ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

RESTful API για Σχεδιασμό Βάσης Δεδομένων MySQL

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Ν. Αναστασάκης

Επιβλέπων : Γεώργιος Στασινόπουλος
Καθηγητής Ε.Μ.Π

Αθήνα, Οκτώβριος 2019

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

RESTful API για Σχεδιασμό Βάσης Δεδομένων MySQL

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Ν. Αναστασάκης

Επιβλέπων : Γεώργιος Στασινόπουλος
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 29η Οκτωβρίου 2019.

.....	.....	.....
Γεώργιος Στασινόπουλος	Ευστάθιος Δ. Συκάς	Ιωάννα Ρουσσάκη
Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Επίκουρη Καθηγήτρια Ε.Μ.Π

Αθήνα, Οκτώβριος 2019

.....

Γεώργιος Ν. Αναστασάκης

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

Περίληψη

Στην παρούσα διπλωματική εργασία παρουσιάζεται ο σχεδιασμός και η ανάπτυξη μιας υπηρεσίας ιστού αρχιτεκτονικής Representational State Transfer καθώς και του συστήματος διεπαφής χρήστη που αλληλοεπιδρά μαζί της. Υλοποιούμε μια διαδικτυακή πλατφόρμα που παρέχει στον χρήστη τη δυνατότητα να δημιουργεί και να επεξεργάζεται δυναμικά το schema της βάσης δεδομένων του. Η επικοινωνία με την ιστοσελίδα διεπαφής χρήστη επιτυγχάνεται μέσω μιας RESTful προγραμματιζόμενης διεπαφής. Για την δημιουργία της πλατφόρμας χρησιμοποιήθηκαν εργαλεία, βιβλιοθήκες και πλαίσια ανάπτυξης που επιλέχθηκαν με βάση τις ανάγκες και τις απαιτήσεις της εργασίας. Πιο συγκεκριμένα, έγινε χρήση της ReactJS, του Spring Boot, του JOOQ και της MySQL βάσης δεδομένων.

Λέξεις Κλειδιά

Schema, REST API, Spring Boot, React Frontend, MySQL, JOOQ

Abstract

In the present diploma thesis we present the design and the development of a Representational State Transfer web service as well as a user interface system that interacts with it. We implement an online platform, which gives the end-user the ability to dynamically create and edit his database schema. By consuming a RESTful application programming interface (API) we achieve communication with the user interface web client. The platform was developed using tools, libraries and frameworks that were selected on the basis of the needs and requirements of the project. More specifically, ReactJS, Spring Boot, JOOQ and MySQL database were used.

Keywords

Schema, REST API, Spring Boot, React Frontend, MySQL, JOOQ

Περιεχόμενα

Ευρετήριο Εικόνων

Κεφάλαιο 1

Εισαγωγή

Η σημερινή εποχή χαρακτηρίζεται ως η εποχή της πληροφορίας. Οι πληροφορίες πλέον μεταφέρονται με τεράστια ευκολία και ταχύτητα σε απίστευτα μεγάλο εύρος. Η εξέλιξη αυτή είναι αδιαμφισβήτητα απόρροια της εξέλιξης και διάδοσης του Internet. Οι περισσότερες διαδικτυακές εφαρμογές τα τελευταία χρόνια εξελίσσονται και τείνουν να υιοθετήσουν χαρακτήρα υπηρεσίας. Έτσι, πρέπει να καλύπτουν τις απαιτήσεις που θέτουν τα σύγχρονα συστήματα. Μία διαδικτυακή υπηρεσία εφαρμόζει τεχνολογίες που προσφέρουν ταχύτητα, απλότητα, χαμηλό κόστος και ανεξαρτησία απο λειτουργικά συστήματα και τεχνολογίες.

Για τον σχεδιασμό και την υλοποίηση μιας διαδικτυακής υπηρεσίας ο προγραμματιστής πρέπει να ακολουθήσει συγκεκριμένες αρχές, πρότυπα και αρχιτεκτονικές. Πιο συγκεκριμένα, η εφαρμογή πρέπει να είναι εύκολα προσβάσιμη μέσω του διαδικτύου ακολουθώντας κλασσικά πρωτόκολλα επικοινωνίας και να επιτυγχάνεται ο πλήρης διαχωρισμός της στην διεπαφή του χρήστη και στον διακομιστή, δηλαδή της ίδιας της υπηρεσίας με την πλατφόρμα που θα καταναλώνει το προϊόν. Έτσι, το σύστημα πρέπει να αποτελείται απο δύο ανεξάρτητα μέρη που δεν παρουσιάζουν περιορισμούς μεταξύ τους και να μπορούν να κατασκευαστούν ξεχωριστά.

Σκοπός της εργασίας αυτής, είναι να αναδείξει εργαλεία κατασκευής μιας διαδικτυακής υπηρεσίας τα οποία καλύπτουν τις ανάγκες υλοποίησης και εξασφαλίζουν τις αρχές σχεδιασμού της εφαρμογής. Χρησιμοποιώντας συγκεκριμένα εργαλεία, γνωστά ως frameworks, κατφέραμε να δημιουργήσουμε δύο αυτόνομα επιμέρους κομμάτια της εφαρμογής και να διασφαλίσουμε ότι επικοινωνούν σωστά μεταξύ τους. Η εφαρμογή θα προφέρει την δυνατότητα στον χρήστη να σχεδιάζει την βάση δεδομένων του, έχοντας πρόσβαση σε αυτή μέσω του διαδικτύου.

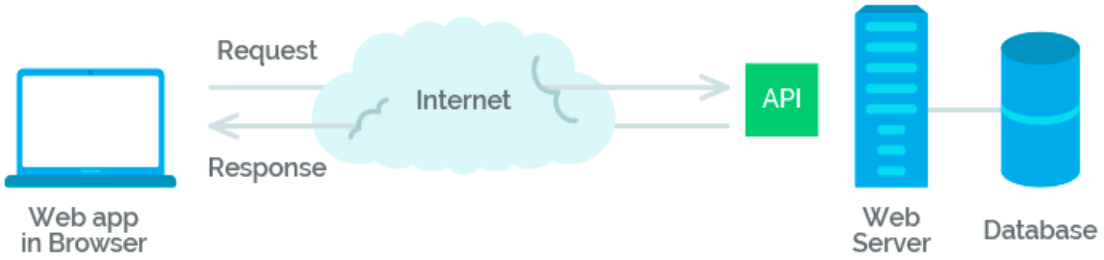
1.1 Οργάνωση Κειμένου

Η παρούσα διπλωματική εργασία αποτελείται απο 5 κεφάλαια, συμπεριλαμβανομένου του παρόντος και πρώτου κεφαλαίου όπου παρουσιάζεται η εισαγωγή και η οργάνωση της εργασίας. Το δεύτερο κεφάλαιο αναφέρεται στο θεωρητικό πλαίσιο στο οποίο βασίζεται η διαδικτυακή πλατφόρμα, καθώς και στις τεχνολογίες που χρησιμοποιήθηκαν. Στο τρίτο κεφάλαιο σκιαγράφεται η αρχιτεκτονική του συστήματος, ενώ αναλύονται τα διαθέσιμα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη βασικών τμημάτων του προγράμματος. Το τέταρτο κεφάλαιο εμπεριέχει την υλοποίηση των βασικότερων κομματιών του συτήματος και παραθέτει ένα παράδειγμα λειτουργίας του. Στο πέμπτο και τελευταίο κεφάλαιο προτείνονται μελλοντικές επεκτάσεις της εργασίας.

Κεφάλαιο 2

Θεωρητικό Πλαίσιο – Τεχνολογίες

2.1 API για την ανάπτυξη RESTful υπηρεσιών ιστού (Web Services)



1. Απεικόνιση Μοντέλου Client-Server

Ένα RESTful API αποτελεί μια διεπαφή προγραμματισμού εφαρμογών (application programming interface), και καθορίζει ένα σύνολο κανόνων και μηχανισμών με τους οποίους μια εφαρμογή ή ένα στοιχείο (component-microservice) αλληλεπιδρά με άλλα προγράμματα λογισμικού (User Interfaces, external services, third-party APIs κλπ).

Όταν καλείται ένα RESTful API, ο server θα μεταφέρει στον client μια αναπαράσταση της κατάστασης του ζητούμενου πόρου.Τέτοια είδους συμπεριφορά βασίζεται στο μοντέλο REST (**Representational State Transfer**).

Συνοπτική περιγραφή των 6 REST κανόνων και μηχανισμών αρχιτεκτονικής [1]:

1. **Uniform Interface (UI).** Οι πόροι πρέπει να απεικονίζονται με ένα μονοσήμαντο-μοναδικό τρόπο (request URL) και μόνο μέσω των HTTP μεθόδων και των ιδιοτήτων τους (headers, query parameters, form data, path parameters & request payload/body) θα είναι δυνατή η ανάκτηση και η επεξεργασία των πόρων.
2. **Client-Server Model.** Η υπηρεσία REST τρέχει σε έναν κεντρικό διακομιστή και οι πελάτες/χρήστες αιτούνται πόρους απο αυτή. Το λογισμικό διεπαφής του χρήστη (**front-end**) επικοινωνεί με το λογισμικό του διακομιστή (**back-end**) μέσω διακριτών HTTP αιτημάτων (**requests**), τα οποία απαντώνται απο τον server με διακριτές απαντήσεις (**responses**).

3. **Stateless Operations.** Ο διακομιστής δεν διατηρεί κάποια πληροφορία ως προς την κατάσταση της σύνδεσης, αλλά επεξεργάζεται κάθε αίτηση το ίδιο.
4. **RESTful resource cashing.** Τα δεδομένα πρέπει να δηλώνονται είτε άμεσα είτε έμμεσα ως προς την δυνατότητα τους να αποθηκευτούν. Έτσι, ο χρήστης μπορεί να τα ξαναχρησιμοποιήσει μέσα σε κάποιο ορισμένο χρονικό διάστημα χωρίς να ξανακάνει αίτημα στον διακομιστή.
5. **Layered system.** Ο χρήστης της υπηρεσίας δεν πρέπει να έχει την δυνατότητα να αντιληφθεί αν είναι συνδεδεμένος στον κεντρικό διακομιστή ή σε κάποιον ενδιάμεσο.
6. **Code on demand.** Τις περισσότερες φορές ο διακομιστής θα απαντήσει στον χρήστη με δεδομένα τα οποία θα είναι σε μορφή JSON ή XML. Όταν είναι απαραίτητο, έχει την δυνατότητα να μεταφέρει στον πελάτη τον εκτελέσιμο κώδικα.

Με άλλα λόγια, τα RESTful APIs αποτελούν τον κώδικα ο οποίος όταν εκτελείται επιτρέπει την αξιόπιστη επικοινωνία μεταξύ δύο προγραμμάτων λογισμικού (ζήτηση πόρων-δεδομένων απο ένα λειτουργικό σύστημα ή μια εφαρμογή).

Η επικοινωνία αυτή πραγματοποιείται μέσω του πρωτόκολλου **HTTP** (RFC 2616) και των μεθόδων που αυτό υποστηρίζει, όπως αναφέρονται παρακάτω [2] :

- **GET**
  Τα GET requests χρησιμοποιούνται για την απεικόνιση των πόρων του server. Τα ερωτήματα αυτά απο τη μεριά του server πρέπει μόνο να ανακτούν τα δεδομένα των πόρων, χωρίς να έχουν κάποια άλλη παρενέργεια ή συνέπεια στην κατάσταση τους (**safe-read requests**). Τέτοιου είδους ερωτήματα συνήθως δεν περιέχουν request payload και οι επαναλαμβανόμενες κλήσεις τους δεν πρέπει να επηρεάζουν το αποτέλεσμα των πόρων στον server (**idempotent** requests).

- **POST**
  Με ένα POST request ο client έχει τη δυνατότητα να στείλει δεδομένα στον server (έναντι της μεθόδου GET που διέπραττε το αντίστροφο), ώστε να δημιουργήσει μια νέα οντότητα στην βάση δεδομένων (απεικόνιση κάποιου αντικειμένου). Ο server διαβάζει το request body, πιστοποιεί αν τηρεί τις προδιαγραφές (δομή, τύπος δεδομένων κλπ), εκτελεί τον κώδικα της κρίσιμης λογικής (business logic) και απαντάει στο χρήστη με κάποιο response body, το οποίο πιθανόν να περιέχει επιπρόσθετα properties.

- **PUT**
  Για την τροποποίηση ενός ήδη υπάρχοντος πόρου (**Update/Replace**). Η μέθοδος απαιτεί το αναγνωριστικό κλειδί του πόρου (id), ώστε ο server να μπορεί να βρεί τη συγκεκριμένη εγγραφή απο την βάση δεδομένων και ολόκληρο το request body με ανανεωμένα μόνο τα properties της οντότητας που επιθυμεί ο client να αλλάξει. Αν κάποιο property δεν συμπεριληφθεί στο payload, η τιμή του θα αντικατασταθεί με 0 ή null.

- **DELETE**
  Για την διαγραφή μιας οντότητας απο την βάση δεδομένων (**idempotent** requests).

- **PATCH**
  Για την μερική τροποποίηση μιας εγγραφής (**partial update/Modify**). Εδώ σε αντίθεση με τα PUT requests, τα request data του client περιλαμβάνουν μόνο τα properties τα οποία επιθυμεί να τροποποιήσει, ενώ τα υπόλοιπα διατηρούν την προηγούμενη τιμή τους.

Όλες οι απαντήσεις του server συνοδεύονται με συγκεκριμένο HTTP status code, ανάλογα με την λειτουργικότητα και την επιτυχία του κάθε request. Οι κωδικοί αυτοί χωρίζονται σε 5 κλάσεις, όπως φαίνεται παρακάτω :

- **1xx** – Informational response
- **2xx** – Success
- **3xx** – Redirection
- **4xx** – Client Errors
- **5xx** – Server Errors

Ακολουθώντας το πρότυπο του REST, ο server της παρούσας εφαρμογής απαντά με **200 (OK)** όταν ένα request πραγματοποιηθεί επιτυχώς, με **201 (Created)** μετά το επιτυχές αποτέλεσμα της δημιουργίας μιας νέας οντότητας στην βάση δεδομένων, **με 204 (No Content)** όταν αφαιρείται επιτυχώς ένας πόρος απο την συλλογή των καταγραφών (resource collection), με **404 (Resource Not Found)** όταν ο client ζητάει την απεικόνηση ενός πόρου με απροσδιόριστο ή ανύπαρκτο στη βάση δεδομένων path παράμετρο (μοναδικός κωδικός για κάθε entity), με **400 (Bad Request)** όταν οποιαδήποτε παράμετρος του request περιέχει μη έγκυρο τύπο δεδομένων ή έρχεται σε αντίθεση με το business logic της εφαρμογής και τέλος με **500 (Internal Server Error)** όταν παρουσιαστεί κάποια απροσδόκητη συμπεριφορά στον server. Συνήθως σε αυτήν την περίπτωση, το response περιέχει ένα κάποιο αφηρημένο μήνυμα ή το stack-trace του runtime exception.

2.1.1 JSON

Όπως αναφέρθηκε παραπάνω, στην παρούσα εφαρμογή ο server μέσω του REST API θα αναπαριστά τα ζητούμενα δεδομένα στον client στη μορφή JSON.

Το JSON (JavaScript Object Notation) είναι ένας ελαφρύς μορφότυπος, ο οποίος χρησιμοποιείται για την ανταλλαγή δεδομένων. Έχει μορφή η οποία είναι κατανοητή στο ανθρώπινο μάτι και είναι βασισμένο στο ECMAScript. Για αυτό το λόγο, έχει πλεόν αντικαταστήσει το XML, το οποίο χρησιμοποιούνταν κατά κόρον στο παρελθόν για τη σειριοποίηση και αποσειροποίηση δεδομένων στον server. Το JSON ενώ δεν μπορεί να θεωρηθεί γλώσσα προγραμματισμού, χρησιμοποιεί συμβάσεις οι οποίες είναι κοινές σε πολλές γώσσες καθιστώντας το ως ένα απο τα ιδανικότερα μέσα ανταλλαγής δεδομένων [3].

Το JSON αποτελείται απο δύο βασικές δομές δεδομένων : τα αντικέιμενα (Objects) και τους πίνακες (Arrays). Τα αντικείμενα είναι μη ταξινομημένα σύνολα αποτελούμενα απο ζεύγη ονομάτων και τιμών, ενώ οι πίνακες είναι σύνολα απο τιμές. Μια τιμή μπορεί να είναι είτε κείμενο (σειρά απο χαρακτήρες) είτε αριθμός είτε λογική τιμή Boolean είτε και αντικείμενο ή πίνακας.

2.1.2 Jackson Σειριοποίηση/Αποσειριοποίηση Δεδομένων

Για τη διευκόλυνση της σειριοποίησης και αποσειριοποίησης αντικειμένων στον server, στην παρούσα εφαρμογή χρησιμοποιήθηκε η δεύτερη έκδοση της βιβλιοθήκης Jackson. Ο τρόπος που το επιτυγχάνουμε αυτό, είναι να αντιστοιχίσουμε κάθε πεδίο των επιθυμητών κλάσεων σε μία συμβολοσειρά που αντιπροσωπεύει το πεδίο του αντικειμένου σε μορφή JSON. Έτσι, έχοντας εφαρμόσει την κατάλληλη αντιστοίχηση, όταν ο Server λαμβάνει ένα HTTP request (POST ή PUT συνήθως) που περιέχει ένα ή περισσότερα αντικείμενα στο payload, μπορούμε με την κλήση μόνο μίας μεθόδου, να μεταβούμε απο JSON Objects σε Java Classes (POJOs – Plain Old Java Object). Κατα την σειριοποίηση αυτή, έχουμε πλεόν τα δεδομένα του χρήστη σε Java αντικείμενα, τα επεξεργαζόμαστε κατάλληλα ακολουθώντας την κρίσιμη λογική του προγράμματος και στη συνέχεια γίνονται οι απαραίτητες ενέργειες με τη βάση δεδομένων.

Αντίστοιχα, κατα την αποσειριοποίηση των δεδομένων, ο Server συλλέγει τους ζητούμενους πόρους απο τη βάση δεδομένων σε Java Objects και τους παρέχει στον χρήστη σε μορφή JSON.

Στην περίπτωση που επιθυμούμε να αγνοήσουμε ορισμένα πεδία στο response body, είτε επειδή εχουν τιμή null είτε επειδή είναι αχρείαστα metadata, μπορούμε να χρησιμοποιήσουμε την επισήμανση @JsonIgnore ή @JsonInclude στα properties της αντίστοιχης Java κλάσης.

2.2 MySQL Βάση Δεδομένων

Η MySQL είναι ένα ελεύθερο σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων (Relational Database Management System - RDBMS) που είναι ευρύτατα διαδεδομένη στα UNIX συστήματα. Είναι πολυνηματική και πολυχρηστική και υποστηρίζει τα τελευταία standards της SQL (Structured Query Language), η οποία αποτελεί την πιο κοινή τυποποιημένη γλώσσα επερωτήσεων για την προσθήκη, την πρόσβαση και την επεξεργασία δεδομένων σε μία Βάση Δεδομένων. Η MySQL έχει σχεδιαστεί και έχει βελτιστοποιηθεί για διαδικτυακές εφαρμογές.

