Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Spring Boot Web Service για μετρήσεις ατμοσφαιρικών ρύπων** **Spring Boot Web Service for atmospheric pollutants measurement**
Ονοματεπώνυμο Φοιτητή	**Γεράσιμος Ευαγγελάτος**
Πατρώνυμο	**Σταύρος**
Αριθμός Μητρώου	**ΜΠΣΠ/ 18012**
Επιβλέπων	**Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης **Νοέμβριος 2020**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Κωνσταντίνος Πατσάκης Επίκουρος Καθηγητής

Περιεχόμενα

Περίληψη

Η παρούσα διπλωματική εργασία έχει σαν στόχο την υλοποίηση μιας εφαρμογής αναπαραγωγής δεδομένων ατμοσφαιρικών μετρήσεων τοπικά, τα οποία προέρχονται από μία απομακρυσμένη διεπαφή προγραμματισμού εφαρμογών(Application Programming Interface - API). Συγκεκριμένα τα δεδομένα από την απομακρυσμένη υπηρεσία ατμοσφαιρικών μετρήσεων αντιγράφονται τοπικά σε βάση δεδομένων και δίνεται η δυνατότητα στον χρήστη να έχει πρόσβαση σε αυτά μέσω τοπικής διεπαφής προγραμματισμού εφαρμογών(API), όπως και να τα επεξεργαστεί μέσω αυτής της διεπαφής. Τέλος η εφαρμογή έχει την δυνατότητα εμφάνισης της λίστας των μετρήσεων στον χρήστη με την μορφή ιστοσελίδας.

Abstract

The present thesis's goal is to implement an application that replicates data from atmospheric measurements locally, that originate from a remote API(Application Programming Interface). In particular, the data from the remote atmospheric measurements API are stored locally in a database giving the user the capability to access them through a local API and to process them through this API. Finally the application supports access to the list of the atmospheric measurements through a web site.

Εισαγωγή

Οι ατμοσφαιρικές μετρήσεις οι οποίες αφορούν καιρικά δεδομένα και ατμοσφαιρικούς ρύπους παρέχονται από το Web API του ιστοτόπου https://www.iqair.com/ . Ο ιστότοπος αυτός υποστηρίζει ατμοσφαιρικά δεδομένα για εκατοντάδες πόλεις ανά τον κόσμο, τα οποία ανανεώνονται σε τακτά χρονικά διαστήματα και μπορούμε να αντλήσουμε τοπικά στην εφαρμογή μας.

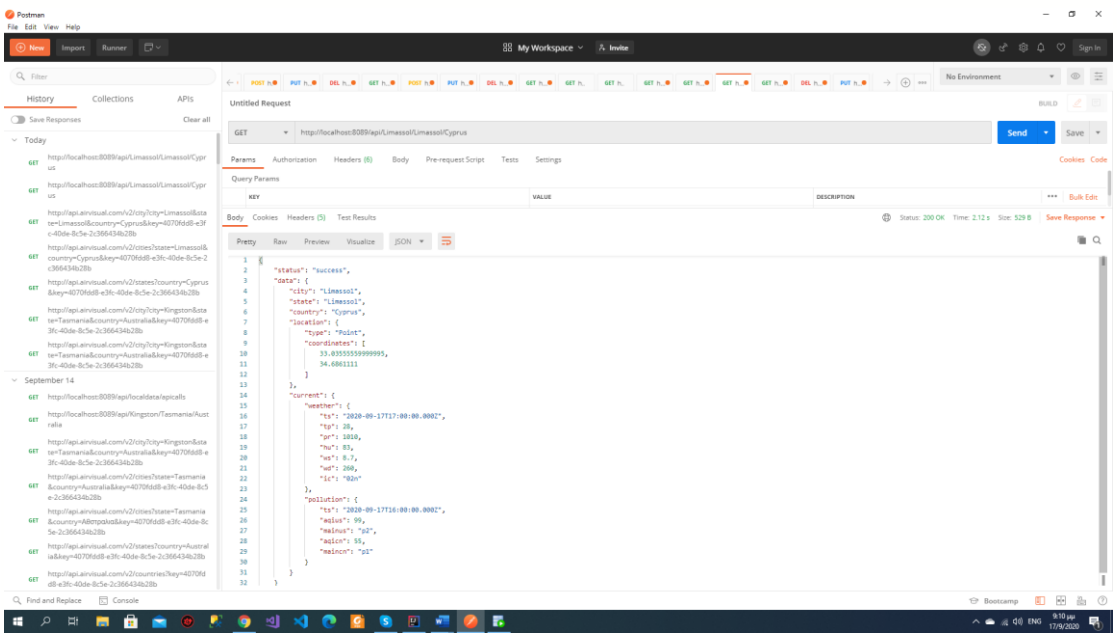
Κατά την εγγραφή στην παραπάνω υπηρεσία παρέχεται ένα μοναδικό κλειδί(key) στους χρήστες με το οποίο μπορούν να πραγματοποιηθούν οι απομακρυσμένες κλήσεις στην υπηρεσία αυτή. Το κλειδί το οποίο μας δόθηκε κατά την εγγραφή είναι το: *4070fdd8-e3fc-40de-8c5e-2c366434b28b* .

Η κλήση έχει την μορφή :

