ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΧΩΡΗΜΕΝΑ ΘΕΜΑΤΑ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΕΦΑΡΜΟΓΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΕΡΓΑΣΙΑ ΓΙΑ ΤΟ ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2021-2022

ΟΜΑΔΑ MDS

ΚΩΝΣΤΑΝΤΙΝΟΣ ΘΑΝΑΣΗΣ, 2895

ΑΝΤΩΝΙΟΣ ΚΩΤΣΗΣ, 3018

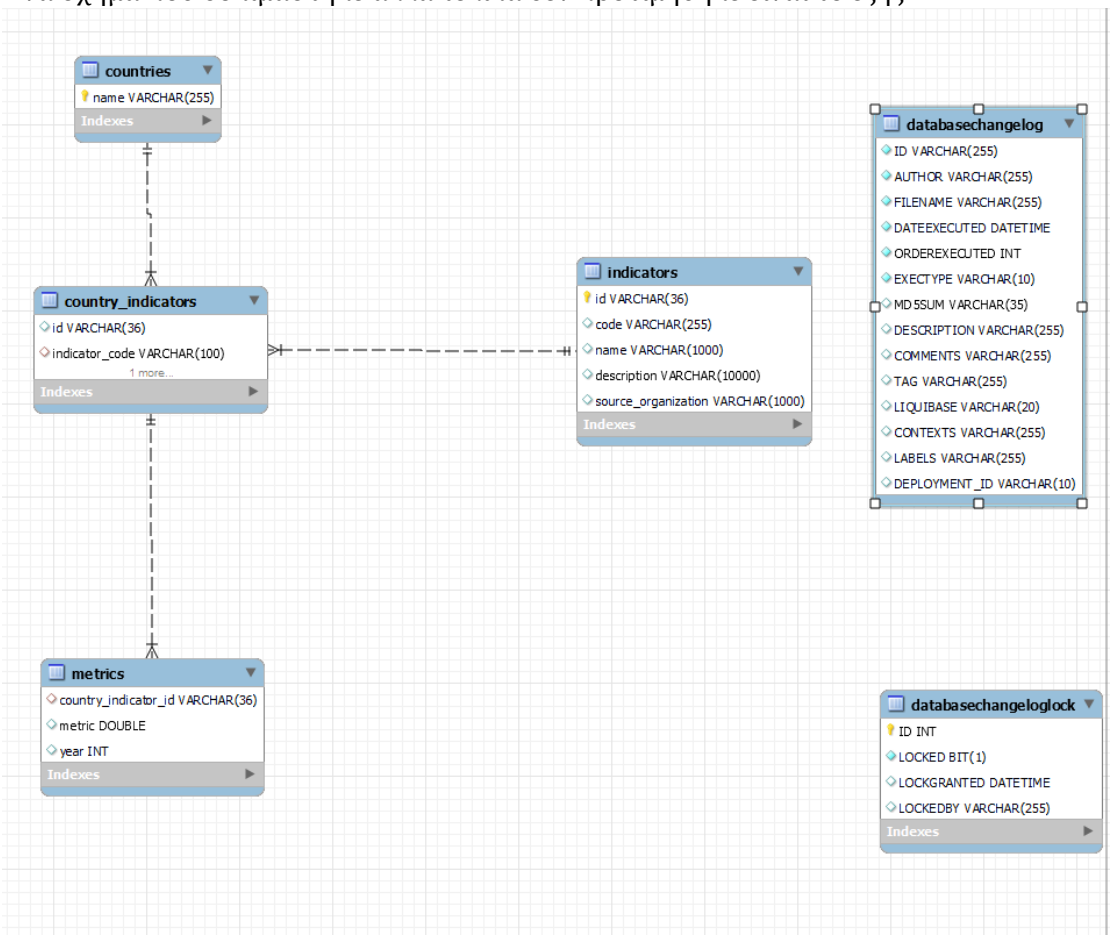
ΤΕΛΙΚΗ ΑΝΑΦΟΡΑ

ΜΑΪΟΣ 2022

1 ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

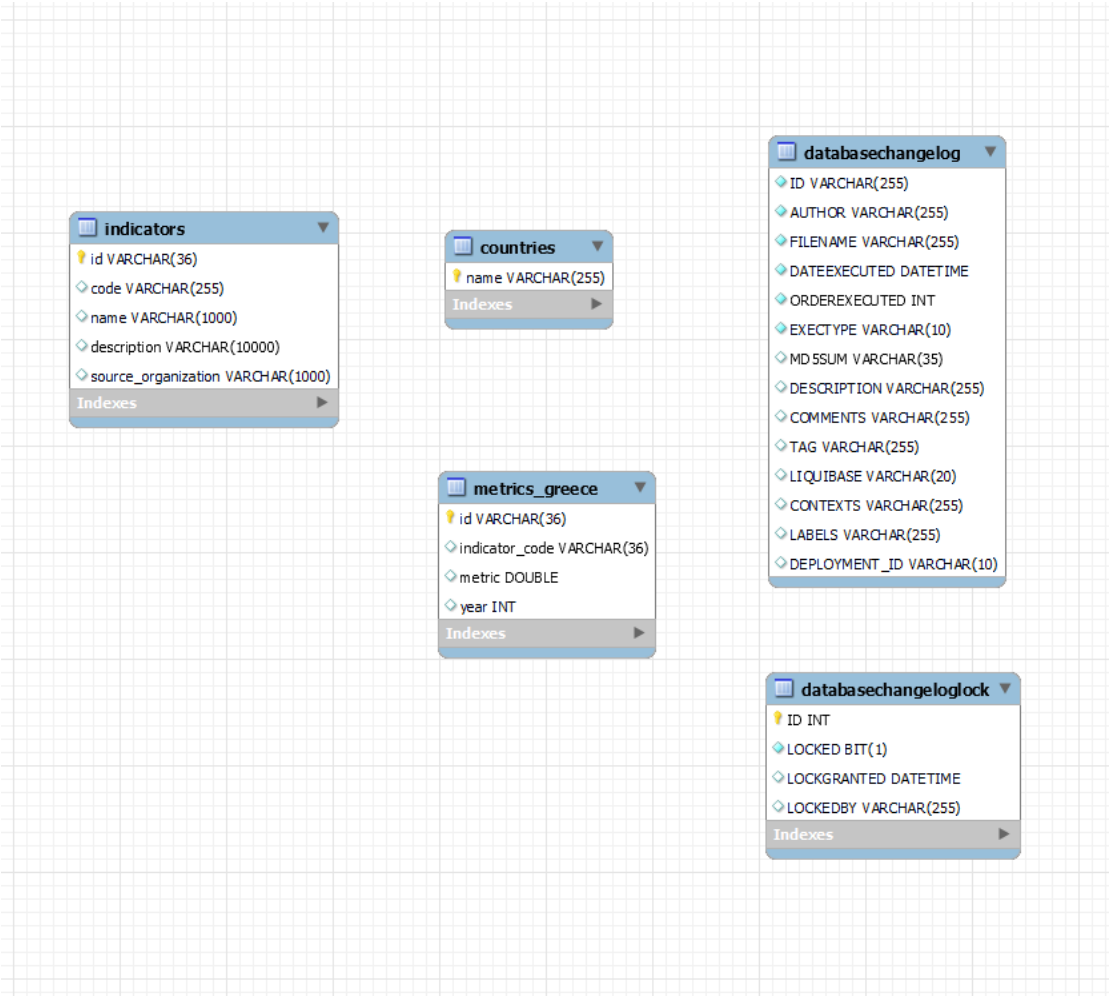
1.1 RUNNER UP SCHEMA

Ένα σχήμα που δοκιμάστηκε αλλά τελικά δεν προτιμήθηκε είναι το εξής



Εδώ όλες οι μετρικές για όλες τις χώρες βρίσκονται στον πίνακα metrics. Εκει με το foreign key βλέπουμε για ποια χώρα και ποιον κωδικό είναι η εκάστοτε μετρική. Ακόμα και με indexing είχαμε 5 δευτερόλεπτα χρόνου απόκρισης(μεταξύ request fe-be-βάση και πίσω). Έτσι επειδή πολλά δεδομένα θα είναι static και δεν περιμένουμε μεγάλες αλλαγές στην βάση το τελικό σχήμα είναι το ακόλουθο.

1.2 WINNER SCHEMA



Η τελική μορφή της βάσης είναι αρκετά απλή. Τα δυο tables στα δεξιά έχουν να κάνουν με το εργαλείο Liquibase και θα αναλυθούν στην συνέχεια. Η βάση κατά τα άλλα έχει 2 σχέσεις, μία να κρατάει πληροφορία για τις χώρες, που στην φάση αυτή είναι μόνο το όνομα αλλά θα μπορούσαν να προστεθούν και επιπλέον πληροφορίες στο μέλλον. Έτσι προτιμήθηκε να γίνει ξεχωριστός πίνακας. Η άλλη έχει πληροφορία για κάθε μοναδικό indicator, δηλαδή το όνομα την πηγή τον κωδικό κλπ κλπ.
Έπειτα έχουμε για κάθε χώρα στο σύστημα ένα table metrics_<όνομα χώρας> όπου διατηρείται πληροφορία για κάθε μετρητή για κάθε έτος της συγκεκριμένης χώρας.

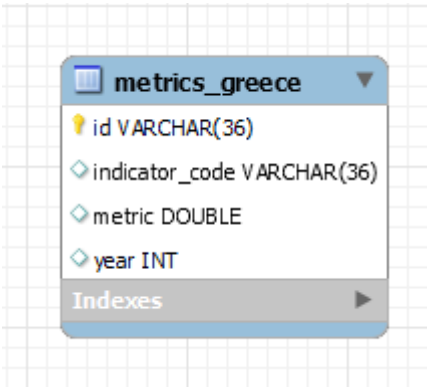
Αυτή η δομή της βάσης έχει 3 σημαντικά trade-offs

1. Για τα περισσότερα, αλλά και συχνότερα query δεν απαιτείται κανένα join, και η πληροφορία αναζητείται με βάση Indexed πεδία
2. Για κάθε νέα χώρα που θα προστίθεται θα πρέπει να κάνουμε καινούριο table.
3. Πρέπει δυναμικά να αποφασίζουμε σε ποιο table να εκτελέσουμε το query, πράγμα που φωνάζει sql-injection αν δεν διαχειριστεί κατάλληλα

Παραλείπεται εδώ για να μην μεγαλώσει τρομακτικά το documentation, για αναλυτικά creates και γενικά το evolve του σχήματος της βάσης, υπάρχουν τα Liquibase αρχεία.

1.2.1 ΡΥΘΜΙΣΗ ΤΟΥ ΦΥΣΙΚΟΥ ΣΧΗΜΑΤΟΣ ΤΗΣ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ

Στο API αναμένουμε κάποια queries τα οποία είναι και συχνά αλλά και σε πίνακες πολλών δεδομένων, με κοινές όμως κάποια μεταβλητές. Θα είναι πάνω στους πίνακες metrics_xxx με την ακόλουθη δομή και θα έχουν υποχρεωτικά κάποιο indicator code



Δηλαδή θα είναι της μορφής

SELECT * from metrics_greece WHERE indicator_code = “xxx” | and year < something | | <and year > something>” |

Με τα δυο year checks να είναι προαιρετικά. Δημιουργώντας τα παρακάτω indexes:

