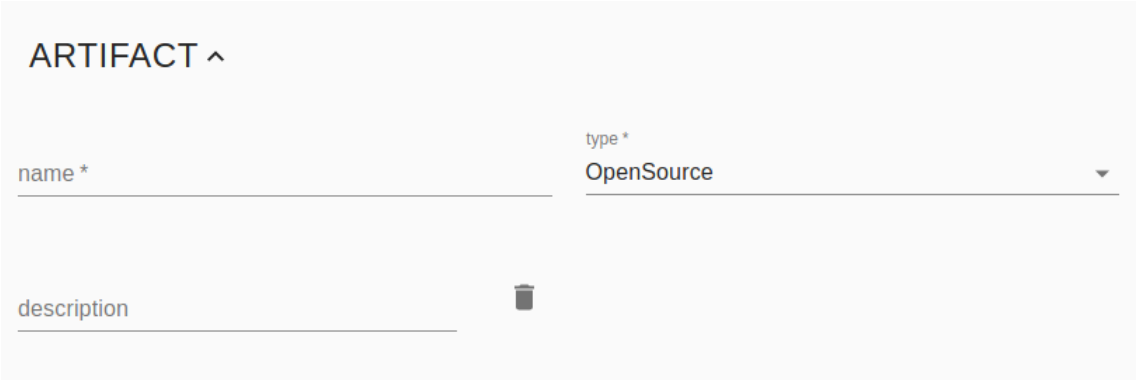
Στο παράδειγμα της εικόνας 4.15 βλέπουμε ένα πεδίο enum στο οποίο έχει επιλεχθεί η τιμή "other. Ο editor υποστηρίζει, σε περίπτωση που υπάρχει επιλογή "other" ή "Other" την εμφάνιση επιπλέον πεδίου συμβολοσειράς ώστε ο χρήστης να πληκτρολογήσει την τιμή που επιθυμεί.

Εμφάνιση στοιχείων "object"

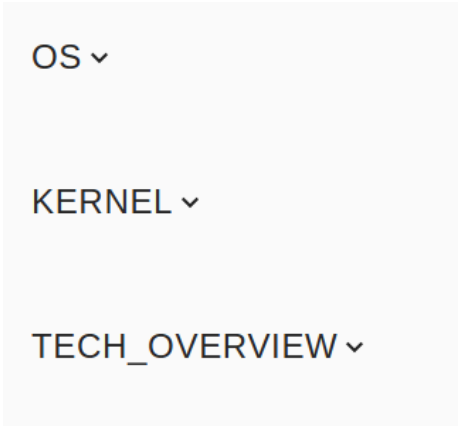
Όταν το αντικείμενο προς εμφάνιση είναι τύπου "object", αυτό θα εμφανιστεί με τον εξής τρόπο (εικόνα 4.16). Αρχικά εμφανίζεται το όνομά του. Ακριβώς από κάτω εμφανίζονται ένα-ενα τα επιμέρους πεδία ή αντικείμενα που περιέχει ακολουθώντας την ίδια αναδρομική διαδικασία. Αν πρόκειται για object που βρίσκεται στο πρώτο επίπεδο της ιεραρχίας, δηλαδή είναι είναι ακριβώς κάτω από το head property, δίπλα σε αυτό το όνομα εμφανίζεται ένα βέλος. Στην περίπτωση αυτή (εικόνες 4.17 & 4.18) όλο το όνομα λειτουργεί σαν πλήκτρο το οποίο "αναδιπλώνει" ή "μαζεύει"/"κρύβει" τα περιεχόμενα του αντικειμένου. Αυτό συμβαίνει ώστε να μην κατακλύζεται ο χρήσης από μετρητά πεδία και να μπορεί να επικεντρώνεται μόνο στο σημείο που επιθυμεί.



Εικόνα 4.16: *Αντικείμενο "object" με το περιεχόμενό του*



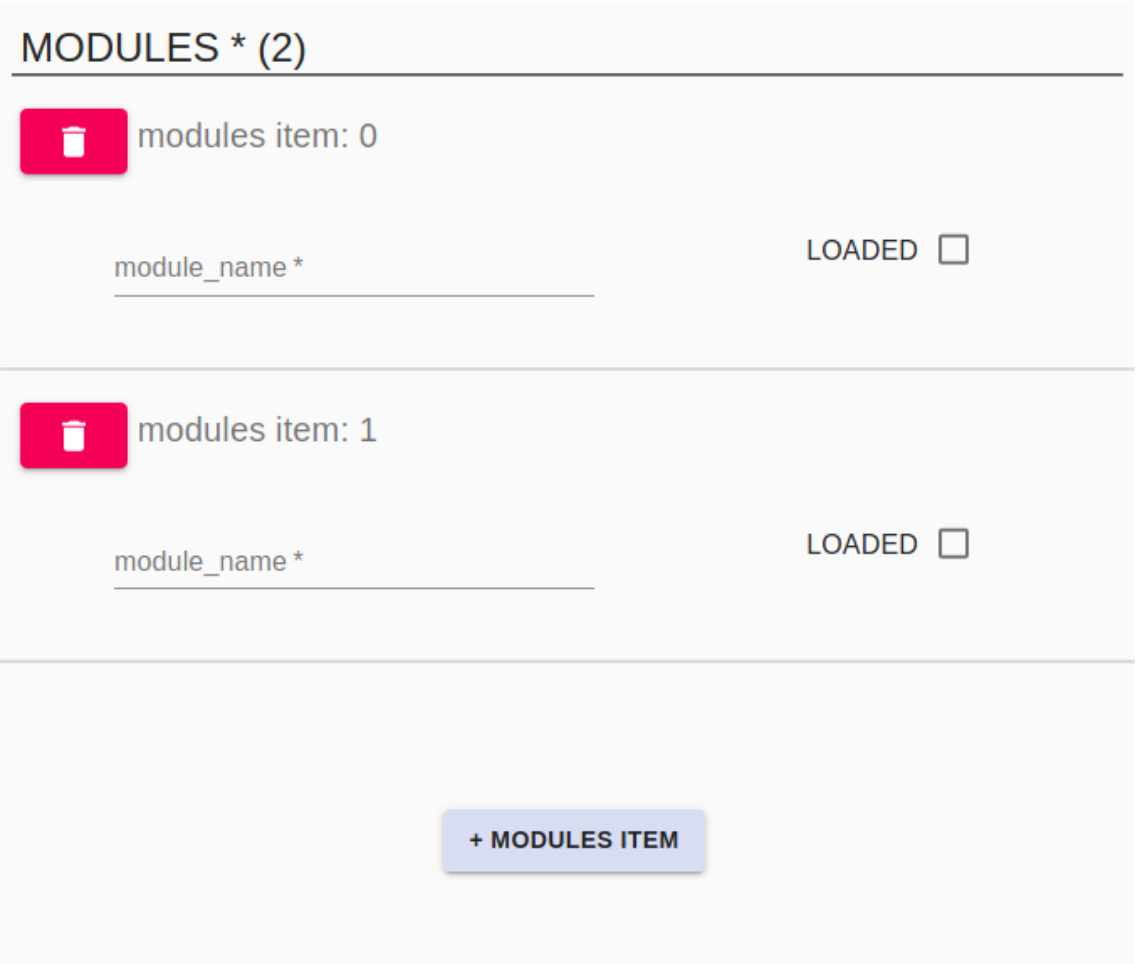
Εικόνα 4.17: *Αντικείμενο "object" με πλήκτρο αναδίπλωσης και τα περιεχόμενά του*