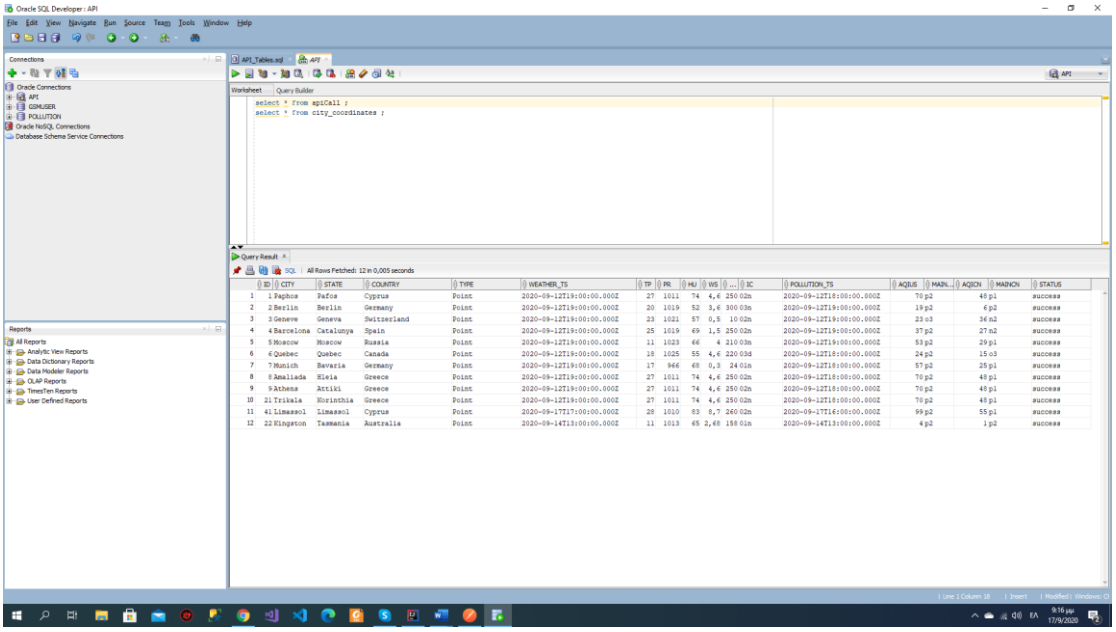
http://api.airvisual.com/v2/city?city=Limassol&state=Limassol&country=Cyprus&key=4070fdd8-e3fc-40de-8c5e-2c366434b28b

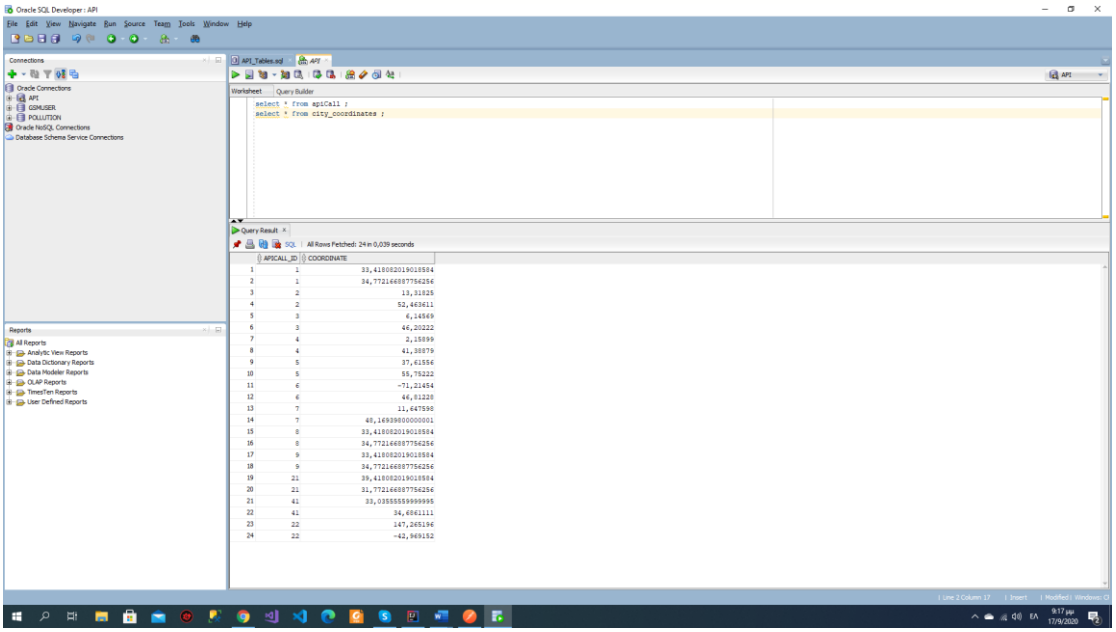
Συμπληρώνουμε πόλη , επαρχία και χώρα στα αντίστοιχα πεδία *city*, *state* & *country* για να αντλήσουμε τις ατμοσφαιρικές μετρήσεις για την συγκεκριμένη πόλη τοπικά στον Η/Υ μας και να τα αποθηκεύσουμε στην βάση δεδομένων .

Με την παρούσα εφαρμογή ο χρήστης καλώντας ένα τοπικό URL(Uniform Resource Locator) στον localhost το οποία περιλαμβάνει χώρα, επαρχία και πόλη καλεί το απομακρυσμένο Web API της https://www.iqair.com/ και αντλεί τις ατμοσφαιρικές μετρήσεις τοπικά όπως φαίνεται στο παρακάτω στιγμιότυπο :



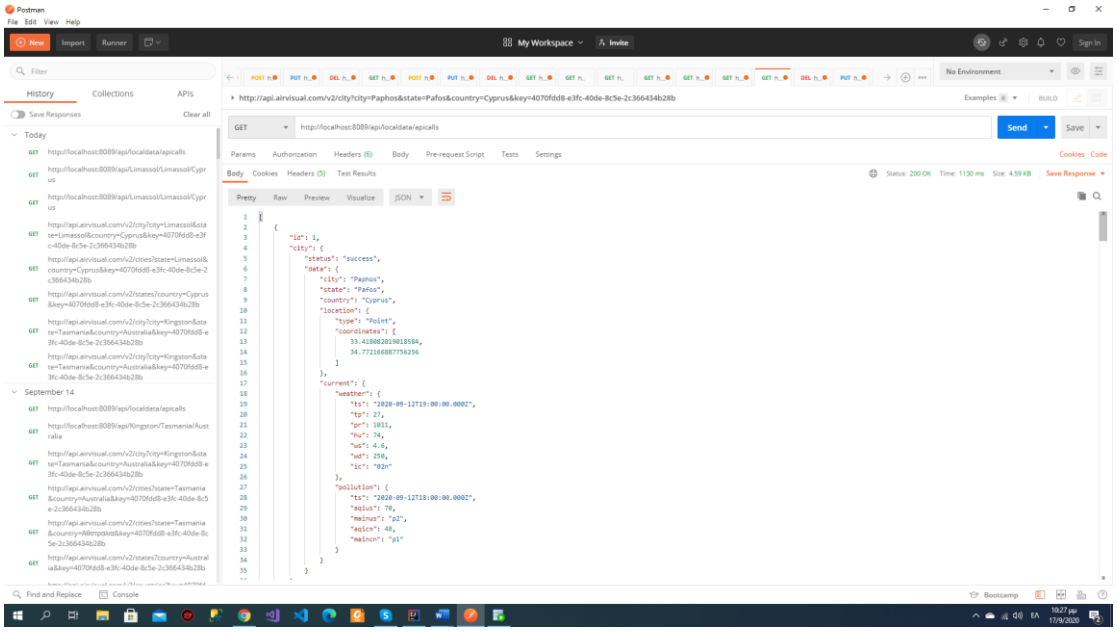
Οι ατμοσφαιρικές μετρήσεις αποθηκεύονται τοπικά σε δύο πίνακες στην σχεσιακή βάση δεδομένων όπως φαίνεται παρακάτω :