Βασικά χαρακτηριστικά και πλεονκετήματα που την απαρτίζουν είναι τα ακόλουθα [4] :

1) Ασφάλεια Δεδομένων. Η αξιοπιστία της σε κρίσιμες συναλαγές πληροφοριών και χρημάτων έχει αποδειχθεί, αφού χρησιμοποιείται πλήρως απο τους πιο γνωστούς παγκόσμια διαδυκτιακούς ιστότοπους όπως είναι το Facebook, το Twitter και το WordPress.
2) On – Demand Scalability. Μπορεί να διαχειριστεί μεγάλο όγκο συναλαγών, κλιμακώνοντας το φορτίο, όσον αφορά το διάβασμα, το γράψιμο και την επεξεργασία των SQL ερωτημάτων, ανάλογα με τις εμπορικές ανάγκες της κάθε επιχείρησης.
3) Υψηλή απόδοση.
4) Χαμηλό κόστος.
5) Μεταφερσιμότητα. Μπορεί να χρησιμοποιηθεί σε διαφορετικά συστήματα UNIX όπως επίσης και στα Microsoft Windows.
6) Πηγαίος Κώδικας. Είναι δυνατή η κατάκτηση και η τροποποίηση του κώδικα προέλευσης της MySQL.

Κεφάλαιο 3

Επιλογή Εργαλείων

Στο παρόν κεφάλαιο γίνεται μια αναφορά αρχικά στην αρχιτεκτονική του συστήματος και στη συνέχεια στα εργαλεία ανοιχτού κώδικα που είναι διαθέσιμα στην αγορά και μπορούν να χρησιμοποιηθούν στην ανάπτυξη των βασικών τμημάτων του προγράμματος.

Η εφαρμογή χωρίζεται σε τρία βασικά τμήματα :

1) το Front-End localhost:3000.
2) ένα REST API το οποίο αναλαμβάνει τον ρόλο του Back-End μαζί με τον διακομιστή Tomcat localhost:8181.
3) την βάση δεδομένων jdbc:mysql://localhost:3306/thesisDB.

Το Front-End αποτελεί το γραφικό περιβάλλον και παρέχει τη δυνατότητα στον χρήστη να δημιουργεί με εύκολο τρόπο tables στην βάση δεδομένων με όσα columns εκείνος χρειάζεται, ενώ παράλληλα τον αποτρέπει να κάνει λάθος στον σχεδιασμό (error handling). Ακόμα, του παρέχει ένα αναλυτικό ιστορικό με το τι έχει φτιάξει και τη δυνατότητα αναζήτησης/διαγραφής μεμονωμένου table απο την βάση δεδομένων. Αυτό το περιβάλλον θα έχει τη μορφή ιστοσελίδας.

2. Full Stack Architecture

Το Back-End θα έχει τη μορφή ενός REST API και θα είναι υπεύθυνο να λαμβάνει τις επιλογές του χρήστη απο το Front-End ώστε να εκτελεί τα κατάλληλα transactions με την βάση δεδομένων. Η μορφή του API γίνεται ιδιαίτερα εμφανής στην Java κλάση Controller, η οποία περιλαμβάνει όλα τα εκτεθειμένα endpoints στο UI και έχει την επισήμανση @RestController. Η λειτουργία της θα περιγραφεί αναλυτικά στο κεφάλαιο 4.

Ως γλώσσα ανάπτυξης του συστήματος για τον διακομιστή (server-side) επιλέχθηκε η Java, ενώ για τον πελάτη (client-side) επιλέχθηκε η JavaScript.

3.1 Εργαλεία Ανάπτυξης Front-End

3.1.1 ReactJS

Η React (ReactJS) είναι μια JavaScript βιβλιοθήκη, η οποία ειδικεύεται στη δημιουργία διαδραστικών διεπαφών χρήστη (UIs) δυναμικού χαρακτήρα με εξαιρετικά γρήγορη απόδοση, ενώ συντηρείται και συνεχίζει να αναπτύσεται απο κοινότητα εταιριών και χρηστών, ως έργο ανοιχτού κώδικα.

Μερικά χαρακτηριστικά της React τα οποία την ξεχωρίζουν απο τα υπόλοιπα frameworks, είναι τα εξής [5]:

- Το virtual DOM της React την κάνει εξεραιτικά γρήγορη, αφού μπορεί να κάνει ανανέωση ένα μέρος της σελίδας, αντί για το συμβατικό μοντέλο πλήρους ανανέωσης (conventional full refresh model).
- Εύκολη δημιουργία ποικίλων Test Cases στο UI .
- Εύκολη επαναχρησιμοποίηση των Components του κώδικα.
- Μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών Android και iOS (React Native – Hybrid Applications).
- Προσφέρει one-way data binding απο το parent component στο child component, αποφεύγοντας έτσι ανεπιθύμητες και απρόβλεπτες ενέργειες.
- Έχει τεράστια στήριξη βοηθητικού υλικού τόσο απο επίσημες πηγές (Facebook), όσο και απο την κοινότητα.
- Βελτιώνει αποτελεσματικά τον χρόνο στο debugging και με διάφορες επεκτάσεις του Chrome το κάνει ευκολότερο.
- Χρησιμοποιείται και για server-side ανάπτυξη.
- Μπορεί να απεικονίσει τεράστιο αριθμό Components γρήγορα και αποτελεσματικά.
- Τα Components μπορούν γραφούν με τη χρήση της JSX (JavaScript Syntax Extension), η οποία αποτελεί μία επέκταση της JavaScript, χρησιμοποιώντας σύνταξη παρόμοια με την HTML.
- Μπορεί να χρησιμοποιηθεί με την TypeScript και την Flow.

Απο την άλλη πλευρά, η React χαρακτηρίζεται και απο κάποια μειονεκτήματα που θέτουν περιορισμούς στη χρήση της, όπως αναφέρεται παρακάτω :

- Η μικτή σύνταξη της React (JavaScript/JSX - HTML) μπορεί να αποθαρρύνει ή να συγχίσει τους προγραμματιστές, αφού απαιτεί βαθιά κατάνοηση λόγω της πολυπλοκότητας της.
- Η χρήση του virtual DOM επιφέρει αυξημένες ανάγκες μνήμης (RAM), αφού κρατιέται διαρκώς ένα αντίγραφο του DOM στη μνήμη.

3.1.2 NodeJS

Το Node.js είναι μια ανοιχτού κώδικα πλατφόρμα ανάπτυξης λογισμικού βασισμένη στην JavaScript. Παρέχει ένα περιβάλλον εκτέλεσης JavaScript έξω από τον περιηγητή ιστού, βασισμένο στο Google V8 JavaScript Engine. Το V8 JavaScript Engine είναι μία μηχανή εκτέλεσης JavaScript ανοιχτού κώδικα, η οποία βρίσκεται πίσω απο το Chrome και όλους τους υπόλοιπους περιηγητές ιστού βασισμένους στο Chromium [6]. Στόχος του Node είναι να παρέχει ένα εύκολο τρόπο δημιουργίας κλιμακωτών διαδικτυακών εφαρμογών. Σε αντίθεση απο τα περισσότερα σύγχρονα περιβάλλοντα ανάπτυξης εφαρμογών δικτύων, μία διεργασία node δεν στηρίζεται στην πολυνηματικότητα αλλά σε ένα μοντέλο ασύγχρονης επικοινωνίας εισόδου/εξόδου.

3.1.3 Node Package Manager

Το npm είναι ο προκαθορισμένος διαχειριστής πακέτων (package manager) για τον node-js server. Αποτελείται απο μία διεπαφή γραμμής εντολών (CLI) και μια online βάση δεδομένων. Το εργαλείο γραμμής εντολών επικοινωνεί με την βάση δεδομένων, η οποία παρέχει δημόσια ή ιδιόκτητα πακέτα nmp. Το npm προσφέρει στον προγραμματιστή τη δυνατότητα να δημοσιεύει και να μοιράζεται τον κώδικά του σε μορφή πακέτων (modules). Τέλος, έχει σχεδιαστεί για να διευκολύνει το κατέβασμα, την εγκατάσταση και την συντήρηση των βιβλιοθηκών JavaScript [7].

Για να χρησιμοποιήσουμε οποιαδήποτε JavaScript βιβλιοθήκη, αρκεί να πληκτρολογήσουμε στο εργαλείο γραμμής εντολών την εντολή **npm install <package/module name>**.

3.1.4 Material-UI

Το Material-UI είναι μία JavaScript βιβλιοθήκη ανοιχτού κώδικα, η οποία διαθέτει React components που υλοποιούν το Material Design της Google. Η πρώτη της έκδοση δημοσιεύτηκε το 2014 και απο τότε αποτελεί μία απο τις κορυφαίες UI βιβλιοθήκες για την React. Προσφέρει τη δυνατότητα δημιουργίας δυναμικών και καλά σχεδιασμένων ιστοσελίδων χωρίς την χρήση CSS απο τον προγραμματιστή. Τα components είναι αυτοϋποστηριζόμενα και μπορούν να εισαχθούν ένα προς ένα σε μία ιστοσελίδα μειώνοντας έτσι το μέγεθος των εισαγωγών (imports) [8].

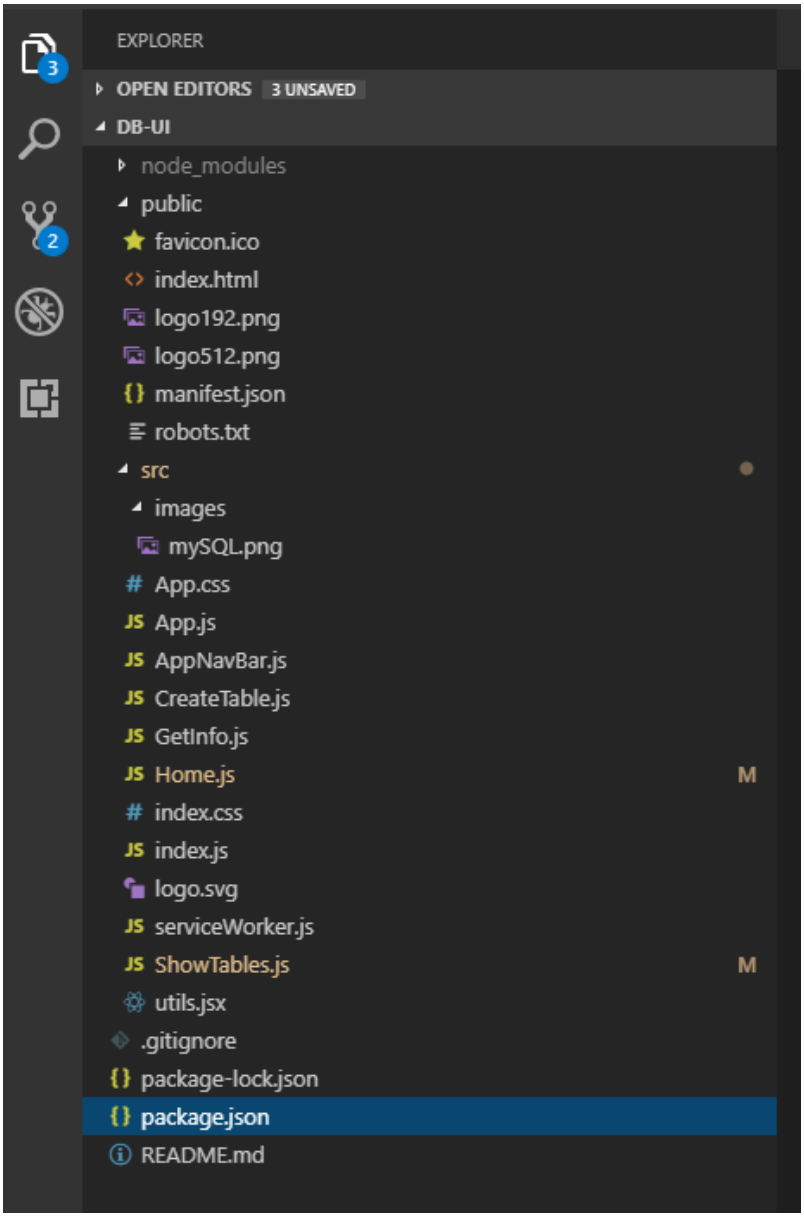
Για την ανάπτυξη της διεπαφής χρήστη αυτής της εφαρμογής, χρησιμοποιήθηκαν τα παρακάτω Material-UI components [9]:

<MuiThemeProvider>, <AppBar>, <Toolbar>, <Typography>, <InputBase>, <HomeIcon>, <SearchIcon>, <Tooltip>, <Link>, <ErrorIcon>, <CloseIcon>, <IconButton>, <Snackbar>, <SnackbarContent>, <GridOnIcon>, <Table>, <TableBody>, <TableCell>, <SaveIcon>, <TableRow>, <Paper>, <MenuItem>, <Checkbox>, <AddBoxIcon>, <IconButton>, <DeleteIcon>, <Container>, <TableHead>, <TableRow>, <EditIcon>, <Grid>, <Button>, <ListSubheader>, <ListItem>, <ListItemText>, <Collapse>, <ExpandLess>, <ExpandMore>, <VpnKeyRoundedIcon>, <Divider>, <ListItemAvatar>, <TabletRoundedIcon>.

Παρακάτω, μπορούμε να δούμε εύκολα όλα τα dependencies του project, δηλαδή όλες τις εξωτερικές JavaScript βιβλιοθήκες που χρησιμοποιεί το Front-End.

package.json

```
{
  "name": "db-ui",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@material-ui/core": "^4.3.3",
    "@material-ui/icons": "^4.2.1",
    "@material-ui/styles": "^4.3.3",
    "material-table": "^1.50.0",
    "material-ui": "^0.20.2",
    "object-path-immutable": "^3.1.1",
    "react": "^16.9.0",
    "react-dom": "^16.9.0",
    "react-router-dom": "^5.0.1",
    "react-scripts": "3.1.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "proxy": "http://localhost:8181",
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```



3. Front-End Folder Structure

Εδώ βλέπουμε την δομή όλων των JavaScript αρχείων του Front-End της εφαρμογής.

- **node_modules**

Περιλαμβάνει κάθε JavaScript βιβλιοθήκη και διαμόρφωση (configuration).

- **src**

Όλα τα **React Components** της εφαρμογής.

3.2 Εργαλεία Ανάπτυξης Back-End (REST API)

3.2.1 Gradle

Το Gradle είναι ένα build-automation software εργαλείο, το οποίο βασίζεται στις γλώσσες Groovy και Kotlin. Το Gradle υποστηρίζει την αυτόματη λήψη και παραμετροποίηση (configuration) των java βιβλιοθηκών (dependencies) του project. Όλες αυτές τις βιβλιοθήκες τις κατεβάζει απο τα Apache Ant και Apache Maven repositories, χρησιμοποιώντας μία domain-specific γλώσσα (DSL) βασισμένη στην Groovy, αντί της XML φόρμας που χρησιμοποιεί το Maven (pom.xml) για την όλη διαχείρηση/διαμόρφωση του project [10].

Η κύρια λειτουργία του Gradle είναι να κάνει build ένα project και να διαχειρίζεται το classpath αυτού. Ένα project αποτελείται απο tasks. Κάθε task αναπαριστά ένα κομμάτι της διαδικασίας που εκτελεί το build του project, όπως για παράδειγμα, την μεταγλώττιση του πηγαίου κώδικα ή την δημιουργία ενός Javadoc.

Ένα project που χρησιμοποιεί Gradle, περιγράφει το build του, μέσω του αρχείου **build.gradle**. Το αρχείο αυτό βρίσκεται στο root directory του project και μπορεί να δημιουργεί και να επεκτείνει δυναμικά οποιοδήποτε task κατά ττην διαδικασία εκτέλεσης της εφαρμογής.

Δηλώνοντας σε αυτό το αρχείο, το target URL, δηλαδή το απομακρυσμένο repository απο το οποίο το Gradle θα κατεβάσει τις Java βιβλιοθήκες που του έχουμε ορίσει μοναδικά με ένα **groupId**, **artifactId** και **version**, μπορούμε να έχουμε τον πλήρη έλεγχο και ότι πληροφορία υπάρχει απο τις επίσημες πηγές (Java API - docs) για τα χαρακτηριστικά και τις αλλαγές των κλάσσεων και μεθόδων που χρησιμοποιούμε στο project, όπως φαίνεται στο παρακάτω παράρτημα κώδικα.

build.gradle

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:2.1.3.RELEASE")
    }
}

apply plugin: 'java'
apply plugin: 'org.springframework.boot'
apply plugin: 'io.spring.dependency-management'

group 'org.anastasakis'
version '1.0-SNAPSHOT'

sourceCompatibility = 1.8

/**
   Target URL
   Represents the remote Maven repository
   from which Gradle searches and downloads the below dependencies
*/
repositories {
    mavenCentral()
}

/**
   A Java library is identified by Gradle via
   its project's groupId:artifactId:version (also known as GAV in Maven).
   This GAV uniquely identifies a library in a certain version.
*/
dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
    compile("mysql:mysql-connector-java:8.0.15")
    compile("org.jooq:jooq:3.11.12")
    compile("org.jooq:jooq-meta:3.11.12")
    compile("com.fasterxml.jackson.core:jackson-databind:2.9.7")
}
```

3.2.2 Spring Boot

Το Spring Boot είναι ένα framework ανοιχτού κώδικα βασισμένο στην Java, το οποίο χρησιμοποιείται για την ανάπτυξη ενός micro Service. Χρησιμοποείται πλέον απο τις περισσότερες εταιρίες, στις οποίες η πολυπλοκότητα του προϊόντος απαιτεί τον χωρισμό του σε μικρότερα μεμονωμένα κομμάτια. Με το spring Boot μπορεί κάποιος να χτίσει μία stand-alone και production ready εφαρμογή, η οποία θα αποτελείται απο ξεχωριστά micro services και το καθ' ένα απο αυτά θα λειτουργεί ανεξάρτητα απο το άλλο. Αυτό σημαίνει ότι η αρχιτεκτονική του κάθε micro Service θα επιτρέπει στους προγραμματιστές να αναπτύσσουν και να κάνουν deploy τα services ανεξάρτητα. Κάθε service που τρέχει στον server έχει την δική του διαδικασία και λειτουργικότητα και αυτός είναι ο λόγος που καθιστά το Spring Boot αναγκαίο για την ανάπτυξη εμπορικών εφαρμογών. Έτσι, η κάθε εταρεία μπορεί να χωρίσει την ανάπτυξη του back-end σε διαφορετικές ομάδες προγραμματιστών ανάλογα με τις παρούσες ανάγκες της.

Τα micro services προσφέρουν στους προγραμματιστές εύκολο deployment, απλή κλιμάκωση, ελάχιστη διαμόρφωση/παραμετροποίηση και λιγότερο χρόνος παραγωγής του project.

Ο κύριος στόχος του Spring Boot είναι να μειώσει αισθητά τον χρόνο της ανάπτυξης της εφαρμογής/micro-service και του Unit/Integration testing αυτής. Κάποια απο τα βασικά χαρακτηριστικά του είναι τα εξής [11]:

- **Opinionated Defaults Configuration** προσέγγιση. Με το Spring Boot είναι εύκολο πλέον για τον προγραμματιστή να στήσει και να αναπτύξει Java Spring εφαρμογές, αυξάνωντας την παραγωγικότητα και μειώνοντας αισθητά τον χρόνο για την ανάπτυξη επαναλαμβανόμενου ή περίπλοκου κώδικα όπως για παράδειγμα, το XML/Java Beans configuration και τα transactions με τη βάση δεδομένων. Το Spring Boot έχει χτιστεί πάνω στο Spring Framework με αποτέλεσμα να αποφεύγεται ο boilerplate κώδικας και τα χειροκίνητα configurations.