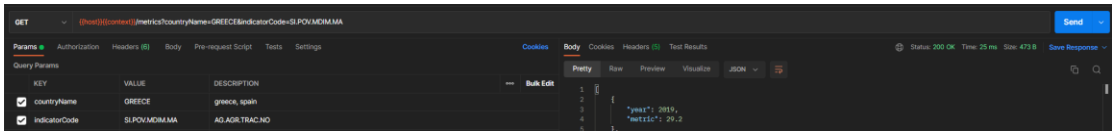
```
create table metrics_greece
(
    id                  varchar(36) ,
    indicator_code      varchar(36) ,
    metric              double precision,
    year                integer,
    primary key (id)
);

CREATE INDEX metrics_greece_indicator_code_index
    ON metrics_greece (indicator_code);

CREATE INDEX metrics_greece_indicator_code_year_index
    ON metrics_greece (indicator_code,year);
```

```
mysql> explain select * from metrics_greece where indicator_code = "FM.AST.PRVT.GD.ZS";
+----+-------------+----------------+------------+------+-----------------------------------------------------------------------------------+-----------------------------------+---------+-------+------+----------+-------+
| id | select_type | table          | partitions | type | possible_keys                                                                     | key                               | key_len | ref   | rows | filtered | Extra |
+----+-------------+----------------+------------+------+-----------------------------------------------------------------------------------+-----------------------------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | metrics_greece | NULL       | ref  | metrics_greece_indicator_code_index,metrics_greece_indicator_code_year_index | metrics_greece_indicator_code_index | 147     | const |   20 |   100.00 | NULL  |
+----+-------------+----------------+------------+------+-----------------------------------------------------------------------------------+-----------------------------------+---------+-------+------+----------+-------+
```

```
mysql> explain select * from metrics_greece where indicator_code = "FM.AST.PRVT.GD.ZS" and year > 1970;
+----+-------------+----------------+------------+------+---------------------------------------------------------------------+-----------------------------------+---------+-------+------+----------+-------------+
| id | select_type | table          | partitions | type | possible_keys                                                       | key                               | key_len | ref   | rows | filtered | Extra       |
+----+-------------+----------------+------------+------+---------------------------------------------------------------------+-----------------------------------+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | metrics_greece | NULL       | ref  | metrics_greece_indicator_code_index,metrics_greece_indicator_code_year_index | metrics_greece_indicator_code_index | 147     | const |   20 |    33.33 | Using where |
+----+-------------+----------------+------------+------+---------------------------------------------------------------------+-----------------------------------+---------+-------+------+----------+-------------+
```

```
mysql> explain select * from metrics_greece where indicator_code = "FM.AST.PRVT.GD.ZS" and year > 1970 and year < 2000;
+----+-------------+----------------+------------+-------+---------------------------------------------------------------------+-----------------------------------+---------+------+------+----------+-----------------------+
| id | select_type | table          | partitions | type  | possible_keys                                                       | key                               | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+----------------+------------+-------+---------------------------------------------------------------------+-----------------------------------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | metrics_greece | NULL       | range | metrics_greece_indicator_code_index,metrics_greece_indicator_code_year_index | metrics_greece_indicator_code_year_index | 152     | NULL |    1 |   100.00 | Using index condition |
+----+-------------+----------------+------------+-------+---------------------------------------------------------------------+-----------------------------------+---------+------+------+----------+-----------------------+
```

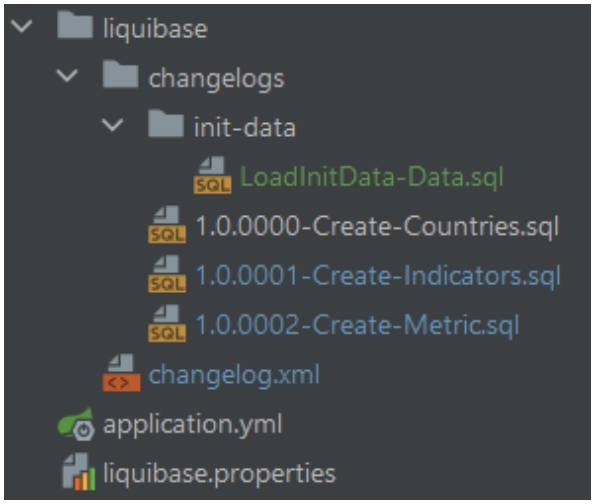
Όπως φαίνεται από τα άνωθι screen dumps, στα αναμενόμενα query στο table έχουμε indexes τα οποία καταφέρνουν να αξιοποιηθούν στο έπακρο καταφέρνοντας filtering ~100%. Εδώ έχουμε πληρώσει indexes σε τόσα tables όσο και χώρες * 2(2 διαφορετικά Index για κάθε table) αλλά έχουμε κατέβει σε χρόνους. Ενδεικτικά ένα query στον πίνακα αυτό με την αρχική δομή έκανε περίπου 5.7sec ενώ τώρα η διαδρομή με Rest σε back-end βάση και πίσω είναι στα 25 ms



1.3 DB MIGRATION/ BACK UP

Για την αρχική κατασκευή της βάσης, αλλά και για την διατήρηση ενός back up χρησιμοποιήθηκε το Liquibase. Με την χρήση αυτού η βάση χτίζεται προεδευτικά με μικρά scriptakia ή αλλιώς changelogs. Αυτά τρέχουν μία φορά(αν δεν οριστεί αλλιώς) όταν ξεκινήσει το πρόγραμμα η on demand. Έτσι μέσω αυτού και του αντίστοιχου plugin της maven, μπορούμε να δημιουργήσουμε τη βάση από το 0.

Η δομή των αρχείων που θα χρειάζεται το Liquibase είναι η εξης:



Τα scriptakia τρέχουν με την σειρά που ορίζεται στο αρχείο changelog

```xml
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog logicalFilePath="li"
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
      http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.1.xsd">

    <include relativeToChangelogFile="true" file="changelogs/1.0.0000-Create-Countries.sql"/>
    <include relativeToChangelogFile="true" file="changelogs/1.0.0001-Create-Indicators.sql"/>
    <include relativeToChangelogFile="true" file="changelogs/1.0.0002-Create-Metric.sql"/>
<!--    <include relativeToChangelogFile="true" file="changelogs/init-data/LoadInitData-Data.sql"/>-->

</databaseChangeLog>
```

Ενώ στην βάση δημιουργούνται 2 tables για την διατήρηση στοιχείων σχετικά με το ποια scripts έχουν τρέξει και πότε.

```
189 •  SELECT * from databasechangelog;
190
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

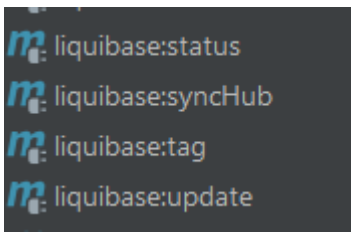
ID	AUTHOR	FILENAME	DATEEXECUTED	ORDEREXECUTED	EXECTYPE	MD5SUM	DESCRIPTION	COMMENTS	TAG	LIQUIBASE	CONTEXTS	LABELS	DEPLOYMENT_ID
0	mds	src/main/resources/liquibase/changelogs/1.0.00...	2022-05-24 17:43:21	1	EXECUTED	8:b20158f08cab990014f95ccd9f54c19a	sql		NULL	4.2.0	NULL	NULL	3403401328
1	mds	src/main/resources/liquibase/changelogs/1.0.00...	2022-05-24 17:43:21	2	EXECUTED	8:f909819395d8869440e0609bbaeba259	sql		NULL	4.2.0	NULL	NULL	3403401328
2	mds	src/main/resources/liquibase/changelogs/1.0.00...	2022-05-24 17:43:25	3	EXECUTED	8:d53e2f5502fdbfb3ce22263a08e87d7c	sql		NULL	4.2.0	NULL	NULL	3403401328

Ενώ το author και id ορίζονται μέσα στα εκάστοτε αρχεία.