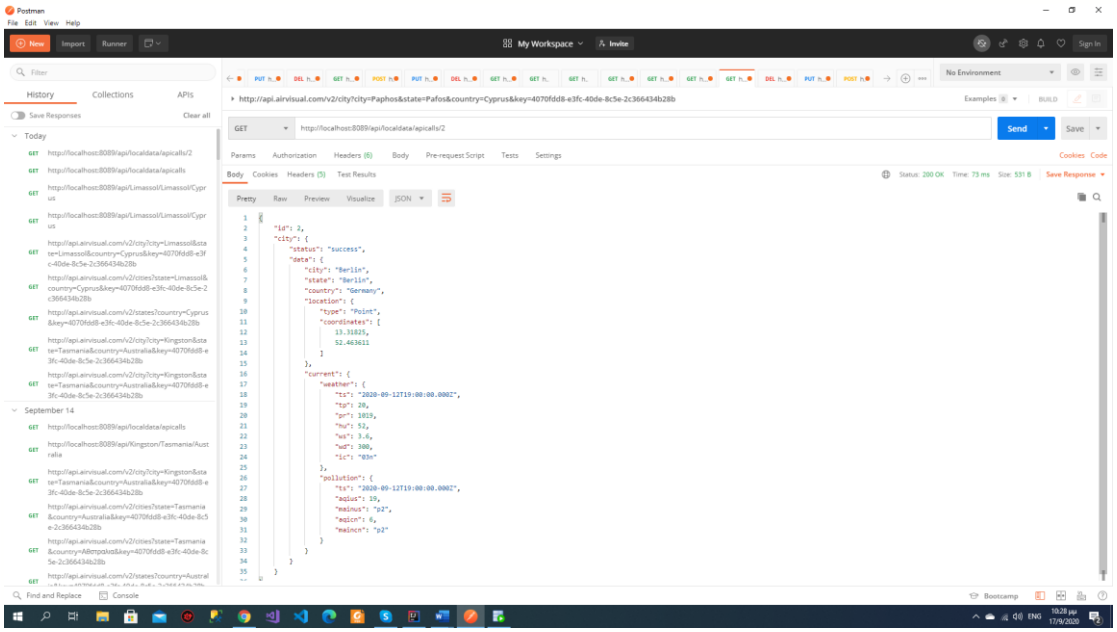




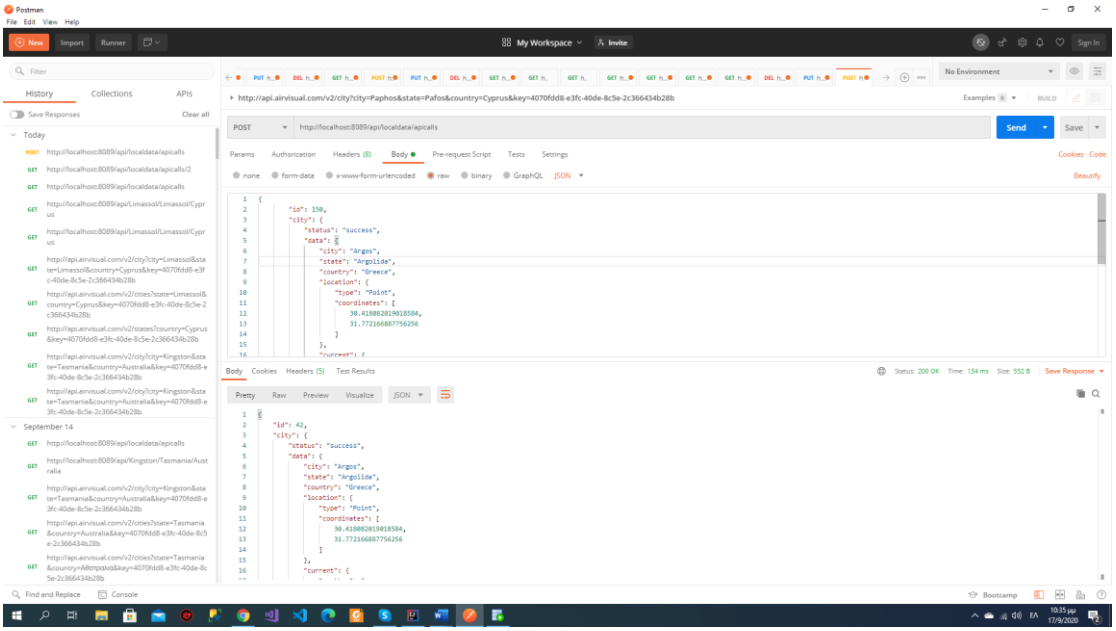
Επιπλέον η εφαρμογή διαθέτει τοπικό Web API το οποίο δίνει την δυνατότητα στον χρήστη να εισάγει, διαβάσει, ενημερώσει και να διαγράψει(Create Read Update Delete - CRUD) τα δεδομένα τα οποία είναι αποθηκευμένα τοπικά στην βάση δεδομένων.