- Η ενσωμάτωση του με το Spring οικοσύστημα (Spring JDBC, Spring Data, Spring JPA, Spring Security κλπ).

- Παρέχει ενσωματωμένους HTTP servers (Tomcat/Jetty), ώστε ο προγραμματιστής να αναπτύσει και να τεστάρει ευκολότερα την διαδικτυακή εφαρμογή, διαχειρίζοντας κατάλληλα τα REST API endpoints αυτής.

- Παρέχει μία διεπαφή γραμμής εντολών και ποικίλα plugins, μέσω των οποίων είναι εύκολο το testing και η ανάπτυξη Spring Boot εφαρμογών, χρησιμοποιώντας build εργαλεία όπως το Gradle ή το Maven.

- Προσφέρει annotation-based Spring εφαρμογές και περιλαμβάνει ενσωματωμένο Servlet Container.

- **Dependency Injection**. Αποτελεί μία απο τις κυριότερες λειτουργίες του Spring IOC (Inversion of Control) και επιλύει το πρόβλημα της εξάρτησης μεταξύ κλάσεων-αντικειμένων. Πιο συγκεκριμένα, όταν σε διαφορετικά σημεία του κώδικα μίας εφαρμογής χρειαζόμαστε να χρησιμοποιήσουμε την λειτουργικότητα μίας συγκεκριμένης κλάσης που προσφέρουν οι μεθόδοι της, θα πρέπει να αρχικοποιούμε κάθε φορά πολλά instances-objects απο αυτή. Κάτι τέτοιο προσδίδει σύγχιση στον κώδικα, ενώ είναι αρκετά εύκολο να γίνει λάθος σε μεγάλες εφαρμογές όταν τα αντικείμενα αυτά έχουν εξάρτηση μεταξύ τους. Αυτό αποφεύγεται πλήρως με το Dependency Injection (Constructor/Setter/File based), αφού η δημιουργία (injection) αντικειμένων σε άλλα αντικείμενα γίνεται αυτόματα με την επισήμανση @Autowired του Spring framework [12].

Όπως φαίνεται στο build.gradle αρχείο, το παρών project χρησιμοποιεί όλες τις βιβλιοθήκες του Spring Boot framework που χρειάζεται μέσω του dependency **spring-boot-starter-web**. Μέσω αυτού, το project εισάγει τα απαραίτητα packages, ώστε να δημιουργήσει τα REST endpoints στον Tomcat server (προκαθορισμένος ενσωματωμένος container) με την ελάχιστη παραμετροποίηση.

3.2.3 JOOQ

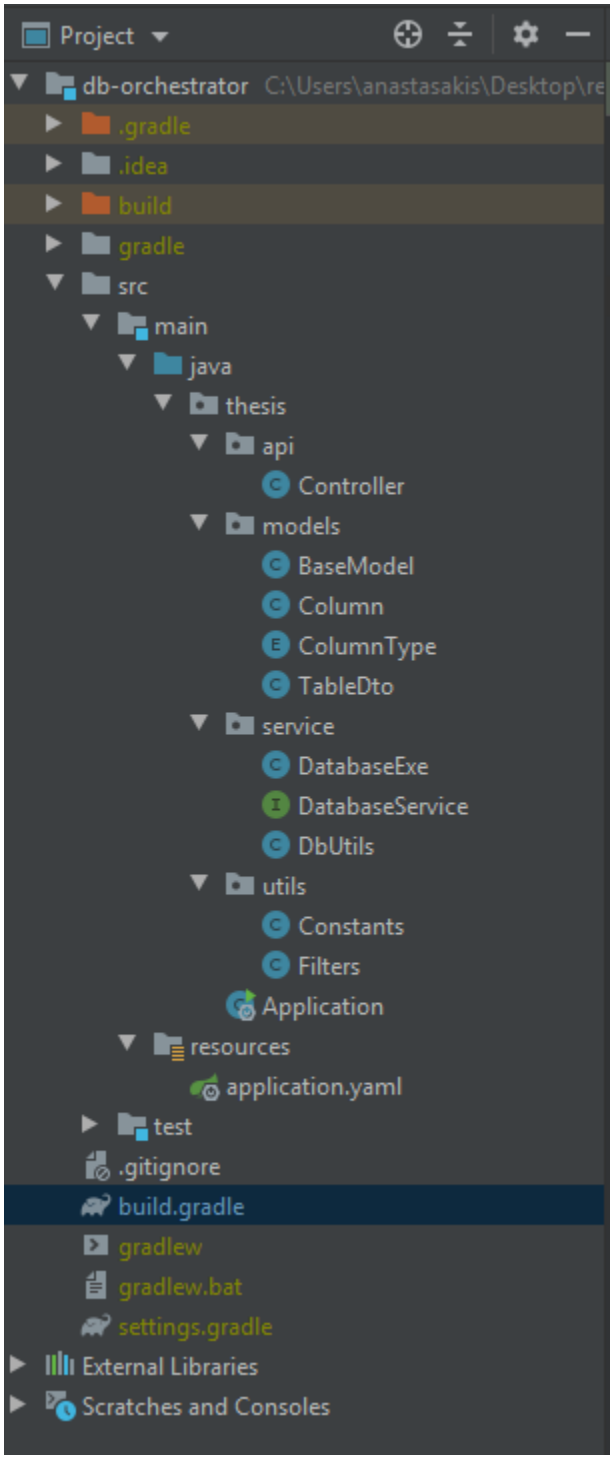
Το jOOQ Object Oriented Querying είναι μία Java βιβλιοθήκη ανοιχτού κώδικα, η οποία είναι υπεύθυνη για την αποθήκευση δεδομένων ως αντικείμενα σε σχεσιακές βάσεις (**active record pattern**), όπως η mySQL. Αποτελεί ένα high level fluent API, μέσω του οποίου ο προγραμματιστής μπορεί με εύκολο και ευανάγνωστο τρόπο να χτίσει την επικοινωνία του server-side κώδικα με την βάση δεδομένων. Το jOOQ χρησιμοποιεί κατά κόρον το JDBC API (Java Database Connectivity) και παρέχει μία domain-specific γλώσσα (DSL) για την κατασκευή SQL ερωτημάτων (queries) απο jOOQ αντικείμενα (Java κλάσεις) και απο διαμορφωμένο κώδικα απο το schema της βάσης δεδομένων (meta-data). Η ανάγκη δημιουργίας του jOOQ API εμφανίστηκε όταν πολλά χαρακτηριστικά της SQL δεν μπορούσαν να αξιοποιηθούν απο αντικειμενοστραφείς γλώσσες προγραμματισμού, όπως η Java, αλλά και απο σχεσιακές βάσεις δεδομένων [13].

Έτσι ο προγραμματιστής, μέσω Java κλάσεων μπορεί να :

1) Αποφύγει συντακτικά λάθη κατά την εκτέλεση των ερωτημάτων ή προβλήματα αντιστοίχησης τύπων δεδομένων (Data mapping).
2) Δημιουργήσει σύνθετα και συνδυαστικά SQL ερωτήματα τα οποία να περιλαμβάνουν enum types, aliasing, unions, εμφωλευμένα selects και περίπλοκα joins.
3) Δημιουργήσει stored procedures και native functions.

Οι περισσότερες Spring εφαρμογές χρησιμοποιούν το JPA (Java Persistence API) για να μεταφέρουν τα δεδομένα του χρήστη απο και πρός την βάση δεδομένων, μέσω προκαθορισμένων entities (Java Beans/Models classes). Στην παρούσα εφαρμογή κάτι τέτοιο δεν υφίσταται, αφού τα δεδομένα του χρήστη αφορούν το schema της βάσης και όχι στοιχεία τα οποία θα αποθηκευτούν σε υπάρχοντα tables. Έτσι, με ειδικά διαμορφωμένα POJOs για την μεταφορά αυτή, δηλαδή Java κλάσεις (Data Transfer Objects), το jOOQ λαμβάνει τις απαραίτητες τιμές, τις επεξεργάζεται και δημιουργεί το ζητούμενο table.

Παρακάτω, βλέπουμε αντοίστιχα την δομή όλων των αρχείων του back-end project.



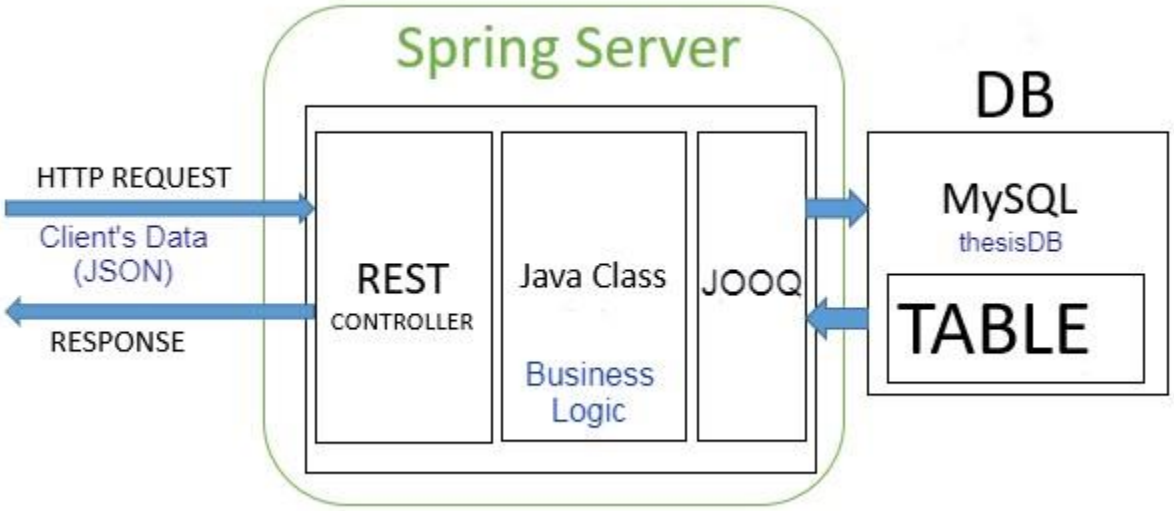
4. Back-End Folder Structure

Κεφάλαιο 4

Υλοποίηση – Λειτουργία Συστήματος

Στο παρών κεφάλαιο επεξηγείται η υλοποίηση των βασικών κομματιών της εφαρμογής και δίνεται ένα παράδειγμα λειτουργίας της. Όλη η ανάπτυξη του back-end συστήματος έγινε στο Intellij Integrated Development Environment (Java έκδοση : **jdk 1.8.0_191**), ενώ του front-end στο Visual Studio Code.

4.1 Υλοποίηση Server (Back-End)



5. RESTful API Design

Κατά την εκκίνηση του back-end συστήματος σηκώνεται ο Tomcat server στην πόρτα 8181 του τοπικού διακομιστή (localhost), όπως έχουμε ορίσει στο αρχείο **application.yaml** (configuration file) και έτσι όλα τα services (API endpoints) είναι διαθέσιμα απο εκεί (**baseUri** = http://localhost:8181).

Application.java

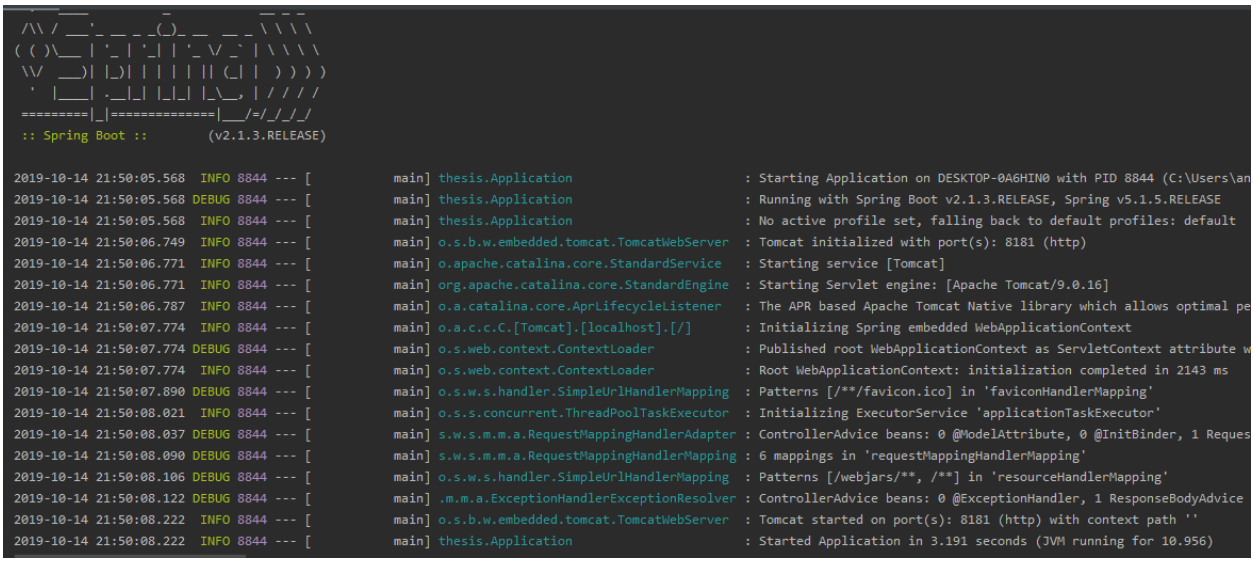
```
package thesis;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Η κλάση Application χρησιμοποιείται για την εκκίνηση της Spring REST εφαρμογής μέσω της Java main μεθόδου, όπως βλέπουμε παραπάνω, αφού πρώτα δημιουργήσει το ApplicationContext (πρόσβαση στον Spring Container ώστε να μπορούμε να χρησιμοποιήσουμε τη λειτουργικότητα του Dependency Injection) απο το classpath του πρότζεκτ και σκανάρει όλες τις κλάσεις που είναι υπεύθυνες για κάθε πιθανό configuration.

Στο παρακάτω στιγμιότυπο βλέπουμε τα logs του server στην κονσόλα του Intellij IDE, τη στιγμή που εκκινείται η εφαρμογή.



6. Intellij Console

application.yaml

```yaml
server:
  port: 8181
logging:
  level:
    thesis: DEBUG
    org:
      springframework:
        web: DEBUG
datasource:
  url: jdbc:mysql://localhost:3306/thesisDB
  username: supadmin
  password: root
```

Το Spring Boot κατά την εκκίνησή του, θα φορτώσει και θα αναλύσει (parsing) όλα τα properties του αρχείου application.yaml το οποίο βρίσκεται κάτω απο το /src/resources package. Έτσι, γνωρίζει εξ'αρχής ότι :

1) Η πόρτα που θα ακούει ο tomcat server είναι η 8181.
2) Όλα τα προκαθορισμένα logs του Spring framework (HTTP request/response, transactions με την βάση δεδομένων κλπ) που είναι σε DEBUG level θα εμφανίζονται στην κονσόλα.
3) Όπου έχουμε χρησιμοποιήσει τον προκαθορισμένο Logger του Spring Boot, τα μηνύματα θα εμφανίζονται με την σημαία DEBUG (root package level : **thesis**). Έτσι κάθε φορά που ο χρήστης θα προσπαθεί να δημιουργήσει ένα table (Controller.java), θα βλέπουμε το request body που έχει στείλει στα logs του server, όπως φαίνεται παρακάτω στιγμιότυπο :

```
2019-10-15 23:01:46.004 DEBUG 15820 --- [nio-8181-exec-7] o.s.web.servlet.DispatcherServlet        : POST "/api/v1.0/schema/tables", parameters={}
2019-10-15 23:01:46.006 DEBUG 15820 --- [nio-8181-exec-7] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped to public org.springframework.http.ResponseEntity thesis.api.Controller
2019-10-15 23:01:46.067 DEBUG 15820 --- [nio-8181-exec-7] m.m.a.RequestResponseBodyMethodProcessor : Read "application/json;charset=UTF-8" to [thesis.models.TableDto@686e40ac]
2019-10-15 23:01:46.162 DEBUG 15820 --- [nio-8181-exec-7] thesis.api.Controller                    : ------------------  REQUESTED TABLE
 {
  "name" : "product",
  "columns" : [ {
    "name" : "product_id",
    "type" : "LONG",
    "primaryKey" : true
  }, {
    "name" : "description",
    "type" : "STRING"
  }, {
    "name" : "suppliers",
    "type" : "INTEGER"
  }, {
    "name" : "price",
    "type" : "DOUBLE"
  }, {
    "name" : "is_sold",
    "type" : "BOOLEAN"
  } ]
}
```

7. Requested table - DEBUG

Controller.java

```
package thesis.api;

import thesis.models.TableDto;
import thesis.service.DatabaseService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import javax.validation.Valid;
import java.io.IOException;
import java.util.List;
import static org.springframework.http.MediaType.APPLICATION_JSON_VALUE;

@RestController
@RequestMapping("/api/v1.0/schema/tables")
public class Controller {

    private Logger log = LoggerFactory.getLogger(Controller.class);

    @Autowired
    @Qualifier("databaseExe")
    private DatabaseService service;

    @PostMapping(consumes = APPLICATION_JSON_VALUE)
    public ResponseEntity createTable(@RequestBody @Valid TableDto table)
            throws IOException {
        log.debug("------------------  REQUESTED TABLE\n {} ", table.toJsonString());
        return service.createTable(table);
    }

    @GetMapping(path = "/{name}")
    public ResponseEntity describe(@PathVariable String name) {
        return service.getInfo(name);
    }

    @DeleteMapping(path = "/{name}")
    public ResponseEntity dropTable(@PathVariable String name) {
        return service.dropTable(name);
    }

    @GetMapping
    public List<TableDto> showTables() {
        return service.showTables();
    }
}
```

Στην παραπάνω κλάση βρίσκονται όλα τα διαθέσιμα HTTP API endpoints στο UI, δηλαδή όλοι οι πόροι που παρέχονται στο front-end.

Πιο συγκεκριμένα,

- **POST /api/v1.0/schema/tables**

  Το endpoint αυτό δέχεται ως request body ένα αντικείμενο (instance) της κλάσης **TableDto.java** σε σειριοποιημένη μορφή (απο JSON σε Java class) το οποίο αποτελεί την απεικόνιση του προς-δημιουργία table. Στη συνέχεια εκτελείται η μέθοδος createTable(TableDto tableDto) της κλάσης **DatabaseExe.java** όπου γίνεται η τελική επεξεργασία των δεδομένων του χρήστη, η οποία θα αναλυθεί παρακάτω.

  Παράδειγμα μορφής ενός σωστού (@Valid) request payload που απαιτεί η μέθοδος αυτή είναι η εξής:

```
{
   "name":"employee",
   "columns":[
      {
         "name":"employee_id",
         "type":"LONG",
         "primaryKey":true
      },
      {
         "name":"username",
         "type":"STRING"
      },
      {
         "name":"phone_number",
         "type":"LONG"
      },
      {
         "name":"salary",
         "type":"DOUBLE"
      },
      {
         "name":"bonus",
         "type":"BOOLEAN"
      },
      {
         "name":"age",
         "type":"INTEGER"
      }
   ]
}
```

8. Δείγμα ζητούμενου table σε μορφή JSON

Η μέθοδος απαιτεί τα ακόλουθα request Headers :

1) **Content-Type** = application/json λόγω του JSON request payload
2) **accept** = application/json λόγω του JSON response του server.

Στο response δεν χρειάζεται να δημιουργηθεί κάποιο μοναδικό αναγνωριστικό id, αφού το κάθε table στην βάση δεδομένων έχει μοναδικό όνομα. Έτσι, για να επιτευχθεί μοναδικά η αναζήτηση, αποτελεί path παράμετρο στα επόμενα endpoints.

- **GET /api/v1.0/schema/tables/{name}**

  Με αυτό το endpoint, ο χρήστης θα ανακτά πληροφορίες για το ζητούμενο table (column names/types/length) όπως ακριβώς κάνει και το MySQL statement **describe table**.

- **DELETE /api/v1.0/schema/tables/{name}**

  Εντολή διαγραφής συγκεκριμένου table από την βάση (**drop table**).

- **GET /api/v1.0/schema/tables**

  Ο χρήστης θα ανακτά σε μια λίστα όλα τα tables της βάσης που έχει δημιουργήσει και τις πληροφορίες τους σε εμφωλευμένη μορφή (array με nested JSON objects).

Με χρήση της επισήμανσης @RequestMapping έχουμε δηλώσει το **basePath** (/api/v1.0/schema/tables), το οποίο είναι κοινό για όλα τα endpoints όπως φαίνεται παραπάνω.

TableDto.java

```
package thesis.models;

import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.annotation.JsonPropertyOrder;

import javax.validation.constraints.NotNull;
import java.util.LinkedList;
import java.util.List;

@JsonPropertyOrder({
        "name",
        "columns"
})
public class TableDto extends BaseModel{
    @NotNull
    @JsonProperty("name")
    private String name;

    @JsonProperty("columns")
    private List<Column> columns;

    public TableDto() {
        this.columns = new LinkedList<>();
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public List<Column> getColumns() {
        return columns;
    }

    public void setColumns(List<Column> columns) {
        this.columns = columns;
    }

    public void setColumn(Column column) {
        this.columns.add(column);
    }
}
```

Column.java

```
package thesis.models;
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.annotation.JsonPropertyOrder;
import javax.validation.constraints.NotNull;

@JsonPropertyOrder({
        "name",
        "type",
        "primaryKey"
})
public class Column {
    @NotNull
    @JsonProperty("name")
    private String name;
    @NotNull
    @JsonProperty("type")
    private ColumnType type;
    @JsonInclude(JsonInclude.Include.NON_DEFAULT)
    @JsonProperty("primaryKey")
    private boolean primaryKey = false;

    public Column(String name, ColumnType type) {
        this.name = name;
        this.type = type;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public ColumnType getType() {
        return type;
    }

    public void setType(ColumnType type) {
        this.type = type;
    }

    public boolean isPrimaryKey() {
        return primaryKey;
    }
    public Column setPrimaryKey(boolean primaryKey) {
        this.primaryKey = primaryKey;
        return this;
    }
}
```

ColumnType.java

```
package thesis.models;

import org.jooq.DataType;
import org.jooq.impl.SQLDataType;

public enum ColumnType {
    STRING(SQLDataType.VARCHAR.length(255).nullable(true)),
    INTEGER(SQLDataType.INTEGER.length(11).nullable(false)),
    BOOLEAN(SQLDataType.BOOLEAN.defaultValue(false).nullable(false)),
    LONG(SQLDataType.BIGINT.length(20).nullable(false)),
    DOUBLE(SQLDataType.DOUBLE.nullable(false));

    private DataType dataType;

    ColumnType(DataType dataType) {
        this.dataType = dataType;
    }

    public DataType getDataType() {
        return this.dataType;
    }
}
```

BaseModel.java

```
package thesis.models;

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;

public class BaseModel {

    public String toJsonString() throws IOException {
        return new ObjectMapper()
                .writerWithDefaultPrettyPrinter()
                .writeValueAsString(this);
    }
}
```

Οι τέσσερις παραπάνω κλάσεις ανήκουν στο **data transfer layer** και αποτελούν τα μοντέλα μέσω των οποίων γίνεται η μεταφορά των δεδομένων του χρήστη από το UI μέχρι την βάση δεδομένων και το αντίστροφο.

Πιο συγκεκριμένα, οι πληροφορίες για την δομή του υπο δημιουργία table ενθηλακώνονται στα properties της κλάσης TableDto.java και μέσω των μεθόδων της (getters) εξάγει τις ζητούμενες τιμές στην κλάση DatabaseExe.java όπου και γίνεται η εκτέλεση της κρίσιμης λογικής του προγράμματος (business logic), δηλαδή διαμορφώνονται τα SQL ερωτήματα και μεταφέρονται οι τιμές στην βάση δεδομένων.

Όπως βλέπουμε, έχει γίνει χρήση διάφορων επισημάνσεων (annotations) της βιβλβιοθήκης Jackson και του javax-validation API, οι οποίες αναλύονται παρακάτω :

1) @JsonPropertyOrder : Καθορίζει τη σειρά με την οποία θα φαίνονται τα properties της κλάσης στο JSON response του server.
2) @JsonProperty : Καθορίζει το όνομα του property που θα εμφανίζεται στο τελικό JSON response και δίνει τη δυνατότητα στην αντίστοιχη Java μεταβλητή να έχει διαφορετικό όνομα.
3) @JsonInclude : Με αυτή την επισήμανση, όσα πεδία **primaryKey** τύπου boolean έχουν τιμή false (προκαθορισμένη τιμή) δεν θα εμφανίζονται στο request και στο response, ώστε να είναι πιο ευανάγνωστο το JSON.
4) @NotNull : Όσα πεδία έχουν αυτή την επισήμανση και το JSON request του χρήστη δεν έχει τιμή σε αυτά, μόλις καλεστεί η μέθοδος createTable του Rest Controller η επισήμανση @Valid (request payload argument) θα παράξει στον server ένα exception με περιγραφή ότι η τιμή του συγκεκριμένου πεδίου ήταν άδεια και ότι δεν γίνεται να συνεχιστεί η διαδικασία. Αυτό μας αποτρέπει απο πολλά πιθανά λάθη αφού ένα λανθασμένα χτισμένο request body απο τον χρήστη δεν θα φτάσει καν στην κρίσιμη λογική του προγράμματος.

Η enumeration κλάση ColumnType αντιπροσωπεύει μία ομάδα απο σταθερές μεταβλητές και χρησιμοποιήθηκε για να αντιστοιχίζει απο το JSON request payload το custom πεδίο του ζητούμενου column του χρήστη (STRING, INTEGER, BOOLEAN, LONG και DOUBLE) σε μορφή **DataType** που περιμένει το jOOQ API για να χτίσει σωστά το SQL query. Όπως φαίνεται στην κλάση έχει χρησιμοποιηθεί προκαθορισμένο μήκος για κάθε πεδίο, ενώ μόνο ο τύπος varchar μπορεί να είναι null όταν θα εισάγονται δεδομένα στο συγκεκριμένο table.

Τέλος, η BaseModel κλάση χρησιμοποιήθηκε για λόγους debugging αφού μετατρέπει μέσω του Jackson ObjectMapper το αντικείμενο TableDto σε μορφοποιημένο JSON String, ώστε να εμφανίζεται κάθε φορά στα logs του server, όπως είδαμε παραπάνω.

Κρίσιμη Λογική (Service Layer)

DatabaseExe.java

```
package thesis.service;

import org.jooq.DSLContext;
import org.jooq.Table;
import org.jooq.impl.DSL;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import thesis.models.Column;
import thesis.models.ColumnType;
import thesis.models.TableDto;

import java.util.List;
import java.util.stream.Collectors;

import static thesis.utils.Constants.ignoredTables;

@Service
public class DatabaseExe implements DatabaseService {

    @Autowired
    private DbUtils dbUtils;

    @Override
    public ResponseEntity<?> createTable(TableDto table) {
        DSLContext dslContext = dbUtils.dslContext();
        String name = table.getName();

        List<Column> pkColumns = table.getColumns()
                .stream()
                .filter(Column::isPrimaryKey)
                .collect(Collectors.toList());

        if (pkColumns.size() == 1) {
            Column pkColumn = pkColumns.get(0);
            String pkName = pkColumn.getName();
            ColumnType pkType = pkColumn.getType();

            dslContext.createTable(name)
                    .column(pkName, pkType.getDataType().identity(true))
                    .constraint(DSL.primaryKey(pkName))
                    .execute();
```

```
        table.getColumns()
                .stream()
                .filter(column -> !column.isPrimaryKey())
                .forEach(column -> dslContext.alterTable(name)
                        .addColumn(column.getName(), column.getType().getDataType())
                        .execute());

        dbUtils.closeJdbcResource();
        return new ResponseEntity<>(table, HttpStatus.CREATED);
    } else if (pkColumns.size() == 0) {
        return new ResponseEntity<>("NO PRIMARY_KEY defined for table " + name,
HttpStatus.INTERNAL_SERVER_ERROR);
    } else {
        return new ResponseEntity<>("Multiple PRIMARY_KEYS in a single table.",
HttpStatus.BAD_REQUEST);
    }
}

@Override
public ResponseEntity<?> getInfo(String name) {

    List<Table> tables = dbUtils.dslContext()
            .meta()
            .getTables()
            .stream()
            .filter(table -> table.getName().equals(name))
            .collect(Collectors.toList());

    if (tables.size() == 1) {
        Table table = tables.get(0);
        TableDto tableDto = dbUtils.getTableDto(table);

        dbUtils.closeJdbcResource();
        return new ResponseEntity<>(tableDto, HttpStatus.OK);
    } else return new ResponseEntity<>("Table " + name + " does not exist.",
HttpStatus.NOT_FOUND);
}

@Override
public ResponseEntity dropTable(String name) {
    List<Table> tables = dbUtils.dslContext()
            .meta()
            .getTables()
            .stream()
            .filter(table -> table.getName().equals(name))
            .collect(Collectors.toList());
```

```
        if (tables.size() == 1) {
            dbUtils.dslContext()
                    .dropTable(tables.get(0).getName())
                    .execute();
            dbUtils.closeJdbcResource();
            return new ResponseEntity(HttpStatus.NO_CONTENT);
        } else return new ResponseEntity<>("Table " + name + " does not exist.",
HttpStatus.NOT_FOUND);
    }

    @Override
    public List<TableDto> showTables() {
        return dbUtils.dslContext()
                .meta()
                .getTables()
                .stream()
                .filter(table -> !ignoredTables.contains(table.getName()))
                .map(table -> dbUtils.getTableDto(table))
                .collect(Collectors.toList());
    }
}
```

DatabaseService.java

```
package thesis.service;

import thesis.models.TableDto;
import org.springframework.http.ResponseEntity;

import java.util.List;

public interface DatabaseService {
    ResponseEntity<?> createTable(TableDto table);

    ResponseEntity<?> getInfo(String name);

    ResponseEntity dropTable(String name);

    List<TableDto> showTables();
}
```

DbUtils.java

```java
package thesis.service;

import org.jooq.DSLContext;
import org.jooq.SQLDialect;
import org.jooq.Table;
import org.jooq.TableField;
import org.jooq.impl.DSL;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import thesis.models.Column;
import thesis.models.ColumnType;
import thesis.models.TableDto;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

@Service
public class DbUtils {
    @Value("${datasource.url}")
    private String url;
    @Value("${datasource.username}")
    private String username;
    @Value("${datasource.password}")
    private String password;

    private Connection connection = null;

    public DSLContext dslContext() {
        try {
            connection = DriverManager.getConnection(url, username, password);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return DSL.using(connection, SQLDialect.MYSQL_8_0);
    }

    public TableDto getTableDto(Table<?> table) {
        TableDto tableDto = new TableDto();
        tableDto.setName(table.getName());

        TableField pkField = table.getPrimaryKey().getFields().get(0);
        String pkName = pkField.getName();
        String pkType = pkField.getType().getSimpleName().toUpperCase();
```

```
        tableDto.setColumn(new Column(pkName, ColumnType.valueOf(pkType))
                .setPrimaryKey(true));

        table.fieldStream()
                .filter(field -> !field.getName().equals(pkName))
                .forEach(field -> {
            String columnName = field.getName();
            String columnType = field.getType().getSimpleName().toUpperCase();
            tableDto.setColumn(new Column(columnName,
ColumnType.valueOf(columnType)));
        });
        return tableDto;
    }

    public void closeJdbcResource() {
        if (connection != null) try {
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Οι κλάσεις DatabaseExe.java και DbUtils.java έχουν την επισήμανση @Service του Spring framework. Αυτό σημαίνει οτι μπορούν να χρησιμοποιηθούν ως πάροχοι υπηρεσιών (service providers) και εκτελούν κώδικα της κρίσιμη λογικής της εφαρμογής σε ξεχωριστό layer απο τον RestController. Το Spring τις ανιχνεύει αυτόματα λόγω του @service annotation και δημιουργεί αντικείμενα απο αυτές όπου καλούνται (μέσω της επισήμανσης @Autowired – Dependency Injection), ώστε να χρησιμοποιηθεί η λειτουργικότητα των μεθόδων τους.

Ο Rest Controller εκτελεί τις μεθόδους της κλάσης DatabaseExe, μέσω του interface DatabaseService και δηλώνει με την επισήμανση @Qualifier ότι χρειάζεται ένα instance της κλάσης DatabaseExe κάθε φορά που εκτελεί μία μεθοδό της, αφου αυτή υλοποιεί το interface. Έτσι, μεταφέρει τα αποτελέσματα του server στον client. Η επισήμανση Qualifier χρησιμοποιείται όταν ένα interface υλοποιείται απο δύο ή παραπάνω κλάσεις και το Spring δεν ξέρει ποιανής κλάσης το object να κάνει ‘inject’ κάθε φορά. Αντίστοιχα, η DatabaseExe κλάση εκτελεί τις μεθόδους της DbUtils.

Το interface DatabaseService, έχει τέσσερις αφηρημένες μεθόδους οι οποίες μαρτυρούν τη βασικά χαρακτηριστικά του προγράμματος. Ουσιαστικά αποτελεί ένα Database Service API, μέσω του οποίου μπορούμε να δημιουργήσουμε, να επεξεργαστούμε, να πάρουμε πληροφορίες και να διαγράψουμε ενα table απο την βάση δεδομένων (CRUD operation).

Όλες οι μεθόδοι επιστρέφουν αντικείμενα της κλάσης ResponseEntity, μέσω της οποίας μπορούμε να παραμετροποιήσουμε όπως θέλουμε το HTTP response του server, όσον αφορά το status code, τα headers και τα δεδομένα (JSON ή String), ανάλογα με την συμπεριφορά του χρήστη. Πιο συγκεκριμένα, όταν ο χρήστης τοποθετήσει στο primaryKey property την τιμή true σε παραπάνω απο ένα column, θα του επιστραφεί **HTTP/1.1 400 Bad Request** με String μήνυμα περιγραφής : **Multiple PRIMARY_KEYS in a single table**. Εάν δεν ορίσει κανένα column ως primary key θα του επιστραφεί **HTTP/1.1 500 Internal Server Error** με μήνυμα : **NO PRIMARY_KEY defined for table <table_name>**, όπως φαίνεται στην μέθοδο DatabaseExe.createTable(). Ακόμα, εάν αναζητήσει κάποιο table με όνομα το οποίο δεν υπάρχει, ο server θα του επιστρέψει **HTTP/1.1 404 Not Found** με μήνυμα : **Table <table_name> does not exist**. Αντίστοιχα, σε κάθε επιτυχημένο request για την δημιουργία ενός table θα του επιστραφεί **HTTP/1.1 201 Created**, ενώ για την απεικόνιση πόρων (GET method) θα του επιστραφεί **HTTP/1.1 200 OK** με δεδομένα τις πληροφορίες του table σε μορφή JSON. Τέλος, μετά απο την επιτυχημένη διαγραφή ενός table θα του επιστραφεί **HTTP/1.1 204 No Content** χωρίς δεδομένα στο body.

Η δημιουργία σύνδεσης με τη βάση δεδομένων, την οποία έχουμε βάλει στατικά στο datasource property του application.yaml (url, username και password) επιτυγχάνεται στην μέθοδο **dslContext()** της DbUtils κλάσης. Η μέθοδος αυτή χρησιμοποιεί το configuration της υπάρχουσας βάσης δεδομένων (java.sql.Connection - JDBC) και το SQL-dialect (MySQL version 8) αφού το jOOQ μπορεί να δουλέψει με ποικίλες σχεσιακές βάσεις (H2, Oracle, PostgreSQL, SQLite κλπ) και επιστρέφει ενα object της κλάσης DSLContext, μέσω του οποίου μπορούμε να δημιουργήσουμε όλα τα MySQL statements. Αντίστοιχα, η μέθοδος **closeJdbcResourse()** κλείνει την σύνδεση κάθε φορά που μία μέθοδος της κλάσης DatabaseExe εκτελείται επιτυχώς.

- Η μέθοδος **createTable(TableDto table)** παίρνει ως όρισμα το request payload του χρήστη που έρχεται απο τον Rest Controller σαν αντικείμενο της κλάσης TableDto. Συλλέγει σε μία λίστα το αντικείμενο Column που περιέχει στο primaryKey πεδίο την τιμή true και μέσω του dslContext χτίζει μια αλυσίδα εκτελέσιμων μεθόδων για να δημιουργήσει το table. Το table που έχει δημιουργήσει αρχικά περιέχει μόνο το primary key (όνομα, τύπο και μήκος). Στη συνέχεια, κάνει μία επανάληψη (iteration) στην λίστα με τα υπόλοιπα Column αντικείμενα του request body και για κάθε ένα απο αυτά μέσω των μεθόδων alterTable(String tableName) και addColumn(String field, DataType type) του jOOQ, προσθέτει την αντίστοιχη στήλη στο υπάρχων table. Τέλος, κλείνει την σύνδεση με τη βάση και επιστρέφει στον χρήστη το ίδιο JSON (echo).

- Η μέθοδος **getInfo(String name)** παίρνει ως όρισμα το όνομα ενός table που υπάρχει ήδη στην βάση δεδομένων, το οποίο αποτελεί path παράμετρο στο endpoint που έχει σταλεί στον Rest Controller. Έχει πρόσβαση στα meta-data της βάσης μέσω της μεθόδου **meta()** του jOOQ και απο εκεί παίρνει όλα τα διαμορφωμένα tables ενθυλακωμένα σε αντικείμενα της κλάσης org.jooq.Table, τα φιλτράρει και όποιο έχει το ίδιο όνομα με το ζητούμενο, το συλλέγει σε μία λίστα. Αν το μήκος της λίστας είναι ίσο με 1, το ζητούμενο table είναι το σωστό. Στη συνέχεια, με χρήση της μεθόδου **getTableDto(org.jooq.Table table)** της κλάσης DbUtils μετατρέπει το jOOQ αντικείμενο σε αντικείμενο της κλάσης TableDto μέσω των **setter** μεθόδων της όπου και τελικά το επιστρέφει στον χρήστη.