```sql
--liquibase formatted sql

--changeset mds:0
create table countries
(
    name varchar(255) not null,
    primary key (name)
);
```

Τέλος στο Liquibase.properties ορίζονται οι μεταβλητές που θα χρειαστεί το Liquibase για να συνδεθεί στην βάση.

Όπως αναφέρθηκε και πριν το να αναδημιουργήσουμε την βάση από το 0, είναι αρκετά απλό χρησιμοποιώντας το update goal του Liquibase plugin



2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΛΟΓΙΣΜΙΚΟΥ

2.1 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ ΔΟΜΗ ETL

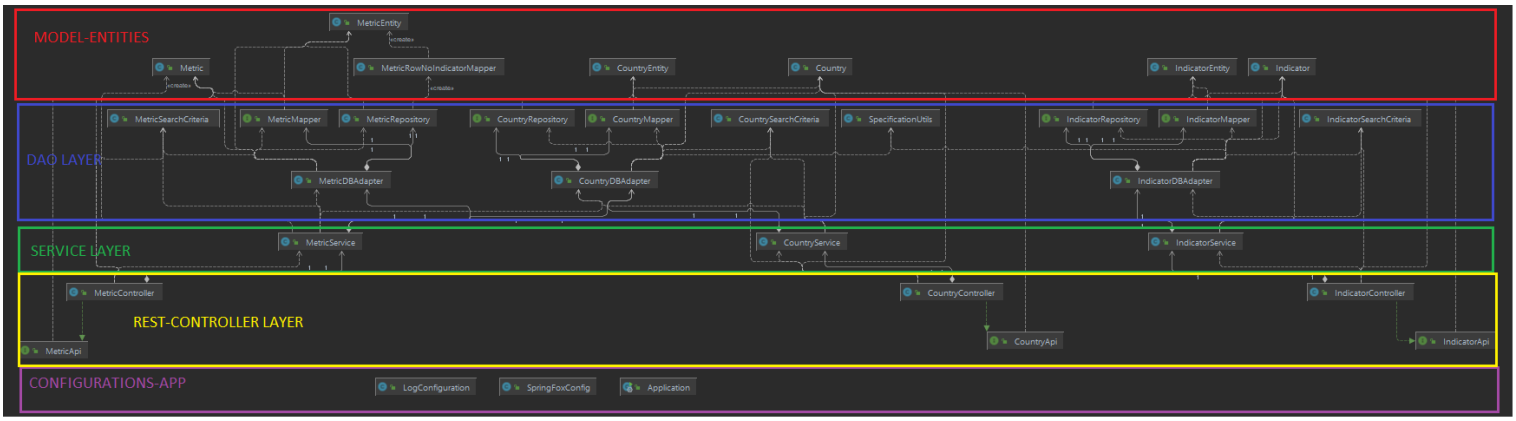
Τα αρχεία που επιλέξαμε είναι 25 χώρες της Ευρώπης και όλες οι μετρικές αυτών για όλα τα έτη. Έχοντας τα αντίστοιχα αρχεία κατεβασμένα τοπικά και με την χρήση ενός python script, μετατράπηκαν σε csv αρχεία αντίστοιχα των tables που έχουμε στην βάση. Στην συνέχεια με την χρήση του Liquibase που αναφέρεται παραπάνω δημιουργήσαμε την βάση και φορτώσαμε τα αρχεία αυτά.

Τα αρχικά(και μοναδικά δεδομένα) φορτώνονται στην βάση μέσω του **LoadInitData.sql** αρχείου, το οποίο περιέχει ένα μάτσο Load Data Infile. Η διαδικασία για να επαναληφθεί σε περίπτωση ολοκληρωτικής καταστροφής της βάσης χωρίζεται σε 2 απλά βήματα.

1. Τρέξιμο του python script για να δημιουργήσει τα αρχεία
2. Τρέξιμο του Liquibase που δημιουργεί την βάση από το 0 και φορτώνει τα εν λόγο αρχεία

2.2 ΔΙΑΓΡΑΜΜΑΤΑ ΚΛΑΣΕΩΝ

Το διάγραμμα κλάσεων είναι το εξής:



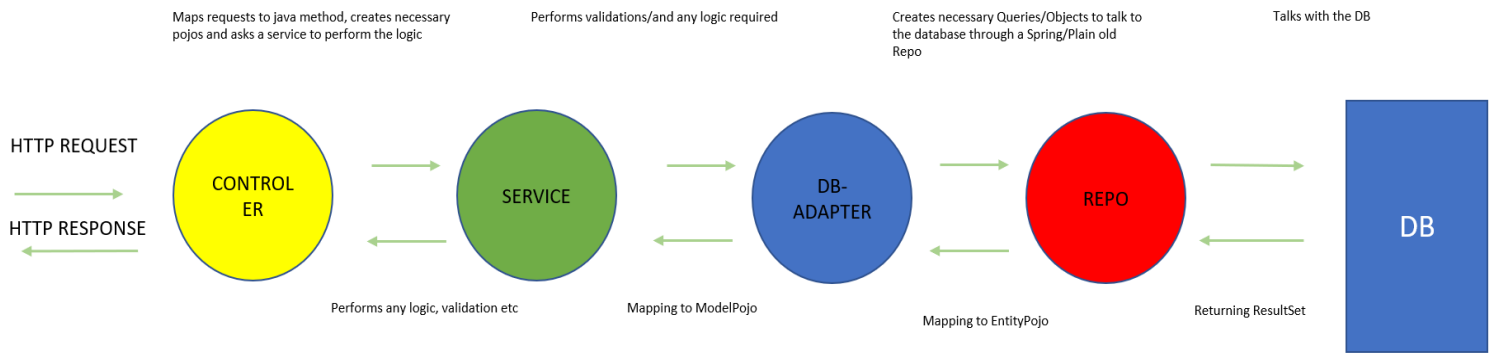
Κάτω κάτω(η πάνω-πάνω) έχουμε τα entities και τα model pojos. Πάνω από αυτά είναι το DAO layer με τα Repositiories και του «διαχειριστές» αυτών.

Στο επόμενο layer έχουμε τα Services όπου υλοποιούν την όποια λογική η απλά συνδέουν τους controllers με το DAO layer.

Στο τελευταίο layer έχουμε τους Controllers που μαραρουν τα HTTP Requests και τα πασάρουν στο αντίστοιχο Service.

Τέλος έχουμε το App και τα όποια configurations.

Γενικά η λογική που ακολουθείται σε κάθε “διαδρομή” rest-db και πίσω είναι:



3 ΥΠΟΔΕΙΓΜΑΤΑ ΕΡΩΤΗΣΕΩΝ ΚΑΙ ΑΠΑΝΤΗΣΕΩΝ

Μόλις το request φτάσει στον back-end το διαχειρίζεται ο indicator-controller (μιας και θέλουμε indicator resources)

Ο controller φτιάχνει το αντίστοιχο pojo