Παρακάτω δίνονται δύο παραδείγματα χρήσης του τοπικού Web API όπου γίνεται αναζήτηση ολόκληρης της λίστας των αποθηκευμένων εγγραφών - ατμοσφαιρικών μετρήσεων και αναζήτηση με βάση τον αύξοντα αριθμό εγγραφής, αντίστοιχα :

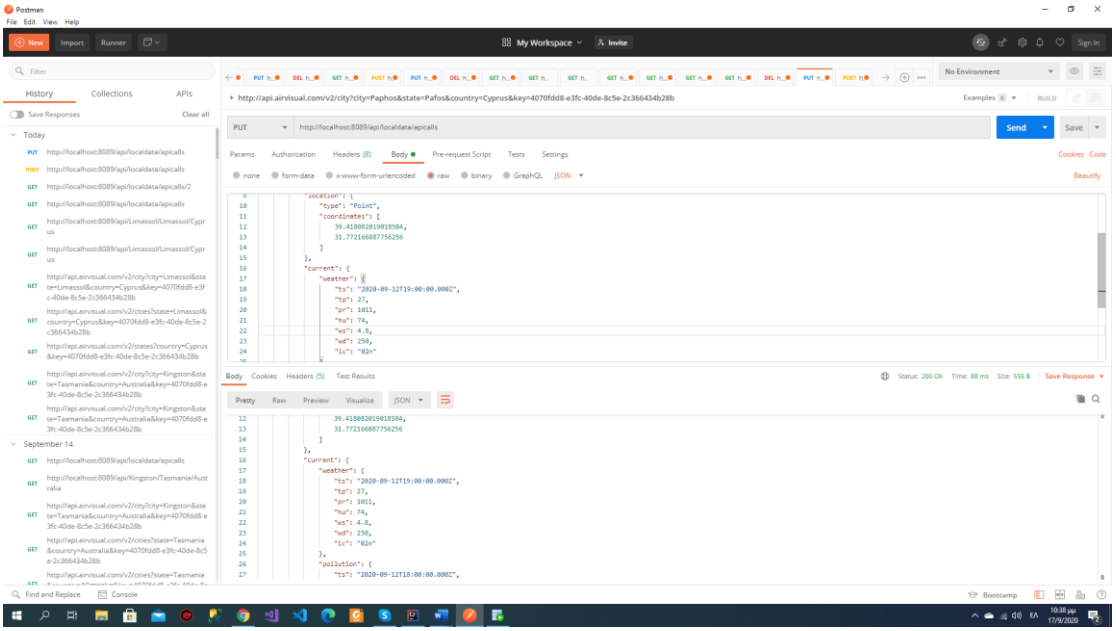




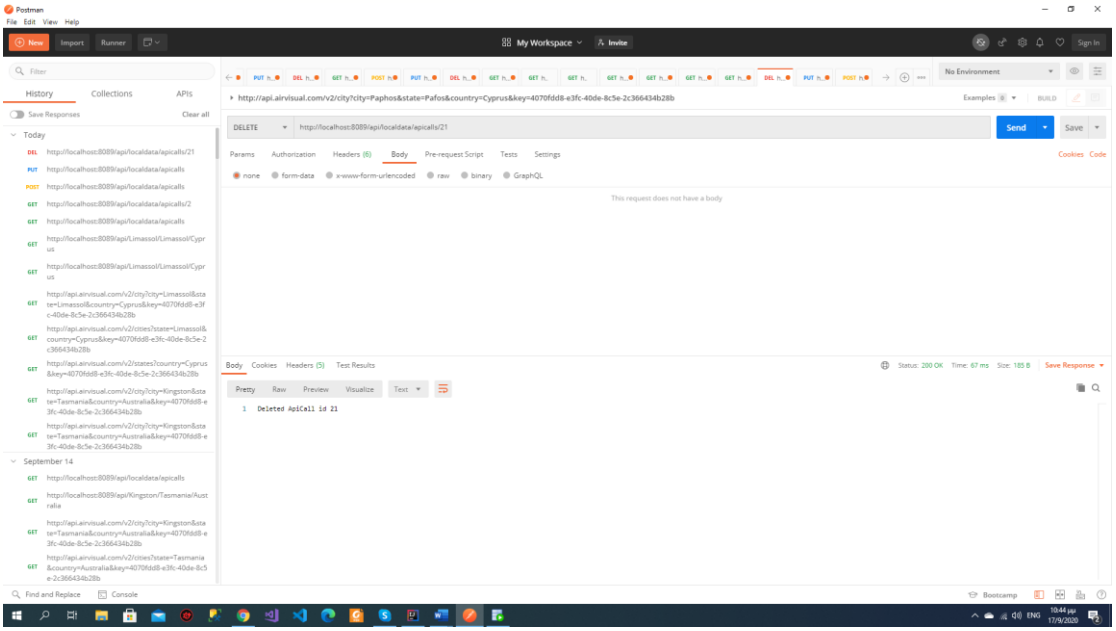
Παράδειγμα δημιουργίας νέας εγγραφής – ατμοσφαιρικής μέτρησης στην τοπική βάση δ. με την βοήθεια του τοπικού Web API :



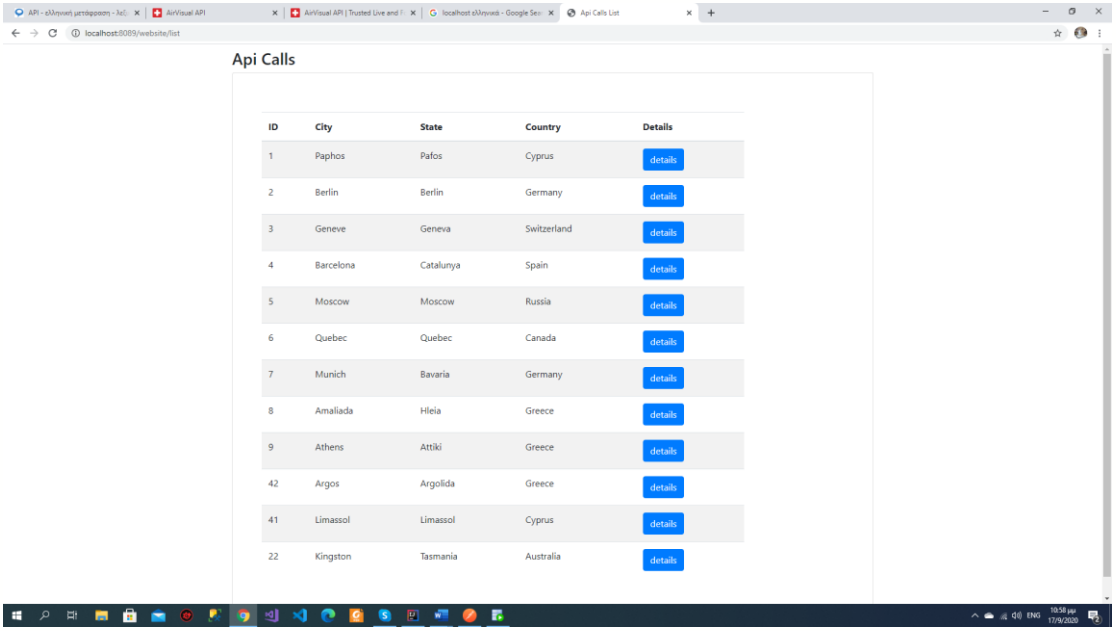
Παράδειγμα ενημέρωσης υπάρχουσας εγγραφής στο τοπικό Web API :