Εικόνα 4.18: *Αντικείμενα "object" με "κρυμμένα" τα περιεχόμενα τους*

Εμφάνιση στοιχείων "array"

Ένα αντικείμενο τύπου "array" (εικόνα 4.19) είναι ίσως το ποιο περίπλοκο/εμπλουτισμένο αντικείμενο που πρέπει να δημιουργηθεί στα πλαίσια της εφαρμογής. Αρχικά εμφανίζεται το όνομα μαζί με μία παρένθεση που περιλαμβάνει τον αριθμό των στοιχείων που περιέχει. Από κάτω, αν έχει οριστεί, υπάρχει η περιγραφή του. Ακριβώς μετά από αυτά εμφανίζονται ένα-ένα όλα τα αντικείμενα ή πεδία που περιέχει με τον ίδιο ακριβώς τρόπο που περιγράφηκε προηγουμένως. Επιπλέον, όμως, κάθε ένα από αυτά περιέχει έναν τίτλο που αναφέρει τον αύξοντα αριθμό του καθώς και ένα κόκκινο πλήκτρο με μορφή κάδου απορριμάτων. Το τελευταίο χρησιμοποιείται για τη διαγραφή του εκάστοτε πεδίου/αντικειμένου. Όταν εμφανιστούν όλα τα στοιχεία εμφανίζεται στο κάτω μέρος του πίνακα ένα πλήκτρο προσθήκης νέου στοιχείου με την επιγραφή "+ <array_name> item", όπου στη θέση του <array_name> βρίσκεται το όνομα του εκάστοτε πίνακα. Τέλος, σημειώνεται ότι ακριβώς όπως και με τα objects έτσι και με τους πίνακες υποστηρίζεται το πλήκτρο αναδίπλωσης περιεχομένων.



Εικόνα 4.19: *Πίνακας με δύο στοιχεία*

Περιπτώσεις "oneOf"

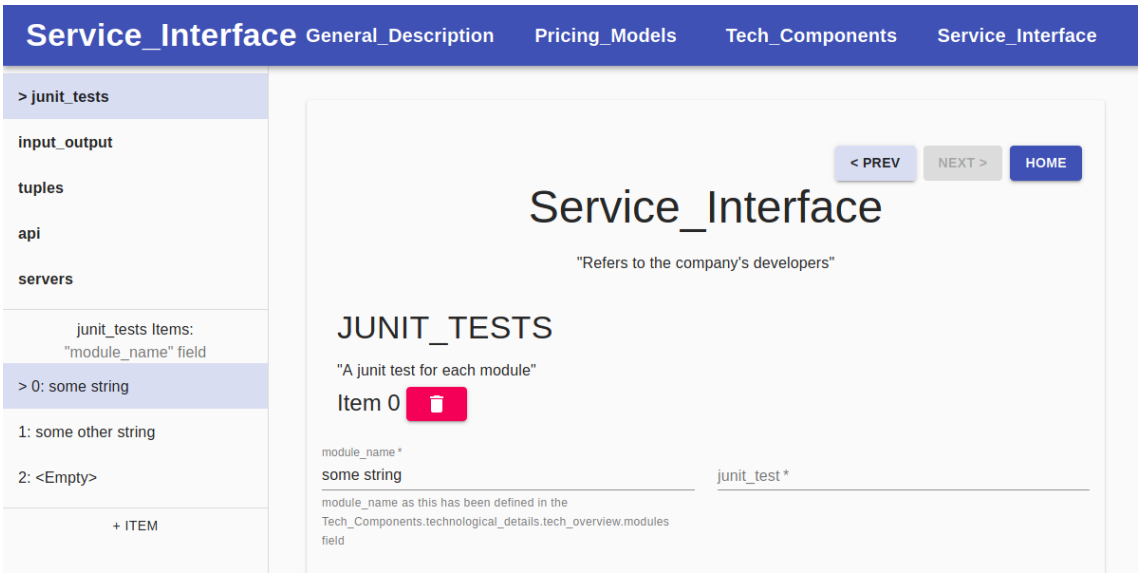
Όταν για κάποιο αντικείμενο έχει οριστεί ο κανόνας "oneOf" τότε πρέπει να είναι δυνατή η επιλογή περιπτώσεων. Για τον λόγο αυτό εμφανίζονται στα δεξιά του ονόματος του πεδίου κατάλληλα πλήκτρα με την ένδειξη "case #" τα οποία περιέχουν τον αριθμό κάθε περίπτωσης. Η ενεργή περίπτωση φαινεται από την τιμή του "_current_case" (στην απλή περίπτωση) ή από τη λίστα "casesNOA" σε περίπτωση που μεσολαβούν πίνακες στη διαδρομή. Όταν μία περίπτωση είναι ενεργή και τα περιεχόμενά της φαίνονται τον χρήστη, το αντίστοιχο πλήκτρο είναι απενεργοποιημένο. Κάτω από αυτά τα πλήκτρα εμφανίζεται εφόσον υπάρχει η αντίστοιχη περιγραφή. Με βάση τον αριθμό της περίπτωσης τροφοδοτούνται τα κατάλληλα output και info στην διαδικασία που περιγράφηκε προηγουμένως ώστε να εμφανιστούν τα πεδία.

Εικόνα 4.20: *Αντικείμενο με κανόνα "oneOf"*

Array Head Properties

Στην περίπτωση που ένα head property είναι τύπου "array" εμφανίζεται μόνο ένα αντικείμενό του τη φορά. Στην κορυφή εμφανίζεται αρχικά ο τίτλος του head property και η περιγραφή του, εφόσον υπάρχει. Κάτω από αυτό φαίνεται ο αύξων αριθμός του αντικειμένου και υπάρχει πάλι ένα κόκκινο πλήκτρο με μορφή κάδου απορριμάτων το οποίο χρησιμοποιείται για τη διαγραφή του εκάστοτε αντικειμένου.