```
@Override
public ResponseEntity<List<Indicator>> findByCriteria(List<String> id, List<String> code, String name, String sourceOrganization) {

    IndicatorSearchCriteria criteria = IndicatorSearchCriteria.builder().id(id)
            .code(code)
            .name(name)
            .sourceOrganization(sourceOrganization)
            .build();

    return ResponseEntity.ok(indicatorService.findByCriteria(criteria));
}
```

και καλεί το αντίστοιχο service

Το service τρέχει το αντίστοιχο logic αν υπάρχει και κάνει μιλάει με την βάση μέσω ενός adapter.

```
public List<Indicator> findByCriteria(IndicatorSearchCriteria criteria) {
    return indicatorDBAdapter.findByCriteria(criteria);
}
```

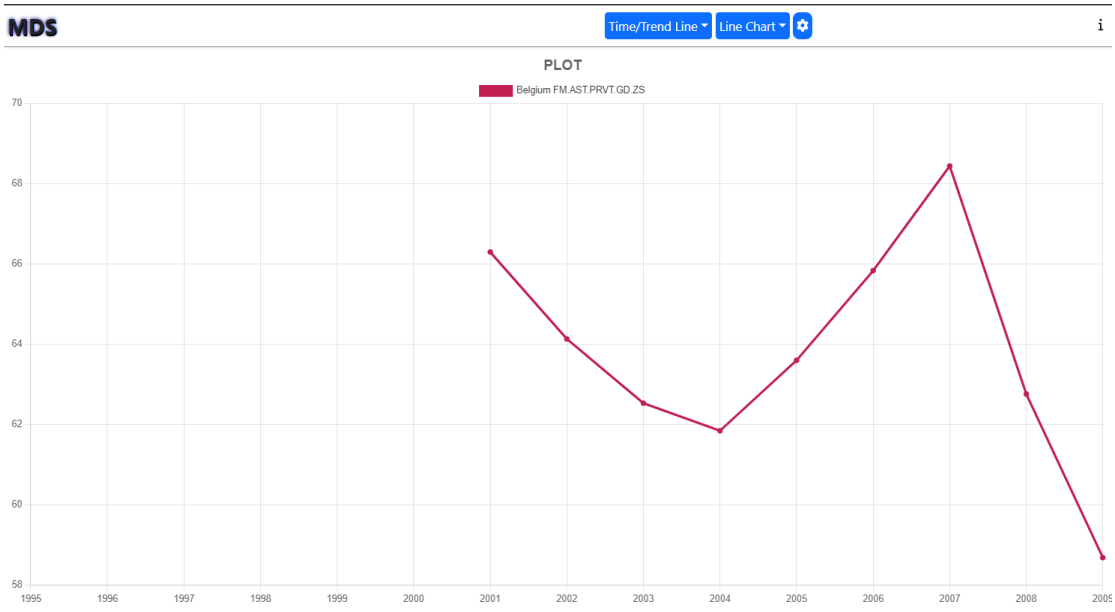
Ο db adapter φτιάχνει τα αντίστοιχα Specification αν μιλάει με SpringRepo η Query αν μιλάει με JavaRepository

```
public List<Indicator> findByCriteria(IndicatorSearchCriteria criteria) {
    Specification<IndicatorEntity> specification = where(hasIdIn(criteria.getId()))
            .and(hasCodeIn(criteria.getCode()))
            .and(hasShortDescriptionLike(criteria.getName()))
            .and(hasSourceOrganizationLike(criteria.getSourceOrganization()));

    List<IndicatorEntity> indicatorEntities = indicatorRepository.findAll(specification);