Παράδειγμα διαγραφής εγγραφής – ατμοσφαιρικής μέτρησης τοπικού Web API :



Τέλος, η εφαρμογή παρέχει την δυνατότητα παρουσίασης λίστας των τοπικά αποθηκευμένων μετρήσεων σε γραφικό περιβάλλον(Graphic User Interface - GUI) μέσω ιστοσελίδας :



Χρήση Εφαρμογής

Η εφαρμογή μας χρησιμοποιεί την πόρτα *8089* του *localhost* για τις κλήσεις τις οποίες πραγματοποιούν οι χρήστες.

Α) Κλήση απομακρυσμένου Web API της www.iqair.com μέσω της εφαρμογής μας

Η κλήση GET είναι της μορφής : *http://localhost:8089/api/{city}/{state}/{country}*

Όπου ο χρήστης συμπληρώνει μία από τις υποστηριζόμενες από το απομακρυσμένο Web API της iqair, πόλη({city}), επαρχία({state}) και χώρα({country}).

Εάν η κλήση αυτή είναι επιτυχής, τα δεδομένα των ατμοσφαιρικών μετρήσεων αποθηκεύονται τοπικά σε βάση δεδομένων της Oracle. Συγκεκριμένα στους πίνακες APICALL και CITY_COORDINATES. Η HTTP αίτηση η οποία χρησιμοποιείται στις κλήσεις είναι η GET.

Ο χρήστης έχει την δυνατότητα να εμφανίσει λίστα με τις υποστηριζόμενες πόλεις, επαρχίες και χώρες για τις οποίες υπάρχουν διαθέσιμες ατμοσφαιρικές μετρήσεις, καλώντας απευθείας το Web API της iqair. Παρακάτω παρατίθενται οι απομακρυσμένες κλήσεις :

1) Απομακρυσμένη κλήση GET για εμφάνιση λίστας διαθέσιμων χωρών :
   http://api.airvisual.com/v2/countries?key=4070fdd8-e3fc-40de-8c5e-2c366434b28b

2) Απομακρυσμένη κλήση GET για εμφάνιση λίστας διαθέσιμων επαρχιών για συγκεκριμένη χώρα. Η χώρα ορίζεται μετά την παράμετρο *country=* και σε αυτήν την περίπτωση είναι η Κύπρος(Cyprus) :
   http://api.airvisual.com/v2/states?country=Cyprus&key=4070fdd8-e3fc-40de-8c5e-2c366434b28b

3) Απομακρυσμένη κλήση GET για εμφάνιση λίστας διαθέσιμων πόλεων για συγκεκριμένη χώρα και επαρχία. Η χώρα ορίζεται μετά την παράμετρο *country=* και η επαρχία μετά την παράμετρο *state=* και είναι η Κύπρος(Cyprus) και η Λεμεσσός(Limassol) αντίστοιχα :
   http://api.airvisual.com/v2/cities?state=Limassol&country=Cyprus&key=4070fdd8-e3fc-40de-8c5e-2c366434b28b

Β) Κλήση τοπικού Web API της εφαρμογής μας

Η κλήση έχει την μορφή : *http://localhost:8089/api/localdata/apicalls* και *http://localhost:8089/api/localdata/apicalls/{id}* .

Ο χρήστης έχει την δυνατότητα CRUD(Create Read Update Delete) στα δεδομένα ατμοσφαιρικών μετρήσεων τα οποία είναι αποθηκευμένα τοπικά στην βάση δεδομένων μας.

Για την πραγματοποίηση των HTTP αιτήσεων GET, POST, PUT & DELETE με τις οποίες επιτυγχάνεται το CRUD στο Web API, χρησιμοποιήθηκε η εφαρμογή POSTMAN.

Συγκεκριμένα οι διαθέσιμες κλήσεις είναι οι εξής :

1) Τοπική κλήση GET για εμφάνιση όλων των αποθηκευμένων μετρήσεων : *http://localhost:8089/api/localdata/apicalls*

2) Τοπική κλήση GET για αναζήτηση αποθηκευμένης μέτρησης με βάση το id : *http://localhost:8089/api/localdata/apicalls/{id}*

3) Τοπική κλήση DELETE για διαγραφή αποθηκευμένης μέτρησης με βάση το id: *http://localhost:8089/api/localdata/apicalls/{id}*

4) Τοπική κλήση PUT για ενημέρωση αποθηκευμένης μέτρησης : *http://localhost:8089/api/localdata/apicalls*

   *Η κλήση πρέπει να περιέχει στο σώμα(Body) της, την εγγραφή προς ενημέρωση σε μορφή JSON.

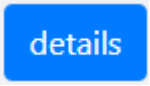
5) Τοπική κλήση POST για δημιουργία καινούριας εγγραφής ατμ. μέτρησης στην τοπική βάση δεδομένων : *http://localhost:8089/api/localdata/apicalls*

   *Η κλήση πρέπει να περιέχει στο σώμα(Body) της, την εγγραφή προς ενημέρωση σε μορφή JSON. Άσχετα με το id του σώματος της κλήσης, η καινούρια εγγραφή η οποία θα δημιουργηθεί θα λάβει σαν id αύξοντα αριθμό, με βάση τις ήδη αποθηκευμένες εγγραφές.