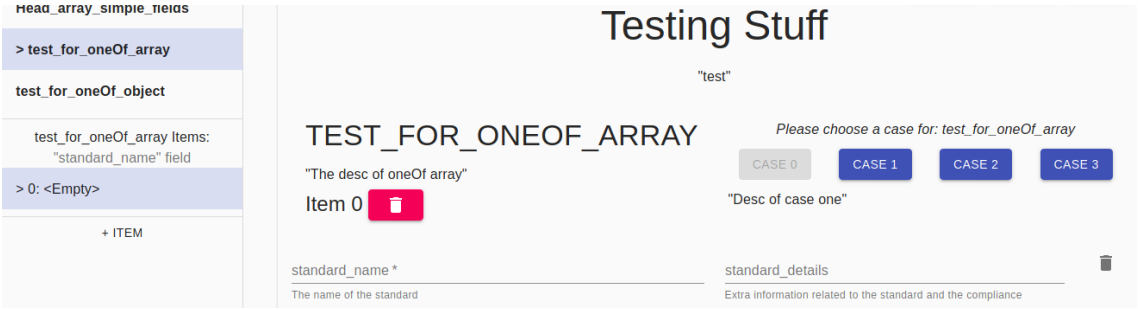
Επιπλέον, το υπο-μενού που βρίσκεται στα αριστερά της οθόνης εμπλουτίζεται. Εμφανίζεται ένα δεύτερο σκέλος (αρχείο MyDrawerLower.js) το οποίο λειτουργεί σαν μενού πλοήγησης στα αντικείμενα του πίνακα. Για να είναι ξεκάθαρο από τον χρήστη στο κάθε αντικείμενο, πέρα από τον αύξοντα αριθμό, εμφανίζεται και το περιεχόμενο του πρώτου απαραίτητου πεδίου, εφόσον αυτό περιέχει κάποια τιμή, αλλιώς εμφανίζεται η τιμή <empty>. Τέλος, στο κάτω μέρος του μενού αυτού υπάρχει πλήκτρο "+ Item" για την προσθήκη νέου στοιχείου στον πίνακα.



Εικόνα 4.21: *Array Head Property με τρία στοιχεία*

4.4.3 "oneOf" σε Head Properties

Τώρα που έχουμε αναλύσει πως ακριβώς εμφανίζονται τα περιεχόμενα ενός head property μπορούμε να ασχοληθούμε με την περίπτωση που έχει οριστεί σε ένα από αυτά ο κανόνας "oneOf". Όπως και στη γενική περίπτωση, έτσι και εδώ εμφανίζονται τα πλήκτρα επιλογής περίπτωσης μαζί με περιγραφή, αν υπάρχει. Ανάλογα με το ποια περίπτωση είναι ενεργή, το αντίστοιχο πλήκτρο απενεργοποιείται και εμφανίζονται τα περιεχόμενα, είτε πρόκειται για object head property είτε για array head property



Εικόνα 4.22: *Array Head Property με κανόνα "oneOf*

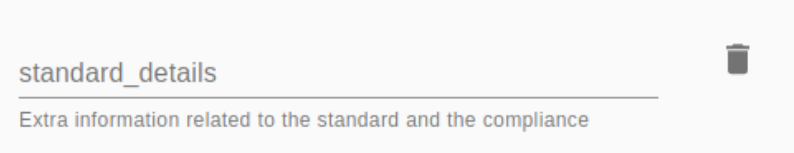
4.4.4 "oneOf" σε sections

Σε συμφωνία με τα προηγούμενα ο κανόνας "oneOf" παρουσιάζεται με τον παρόμοιο τρόπο. Στην κορυφή της σελίδας, υπάρχουν τα πλήκτρα περιπτώσεων μαζί με πριγραφές, αν αυτές έχουν οριστεί. Ανάλογα με την τιμή του property current_oneOf ακολουθείται η διαδικασία εμφάνισης ενός section για τα κατάλληλα output και info.

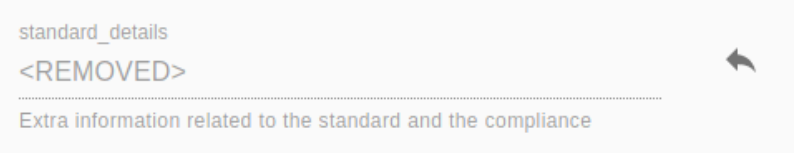
4.4.5 Εμφάνιση πλήκτρου "Remove"/"Restore"

Κατά τη δημιουργία ενός section έγινε αναφορά στο property "req" ενός πεδίου. Το συγκεκριμένο property ορίζει κατά πόσον ένα πεδίο είναι απαραίτητο να συμπληρωθεί σύμφωνα με το JSON-Schema. Αν η τιμή του "req" είναι true, τότε δίπλα στο όνομα του πεδίου εμφανίζεται ένας αστερίσκος “*” που δηλώνει την αναγκαιότητα να συμπληρωθεί το πεδίο αυτό. Από την άλλη, αν η τιμή είναι false τότε ο χρήστης έχει την δυνατότητα να το αφαιρέσει ("Remove") από το Artifact. Τα πεδία που έχουν "req":false εμφανίζουν στα δεξιά τους ένα γκρι κάδο απορριμάτων ο οποίος υπάρχει εκεί για αυτό το σκοπό. Με το πάτημα του το πεδίο αφαιρείται από το αποτέλεσμα αλλά παραμένει στη φόρμα με μία αχνή μορφή και χωρίς τη δυνατότητα συμπλήρωσης του. Αυτό συμβαίνει ώστε σε περίπτωση που ο χρήστης αλλάξει γνώμη για αυτήν την ενέργεια να μπορεί να το επαναφέρει "Restore" πατώντας το ίδιο πλήκτρο, το οποίο αυτή τη φορά θα έχει σχήμα ενός βέλους επαναφοράς.

Στην περίπτωση που το πεδίο βρίσκεται μέσα σε πίνακα, ο έλεγχος του property "req" δεν αρκεί καθώς μπορεί σε ένα αντικείμενο του πίνακα το πεδίο να έχει γίνει remove και σε ένα άλλο όχι. Για το λόγο αυτό ελέγχεται ταυτόχρονα και το κατά πόσον υπάρχει στο output επιτρέποντας έτσι την αφαίρεση πεδίων από κάποια άλλα όχι όλα τα αντικείμενα ενός πίνακα.



Εικόνα 4.23: *Μη απαραίτητο πεδίο με δυνατότητα "αφαίρεσης"*



Εικόνα 4.24: *Μη απαραίτητο πεδίο που έχει "αφαιρεθεί" (Removed) με δυνατότητα επαναφοράς*

4.5 Επεξεργασία του Artifact

Μετά την ανάλυση της δυναμικής δημιουργίας και γραφικής αναπαράστασης του Artifact καταλήγουμε ίσως στο σημαντικότερο κομμάτι της εφαρμογής, την επεξεργασία. Βασικό συστατικό της προσέγγισης που ακολουθήθηκε είναι ότι οι συναρτήσεις πάσης φύσεως επεξεργασίας βρίσκονται στο ίδιο μέρος και ορίζονται όσο το δυνατόν γενικότερα. Το πρώτο μέρος έχει να κάνει με την τεχνική Μοναδικής Πηγής Αληθείας που περιγράφηκε σε προηγούμενο κεφάλαιο. Οι συναρτήσεις βρίσκονται μέσα στο section καθώς εκεί είναι αποθηκευμένες και οι τιμές τις οποίες θα επεξεργαστούν. Το δεύτερο μέρος αφορά την προσπάθεια να χρησιμοποιούνται κατά το δυνατόν λιγότερες συναρτήσεις για περισσότερες περιπτώσεις. Τέλος, για αυτή την ενότητα είναι καλό να γνωρίζουμε ότι ένα component διατηρεί μία λίστα με όνομα "path" όπου αποθηκεύεται η διαδρομή που ακολουθείται μέσα στο output ώστε να καταλήξει κανείς από την κορυφή της "ιεραρχίας" του Artifact στο συγκεκριμένο component.