- Η μέθοδος **dropTable(String name)** κάνει ακριβώς την ίδια διαδικασία με την getInfo μέχρι να βρεί το ζητούμενο table απο τη βάση. Στη συνέχεια, εκτελεί την μέθοδο dropTable(String table) του jOOQ, η οποία με τη σειρά της δημιουργεί το sql statement drop table και το αφαιρεί απο τη βάση.

- Η μέθοδος **showTables()** συλλέγει όλα τα tables της βάσης απο τα meta-data σε μία λίστα, αγνοώντας αυτά που έχουν όνομα το οποίο ανήκει στην λίστα ignoredTables της κλάσης Constants και αφού έχει αντιστοιχήσει (Stream.map(Function mapper)) κάθε org.jooq.Table αντικείμενο σε TableDto επιστρέφει το array στον χρήστη.

Τέλος, η παρακάτω κλάση Constants.java χρησιμοποιήθηκε ως βοηθητική, αφου παίρνουμε όλα τα διαμορφωμένα δεδομένα απο τα meta-data του jOOQ που δεν χρειαζόμαστε, μέσω ενός static Arraylist που καλούμε. Έτσι, στο φίλτρο της μεθόδου showTables απομονώνουμε όλες τις έξτρα πληροφορίες της βάσης (ενθυλακωμένες σε αντικείμενα τύπου **org.jooq.Table**) σε αυτές που έχει δημιουργήσει μόνο ο χρήστης, ώστε να έχουμε καλύτερη απόδοση και επεξεργασία αλλά και περισσότερο ευανάγνωστο JSON response.

Constants.java

```
package thesis.utils;

import java.util.Arrays;
import java.util.List;

public class Constants {
    private static final String COLLATION_CHARACTER_SET_APPLICABILITY =
"COLLATION_CHARACTER_SET_APPLICABILITY";
    private static final String CHARACTER_SETS = "CHARACTER_SETS";
    private static final String COLLATIONS = "COLLATIONS";
    private static final String COLUMN_PRIVILEGES = "COLUMN_PRIVILEGES";
    private static final String COLUMN_STATISTICS = "COLUMN_STATISTICS";
    private static final String COLUMNS = "COLUMNS";
    private static final String ENGINES = "ENGINES";
    private static final String EVENTS = "EVENTS";
    private static final String FILES = "FILES";
    private static final String INNODB_BUFFER_PAGE = "INNODB_BUFFER_PAGE";
    private static final String INNODB_BUFFER_PAGE_LRU = "INNODB_BUFFER_PAGE_LRU";
    private static final String INNODB_BUFFER_POOL_STATS =
"INNODB_BUFFER_POOL_STATS";
    private static final String INNODB_CACHED_INDEXES = "INNODB_CACHED_INDEXES";
    private static final String INNODB_CMP = "INNODB_CMP";
    private static final String INNODB_CMP_PER_INDEX = "INNODB_CMP_PER_INDEX";
    private static final String INNODB_CMP_PER_INDEX_RESET =
"INNODB_CMP_PER_INDEX_RESET";
    private static final String INNODB_CMP_RESET = "INNODB_CMP_RESET";
    private static final String INNODB_CMPMEM = "INNODB_CMPMEM";
    private static final String INNODB_CMPMEM_RESET = "INNODB_CMPMEM_RESET";
    private static final String INNODB_COLUMNS = "INNODB_COLUMNS";
    private static final String INNODB_DATAFILES = "INNODB_DATAFILES";
    private static final String INNODB_FIELDS = "INNODB_FIELDS";
    private static final String INNODB_FOREIGN = "INNODB_FOREIGN";
    private static final String INNODB_FOREIGN_COLS = "INNODB_FOREIGN_COLS";
```

```
private static final String INNODB_FT_BEING_DELETED = "INNODB_FT_BEING_DELETED";
private static final String INNODB_FT_CONFIG = "INNODB_FT_CONFIG";
private static final String INNODB_FT_DEFAULT_STOPWORD =
"INNODB_FT_DEFAULT_STOPWORD";
private static final String INNODB_FT_DELETED = "INNODB_FT_DELETED";
private static final String INNODB_FT_INDEX_CACHE = "INNODB_FT_INDEX_CACHE";
private static final String INNODB_FT_INDEX_TABLE = "INNODB_FT_INDEX_TABLE";
private static final String INNODB_INDEXES = "INNODB_INDEXES";
private static final String INNODB_METRICS = "INNODB_METRICS";
private static final String INNODB_SESSION_TEMP_TABLESPACES =
"INNODB_SESSION_TEMP_TABLESPACES";
private static final String INNODB_TABLES = "INNODB_TABLES";
private static final String INNODB_TABLESPACES = "INNODB_TABLESPACES";
private static final String INNODB_TABLESPACES_BRIEF = "INNODB_TABLESPACES_BRIEF";
private static final String INNODB_TABLESTATS = "INNODB_TABLESTATS";
private static final String INNODB_TEMP_TABLE_INFO = "INNODB_TEMP_TABLE_INFO";
private static final String INNODB_TRX = "INNODB_TRX";
private static final String INNODB_VIRTUAL = "INNODB_VIRTUAL";
private static final String KEY_COLUMN_USAGE = "KEY_COLUMN_USAGE";
private static final String KEYWORDS = "KEYWORDS";
private static final String OPTIMIZER_TRACE = "OPTIMIZER_TRACE";
private static final String PARAMETERS = "PARAMETERS";
private static final String PARTITIONS = "PARTITIONS";
private static final String PLUGINS = "PLUGINS";
private static final String PROCESSLIST = "PROCESSLIST";
private static final String PROFILING = "PROFILING";
private static final String REFERENTIAL_CONSTRAINTS = "REFERENTIAL_CONSTRAINTS";
private static final String RESOURCE_GROUPS = "RESOURCE_GROUPS";
private static final String ROUTINES = "ROUTINES";
private static final String SCHEMA_PRIVILEGES = "SCHEMA_PRIVILEGES";
private static final String SCHEMATA = "SCHEMATA";
private static final String ST_GEOMETRY_COLUMNS = "ST_GEOMETRY_COLUMNS";
private static final String ST_SPATIAL_REFERENCE_SYSTEMS =
"ST_SPATIAL_REFERENCE_SYSTEMS";
private static final String STATISTICS = "STATISTICS";
private static final String TABLE_CONSTRAINTS = "TABLE_CONSTRAINTS";
private static final String TABLE_PRIVILEGES = "TABLE_PRIVILEGES";
private static final String TABLES = "TABLES";
private static final String TABLESPACES = "TABLESPACES";
private static final String TRIGGERS = "TRIGGERS";
private static final String USER_PRIVILEGES = "USER_PRIVILEGES";
private static final String VIEW_ROUTINE_USAGE = "VIEW_ROUTINE_USAGE";
private static final String VIEW_TABLE_USAGE = "VIEW_TABLE_USAGE";
private static final String VIEWS = "VIEWS";
```

```
public static List<String> ignoredTables =
        Arrays.asList(
                CHARACTER_SETS,
                COLLATION_CHARACTER_SET_APPLICABILITY,
                COLLATIONS,
                COLUMN_PRIVILEGES,
                COLUMN_STATISTICS,
                COLUMNS,
                ENGINES,
                EVENTS,
                FILES,
                INNODB_BUFFER_PAGE,
                INNODB_BUFFER_PAGE_LRU,
                INNODB_BUFFER_POOL_STATS,
                INNODB_CACHED_INDEXES,
                INNODB_CMP,
                INNODB_CMP_PER_INDEX,
                INNODB_CMP_PER_INDEX_RESET,
                INNODB_CMP_RESET,
                INNODB_CMPMEM,
                INNODB_CMPMEM_RESET,
                INNODB_COLUMNS,
                INNODB_DATAFILES,
                INNODB_FIELDS,
                INNODB_FOREIGN,
                INNODB_FOREIGN_COLS,
                INNODB_FT_BEING_DELETED,
                INNODB_FT_CONFIG,
                INNODB_FT_DEFAULT_STOPWORD,
                INNODB_FT_DELETED,
                INNODB_FT_INDEX_CACHE,
                INNODB_FT_INDEX_TABLE,
                INNODB_INDEXES,
                INNODB_METRICS,
                INNODB_SESSION_TEMP_TABLESPACES,
                INNODB_TABLES,
                INNODB_TABLESPACES,
                INNODB_TABLESPACES_BRIEF,
                INNODB_TABLESTATS,
                INNODB_TEMP_TABLE_INFO,
                INNODB_TRX,
                INNODB_VIRTUAL,
                KEY_COLUMN_USAGE,
                KEYWORDS,
```

```
                OPTIMIZER_TRACE,
                PARAMETERS,
                PARTITIONS,
                PLUGINS,
                PROCESSLIST,
                PROFILING,
                REFERENTIAL_CONSTRAINTS,
                RESOURCE_GROUPS,
                ROUTINES,
                SCHEMA_PRIVILEGES,
                SCHEMATA,
                ST_GEOMETRY_COLUMNS,
                ST_SPATIAL_REFERENCE_SYSTEMS,
                STATISTICS,
                TABLE_CONSTRAINTS,
                TABLE_PRIVILEGES,
                TABLES,
                TABLESPACES,
                TRIGGERS,
                USER_PRIVILEGES,
                VIEW_ROUTINE_USAGE,
                VIEW_TABLE_USAGE,
                VIEWS
        );
}
```

4.2 Υλοποίηση Γραφικού Περιβάλλοντος (Front-End)

Η ανάπτυξη του front-end πραγματοποιήθηκε με χρήση της βιβλιοθήκης React. Κάθε σελίδα αποτελεί και ένα component, γεγονός που επιτρέπει ευκολότερη επαναχρησιμοποίηση του κώδικα.

Το αρχικό component της εφαρμογής είναι το App component, μέσα στο οποίο γίνεται η αντιστοίχηση κάθε σελίδας (URL) με το React component, με χρήση της βιβλιοθήκης react-router-dom, όπως φαίνεται παρακάτω.

App.js

```
import React, { Component } from 'react';
import './App.css';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Home from './Home';
import CreateTable from './CreateTable'
import GetInfo from './GetInfo'
import MuiThemeProvider from 'material-ui/styles/MuiThemeProvider';
import ShowTables from './ShowTables';

class App extends Component {
  render() {
    return (
      <MuiThemeProvider>
        <Router>
          <Switch>
            <Route path='/' exact={true} component={Home} />
            <Route path='/tables/create' exact={true} component={CreateTable} />
            <Route path='/tables/:name' exact={true} component={GetInfo} />
            <Route path='/tables/' exact={true} component={ShowTables} />
          </Switch>
        </Router>
      </MuiThemeProvider>
    );
  }
}
export default App;
```

Το <Switch> component ομαδοποιεί όλα τα παιδιά components του (<Route>), κάνει μία επανάληψη (iteration) σε αυτά και καλεί (renders) μόνο το πρώτο που θα αντιστοιχηθεί ακριβώς (exact = {true}) με το current pathname. Παρατηρούμε ότι το component GetInfo θα γίνει render όταν στο URL υπάρχει String path παράμετρος ακριβώς δίπλα απο το pathname **/tables**.

Home.js

```
import React, { Component } from 'react';
import './App.css';
import AppNavbar from './AppNavBar';
import { withRouter } from 'react-router-dom';
import Grid from '@material-ui/core/Grid';
import Button from '@material-ui/core/Button';
import mySQL from "./images/mySQL.png";
import Paper from '@material-ui/core/Paper';
import { withStyles } from '@material-ui/core';

const styles = theme => ({
    button: {
        left: 10,
        margin: 25,
        '&:hover': {
            color: 'white',
        },
    },
});

class Home extends Component {

    constructor() {
        super();
    }

    render() {
        const { classes } = this.props;
```

```
        return (
             <div>
                 <AppNavbar />
                 <br></br>
                 <br></br>
                 <Grid container justify="center">
                     <Paper style={{ width: '100%', maxWidth: 210 }}>
                         <img src={mySQL} />
                         <br></br>
                         <Button className={classes.button}
                             variant="contained"
                             href="/tables/create"
                             color="primary" >
                             Create Table
                          </Button>
                     </Paper>
                 </Grid>
             </div >
         );
    }
}
export default withRouter(withStyles(styles)(Home));
```

Η παραπάνω κλάση αποτελεί την αρχική σελίδα της εφαρμογής και ανοίγει στην root διεύθυνση http://localhost:3000/. Το Home component καλεί και εμφανίζει το AppNavBar component που έχουμε φτιάξει και το Button component της material-UI βιβλιοθήκης, το οποίο εδώ λειτουργεί σαν σύνδεσμος (href=”/tables/create”). Με το πάτημά του, η Home σελίδα θα μας ανακατευθύνει στην σελίδα http://localhost:300/tables/create, δηλαδή θα γίνει render το component CreateTable μέσω του Router που είδαμε πριν.

CreateTable.java

```
import { TextField, Container,  TableHead, Tooltip } from "@material-ui/core";
import React, { Component } from 'react';
import AppNavbar from './AppNavBar';
import Table from '@material-ui/core/Table';
import TableBody from '@material-ui/core/TableBody';
import TableCell from '@material-ui/core/TableCell';
import SaveIcon from '@material-ui/icons/Save';
import TableRow from '@material-ui/core/TableRow';
import Paper from '@material-ui/core/Paper';
import { withRouter } from 'react-router-dom';
import MenuItem from '@material-ui/core/MenuItem';
import Checkbox from '@material-ui/core/Checkbox';
import AddBoxIcon from '@material-ui/icons/AddBox';
import IconButton from '@material-ui/core/IconButton';
import DeleteIcon from '@material-ui/icons/Delete';
import { set } from 'object-path-immutable';

class CreateTable extends Component {

    constructor(props) {
        super(props);
        this.state = {
            statusCode: '',
            payload: {
                name: '',
                columns: [
                    {
                        name: '',
                        type: '',
                        primaryKey: false
                    },
                ]
            }
        }
        this.handleChange = this.handleChange.bind(this);
        this.handleTableNameChange = this.handleTableNameChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
        this.handleCheck = this.handleCheck.bind(this);
    }
```

```
    handleTableNameChange(event) {
        this.setState({
            payload: {
                ...this.state.payload,
                name: event.target.value,
            },
        });
    }

    handleChange = index => event => {
        const name = event.target.name;
        const value = event.target.value;

        this.setState({
            payload: set(this.state.payload, ['columns', index, name], value)
        });
    }

    handleCheck = (index) => event => {
        const payload = { ...this.state.payload };

        var counter = payload.columns
        .filter((column, idx) => column.primaryKey && idx != index)
        .length;

        if (counter > 0) {
            alert(counter + ' => You can not assign multiple PKs in the same
table');
        } else {
            this.setState({
                payload: set(this.state.payload, ['columns', index,
event.target.name], event.target.checked)
            });
        }
    }
```

```
handleAddRow = () => {
    const item = {
        name: '',
        type: '',
        primaryKey: false
    }
    const object = this.state.payload;
    object.columns = [...object.columns, item];
    this.setState({ payload: object });
}

handleRemoveRow = (index) => () => {
    const columns = [...this.state.payload.columns];
    columns.splice(index, 1);
    this.setState({
        payload: {
            ...this.state.payload,
            columns: columns,
        }
    });
}

async  handleSubmit(event) {
    event.preventDefault();
    //alert('------     ' + JSON.stringify(this.state.payload));

    await fetch('/api/v1.0/schema/tables', {
        method: 'POST',
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(this.state.payload),
    }).then(response => {
        if (response.status === 201) {
            this.props.history.push(`/tables/${this.state.payload.name}`);
        } else {
            alert(response.json());
        }
    }
    )
}
```

```
render() {
    const types = [
        {
            value: 'STRING',
            label: 'varchar(255)',
        },
        {
            value: 'INTEGER',
            label: 'int(11)',
        },
        {
            value: 'BOOLEAN',
            label: 'tinyint(1)',
        },
        {
            value: 'LONG',
            label: 'bigint(20)',
        },
        {
            value: 'DOUBLE',
            label: 'double',
        }
    ];

    return (
        <div >
            <AppNavbar />
            <br></br>
            <Container maxWidth="md">
                <form onSubmit={this.handleSubmit}>
                    <Tooltip style={{ left: 840 }}
                        title="Save Table"
                        placement="left">
                        <IconButton type="submit"
                            onSubmit={this.handleSubmit}>
                            <SaveIcon fontSize="large" />
                        </IconButton>
                    </Tooltip>
                    <br></br>
```

```
<Paper>
    <h4 style={{ padding: 25, display: 'inline' }}
    >Create Table </h4>
    <TextField
        style={{ left: 410, display: 'inline' }}
        required={true}
        autoFocus={true}
        variant="outlined"
        margin="dense"
        label="Table Name"
        name="name"
        value={this.state.payload.name}
        onChange={this.handleTableNameChange}
    />
    <Tooltip
        style={{ left: 415, display: 'inline' }}
        title="Add Column"
        placement="right">
        <IconButton onClick={this.handleAddRow}>
            <AddBoxIcon />
        </IconButton>
    </Tooltip>
    <Table>
        <TableHead>
            <TableRow>
                <TableCell align="center">
                    No.
                    </TableCell>
                <TableCell align="center">
                    Column Name
                    </TableCell>
                <TableCell align="center">
                    Column Type
                    </TableCell>
                <TableCell align="center">
                    Primary Key
                    </TableCell>
                <TableCell align="center">
                    Remove Column
                    </TableCell>
            </TableRow>
        </TableHead>
```

```
<TableBody>
    {this.state.payload.columns
          .map((column, index) => (
              <TableRow key={index}>
                    <TableCell align="center">
                          {index + 1}
                    </TableCell>
                    <TableCell align="center">
                         <TextField
                              id="column-name"
                              name="name"
                              required={true}
                              margin="dense"
                              value={this.state.payload.columns[index].name}
                              onChange={this.handleChange(index)} />
                     </TableCell>
                     <TableCell align="center">
                        <TextField
                          style={{width: 120}}
                          select
                          id="column-type"
                          name="type"
                          required={true}
                          margin="dense"
                          value={this.state.payload.columns[index].type}
                          onChange={this.handleChange(index)}
                         >
                          {types.map(option => (
                              <MenuItem key={option.value}
                                  value={option.value}>
                                        {option.label}
                              </MenuItem>
                          ))}
                         </TextField>
                        </TableCell>
```

```
<TableCell aligh="center">
      <Tooltip title="Primary Key">
           <Checkbox
                name="primaryKey"
                disabled={this.state.disabled}
                checked={this.state.payload.columns[index].primaryKey}
                onChange={this.handleCheck(index)}
            />
       </Tooltip>
 </TableCell>
 <TableCell aligh="center">
       <Tooltip title="Remove Column">
             <IconButton onClick={this.handleRemoveRow(index)} >
                    <DeleteIcon />
             </IconButton>
       </Tooltip>
 </TableCell>
</TableRow>
))}
</TableBody>
</Table>
</Paper>
</form>
</Container>
</div >
);
}}
export default withRouter(CreateTable);
```

Όταν καλείται (renders) το παραπάνω component και ανοίγει η σελίδα http://localhost:3000/tables/create, τα δεδομένα του πίνακα (material-UI/core table) θα είναι άδεια, αφου στο state της κλάσης εσωτερικά το αντικείμενο payload έχει κενές τιμές, όπως του έχουμε ορίσει στατικά. Όταν ο χρήστης θα αρχίσει να εισάγει τιμές σε όλα τα πεδία, το payload θα αρχίζει να χτίζεται μέσω των μεθόδων handleTableNameChange, handleChange, handleCheck, handleAddRow και handleRemoveRow που έχουμε ορίσει.