Γ) Κλήση τοπικής ιστοσελίδας της εφαρμογής μας

Η κλήση χρησιμοποιεί την HTTP αίτηση GET και έχει την μορφή :

http://localhost:8089/website/list

Σε οποιοδήποτε φυλλομετρητή(browser) πληκτρολογήσουμε τον παραπάνω σύνδεσμο, θα εμφανίσουμε όλες τις αποθηκευμένες εγγραφές της βάσης δεδομένων μας, οι οποίες αφορούν ατμοσφαιρικές μετρήσεις για διάφορες πόλεις του κόσμου. Οι εγγραφές αυτές έχουν την μορφή πίνακα και για κάθε εγγραφή εμφανίζεται το id, η πόλη, η επαρχία και η χώρα. Επίσης στο τέλος κάθε εγγραφής υπάρχει το κουμπί 

, πατώντας το οποίο εμφανίζονται οι λεπτομέρειες της συγκεκριμένης εγγραφής οι οποίες αφορούν την ατμοσφαιρική μέτρηση. Οι λεπτομέρειες αυτές εμφανίζονται σε νέα σελίδα με URL : *http://localhost:8089/website/details/{id}* με την μορφή HTML φόρμας, όπου {id} είναι το id της εγγραφής-μέτρησης για την οποία πατήσαμε το κουμπί details.

Αρχιτεκτονική Συστήματος

Για την δημιουργία της εφαρμογής χρησιμοποιήθηκαν οι παρακάτω τεχνολογίες :

Γλώσσα προγραμματισμού : Java Version 11

Περιβάλλον ανάπτυξης λογισμικού : IntelliJ IDEA Community edition 2020.2

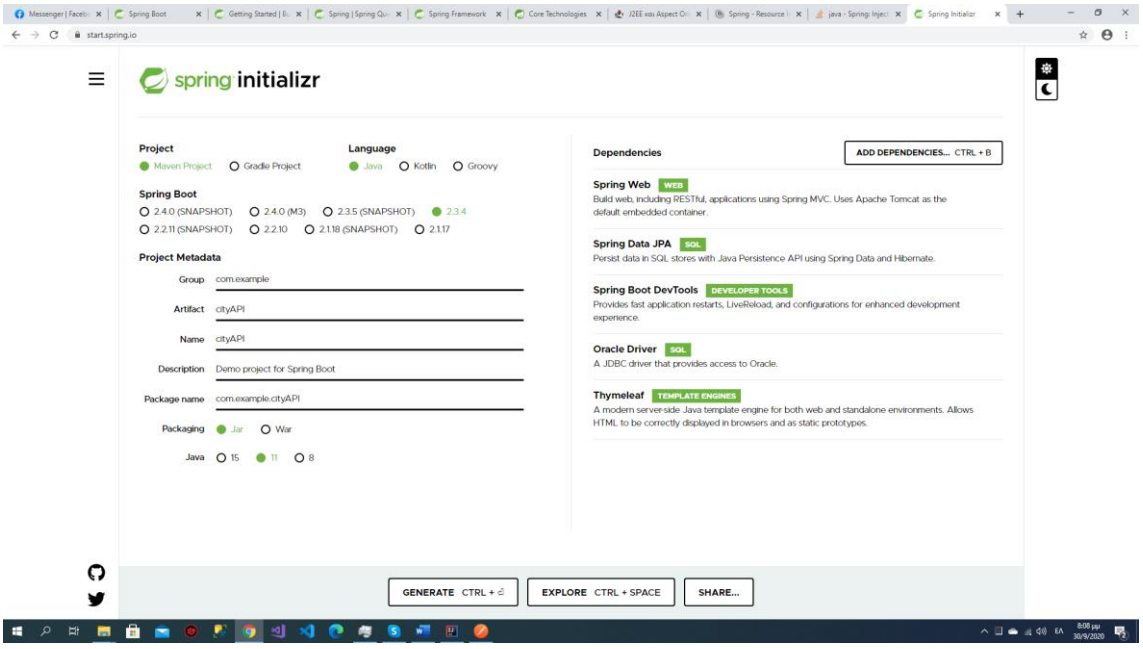
Java Framework : Spring Boot 2.3.4

Εργαλεία :

- *Jackson* project για την επεξεργασία και μετατροπή JSON αρχείων σε αντικείμενα και αντίστροφα,
- *Hibernate* ORM(Object Relational Mapping) για την επικοινωνία μεταξύ σχεσιακής βάσης δεδομένων και αντικειμένων τα οποία δημιουργούνται από κλάσεις,
- *Maven* build automation tool για τον έλεγχο των dependencies της εφαρμογής,
- *Thymeleaf* template engine για την δημιουργία HTML views.

Βάση Δεδομένων : Oracle Express Edition 18.4, Oracle SQL Developer 19.2 IDE(Integrated Development Environment)

Για την κατασκευή του αρχικού project της εφαρμογής σε Spring Boot χρησιμοποιήθηκε το Spring Initializr(https://start.spring.io/) και προστέθηκαν τα dependencies Spring Web, Spring Data JPA, Spring Boot DevTools, Oracle Driver, Thymeleaf όπως φαίνεται παρακάτω :



Αρχικά γίνεται παρουσίαση των τεχνολογιών οι οποίες χρησιμοποιήθηκαν, όπως frameworks και εργαλεία, και στην συνέχεια αναλύεται η αρχιτεκτονική της εφαρμογής.

Spring Framework

Το Spring Framework είναι μία Java πλατφόρμα η οποία παρέχει πλήρη υποστήριξη υποδομής για την ανάπτυξη Java εφαρμογών. Το Spring χειρίζεται την υποδομή έτσι ώστε οι προγραμματιστές να εστιάσουν στις εφαρμογές τους. Το Spring δίνει την δυνατότητα δημιουργίας εφαρμογών από POJOs(Plain Old Java Objects) όπως επίσης και να εφαρμοστούν επιχειρηματικές(enterprise) υπηρεσίες μη-επεμβατικά σε POJOs. Αυτή η δυνατότητα ισχύει στο Java Standard Edition προγραμματιστικό μοντέλο και πλήρως και μερικώς σε αυτό του Java Enterprise Edition.