4.5.1 Επεξεργασία τιμών

Για την εμφάνιση της τιμής ενός πεδίου στον χρήστη απαιτείται να υπάρχει στο component μία αναφορά στο σωστό σημείο του output. Αυτή η αναφορά δίνεται κατά τη δημιουργία του component ενώ παράλληλα εμπλουτίζεται η λίστα "path" με τα ονόματα των αντικειμένων και τα indexes των πινάκων που μεσολαβουν στη διαδρομή. Αν ο χρήστης επιθυμεί να αλλάξει την τιμή ενός πεδίου δεν έχει παρά να γράψει τη νέα τιμή, αν πρόκειται για string, να επιλέξει αν πρόκειται για enum κλπ. Όπως έχουμε αναφέρει, η αλλαγή αυτή γίνεται αυτόματα και χωρίς την ανάγκη άλλης ενέργειας από τον χρήστη, όπως θα ήταν το πάτημα ενός κουμπιού.

Όταν γίνει μία αλλαγή, αρχικά δημιουργείται ένα "γεγονός αλλαγής" (change event). Στη γενική περίπτωση, αυτό ερμηνεύεται από τον editor και καλείται η συνάρτηση handleMasterChange( ). Η παραπάνω συνάρτηση λαμβάνει ως βασικές παραμέτρους το event και το path του πεδίου στο οποίο δημιουργήθηκε η αλλαγή. Στη συνέχεια, μέσω εντολών επανάληψης, ένας δείκτης pointer ξεκινά από την κορυφή του output και, σύμφωνα με το path, καταλήγει στο σημείο όπου πρέπει να γίνει η αλλαγή. Ακολουθεί παράδειγμα διάσχισης του output όπου σε κάθε επανάληψη ο δείκτης "pointer" μετατοπίζεται σε ένα από τα properties του αντικειμένου στο οποίο αρχικά δείχνει σύμφωνα με τη λίστα path:

```
for (i = 1; i < path.length-1; i++) {
    pointer = pointer[path[i]]
}
```

Για να γίνει η αλλαγή καλείται με τα κατάλληλα ορίσματα η συνάρτηση setState( ) μέσω της οποίας γίνεται η ενημέρωση του state στη ReactJS. Σημειώνεται ότι η παραπάνω εντολή προκαλεί την επαναφόρτωση του αντικειμένου το οποίο ενημερώνει. Αυτό, συνεπώς, προκαλεί άμεσα και την επαναφόρτωση του πεδίου το οποίο ο χρήστης επεξεργάστηκε και η αλλαγή εμφανίζεται αμέσως. Ακολουθεί παράδειγμα χρήσης του setState( ) για την ενημέρωση του output (η συγκεκριμένη αποτελεί σύντομη γραφή):

```
this.setState(() => (output))
```

Εξαίρεση στην παραπάνω διαδικασία αποτελεί η περίπτωση του πεδίου object. Σε αυτό, για λόγους που έχουν ήδη περιγραφεί, απαιτείται να πατηθεί το πλήκτρο parse για να ξεκινήσει η προαναφερθείσα διαδικασία. Επιπλέον, η εκτέλεση του parse θα εμφανίσει τυχόν σφάλματα που υπάρχουν στην τιμή του πεδίου η οποία πρέπει να έχει δομή JSON.

4.5.2 Προσθήκη/Διαγραφή στοιχείων σε πίνακα/λίστα

Η προσθαφαίρεση στοιχείων από έναν πίνακα/λίστα ακολουθεί παρόμοια διαδικασία με αυτή της επεξεργασίας τιμών. Αν πατηθεί το κόκκινο πλήκτρο διαγραφής καλείται η συνάρτηση handleMasterDelete( ) η οποία θα λάβει ως παραμέτρους το path μέχρι τον πίνακα και το index του στοιχείου που πρέπει να διαγραφεί. Όπως και στην επεξεργασία τιμών ένας δείκτης καταλήγει με βάση το path στο απαραίτητο σημείο όπου διαγράφεται μέσω της εντολής pointer.splice(index,1) το σωστό στοιχείο. Στη συνέχεια καλείται η setState( ) ώστε να αποθηκευτεί η τιμή και να γίνει επαναφόρτωση.

Από την άλλη, όταν πατηθεί το πλήκτρο εισαγωγής νέου στοιχείου, καλείται η συνάρτηση handleMasterNew( ). Με τον ίδιο ακριβώς τρόπο ένας δείκτης καταλήγει στο σημείο που βρίσκεται ο πίνακας στο output. Επίσης, ένας άλλος δείκτης πάλι με βάση το path "διασχίζει" το info ώστε να βρει εκεί το πρότυπο νέου στοιχείου (template) για τον πίνακα. Στη συνέχεια προσθέτει στο τέλος του πίνακα ένα αντίγραφο του template και καλεί τη setState( ).

Σημειώνεται ότι η λόγω του όγκου και των διαφορετικού τύπου πληροφοριών που αποθηκεύονται για κάθε αντικείμενο στο info είναι απαραίτητη για η διάσχισή του η κλήση της συνάρτησης returnFromInfo( ) η οποία λαμβάνει ως παραμέτρους το path καθώς και το όνομα του property που θέλουμε να επιστραφεί.

4.5.3 Αφαίρεση/Επαναφορά απλών πεδίων

Στην περίπτωση αφαίρεσης και επαναφοράς πεδίων είναι απαραίτητο να διαγραφεί το πεδίο από το output. Συνεπώς, με παρόμοιο τρόπο με πριν, καλείται η συνάρτηση handleMasterRemove( ) όπου ένας δείκτης διαγράφει από το output το συγκεκριμένο πεδίο και στη συνέχεια καλείται η setState( ).

Όταν για ένα πεδίο που έχει αφαιρεθεί ο χρήστης πατήσει το πλήκτρο επαναφοράς καλείται η συνάρτηση handleMasterRestore( ). Στη συνάρτηση αυτή ακολουθείται η κλασική πλέον διαδικασία διάσχισης των output και info από δύο δείκτες. Όταν οι δύο δείκτες φτάσουν στα απαραίτητα σημεία, ο δείκτης του info λαμβάνει τον τύπο του πεδίου. Με βάση αυτόν και τη συνάρτηση returnBasicType( ) επιστρέφεται ένα νέο πεδίο με προεπιλεγμένη τιμή (όπως αυτές ορίστηκαν στην ενότητα 4.3.3). Το πεδίο προστίθεται στο σημείο όπου "δείχνει" ο δείκτης του output και καλείται η συνάρτηση setState( ).