- **handleTableNameChange**: αφορά το attribute του TextField component και κάθε φορά που ο χρήστης θα πληκτρολογεί (**onChange**) το όνομα του table που θέλει να δημιουργήσει, η μέθοδος αυτή θα παίρνει την τιμή και θα την εισχωρεί στο πεδίο **name** του payload.
- **handleChange**: αυτή η μέθοδος παίρνει ως όρισμα το index της συγκεκριμένης σειράς του πίνακα και τοποθετεί την τιμή του χρήστη (column name) στο πεδίο columns[index].name του payload.
- **handleCheck**: αφορά το attribute του Checkbox component και όταν ο χρήστης επιλέξει ως primary key μία μόνο γραμμή του πίνακα, η μέθοδος θα τοποθετήσει την τιμή true στο πεδίο columns[index].primaryKey. Έχουμε περιορίσει τον χρήστη στην επιλογή μόνο μίας σειράς, αφού εάν μπορούσε να στείλει το request payload με ορισμένα 2 primary keys στο ίδιο table, το Back-end θα του επέστρεφε 500 Internal Server Error όπως είδαμε παραπάνω. Έτσι, αποφεύγουμε ανούσια HTTP API calls στον server.
- **handleAddRow**: αφορά το onClick attribute του IconButton. Κάθε φορά που ο χρήστης θα κάνει click στο πεδίο Add Column όπως εμφανίζεται στην HTML σελίδα, η μέθοδος αυτή θα προσθέτει ένα κενό object στην λίστα με τα columns, δηλαδή μία καινούρια σειρά στον πίνακα η οποία θα περιμένει τιμές.
- **handleRemoveRow**: Αντίστοιχα εδώ, έχοντας το συγκεκριμένο index της γραμμής που θέλει να αφαιρέσει ο χρήστης απο το table που επρόκειτο να δημιουργήσει, η μέθοδος διαγράφει το column object ({ name: '', type: '', primaryKey: false }) απο το array και ανανεώνει το payload.

Τέλος, όταν ο χρήστης έχει πλέον ετοιμάσει το request body που θα στείλει στο API και πατήσει το Save Icon Button, εκτελείται η μέθοδος **handleSubmit** όπου και γίνεται το HTTP POST request http://localhost:8181/api/v1.0/schema/tables. Εάν το request πραγματοποιηθεί επιτυχώς και το API επιστρέψει HTTP/1.1 201 Created, η σελίδα θα μας ανακατευθύνει στην διεύθυνση http://localhost:3000/tables/{name}. Έτσι, θα καλεστεί το GetInfo component (**this.props.history.push** - react-router-dom), το οποίο αναλύεται παρακάτω.

GetInfo.java

```
import React, { Component } from 'react';
import AppNavBar from './AppNavBar';
import { Container, Table, TableHead, TableCell, TableBody, Tooltip } from
'@material-ui/core';
import { withRouter } from 'react-router-dom';
import { Paper, TableRow } from 'material-ui';
import IconButton from '@material-ui/core/IconButton';
import DeleteIcon from '@material-ui/icons/Delete';
import EditIcon from '@material-ui/icons/Edit';
import { withStyles } from '@material-ui/core';
import { getDataType } from './utils';

const styles = theme => ({
    root: {
        //backgroundColor: 'red'
        //backgroundColor: theme.palette.primary.light[500],
    },
    inline: {
        padding: 25,
        display: 'inline'
    },
});

class GetInfo extends Component {

    emptyItem = {
        name: '',
        columns: [
            {
                name: '',
                type: '',
                primaryKey: false
            }
        ]
    };
```

```
constructor(props) {
    super(props);
    this.state = {
        table: this.emptyItem
    }
    this.handleDrop = this.handleDrop.bind(this);
    this.handleEdit = this.handleEdit.bind(this);
}

componentWillMount() {
    this.fetchData(this.props.match.params.name);
}

componentWillReceiveProps(nextProps) {
    const currentName = nextProps.match.params.name;
    if (currentName !== this.props.match.params.name) {
        this.fetchData(currentName);
    }
}

async fetchData(table) {
    await fetch(`/api/v1.0/schema/tables/${table}`, {
        method: 'GET'
    })
        .then(response => response.json())
        .then(data => {
            this.setState({ table: data })
        });
}

handleEdit() {}

handleDrop() {
    fetch(`/api/v1.0/schema/tables/${this.state.table.name}`, {
        method: 'DELETE'
    }).then(response => {
        if (response.status === 204) {
            this.props.history.push(`/tables`);
        }
    })
}
```

```
render() {

        const { classes } = this.props;

        function abc(type) {
            if (type === 'STRING') { return 'YES'; } else { return 'NO'; }
        }
        function key(isPrimaryKey) {
            if (isPrimaryKey) return 'PRI';
        }
        function dbDefault(type) {
            if (type === 'BOOLEAN') { return 0; } else { return '[NULL]'; }
        }
        function extra(isPrimaryKey) {
            if (isPrimaryKey) return 'auto_increment';
        }

        return (
            <div className={classes.root}>
                <AppNavBar />
                <br></br>
                <Container maxWidth="md">
                    <Paper>
                       <h4 className={classes.inline}>
                         {this.state.table.name}
                       </h4>
                      <Tooltip title="Drop Table" placement="top">
                         <IconButton className={classes.inline}
                                     onClick={this.handleDrop}>
                              <DeleteIcon />
                          </IconButton>
                      </Tooltip>
                      <Tooltip title="Alter Table" placement="right">
                         <IconButton className={classes.inline}
                                     onClick={this.handleEdit()}>
                             <EditIcon />
                         </IconButton>
                      </Tooltip>
```

```
<Table style={{ minWidth: 650 }} size="small">
    <TableHead>
       <TableRow>
           <TableCell align="left">No.</TableCell>
           <TableCell align="left">Field</TableCell>
           <TableCell align="left">Type</TableCell>
           <TableCell align="left">Null</TableCell>
           <TableCell align="left">Key</TableCell>
           <TableCell align="left">Default</TableCell>
           <TableCell align="left">Extra</TableCell>
        </TableRow>
    </TableHead>
    <TableBody>
        {this.state.table.columns.map((column, index) => (
           <TableRow key={column.name}>
                <TableCell align="left">{index + 1}</TableCell>
                <TableCell align="left">{column.name}</TableCell>
                <TableCell align="left">{getDataType(column.type)}</TableCell>
                <TableCell align="left">{abc(column.type)}</TableCell>
                <TableCell align="left">{key(column.primaryKey)}</TableCell>
                <TableCell align="left">{dbDefault(column.type)}</TableCell>
                <TableCell align="left">{extra(column.primaryKey)}</TableCell>
            </TableRow>
         ))}
      </TableBody>
 </Table>
</Paper>
</Container>
</div>
);
}}
export default withRouter(withStyles(styles)(GetInfo));
```

Βοηθητική συνάρτηση :

utils.jsx

```
import React from 'react';

export function getDataType(customType) {
    let dbType;
    switch (customType) {
        case 'STRING':
            dbType = 'varchar(255)';
            break;
        case 'INTEGER':
            dbType = 'int(11)';
            break;
        case 'BOOLEAN':
            dbType = 'tinyint(1)';
            break;
        case 'LONG':
            dbType = 'bigint(20)';
            break;
        case 'DOUBLE':
            dbType = 'double';
            break;
    }
    return dbType;
}
```

Όταν καλείται το GetInfo component, δηλαδή όταν ο browser ανιχνεύσει στο URL ένα path παράμετρο τύπου String που αποτελεί το όνομα του ζητούμενου table, η πρώτη κίνηση της κλάσης είναι να εξάγει αυτό το όνομα σε μία μεταβλητή μέσω της μεθόδου **componentWillMount()** και να πραγματοποιήσει το GET request http://localhost:8181/api/v1.0/schema/tables/{name} στο back-end. Αυτό επιτυγχάνεται με την μέθοδο **fetchData(name)**, η οποία ανανεώνει το state της κλάσης με το JSON response του server, γεμίζοντας το emptyItem αντικείμενο με τις τιμές που έλαβε. Έτσι, όλες οι πληροφορίες του table απο την βάση δεδομένων απεικονίζονται στον πίνακα της σελίδας. Ακόμα, απο αυτή τη σελίδα έχουμε την δυνατότητα να το διαγράψουμε μέσω του attribute **onClick** του DeleteIcon component, αφού θα εκτελεστεί η μέθοδος **handleDrop()** και θα γίνει το DELETE request http://localhost:8181/api/v1.0/schema/tables/{name} στο RESTful API. Αν ο server επιστρέψει HTTP/1.1 204 No Content, ο χρήστης θα ανακατευθυνθεί στην σελίδα /tables και θα καλεστεί το ShowTables component, το οποίο περιγράφεται παρακάτω.

ShowTables.js

```
import React, { Component } from "react";
import AppNavBar from './AppNavBar';
import List from '@material-ui/core/List';
import ListSubheader from '@material-ui/core/ListSubheader';
import ListItem from '@material-ui/core/ListItem';
import Paper from '@material-ui/core/Paper';
import ListItemText from '@material-ui/core/ListItemText';
import Collapse from '@material-ui/core/Collapse';
import ExpandLess from '@material-ui/icons/ExpandLess';
import ExpandMore from '@material-ui/icons/ExpandMore';
import { withRouter } from 'react-router-dom';
import { getDataType } from './utils';
import VpnKeyRoundedIcon from '@material-ui/icons/VpnKeyRounded';
import Divider from '@material-ui/core/Divider';
import ListItemAvatar from '@material-ui/core/ListItemAvatar';
import TabletRoundedIcon from '@material-ui/icons/TabletRounded';
import Grid from '@material-ui/core/Grid';

class ShowTables extends Component {

    constructor(props) {
        super(props);
        this.state = {
            tables: [{
                name: '',
                columns: [
                    {
                        name: '',
                        type: '',
                        primaryKey: false
                    }
                ],
                collapse: false
            }]
        }
        this.toggle = this.toggle.bind(this);
    }
```

```
    async componentDidMount() {
        await fetch(`/api/v1.0/schema/tables`)
            .then(response => response.json())
            .then(data => {
                this.setState({ tables: data })
            })
    }

    toggle = (tableName) => () => {
        this.setState(prevState => ({
            tables: prevState.tables.map(
                table => (table.name === tableName ? Object.assign(table, {
collapse: !table.collapse }) : table)
            )
        }));
        //alert(this.state.tables.map(table=>(table.name === tableName ? true :
false)))
    }

    render() {
        return (
            <div>
                <AppNavBar />
                <br></br>
                <br></br>
                <Grid container justify="center" spacing={2}>
                    <Paper style={{ width: '100%', maxWidth: 360 }}>
                        <List
                            component="nav"
                            subheader={<ListSubheader
                            component="div">Tables_in_thesisdb</ListSubheader>}>
                            {this.state.tables.map(table => (
                                <div>
                                    <ListItem button
                                              onClick={this.toggle(table.name)}>
                                        <ListItemAvatar>
                                            <TabletRoundedIcon />
                                        </ListItemAvatar>
                                        <ListItemText primary={table.name} />
                                        {table.collapse ?
                                         <ExpandLess /> : <ExpandMore />}
                                    </ListItem>
```

```
                        <Collapse in={table.collapse}
                                  timeout="auto" unmountOnExit>
                              {table.columns.map(column => (
                                  <List component="div" disablePadding>
                                     <ListItem>
                                         <ListItemAvatar>
                                             {column.primaryKey
                                              && <VpnKeyRoundedIcon />}
                                          </ListItemAvatar>
                                          <ListItemText
                                             primary={column.name}
                                             secondary={getDataType(column.type)}
                                           />
                                      </ListItem>
                                          <Divider
                                              variant="inset"
                                              component="li" />
                                    </List>
                                ))}
                            </Collapse>
                          </div>
                      ))}
                  </List>
              </Paper>
          </Grid>
      </div>
         );
     }
}
export default withRouter(ShowTables);
```

Όταν ανοίγει η σελίδα /tables, εκτελείται η μέθοδος **componentDidMount()**, η οποία ερωτά το back-end με GET http://localhost:8181/api/v1.0/schema/tables, και έτσι ανανεώνει το state της κλάσης (array απο objects) με τα δεδομένα που πήρε απο την απάντηση του server. Έτσι, ο χρήστης θα μπορεί να βλέπει όλα τα tables που έχει δημιουργήσει στην βάση δεδομένων σε μία λίστα (material-UI List component) και κάθε φορά που θα επεκτείνει (**Collapse** component) ένα στοιχείο απο αυτή τη λίστα θα βλέπει λεπτομερώς τη δομή του εκάστοτε table.

AppNavBar.js

```
import React, { Component } from 'react';
import AppBar from '@material-ui/core/AppBar';
import Toolbar from '@material-ui/core/Toolbar';
import Typography from '@material-ui/core/Typography';
import InputBase from '@material-ui/core/InputBase';
import { fade, makeStyles } from '@material-ui/core/styles';
import HomeIcon from '@material-ui/icons/Home';
import SearchIcon from '@material-ui/icons/Search';
import { withRouter } from 'react-router-dom';
import Tooltip from '@material-ui/core/Tooltip';
import { Link } from '@material-ui/core';
import PropTypes from 'prop-types';
import clsx from 'clsx';
import ErrorIcon from '@material-ui/icons/Error';
import CloseIcon from '@material-ui/icons/Close';
import IconButton from '@material-ui/core/IconButton';
import Snackbar from '@material-ui/core/Snackbar';
import SnackbarContent from '@material-ui/core/SnackbarContent';
import GridOnIcon from '@material-ui/icons/GridOn';

const variantIcon = {
    error: ErrorIcon
};

const useStyles = makeStyles(theme => ({
    root: {
        flexGrow: 1,
    },
    homeButton: {
        marginRight: theme.spacing(2),
        '&:hover': {
            color: 'inherit',
        },
    },
    title: {
        flexGrow: 1,
        display: 'none',
        [theme.breakpoints.up('sm')]: {
            display: 'block',
        },
    },
```

```
search: {
    position: 'relative',
    borderRadius: theme.shape.borderRadius,
    backgroundColor: fade(theme.palette.common.white, 0.15),
    '&:hover': {
        backgroundColor: fade(theme.palette.common.white, 0.25),
    },
    marginLeft: 0,
    width: '100%',
    [theme.breakpoints.up('sm')]: {
        marginLeft: theme.spacing(1),
        width: 'auto',
    },
},
searchIcon: {
    width: theme.spacing(7),
    height: '100%',
    position: 'absolute',
    pointerEvents: 'none',
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'center',
},
inputRoot: {
    color: 'inherit',
},
inputInput: {
    padding: theme.spacing(1, 1, 1, 7),
    transition: theme.transitions.create('width'),
    width: '100%',
    [theme.breakpoints.up('sm')]: {
        width: 120,
        '&:focus': {
            width: 200,
        },
    },
},
```

```
    grid:{
        marginLeft: theme.spacing(3),
        //marginRight: theme.spacing(2),
        '&:hover': {
            color: 'inherit',
        },
    },

    error: {
        backgroundColor: theme.palette.error.dark,
    },
    icon: {
        fontSize: 20,
    },
    iconVariant: {
        opacity: 0.9,
        marginRight: theme.spacing(1),
    },
    message: {
        display: 'flex',
        alignItems: 'center',
    },
}));

function SnackbarContentWrapper(props) {
    const classes = useStyles();
    const { className, message, onClose, variant, ...other } = props;
    const Icon = variantIcon[variant];

    return (
        <SnackbarContent
            className={clsx(classes[variant], className)}
            aria-describedby="client-snackbar"
            message={
                <span id="client-snackbar" className={classes.message}>
                    <Icon className={clsx(classes.icon, classes.iconVariant)} />
                    {message}
                </span>
            }
```

```
            action={[
                <IconButton key="close" aria-label="close" color="inherit"
onClick={onClose}>
                    <CloseIcon className={classes.icon} />
                </IconButton>,
            ]}
            {...other}
        />
    );
}

SnackbarContentWrapper.propTypes = {
    className: PropTypes.string,
    message: PropTypes.string,
    onClose: PropTypes.func,
    variant: PropTypes.oneOf(['error', 'info', 'success', 'warning']).isRequired,
};

const Bar = ({ handleChange, handleSubmit, tableName, open, handleClose,
errorMessage }) => {
    const classes = useStyles();
    return (
        <div>

            <Snackbar
                anchorOrigin={{
                    vertical: 'bottom',
                    horizontal: 'left',
                }}
                open={open}
                autoHideDuration={9000}
                onClose={handleClose}
            >
                <SnackbarContentWrapper
                    onClose={handleClose}
                    variant="error"
                    message={errorMessage}
                />
            </Snackbar>

            <div className={classes.root}>
                <AppBar position="static">
```

```
<Toolbar>
    <Tooltip title="Home" placement="bottom">
        <Link
            href="/"
            edge="start"
            className={classes.homeButton}
            color="inherit"
        >
            <HomeIcon fontSize="medium"/>
        </Link>
    </Tooltip>
    <Typography className={classes.title} variant="h6"
                noWrap>MySQL designer
    </Typography>
    <Tooltip title="Describe Table" placement="bottom">
        <div className={classes.search}>
            <div className={classes.searchIcon}>
                <SearchIcon />
            </div>
            <form onSubmit={handleSubmit}>
                <InputBase
                    value={tableName}
                    onChange={handleChange}
                    placeholder="Search…"
                    classes={{
                        root: classes.inputRoot,
                        input: classes.inputInput,
                    }}
                />
            </form>
        </div>
    </Tooltip>
    <Tooltip title="Show Tables" placement="bottom">
        <Link
            href="/tables"
            //edge="start"
            className={classes.grid}
            color="inherit"
        >
            <GridOnIcon fontSize="medium"/>
        </Link>
    </Tooltip>
```

```
                    </Toolbar>
                </AppBar>
            </div>
        </div>
    );
}

class AppNavBar extends Component {
    emptyName = '';

    constructor() {
        super();
        this.state = { name: this.emptyName, open: false, statusCode: 0,
                       errorMessage: '' };
        this.handleChange = this.handleChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
        this.handleClose = this.handleClose.bind(this);
    }

    handleChange(event) {
        this.setState({ name: event.target.value });
    }
    async handleSubmit(event) {
        event.preventDefault();
        let table = this.state.name;

        if (table !== '') {
            let response = await fetch(`/api/v1.0/schema/tables/${table}`);

            this.setState({ statusCode: response.status });
            const code = this.state.statusCode;

            if (code === 200) {
                this.props.history.push(`/tables/${table}`);
                this.setState({ name: this.emptyName });
            } else if (code === 404) {
                let text = await response.text();

                this.setState({ errorMessage: text });
                this.setState({ open: true });
            }
        }
    }
```

```
    handleClose() {
        this.setState({ open: false });
        //this.setState({ name: this.emptyName })
    }

    render() {
        return (
            <Bar
                handleChange={this.handleChange}
                handleSubmit={this.handleSubmit}
                tableName={this.state.name}
                open={this.state.open}
                handleClose={this.handleClose}
                errorMessage={this.state.errorMessage}
            />
        );
    }
}
export default withRouter(AppNavBar);
```