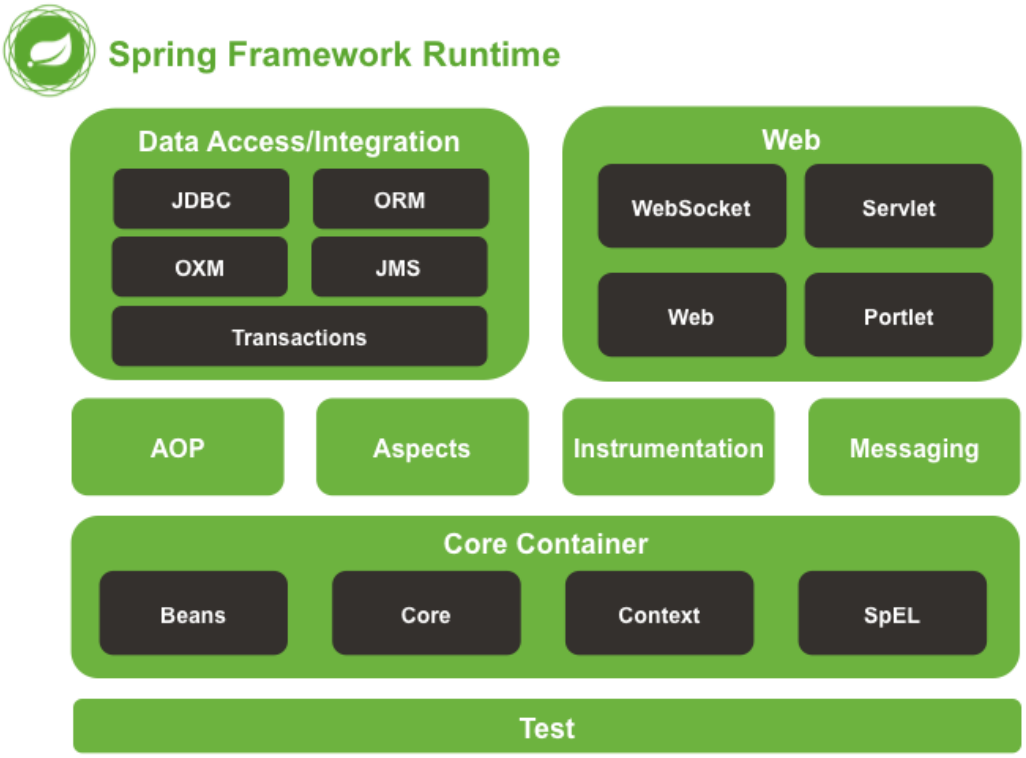
Μερικά παραδείγματα του πως ένας προγραμματιστής εφαρμογών μπορεί να επωφεληθεί από την Spring πλατφόρμα είναι τα εξής :

- Να εκτελέσει μια Java μέθοδο σε ένα database transaction χωρίς να χρειαστεί να χρησιμοποιήσει κάποιο transaction API
- Να μετατρέψει μια τοπική Java μέθοδο σε HTTP endpoint χωρίς να χρειαστεί να χρησιμοποιήσει το Servlet API
- Να μετατρέψει μια τοπική Java μέθοδο σε διαχειριστή μηνυμάτων(message handler) χωρίς να χρειαστεί να χρησιμοποιήσει το JMS API
- Να μετατρέψει μια τοπική Java μέθοδο σε εργασία διαχείρισης(management operation) χωρίς να χρειαστεί να χρησιμοποιήσει το JMX API

Μία Java εφαρμογή τυπικά αποτελείται από αντικείμενα τα οποία συνεργάζονται μεταξύ τους για την άρτια διαμόρφωσή της. Έτσι τα αντικείμενα σε μια εφαρμογή έχουν εξαρτήσεις(dependencies) μεταξύ τους. Παρόλο που η Java πλατφόρμα παρέχει πλούτο λειτουργιών για την ανάπτυξη εφαρμογών, στερείται τα μέσα για την οργάνωση των βασικών δομικών στοιχείων σε ένα ολοκληρωμένο σύνολο, αφήνοντας αυτή την εργασία σε αρχιτέκτονες λογισμικού(architects) και προγραμματιστές. Παρόλο που μπορούν να χρησιμοποιηθούν σχεδιαστικά πρότυπα(design patterns) όπως Factory, Abstract Factory,Builder, Decorator και Service Locator για να συνθέσουν τις διάφορες κλάσεις και αντικείμενα τα οποία αποτελούν μία εφαρμογή, αυτά τα design patterns είναι απλά αυτό : δοκιμασμένες πρακτικές(best practices) στις οποίες έχει δοθεί ένα όνομα, με περιγραφή του τι μπορεί να κάνει το pattern, που μπορεί να εφαρμοστεί, τα προβλήματα στα οποία απευθύνεται και ούτω καθεξής. Τα design patterns είναι επίσημα best practices τα οποία πρέπει να υλοποιήσει ο ίδιος ο προγραμματιστής μέσα σε μία εφαρμογή.

Το Spring Framework Inversion of Control(IoC)-Dependency Injection(DI) συστατικό επιλύει αυτό το ζήτημα παρέχοντας έναν επίσημο τρόπο για την σύνθεση άσχετων μεταξύ τους μερών σε μία λειτουργική εφαρμογή έτοιμη προς χρήση. Το Spring Framework κωδικοποιεί επίσημα design patterns σαν αντικείμενα τα οποία μπορεί κανείς να ενσωματώσει στις εφαρμογές του. Πολυάριθμοι οργανισμοί και ιδρύματα χρησιμοποιούν το Spring Framework κατά αυτόν τον τρόπο για την κατασκευή εύρωστων(robust) και εύκολα συντηρήσιμων εφαρμογών.

Το Spring Framework αποτελείται από χαρακτηριστικά τα οποία είναι οργανωμένα σε περίπου 20 μέρη(modules). Αυτά τα modules είναι οργανωμένα στις υποομάδες Core Container, Data Access/Integration, Web, AOP(Aspect Oriented Programming), Instrumentation, Messaging και Test όπως φαίνεται στο ακόλουθο διάγραμμα :



Core Container

Αποτελείται από τα modules spring-core, spring-beans, spring-context, spring-context-support και spring-expression.