Όπως και με την προσθαφαίρεση στοιχείων από πίνακα/λίστα, έτσι και εδώ, χρησιμοποιείται κατάλληλη συνάρτηση (returnInfoForRemove( )) για τη διάσχηση του info.

4.5.4 Επιλογή περιπτώσεων "oneOf"

Η επιλογή μίας διαφορετικής περίπτωσης από αυτές που προσφέρει ένας κανόνας "oneOf" γίνεται με την επιλογή το αντίστοιχου πλήκτρου. Όταν ο χρήστης επιλέξει μία διαφορετική περίπτωση καλείται η συνάρτηση hasterCaseClicked( ) η οποία δέχεται ως όρισμα το path καθώς και ένα index για την περίπτωση που επιλέχθηκε. Στη συνέχεια γίνεται η γνωστή διαδικασία διάσχησης output και info. Το δείκτης του info με βάση το index θα αντλήσει το κατάλληλο output από το "casesOut" και με αυτό θα αντικαταστήσει το παλαιό πεδίο στο οποίο δείχνει ο δείκτης του output. Στη συνέχεια θα αποθηκευτεί στο info η τρέχουσα πλέον περίπτωση ενημερώνοντας την τιμή του "_current_case" (στην απλή περίπτωση) ή του "casesArrayItems" αν στο path μεσολαβούν πίνακες.

4.5.5 Τεχνικές memo

Οι διαδικασίες που περιγράφηκαν παραπάνω έχουν ένα κοινό, αποθηκεύουν μία τιμή, ενέργεια που αυτόματα προκαλεί την επαναφόρτωση της σελίδας. Αυτό προφανώς γίνεται για κάθε γράμμα μίας συμβολοσειράς που ο χρήστης ενδεχομένως θα πληκτρολογήσει. Είναι, συνεπώς, εύλογη η απορία κατά πόσον αυτό προκαλεί καθυστέρησή (lag) στην εφαρμογή.

Πρώιμοι έλεγχοι που πραγματοποιήθηκαν κατά την ανάπτυξη της εφαρμογής (οι οποίοι θα εξεταστούν στο επόμενο κεφάλαιο) έδειξαν ότι η παραπάνω υπόθεση είναι αληθής. Αιτία του προβλήματος αυτού ήταν ότι η επαναφόρτωση ενός section οδηγούσε στην επαναφόρτωση όλων των components που βρίσκονται από κάτω του. Αυτό δεν είναι πρακτικό καθώς δεν υπάρχει ανάγκη επαναφόρτωσης π.χ. του Κυρίως Μενού ή κάποιου πεδίου κάθε φορά που πληκτρολογείται ένα γράμμα σε κάποιο άλλο πεδίο string.

Λύση στο παραπάνω πρόβλημα αποτέλεσε η χρήση της τεχνικής Memo. Η εντολή memo( ) που προσφέρει η ReactJS είναι ορισμένη για αυτόν ακριβώς το σκοπό. Ο τρόπος με τον οποίο λειτουργεί είναι αρκετά απλός. Σαν πρώτο όρισμα δίνεται μία συνάρτηση/component το οποίο επιθυμούμε να μην επαναφορτώνεται συνεχώς. Σαν δεύτερο όρισμα δίνεται μία συνάρτηση στην οποία δηλώνεται ο έλεγχος ο οποίος θα καθορίζει κατά πόσον το component πρέπει να φωρτωθεί εκ νέου. Π.χ. στο τέλος του αρχείου ItemString.js υπάρχει το εξής:

```
function areEqual (prevProps, nextProps) {
   return JSON.stringify(prevProps.content)===JSON.stringify(nextProps.content)
}

export default React.memo(ItemString, areEqual)
```

Η συνάρτηση areEqual( ) ελέγχει αν το “περιεχόμενο” που είχε δοθεί ως όρισμα (αντικείμενο "props") στο compontent για να εμφανιστεί, είναι διαφορετικό από αυτό που του δίνεται κατά την επαναφόρτωση. Αν είναι ίδια, δηλαδή επιστραφεί η τιμή true, τότε παραμένει στην οθόνη το παλαιό compontent, όπως και θα έπρεπε να γίνει. Αν όχι, προφανώς επιστρέφεται η τιμή false και το component φορτώνεται εκ νέου.

4.6 Παραγωγή Εξόδου

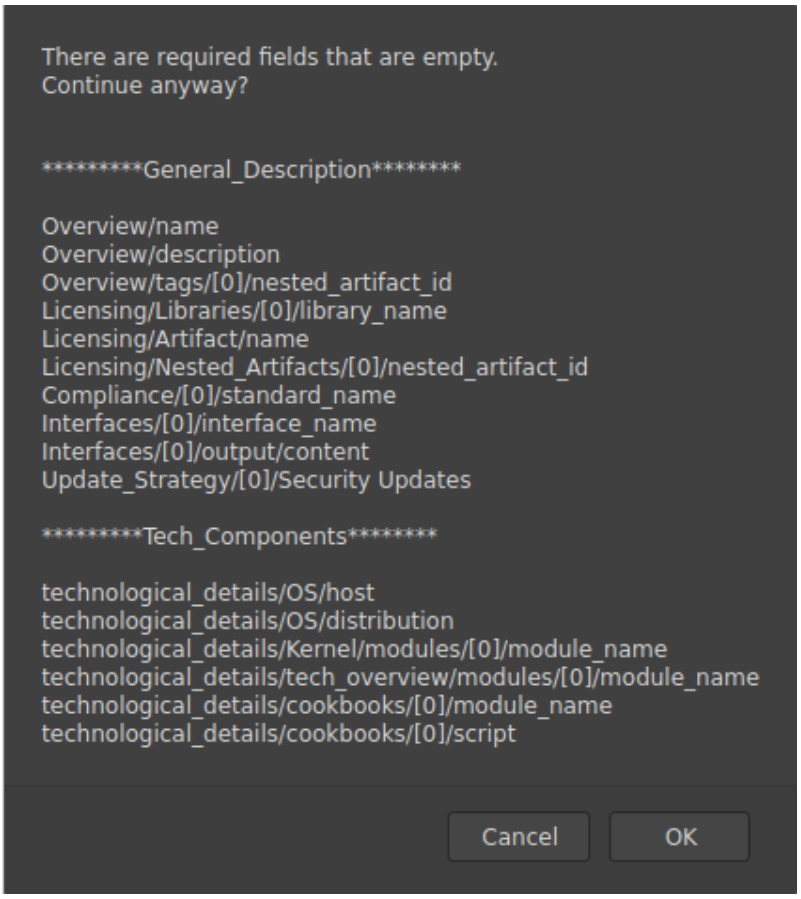
Στην ενότητα αυτή γίνεται η περιγραφή του τρόπου με τον οποίο ελέγχεται και παράγεται το τελικό Artifact. Η διαδικασία αυτή ξεκινάει με το πάτημα του πλήκτρου ¨Παραγωγής αρχείου εξόδου¨ ("Output file").