    return indicatorMapper.entityToModel(indicatorEntities);
}
```

Εκεί αν είναι Spring Repo δεν έχουμε κάτι να κάνουμε αφού γίνεται implement από το Spring Boot

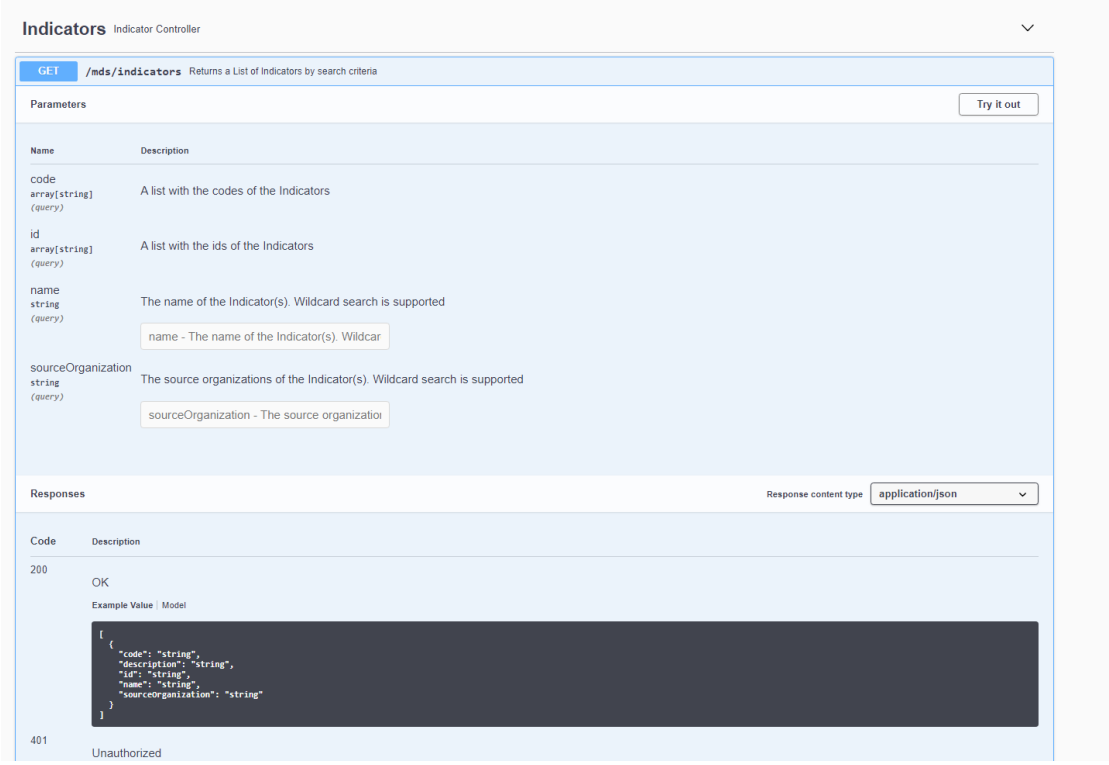
```
@Repository
public interface IndicatorRepository extends JpaRepository<IndicatorEntity, String>, JpaSpecificationExecutor<IndicatorEntity> {
}
```

Από εκεί και έπειτα παίρνουμε τον δρόμο για πίσω και τα το τελικό chart



4 DOCUMENTATION

Ένας από τους στόχους του project είναι το καθαρό documentation από το back-end στο front end. To this end, χρησιμοποιήσαμε το Open-API specification με implementation του Spring. Αυτό με κάποια annotations δικά του αλλά διαβάζοντας και τα αντίστοιχα του Spring Boot κάνει generate ένα web-based HTTP client οποίος παρέχει και το αντίστοιχο Documentation.



Μέσα από εδώ μπορούμε να δούμε τα διαθέσιμα endpoints, τα επιστρεφόμαι dto, καθώς και κάποιο doc που μπορεί να έχει γραφεί εξηγώντας περαιτέρω τι κάνει το κάθε endpoint.

5 FRONT END

5.1 ΔΟΜΗ ΚΑΙ ΔΗΜΙΟΥΡΓΙΑ

Για την υλοποίηση του front end μέρους της εφαρμογής χρησιμοποιήσαμε την γλώσσα TypeScript και το framework React. Επιπλέον χρησιμοποιήσαμε τις βιβλιοθήκες `react-bootstarp, fortAwesome, materialUi, SASS` για το styling, react-chart-2, η οποία είναι υλοποιημένη με `d3.js`, για τα charts και την βιβλιοθήκη `axios` για τα http requests . Επίσης χρησιμοποιήθηκε ο package manager yarn.

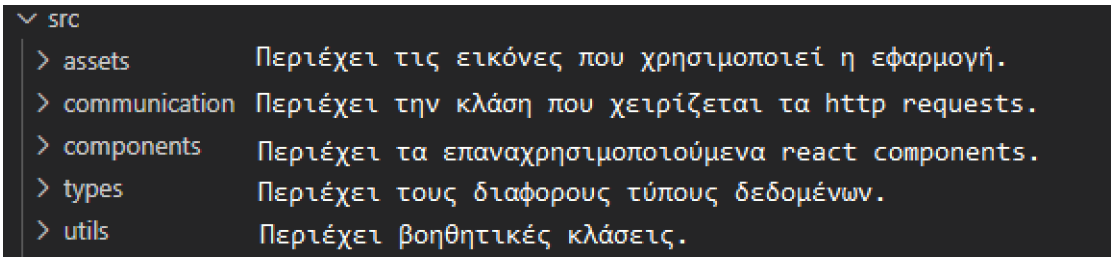
Ο βασικός κορμός της εφαρμογής δημιουργήθηκε με την εντολή