Το AppNavBar component καλείται απο όλες τις σελίδες και αποτελεί την μπάρα περιήγησης της εφαρμογής. Ο χρήστης σε οποιαδήποτε σελίδα και αν βρίσκεται μέσω αυτής της μπάρας μπορεί να ανακατευθυνθεί :

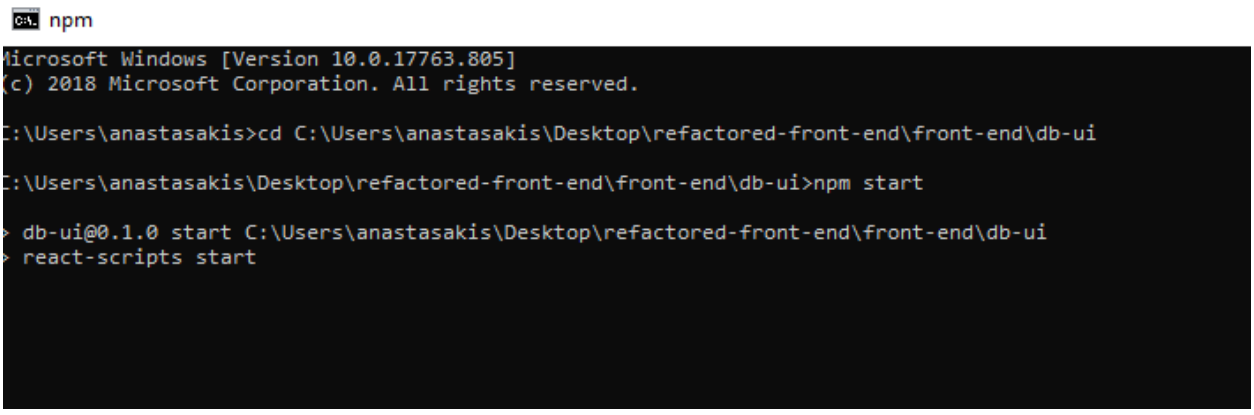
1) Στην αρχική σελίδα της εφαρμογής με την χρήση του HomeIcon component, το οποίο έχουμε ενθυλακώσει με Link στην root διεύθυνση “\”.
2) Στην σελίδα /tables/name αφού πληκτρολογήσει στην περιοχή αναζήτησης το όνομα του table που επιθυμεί να δεί τις λεπτομέρειες του. Όταν ο χρήστης πατήσει enter στην αναζήτηση, εκτελείται η μέθοδος **handleSubmit(event)** και ερωτάται ο server με GET /api/v1.0/schema/tables/{table}. Μόνο αν το ζητούμενο table υπάρχει στην βάση, δηλαδή ο server επιστρεψει HTTP/1.1 200 OK, θα γίνει αυτή η αναδρομολόγηση (this.props.history.push) και ο χρήστης θα αλλάξει σελίδα. Αν βρίσκεται στην ίδια σελίδα και αναζητήσει διαφορετικό table απο αυτό που βλέπει, θα ενημερωθούν τα δεδομένα και θα αλλάξουν οι τιμές του πίνακα (**componentWillReceiveProps(nextProps)** – GetInfo.js). Εάν πατήσει enter με κενό όνομα, δεν θα πραγματοποιηθεί καν το request ούτε καμία άλλη ενέργεια. Τέλος, αν προσπαθήσει να αναζητήσει table με μη έγκυρο όνομα ή όνομα το οποίο δεν υπάρχει στην βάση, ο server θα επιστρέψει HTTP/1.1 404 Not Found με ένα μήνυμα τύπου String.

Σε αυτήν την περίπτωση, η μέθοδος handleSubmit θα ανανεώσει στο state της κλάσης την μεταβλητή errorMessage, η οποία θα περιέχει ακρίβως το μήνυμα που ήρθε απο τον server και την μεταβλητή open σε true, η οποία θα εμφανίσει το SnackBarContent component, ένα πλαίσιο προειδοποίησης ότι το table με το ζητούμενο όνομα δεν υπάρχει, χωρίς να αλλάξει σελίδα ο χρήστης.

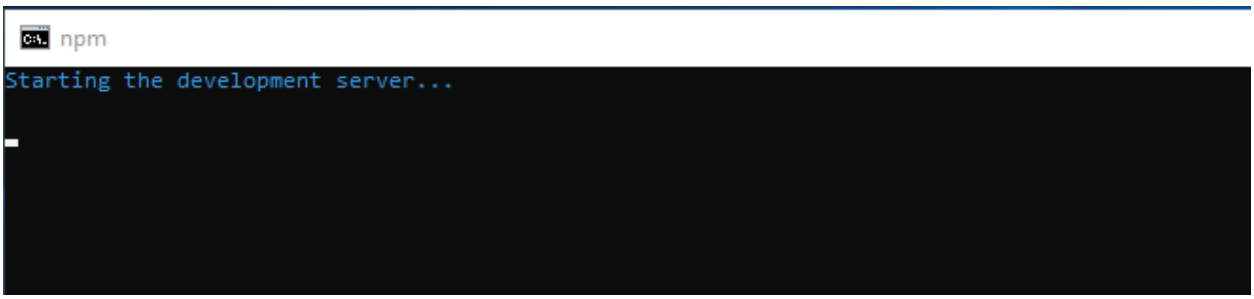
3) Στην σελίδα /tables με την χρήση του GridOnIcon component, το οποίο λειτουργεί ως σύνδεσμος.

4.3 Παράδειγμα Λειτουργίας Συστήματος

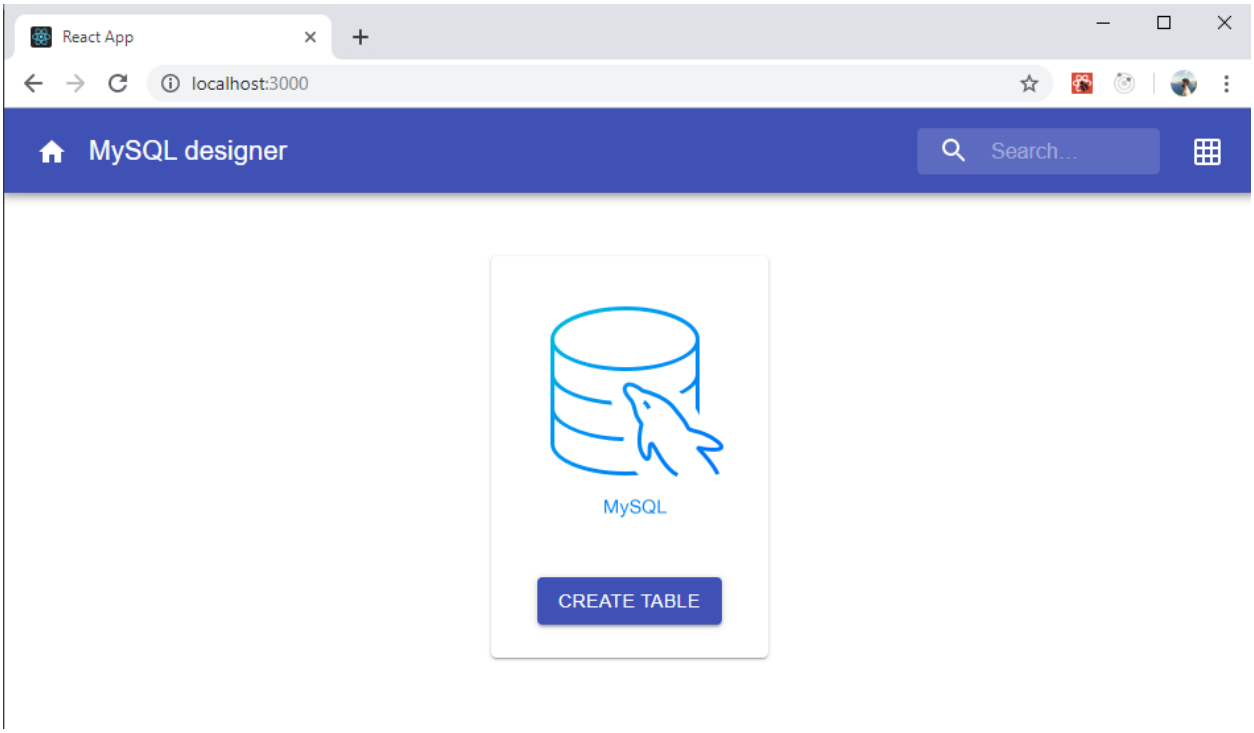
Ανοίγοντας το Command Prompt στην διεύθυνση που βρίσκονται τα αρχεία του front-end πρότζεκτ και πληκτρολογώντας την εντολή **npm start** θα τρέξουμε μία συλλογή απο scripts (react-scripts **package.json**), μέσω των οποίων θα σηκωθεί ο προκαθορισμένος **nodeJS** server, ο οποίος θα εκκινήσει την React εφαρμογη στον τοπικό διακομιστή localhost στην πόρτα 3000. Έτσι θα οδηγηθούμε στην αρχική σελίδα της εφαρμογής, όπως φαίνεται παρακάτω.



9. npm start

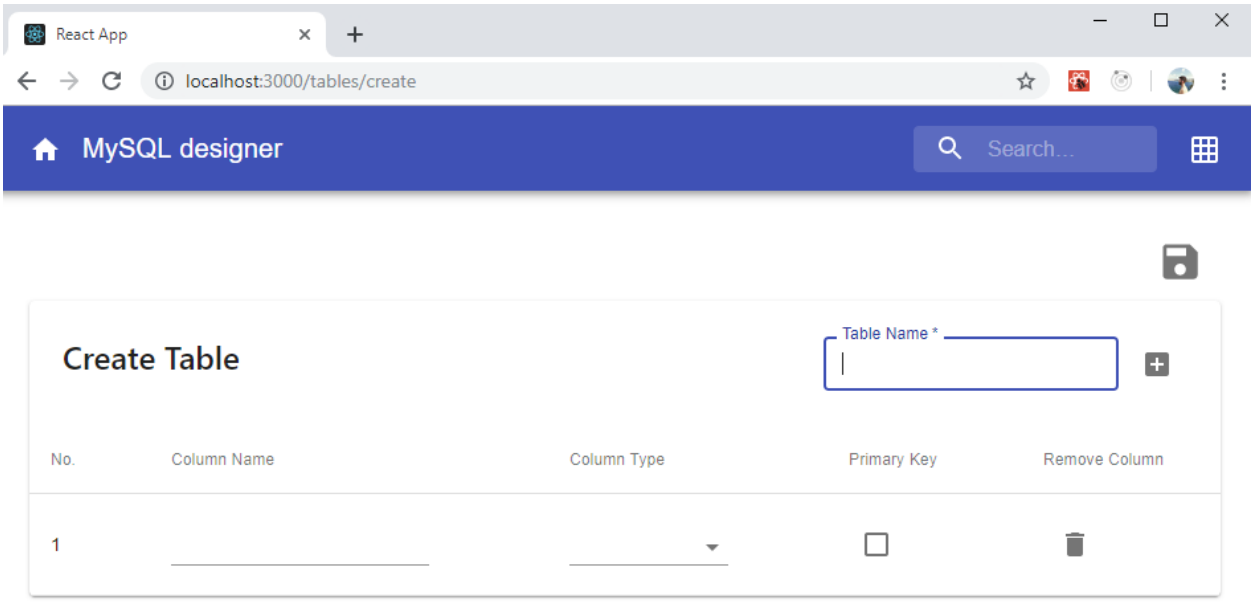


10. UI Server

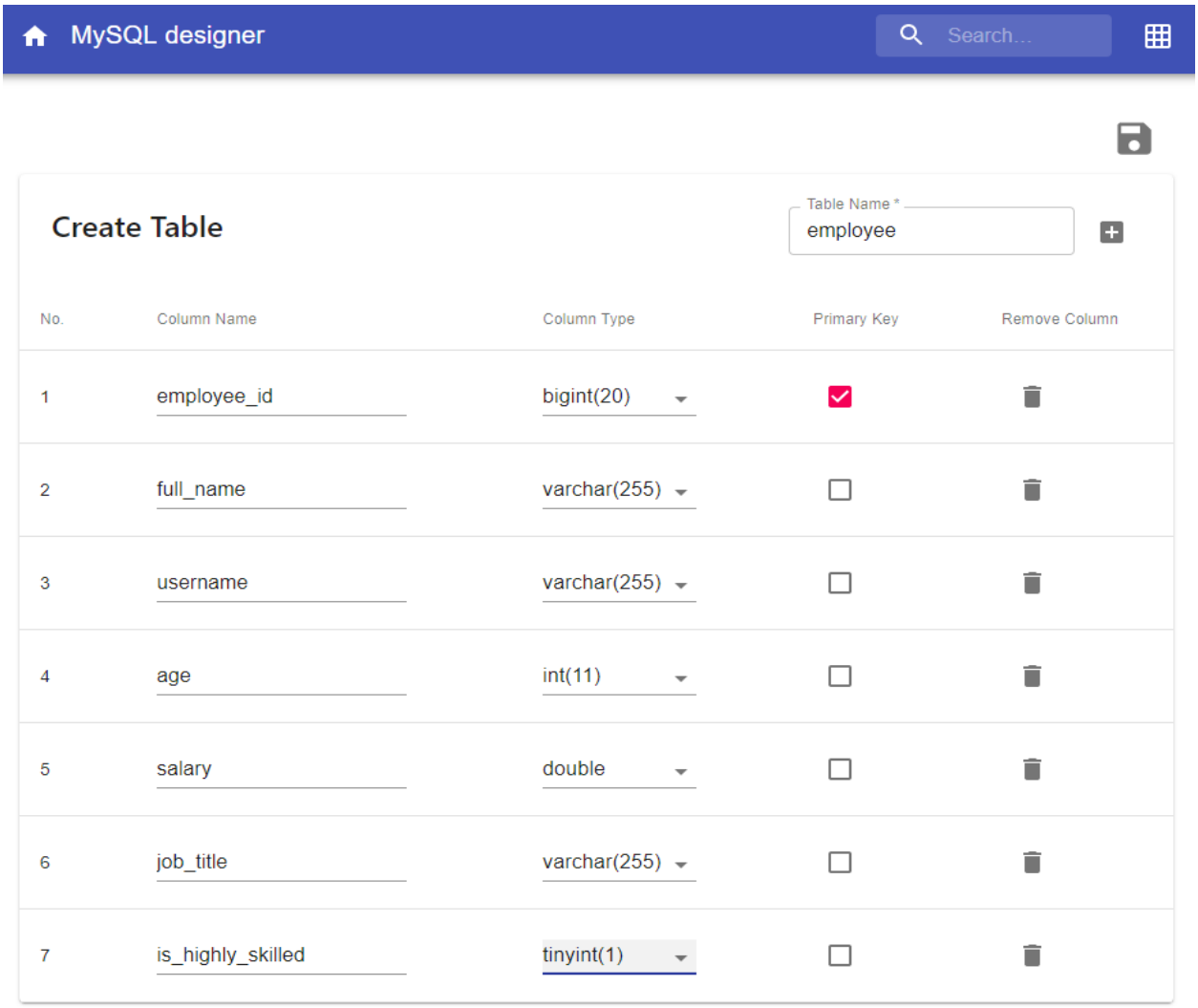


11. Home Component

Ο χρήστης πατώντας το κουμπί CREATE TABLE θα περιηγηθεί στην παρακάτω σελίδα.

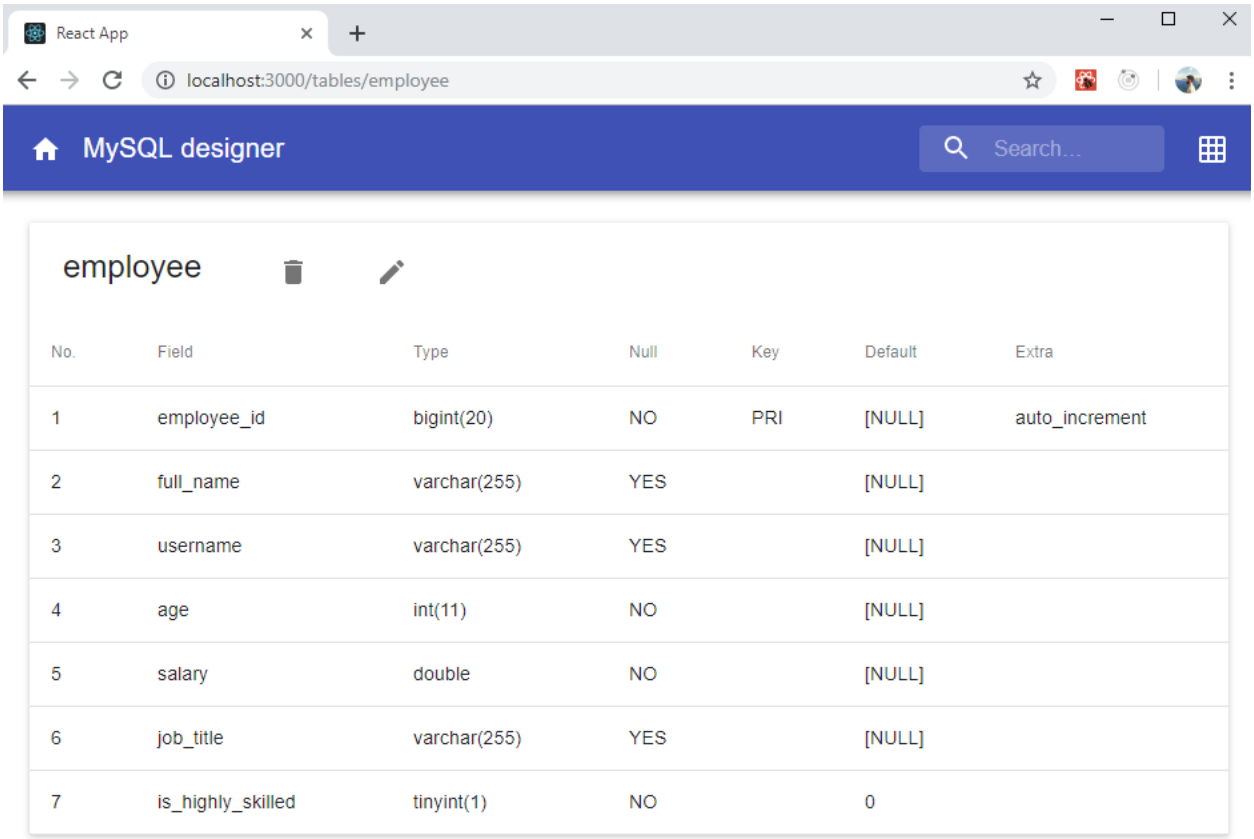


12. CreateTable Component



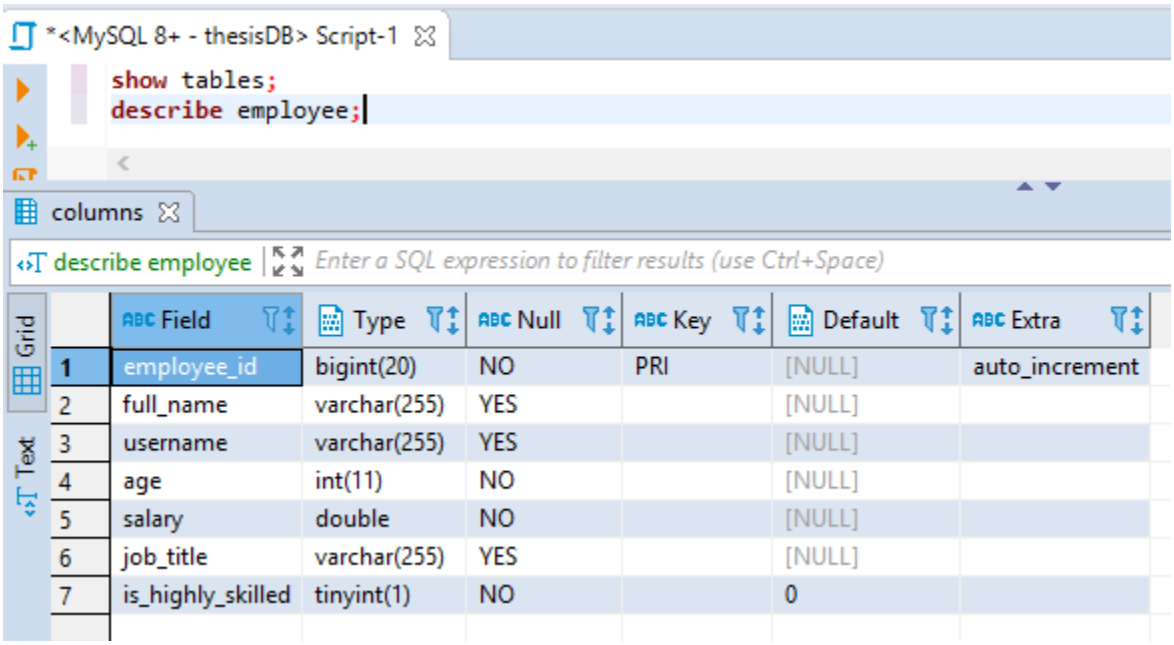
13. Εισαγωγή δεδομένων στο υπο-δημιουργία table

Το επιτυχές αποτέλεσμα της δημιουργίας του employee table, φάινεται στην παρακάτω σελίδα /tables/employee. Αμέσως μετά την δημιουργία του, το GetInfo component ερωτά την βάση μέσω του back-end και μας εμφανίζει όλες τις λεπτομέρειες για την δομή του table απο εκεί. Παράλληλα, πιστοποιούμε οτι έχει δημιουργηθεί σωστά, δηλαδή περιέχει όλες τις ζητούμενες τιμές του χρήστη, παραθέτωντας το αποτέλεσμα του sql statement **describe employee;** στην βάση δεδομένων, μέσω του γραφικού περιβάλλοντος **DBeaver**, όπως φαίνεται στην εικόνα 15.