4.6.1 Έλεγχος απαραίτητων πεδίων

Όπως γνωρίζουμε από προηγούμενα κεφάλαια, ορισμένα πεδία δηλώνονται από την είσοδο ως απαραίτητα. Συνεπώς πρέπει να υπάρχουν ως properties στο τελικό αποτέλεσμα και να έχουν τιμή. Με το πάτημα του πλήκτρου "Output file" καλείται μέσω του Home η συνάρτηση homeCheckRequired( ) με παράμετρο τη λίστα masterStates. Αυτή διασχίζει για κάθε state τα output και info με σκοπό να ελέγξει την τιμή των πεδίων που έχουν στο info "req":true.

Είναι χρήσιμο να τονισθεί ότι ο παραπάνω έλεγχος αφορά μόνο την ύπαρξη τιμής. Με δεδομένες τις προεπιλεγμένες τιμές των πεδίων, βλέπουμε ότι τα μόνα πεδία που ουσιαστικά απαιτούν έλεγχο είναι αυτά με τύπο "string" και "object". Τα δύο παραπάνω έχουν ως προεπιλεγμένη τιμή ""(κενή συμβολοσειρά) και "{ }" (κενό αντικείμενο/στοιχείο) αντίστοιχα. Συνεπώς, αν βρεθούν τέτοια πεδία που είναι μεν απαραίτητα αλλά έχουν τις παραπάνω τιμές, πρέπει να γίνει μία ενημέρωση προς τον χρήστη. Για τους υπόλοιπους τύπους πεδίων δεν είναι απαραίτητος ο έλεγχος καθώς οι προεπιλεγμένες τιμές αποτελούν μία λογική τιμή (υπενθυμίζεται ότι για πεδία τύπου "number" έχουμε αρχική τιμή 0 (μηδέν), για "boolean" έχουμε false και για "enum" έχουμε την πρώτη επιλογή).

Όταν λοιπόν ο έλεγχος ανακαλήψει κάποιο πεδίο "string" ή "object" που είναι απαραίτητο αλλά εξακολουθεί να έχει την προεπιλεγμένη τιμή του, αποθηκεύει το path του σε μία λίστα. Στο τέλος, αν η λίστα δεν είναι κενή, εμφανίζεται στον χρήστη ενημερώνοντας τον, μέσω ενός παραθύρου διαλόγου. Το παράθυρο αυτό δίνει στο χρήστη την επιλογή, παρά τα μη συμπληρωμένα πεδία, να παράξει το Artifact εφόσον αυτός το επιθυμεί. Αν η λίστα είναι κενή είναι προφανές ότι δεν υπάρχουν άλλα πεδία τα οποία χρήζουν απαραίτητα συμπλήρωσης και συνεπώς περνάμε αυτόματα στην παραγωγή της εξόδου.



Εικόνα 4.25: *Παράθυρο διαλόγου με τη λίστα των απαραίτητων πεδίων που δεν έχουν συμπληρωθεί*

4.6.2 Παραγωγή του Artifact

Η δομή με την οποία έχουν αποθηκευτεί τα δεδομένα καθιστά την παραγωγή της εξόδου μία αρκετά απλή διαδικασία. Ουσιαστικά, το Artifact είναι μοιρασμένο σε τόσα τμήματα όσα και τα sections του. Συνεπώς, η μόνη ενέργεια που πρέπει να πραγματοποιηθεί είναι η συλλογή των output από κάθε section σε ένα ενιαίο αντικείμενο. Τέλος, το αντικείμενο αυτό αποτυπώνεται σε ένα αρχείο με όνομα "Artifact.json" και προσφέρεται προς λήψη από τον χρήστη. Η παραπάνω διαδικασία πραγματοποιείται από τη συνάρτηση homeOutputFile( ) η οποία δέχεται ως παράμετρο τη λίστα masterStates (μέρος του κώδικα της συνάρτησης που αφορά στην εγγραφή σε αρχείο μπορεί να βρεθεί στη βιβλιογραφία[16]).

Κεφάλαιο 5

Έλεγχος

Στο κεφάλαιο αυτό γίνεται αναφορά στην δοκιμές που πραγματοποιήθηκαν κατά τη διάρκεια ανάπτυξης αλλά και μετά την ολοκλήρωση της εφαρμογής.

5.1 Έλεγχοι υποστήριξης πολύπλοκων περιπτώσεων εισόδου

Κατά τη διάρκεια ανάπτυξης της εφαρμογής γίνονταν συνεχείς έλεγχοι για το κατά πόσον υποστηρίζεται οποιοδήποτε input πληροί τους κανόνες που έχουν τεθεί. Επειδή, οι πιθανές είσοδοι είναι άπειρες, δόθηκε βάση στον έλεγχο ορισμένων “ακραίων” περιπτώσεων. Αυτές οι περιπτώσεις αφορούσαν εισόδους που περιλάμβαναν π.χ. εμφωλευμένους πίνακες (πινακες μέσα σε πίνακες μέσα σε πίνακες κοκ) ή εμφωλευμένους κανόνες "oneOf". Αυτού του είδους οι έλεγχοι βοήθησαν ώστε να αναγνωριστούν τα όρια της εφαρμογής και να γίνουν οι απαραίτητες διορθώσεις με σκοπό την εξάλειψη του ενδεχομένου εμφάνισης σφάλματος μετά τη δυναμική δημιουργία του Artifact. Ένας από του πρώτους και βασικότερους ελέγχους ήταν αυτός που έλεγξε την περίπτωση που ένας κανόνας "oneOf" εμφανιζόταν μέσα σε αντικείμενο ενός πίνακα. Αυτό οδήγησε στην επέκταση της υλοποίησης του κανόνα "oneOf" με τη δημιουργία του σχήματος αποθήκευσης περιπτώσεων "casesArrayItems" που περιγράφηκε στο προηγούμενο κεφάλαιο.

5.2 Έλεγχοι χρόνου απόκρισης

Ένα σημαντικότατο σημείο του σχεδιασμού μίας εφαρμογής είναι ο χρόνος ανταπόκρισής της (response time) στις εντολές του χρήστη. Ίσως το πιο πρακτικό σημείο όπου αυτό θα έπρεπε να ελεγχθεί είναι κατά τη συμπλήρωση τιμών σε πεδία τύπου "string". Όπως έχει ήδη αναφερθεί, κάθε πλήκτρο που πληκτρολογείται από τον χρήστη οδηγεί σε μερική επαναφόρτωση της σελίδας. Αυτό θα μπορούσε δυνητικά να οδηγήσει σε καθυστερήσεις για έναν χρήστη πληκτρολογεί αρκετά γρήγορα.