```
yarn create react-app <my-app>
```

η οποία δημιουργεί το παρακάτω structure

```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
    └── setupTests.js
```

Συγκεκριμένα, όλος ο κώδικας της εφαρμογής βρίσκεται κάτω από τον φάκελο src με το εξής format.

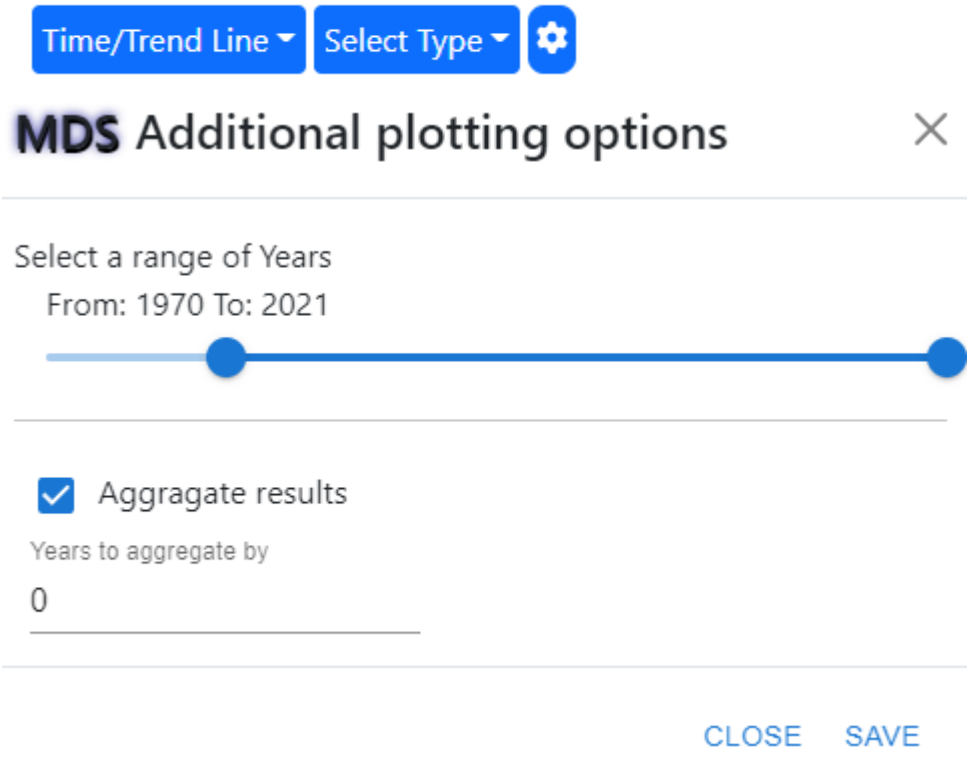


Λόγο της μικρής έκτασης της εφαρμογής δεν χρησιμοποιήθηκε κάποιο state management tool για την επικοινωνία των components μεταξύ τους. Αυτή επιτυγχάνεται με πέρασμα του state του εκάστοτε component στο άλλο μέσω των props του.

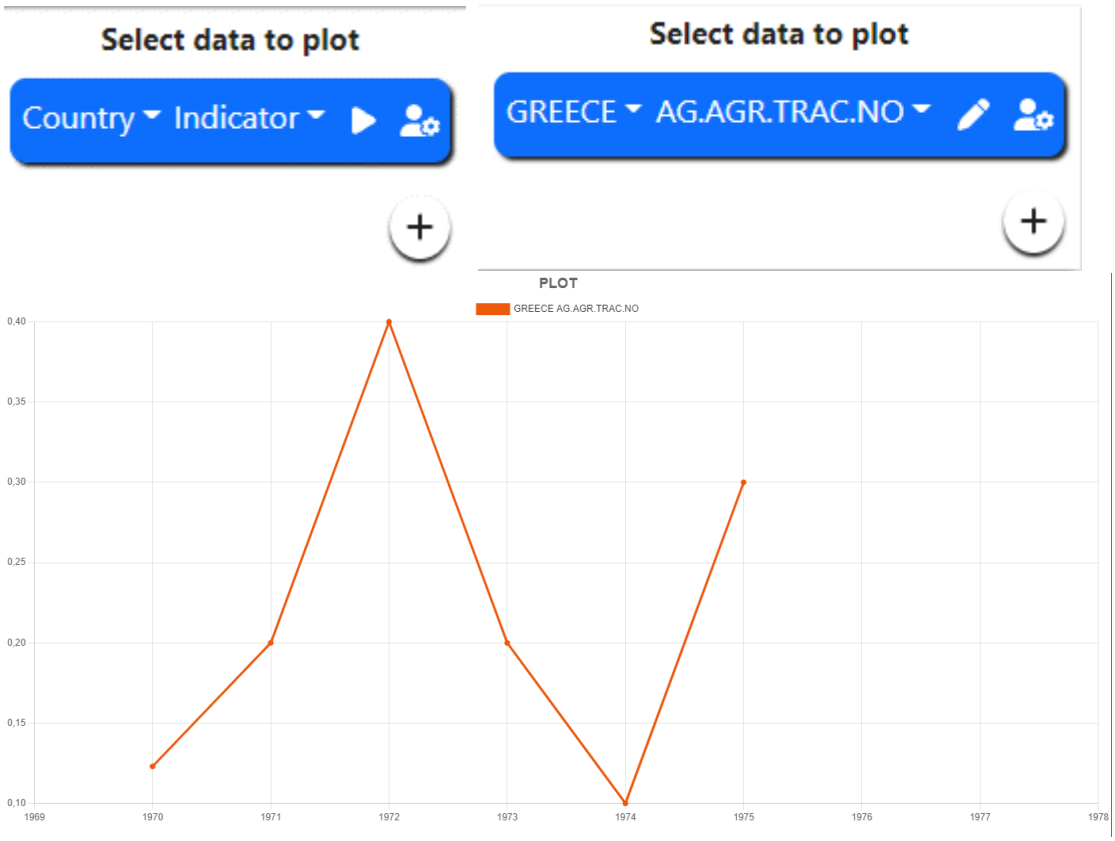
5.2 ΧΡΗΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Ξεκινώντας την εφαρμογή, ο χρήστης βλέπει ένα modal με βασικές οδηγίες χρήσης της εφαρμογής, ενώ παράλληλα στέλνονται δυο requests στο backend, ένα get για να πάρουμε όλες τις χώρες και ένα ακόμα get για να πάρουμε όλους τους indicators τους οποίους και κρατάμε στην μνήμη.

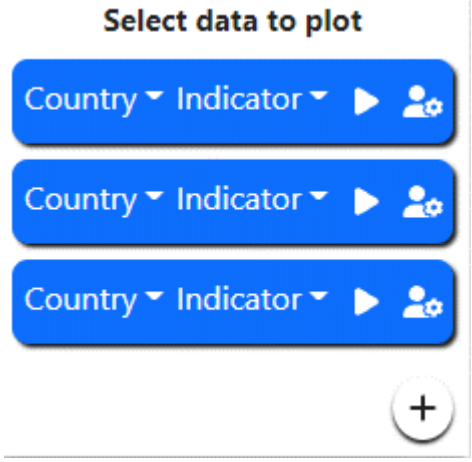
Ο χρήστης στην συνέχεια θα πρέπει να επιλέξει ποια οικογένεια και τι είδος chart θέλει να δει αλλά και να σεταρει εξτρά επιλογές οι οποίες θα ρυθμίσου τις παραμέτρους των requests που θα σταλθούν.



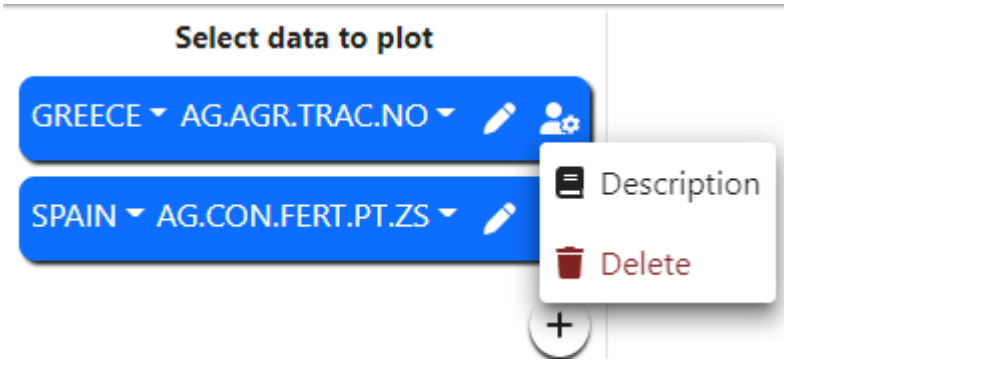