No.	Field	Type	Null	Key	Default	Extra
1	employee_id	bigint(20)	NO	PRI	[NULL]	auto_increment
2	full_name	varchar(255)	YES		[NULL]	
3	username	varchar(255)	YES		[NULL]	
4	age	int(11)	NO		[NULL]	
5	salary	double	NO		[NULL]	
6	job_title	varchar(255)	YES		[NULL]	
7	is_highly_skilled	tinyint(1)	NO		0	

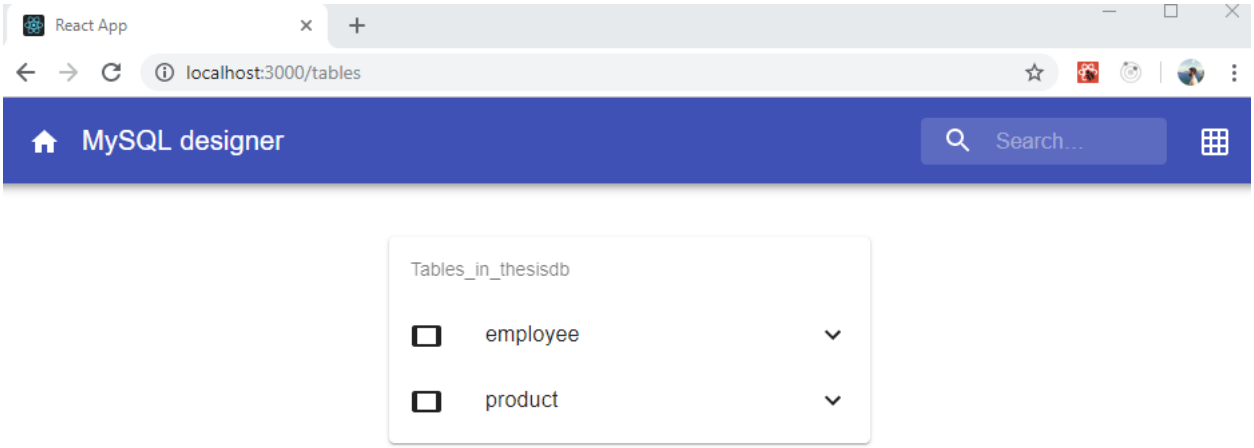
14. GetInfo Component



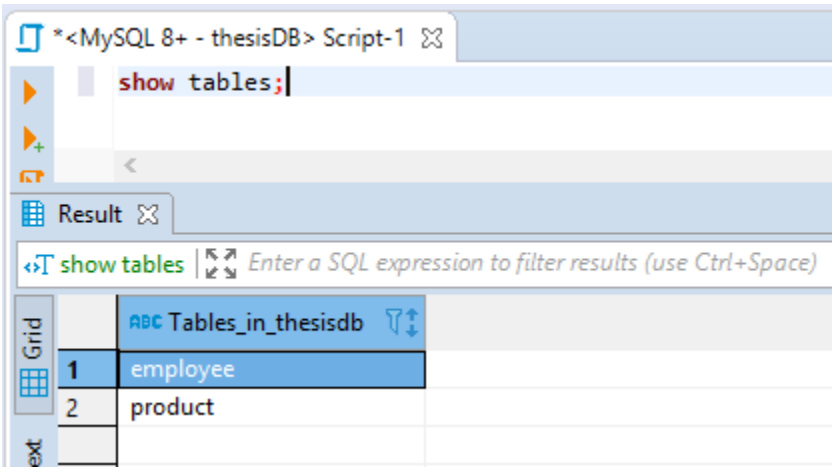
	Field	Type	Null	Key	Default	Extra
1	employee_id	bigint(20)	NO	PRI	[NULL]	auto_increment
2	full_name	varchar(255)	YES		[NULL]	
3	username	varchar(255)	YES		[NULL]	
4	age	int(11)	NO		[NULL]	
5	salary	double	NO		[NULL]	
6	job_title	varchar(255)	YES		[NULL]	
7	is_highly_skilled	tinyint(1)	NO		0	

15. DBeaver describe table command

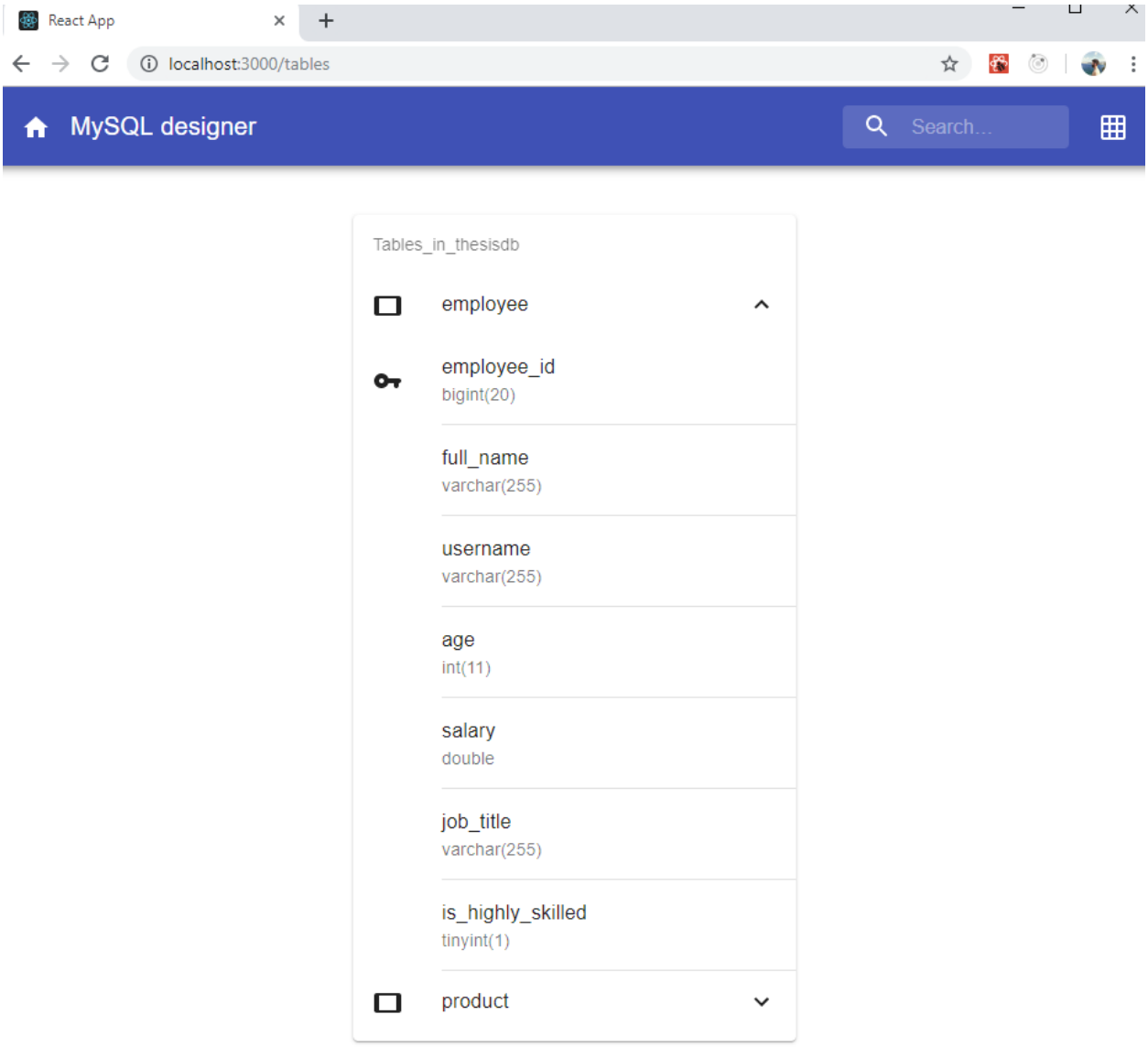
Πατώντας το Grid Icon κουμπί πάνω δεξιά στην μπάρα, ο χρήστης θα μεταφερθεί στην σελίδα /tables, όπου θα εμφανιστούν όλα τα tables που έχει δημιουργήσει. Το ίδιο ακριβώς αποτέλεσμα θα είχε και αν εκτελούσε το sql statement **show tables;** κατευθείαν στην βάση, όπως κάναμε με τη βοήθεια του DBeaver (Εικόνα 17).



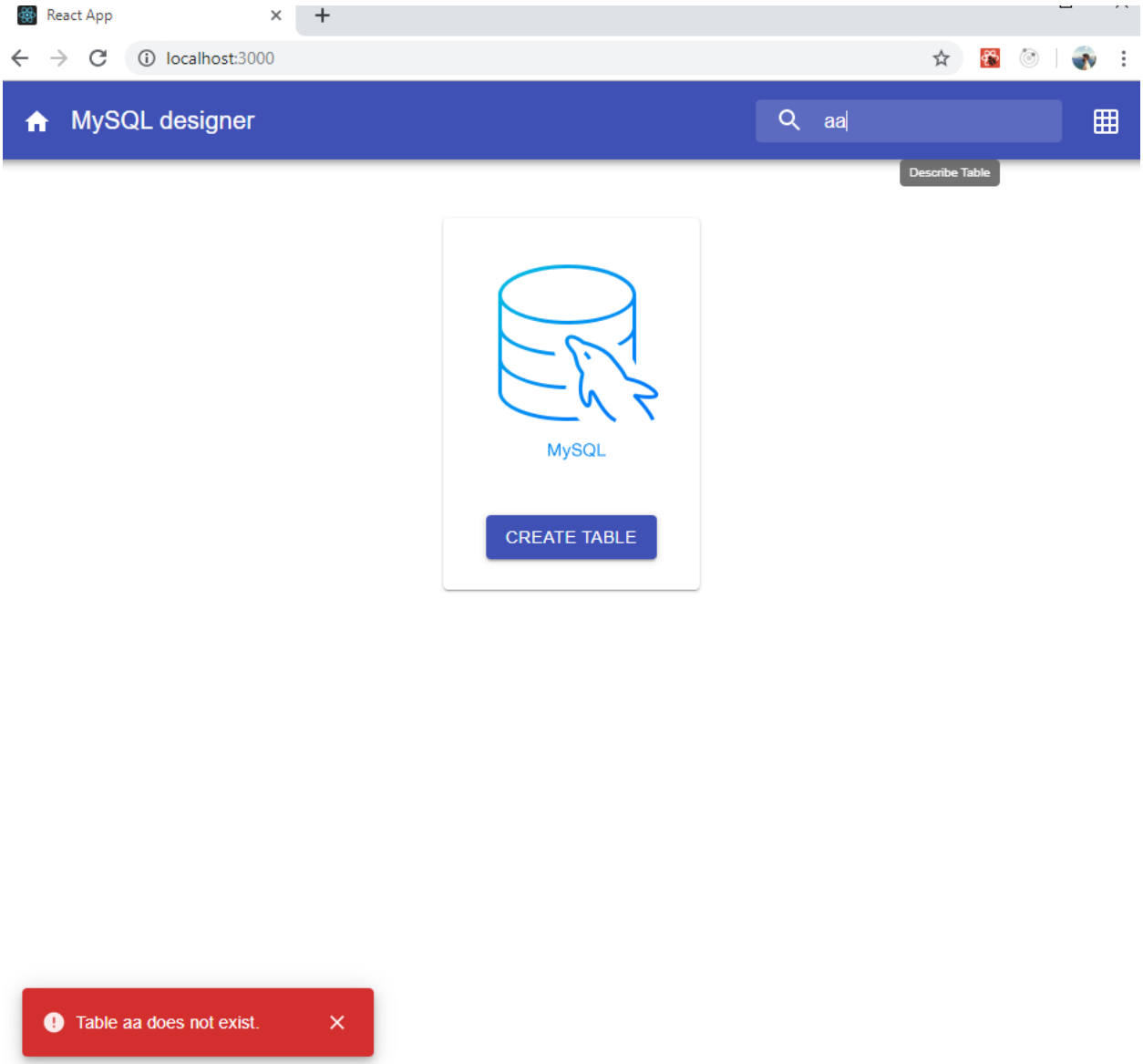
16. ShowTables Component



17. DBeaver show tables command



18. Παράδειγμα επέκτασης στοιχείου απο την λίστα με τα tables



19. Προσπάθεια αναζήτησης μη υπάρχοντος table στην βάση δεδομένων

Κεφάλαιο 5

5.1 Μελλοντικές Επεκτάσεις

Στην παρούσα φάση, η πλατφόρμα έχει στατικά μία μόνο βάση δεδομένων που έχει οριστεί στον back-end κώδικα, όπως είδαμε παραπάνω. Έτσι, όλοι οι χρήστες προς το παρών θα αλληλεπιδρούν με την ίδια βάση, γεγονός που θα προκαλέσει συγκρούσεις και περιορισμούς στην ονομασία των υπο δημιουργία tables. Μία βελτίωση της εφαρμογής θα ήταν, ο κάθε χρήστης να μπορεί να δημιουργεί την δική του βάση δεδομένων μέσω μίας σελίδας (ξεχωριστό API στο οποίο θα αντιστοιχίζεται το όνομα της βάσης με το μοναδικό αναγνωριστικό-όνομα του χρήστη) πριν αρχίσει να διαμορφώνει το schema της. Έτσι, δεν θα έχει καμία εξάρτηση με τους υπόλοιπους χρήστες.

Επιπλέον, μία ιδέα για μελλοντική επέκταση είναι η σύνδεση των tables μταξύ τους. Πιο συγκεκριμένα, να προστεθούν foreign keys ώστε να δίνεται η δυνατότητα στον χρήστη, να συσχετίζει τα tables που έχει δημιουργήσει (one-to-one, one-to-many, many-to-many relationships), ανάλογα με τις ανάγκες του.

Ακόμα μία μελλοντική επέκταση θα ήταν να δίνεται η δυνατότητα στο χρήστη να εισάγει δεδομένα στα tables που μόλις έχει δημιουργήσει, να τα μορφοποιεί και να συλλέγει πληροφορίες σε πίνακες απο δεδομένα διαφορετικών tables, εάν αυτά συνδέονται μεταξύ τους (join queries).

Τέλος, μία κρίσιμη βελτίωση της πλατφόρμας θα ήταν ο χρήστης να μπορεί να επεξεργαστεί το schema που έχει δημιουργήσει, δηλαδή να μπορεί να προσθέτει και να διαγράφει columns απο ένα υπάρχων table, να αλλάζει τον τύπο δεδομένων τους, το μήκος και το όνομα τους.

Βιβλιογραφία

[1] Rouse, M. (2019, June Monday). *RESTful API*. Retrieved from SearchApp Architecture: https://searchapparchitecture.techtarget.com/definition/RESTful-API

[2] REST API Tutorial (2019, March 9). *HTTP Methods*. Retrieved from https://restfulapi.net/http-methods

[3] Crockford, D. (n.d.). *Introducing JSON*. Retrieved from https://www.json.org

[4] Gupta, A. (2018, Aug 2). *Advanatges of MySQL*. Retrieved from Quora: https://www.quora.com/Which-are-the-advantages-of-MySQL

[5] Khirale, A. (2018, Feb 2). *7 Top ReactJS Features Which Makes It Best For Development*. Retrieved from Angular Minds: https://www.angularminds.com/blog/article/7-top-reactjs-features-which-makes-it-best-for-development.html

[6] Wikipedia. (2015, September). *Node.js*. Retrieved from https://en.wikipedia.org/wiki/Node.js

[7] npmjs. (n.d.). *About npm*. Retrieved from https://docs.npmjs.com/about-npm/

[8] freeCodeCamp. (2018, April 18). *Meet Material-UI — your new favorite user interface library*. Retrieved from https://www.freecodecamp.org/news/meet-your-material-ui-your-new-favorite-user-interface-library-6349a1c88a8c/

[9] material-ui. (n.d.). *Components*. Retrieved from https://material-ui.com/components

[10] Wikipedia. (n.d.). *Gradle*. Retrieved from https://en.wikipedia.org/wiki/Gradle

[11] Raffai, Z. (2018, Jul 30). *Spring Boot: The Most Notable Features You Should Know*. Retrieved from DZone: https://dzone.com/articles/what-is-spring-boot

[12] Crusoveanu, L. (2018, Oktober 16). *Intro to Inversion of Control and Dependency Injection with Spring*. Retrieved from Baeldung: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

[13] Wikipedia. (n.d.). *JOOQ Object Oriented Querying*. Retrieved from https://en.wikipedia.org/wiki/JOOQ_Object_Oriented_Querying