Η πρώτη απόπειρα ελέγχου του παραπάνω έγινε με απλή χρήση της εφαρμογής. Αυτό που παρατηρήθηκε ήταν πως, πράγματι, για γρήγορη πληκτρολόγηση σε ένα πεδίο τύπου "string" η ταχύτητα απόκρισης ήταν χαμηλή με αποτέλεσμα η συμβολοσειρά να εμφανίζεται λίγο πιο μετά από τη στιγμή που πληκτρολογήθηκε. Το παραπάνω δεν δημιουργούσε λάθη στις τιμές καθώς κανένα πλήκτρο δεν χανόταν. Παρ' όλα αυτά θεωρήθηκε πως έπρεπε να εξεταστεί περαιτέρω.

Η επέκταση React Development Tools για τον περιηγητή Mozilla Firefox προσφέρει μεταξύ των άλλων, το εργαλείο Profiler. Το συγκεκριμένο εργαλείο επιτρέπει στον προγραμματιστή να "μαγνητοσκοπήσει" τρόπον τινά μία ενέργεια στην εφαρμογή του και στη συνέχεια να λάβει δεδομένα για το χρόνο εκτέλεσής της καθώς και την ενδεχόμενη επαναφόρτωση κάθε component. Η χρήση του παραπάνω component για τη διαδικασία πληκτρολόγησης ενός μόνο πλήκτρου μέσα σε ένα πεδίο τύπου "string" έδειξε τα εξής:

1. Γινόταν επαναφόρτωση πολλών components χωρίς να υπάρχει η ανάγκη
2. Ο χρόνος εμφάνισης του χαρακτήρα που πληκτρολογήθηκε ήταν περισσότερο από 100ms

Από το δεύτερο καταλαβαίνουμε ότι μία ταχύτητα πληκτρολόγησης τουλάχιστον 10 χαρακτήρων/δευτερόλεπτο θα οδηγήσει σε καθυστερήσεις. Γνωρίζουμε ότι ανάλογα με το σύστημα στο οποίο γίνεται ο έλεγχος, τα στοιχεία θα διαφέρουν. Συνεπώς, δεν έχουν ιδιαίτερη αξία οι αντικειμενικές τιμές αλλά η σχέση πριν και μετά την εφαρμογή ειδικών μεθόδων επίλυσης του προβλήματος αυτού.

Οι δύο παρατηρήσεις που προέκυψαν από τον έλεγχο δεν είναι ανεξάρτητες. Το γεγονός ότι ορισμένα components φορτώνουν εκ νέου, χωρίς αυτό να είναι αναγκαίο, καταναλώνει πολύτιμο χρόνο. Συνεπώς, η λύση σε αυτό το πρόβλημα είναι να φορτώνουν εκ νέου μόνο τα components στα οποία αυτό επιβάλλεται. Με βάση τα παραπάνω υλοποιήθηκαν οι λύσεις που προτείνονται στην ενότητα 4.5.5.

Μετά την εισαγωγή των μεθόδων memo πραγματοποιήθηκε εκ νέου ο ίδιος έλεγχος πάνω στο ίδιο σύστημα. Τα αποτελέσματα του ελέγχου που επαναλήφθηκε σε δέκα διαφορετικά πεδία του Default Input παρουσιάζονται στον πίνακα 5.1:

Παρατηρώντας τα αποτελέσματα βλέπουμε μία αισθητή μείωση του χρόνου απόκρισης, πράγμα που φαίνεται και από τη χρήση της εφαρμογής. Ο μέσος όρος του χρόνου απόκρισης που προκύπτει από τον έλεγχο είναι 31,7ms. Συγκρίνοντάς το παραπάνω με την προηγούμενη μέτρηση (πάνω από 100ms) καταλαβαίνουμε ότι η χρήση της συνάρτησης memo στα διάφορα components προσέφερε μία βελτίωση χρόνου απόκρισης πάνω από 70%!

Πίνακας 5.1: *Πίνακας χρόνου απόκρισης κατά την πληκτρολόγηση ενός νέου χαρακτήρα*

Επανάληψη	**Χρόνος Απόκρισης σε ms**
1	36
2	39
3	28
4	21
5	33
6	27
7	31
8	46
9	37
10	19
Μέσος Όρος	31,7

Μέρος III

Επίλογος

Κεφάλαιο 6

Επίλογος

6.1 Συμπεράσματα

Η ανάγκη δημιουργίας ενός VDC Artifact βασισμένο σε ένα JSON-Schema οδήγησε στην εκπόνηση της παρούσας διπλωματική εργασίας. Στα πλαίσιά της αναπτύχθηκε μία εφαρμογή η οποία δίνει τη δυνατότητα στο χρήστη να εισάγει ένα JSON-Schema το οποίο θα καθορίσει τη μορφή του Artifact που επιθυμεί να δημιουργήσει. Η χρήση είναι απλή, εύκολη στην κατανόηση και οδηγεί στην ταχύτατη δημιουργία του ζητούμενου αποτελέσματος. Επιπλέον, δίνεται η εγγύηση ότι το αποτέλεσμα πληροί τόσο τους κανόνες δομής ενός αρχείου json αλλά συνάδει απόλυτα και με το σχήμα που δόθηκε.

Συνεπώς, όλα τα παραπάνω μετατρέπουν τη δημιουργία ενός VDC Artifact από μία χρονοβόρα και δυνητικά επιρρεπή σε λάθη διαδικασία σε μία απλή συμπλήρωση φόρμας που εγγυάται για την ακεραιότητα των δεδομένων και την τήρηση των κανόνων. Κάθε πιθανός χρήστης δεν έχει λόγο να μη τη χρησιμοποιήσει.

6.2 Μελλοντικές επεκτάσεις

Η εφαρμογή της παρούσας εργασίας θα μπορούσε μελλοντικά να επεκταθεί επάνω στους εξής δύο άξονες:

1. Πέρα από τον κανόνα "oneOf" θα ήταν θεμητό να υπάρχει υποστήριξη όλων των κανόνων συνδυασμού JSON-Schemas καθώς και η υποστήριξη αναφορών. Οι κανόνες "allOf" και "anyOf" εμπλουτίζουν τους τρόπους με τους οποίους μπορεί να οριστεί ένα σχήμα. Μία συνήθης πρακτική ορισμού του "allOf" είναι η χρήση αναφοράς "$ref" η οποία επιτρέπει την επαναχρησιμοποίησή τμημάτων του Schema και σε άλλα σημεία του.

   Τα παραπάνω, παρ' ότι δεν είναι απαραίτητα, θα έδιναν στον χρήστη περισσότερη ελευθερία σχετικά με τον ορισμό του σχήματος εισόδου.

2. Υποστήριξη αποθήκευσης εργασίας. Το γνωστό και ως save θα έδινε στον χρήστη τη δυνατότητα να αποθηκεύσει την πρόοδό του όταν ακόμα το Artifact ήταν ημιτελές. Με τον τρόπο αυτό θα μπορούσε σε μελλοντικό χρόνο να συνεχίσει από εκεί που σταμάτησε.

   Το παραπάνω θα αποτελούσε ιδιαίτερα σημαντική ευκολία προς τον χρήστη καθώς δε θα ήταν απαραίτητο να ολοκληρώσει την εργασία του κατευθείαν και με την πρώτη προσπάθεια. Επίσης θα παρείχε την ασφάλεια της αποθήκευσης σε περίπτωση προβλήματος ανεξάρτητο της εφαρμογής το οποίο θα οδηγούσε σε τερματισμό της.