Με το που γίνει η επιλογή του τύπου του chart, εμφανίζεται στα αριστερά της οθόνης μια μπάρα στην οποία υπάρχουν 2 dropdowns ένα με τις διαθέσιμες χώρες και ένα με τις διαθέσιμες indicators. Πατώντας το εικονίδιο play, στέλνεται στο backend request με τις αντίστοιχες παραμέτρους που έχει διαλέξει ο χρήστης.

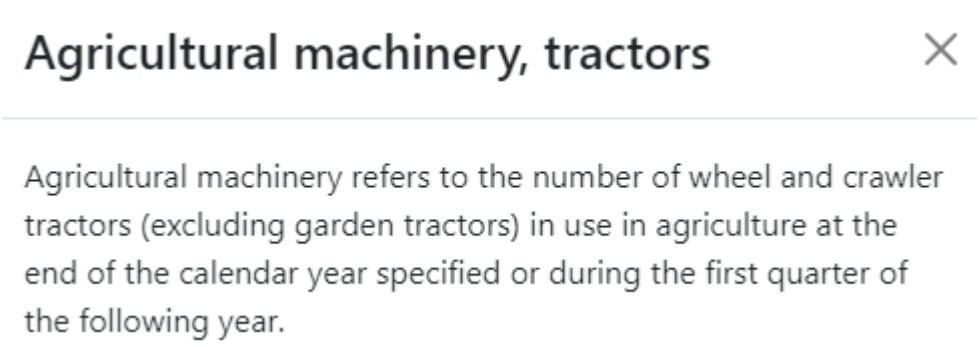


Με το εικονίδιο + μπορεί να προσθέσει εξτρά επιλογή

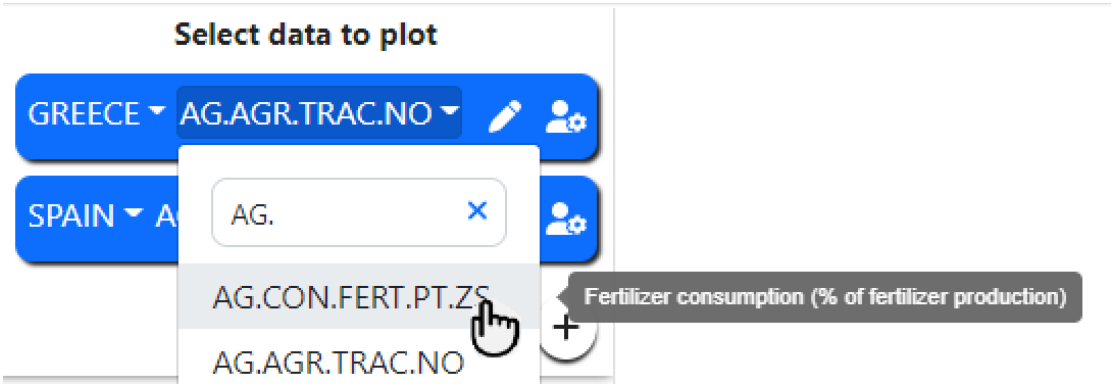


Ενώ πατώντας το εικονίδιο με το ανθρωπάκι εμφανίζονται δυο επιλογές. Η πρώτη είναι να δει την λεπτομερή περιγραφή του δείκτη που επέλεξε ενώ η άλλη είναι να διαγράψει την επιλογή του.





Παράλληλα, όταν ο χρήστης κάνει hover πάνω από μια τιμή του dropdown του εμφανίζεται tooltip με το όνομα του δείκτη, ενώ μπορεί και να φιλτράρει τις επιλογές μέσω του search field.



6 SET-UP IN LOCAL MACHINE

Για να τρέξει τοπικά το project από πλευράς be χρειάζεται maven installed και ένα my-sql spring compatible version. Έπειτα δημιουργούμε την βάση και βάζουμε τα αντίστοιχα properties στα αρχεία application.properties και Liquibase properties.

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/localdb
    username: root
    password: K59785978@m
```

```
changeLogFile=src/main/resources/liquibase/changelog.xml
url=jdbc:mysql://localhost:3306/localdb
username=root
password=K59785978@m
logLevel=info
driver=com.mysql.cj.jdbc.Driver
```