Βιβλιογραφία

[1] *Node-RED*. https://nodered.org. Ημερομηνία πρόσβασης: 19-02-2021.

[2] Vrettos Moulos Achilleas Marinakis George Chatzikyriakos David García Pérez Jose Antonio Sanchez Pierluigi Plebani David Bermbach Marco Peise Sebastian Werner Ma ya Anderson Aitor Fernández Grigor Pavlov Peter Gray. *Data-intensive applications Improvement by moving daTA and computation inmixed cloud/fog environmentS*. Τεχνική Αναφορά με αριθμό No.731945, European Union's Horizon research and innovation programme, 2017.

[3] *What is JSON?* https://www.w3schools.com/whatis/whatis_json.asp. Ημερομηνία πρόσβασης: 21-02-2021.

[4] LarsÅke Fredlund, Clara Benac Earle, Ángel Herranz και Julio Mariño. *Property-Based Testing of JSON Based Web Services*. *2014 IEEE International Conference on Web Services*, Anchorage, AK, USA, 2014.

[5] Philipp Wehner, Christina Piberger και Diana Göhringer. *Using JSON to manage communication between services in the Internet of Things*. *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, Montpellier, France, 2014.

[6] Angelo Augusto Frozza, Ronaldodos Santos Mello και Felipede Souza da Costa. *An Approach for Schema Extraction of JSON and Extended JSON Document Collections*. *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, Salt Lake City, UT, USA, 2018.

[7] *What is JSON Schema?* https://restfulapi.net/json-schema/. Ημερομηνία πρόσβασης: 21-02-2021.

[8] *genericSchema.json*. https://github.com/thevdc/virtualdatacontainer/blob/master/jsonSchemas/genericSchema.json. Ημερομηνία πρόσβασης: 26-02-2021.

[9] *DITAS-Project*. https://www.ditas-project.eu/. Ημερομηνία πρόσβασης: 19-02-2021.

[10] *ditasSchema.json*. https://github.com/thevdc/virtualdatacontainer/blob/master/jsonSchemas/ditasSchema.json. Ημερομηνία πρόσβασης: 23-02-2021.

[11] *DITAS JSON Schema Editor*. https://github.com/thevdc/virtualdatacontainer/tree/master/DITASjsonSchemaEditor. Ημερομηνία πρόσβασης: 23-02-2021.

[12] *Type-specific keywords*. https://json-schema.org/understanding-json-schema/reference/type.html. Ημερομηνία πρόσβασης: 26-02-2021.

[13] *React*. https://reactjs.org/. Ημερομηνία πρόσβασης: 27-02-2021.

[14] *Material - UI*. https://material-ui.com/. Ημερομηνία πρόσβασης: 27-02-2021.

[15] *Enable Tab Character Insertion Inside Textarea*. https://jsfiddle.net/2wAzx/13/. Ημερομηνία πρόσβασης: 04-03-2021.

[16] *How to save my input values to text file with ReactJS?* https://stackoverflow.com/questions/61237355/how-to-save-my-input-values-to-text-file-with-reactjs/61238960#61238960. Ημερομηνία πρόσβασης: 05-03-2021.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

βλπ	βλέπε
κ.λπ.	και λοιπά
κ.ο.κ	και ούτω καθεξής
π.χ.	παραδείγματος χάρην
API	Application Programming Interface
CAF	Corporate Address File
DITAS	Data-intensice applications Improvement by moving daTA and computation in mixed cloud/fog environmentS
JSON	JavaScript Object Notation
REST	REpresentational State Transfer
SSoT	Single Source of Truth
UI	User Interface
VDC	Virtual Data Container

Απόδοση ξενόγλωσσων όρων

Ξενόγλωσσος όρος	Απόδοση
abstract	αφηρημένος
add item button	πλήκτρο προσθήκης αντικειμένου
Application Programming Interfaces	Διασυνδέσεις Προγραμματισμού Εφαρμογών
array	πίνακας/λίστα
arrow button	πλήκτρο με μορφή βέλους
boolean	πεδίο δυαδικής τιμής
change event	γεγονός αλλαγής
checkbox	πλαίσιο ελέγχου
cloud	νέφος
component	συστατικό/μέρος εφαρμογής
compression	συμπίεση
Corporate Address File	αρχείο εταιρικής διεύθυνσης
current case	τρέχουσα περίπτωση
datasources	πηγές δεδομένων
default input	προεπιλεγμένη είσοδος
delete item button	πλήκτρο διαγραφής αντικειμένου
dependencies	εξαρτήσεις
deployment	ανάπτυξη
description	περιγραφή
download	λήψη (αρχείου)
dynamic	δυναμικός, μη στατικός, μη προκαθορισμένος
edge divices	συσκευές αιχμής
editor	εφαρμογή σύνταξης
element	στοιχείο, αντικείμενο
encryption	κωδικοποίηση
false	ψευδής
fields	πεδία
generate	παράγω
head properties	κύριες ιδιότητες
home	αρχική οθόνη
index	αύξων αριθμός

info	πληροφορίες
input	δεδομένα εισόδου
JSON-Schema	δομικό σχήμα JSON
lag	καθυστέρηση
library	βιβλιοθήκη
middleware	μεσάζον λογισμικό
modules	τμήματα, μέρη
multi-page application	εφαρμογή πολλαπλών σελίδων
next button	πλήκτρο “Επόμενο”
number	αριθμός
object	αντικείμενο
object type field	πεδίο τύπου αντικειμένου
output file button	πλήκτρο δημιουργίας αρχείου εξόδου
parent element	αντικείμενο “γονέας”
parse button	πλήκτρο ανάλυσης
path	διαδρομή
pipelines	ροές δεδομένων
pointer	δείκτης
previous button	πλήκτρο “Προηγούμενο”
properties	ιδιότητες
remove button	πλήκτρο αφαίρεσης πεδίου
REpresentational State Transfer	Αντιπροσωπευτική Μεταφορά Κατάστασης
required	απαραίτητος
response time	χρόνος απόκρισης
restore button	πλήκτρο επαναφοράς πεδίου
rules	κανόνες
save	αποθήκευση εργασίας
scalability	επεκτασιμότητα
script	κώδικας
scroll	μετακίνηση πάνω σε σελίδα
simple field arrays	πίνακες/λίστες απλών πεδίων
single-page application	εφαρμογή μοναδικής σελίδας
Single Source of Truth	Μοναδική Πηγή Αληθείας
string	συμβολοσειρά
text box	πλαίσιο κειμένου
true	αληθής (τιμή)
user interface	διεπαφή χρήστη
upload	μεταφόρτωση αρχείου
validate	επικυρώνω
versioning	εκδόσεις

Virtual Data Container	Ψηφιακό Κιβώτιο Δεδομένων
web browser	περιηγητής ιστού