Στην συνέχεια είτε τρέχουμε το Liquibase

1. Είτε με το χέρι μεσω του plugin
2. Ειτε μεσω terminal -> `mvn liquibase:update`
3. Είτε το βάζουμε να τρέξει κατά το start-up με το αντίστοιχο property

```
liquibase:
  enabled: false
  drop-first: false
  change-log: liquibase/changelog.xml
```

.

Αν θέλουμε και τα demo δεδομένα θα πρέπει να τρέξουμε το scriptακι της python και να τοποθετήσουμε τα αρχεία που θα παράξει σε ένα secure location στο local μας για να τα τραβήξει η sql.

Μπορούμε να δούμε το secure local dir μέσω:

- SHOW VARIABLES LIKE `secure_file_priv`

```
190
191 •  SHOW VARIABLES LIKE 'secure_file_priv';
192
```

Result Grid | Filter Rows: | Export: | Wrap Cell C

Variable_name	Value
secure_file_priv	C:\ProgramData\MySQL\MySQL Server 8.0\Upl...

Τέλος τρέχουμε το App είτε μέσω terminal:

- mvn spring-boot:run

Είτε πηγαίνοντας στο αρχείο App.java και εκτελώντας το μέσω του πράσινου βέλους του ide

Για να τρέξει το front-end θα πρέπει να υπάρχει εγκατεστημένη έκδοση του

nodeJS > =16.14.2 (recommended 16.14.2) καθώς και yarn >= 1.22.8.

Στον φάκελο ui/db αρχικά θα πρέπει να τρέξει η εντολή

- yarn install

ώστε να εγκατασταθούν όλα τα απαραίτητα πακέτα και στην συνέχεια το app είναι έτοιμο να ξεκινήσει με την εντολή

- yarn start

Το app θα ξεκινήσει αυτόματα στο port 3000.