Τα spring-core και spring-beans modules παρέχουν τα βασικά μέρη του framework, συμπεριλαμβανομένων των Dependency Injection χαρακτηριστικών. Το BeanFactory είναι μια εξελιγμένη υλοποίηση του Factory design pattern. Αφαιρεί την ανάγκη για προγραμματιστικά singletons(μέχρι ένα αντικείμενο μπορεί να κατασκευαστεί για κάποια κλάση) και επιτρέπει την αποσύνδεση(decoupling) της διαμόρφωσης και των χαρακτηριστικών των dependencies από την πραγματική προγραμματιστική λογική.

Το spring-context module χτίζεται πάνω στην βάση την οποία παρέχουν τα Core και Bean modules. Αποτελεί ένα μέσο πρόσβασης σε αντικείμενα με έναν τρόπο παραπλήσιο με αυτόν ενός framework. Το Context module κληρονομεί τα χαρακτηριστικά του από το Beans module και προσθέτει υποστήριξη για διεθνοποίηση, αναπαραγωγή γεγονότων(event propagation), φόρτωση πόρων(resource loading) και την διάφανη δημιουργία(transparent creation) των contexts από π.χ. ένα Servlet container. Το Context module επίσης υποστηρίζει Java Enterprise Edition(JEE) χαρακτηριστικά όπως Enterprise Java Beans(EJB), Java Management Extensions(JMX) και basic remoting. Το ApplicationContext interface αποτελεί το κεντρικό σημείο του Context module.

Το spring-context-support module παρέχει υποστήριξη για ενσωμάτωση κοινών βιβλιοθηκών τρίτων, στο Spring application context, για λειτουργίες caching(EhCache, Guava, JCache), λειτουργίες email(JavaMail), λειτουργίες χρονοδιαγράμματος(CommonJ, Quartz) και template engines(FreeMarker, JasperReports, Velocity).

Το spring-expression module παρέχει μία ισχυρή γλώσσα(Expression Language) για την διενέργεια ερωτημάτων(querying) και την διαχείριση ενός γράφου αντικειμένου(object graph) κατά την εκτέλεση(runtime). Είναι μια επέκταση της γλώσσας unified EL όπως προσδιορίζεται στην προδιαγραφή JSP 2.1. Η γλώσσα αυτή υποστηρίζει την

ανάθεση(set) και επιστροφή(get) τιμών ιδιοτήτων(properties), ανάθεση(assignment) ιδιοτήτων, κλήση μεθόδων, πρόσβαση στα περιεχόμενα πινάκων, συλλογών(collections) και ευρετηρίων(indexers), λογικών και αριθμητικών τελεστών, τις έγκυρες μεταβλητές, και την ανεύρεση αντικειμένων μέσω του ονόματός τους από το Spring Container. Επίσης υποστηρίζει προβολή λίστας και επιλογή λίστας, όπως επίσης και λειτουργίες συνόλων(aggregations) σε λίστα.

AOP and Instrumentation

Το spring-aop module παρέχει μία AOP Alliance-συμβατή υλοποίηση πτυχοστρεφούς /θεματοστρεφούς προγραμματισμού επιτρέποντας έτσι τον ορισμό, για παράδειγμα, method interceptors και pointcuts για να αποσυνδεθεί ο κώδικας ο οποίος υλοποιεί κάποια λειτουργικότητα η οποία πρέπει να χωριστεί. Χρησιμοποιώντας λειτουργικότητα μεταδεδομένων(metadata) επιπέδου πηγαίου κώδικα(source), μπορούμε να ενσωματώσουμε συμπεριφοριστική(behavioral) πληροφορία μέσα στον κώδικα, με έναν παρόμοιο τρόπο με αυτόν των .NET attributes.

Το ξεχωριστό spring-aspects module ενσωματώνει την τεχνολογία AspectJ.

Το spring-instrument module παρέχει υποστήριξη για ενορχήστρωση (instrumentation) κλάσεων και υλοποιήσεις φόρτωσης κλάσεων(classloader) για χρήση σε συγκεκριμένους διακομιστές εφαρμογών(application servers). Το spring-instrument-tomcat module περιέχει τον Spring instrumentation παράγοντα(agent) για τον Tomcat Web Server.

Messaging

Το Spring Framework περιλαμβάνει ένα spring-messaging module με σημαντικές αφηρημένες έννοιες(abstractions) οι οποίες προέρχονται από το Spring Integration project, όπως Message, MessageChannel, MesageHandler και άλλες που αποτελούν θεμέλιο στις μηνυματοκεντρικές(messaging-based) εφαρμογές. Το module επίσης περιλαμβάνει μια ομάδα από σχόλια(annotations) για την αντιστοίχιση(mapping) μηνυμάτων με μεθόδους, παρόμοια με το Spring MVC προγραμματιστικό μοντέλο, το οποίο επίσης βασίζεται σε annotations.

Data Access/Integration

Το Data Access/Integration επίπεδο αποτελείται από τα JDBC, ORM, OXM, JMS και Transaction modules.

Το spring-jdbc module παρέχει ένα JDBC επίπεδο αφηρημένης έννοιας(abstraction) το οποίο αφαιρεί την ανάγκη συγγραφής κουραστικού JDBC κώδικα όπως και της ανάλυσης των ειδικών κωδικών σφάλματος ανάλογα με την διαφορετική τεχνολογία βάσης δεδομένων που χρησιμοποιείται.

Το spring-tx module υποστηρίζει προγραμματιστική και δηλωτική διαχείριση συναλλαγών(transactions) για κλάσεις οι οποίες υλοποιούν ειδικά interfaces και για όλα τα POJOs.

Το spring-orm module παρέχει επίπεδα ενσωμάτωσης(integration) για δημοφιλή Object-Relational-Mapping APIs, συμπεριλαμβανομένων των JPA, JDO και Hibernate. Χρησιμοποιώντας το spring-orm module μπορούμε να χρησιμοποιήσουμε όλα αυτά τα ORM frameworks σε συνδυασμό με όλα τα άλλα χαρακτηριστικά που προσφέρει το Spring, όπως το χαρακτηριστικό της δηλωτικής διαχείρισης transaction που αναφέρθηκε παραπάνω.

Το spring-oxm module παρέχει ένα abstraction επίπεδο το οποίο υποστηρίζει υλοποιήσεις mapping αντικειμένων(Object) με XML αρχεία όπως, JAXB, Castor, XMLBeans, JiBX και XStream.

Το spring-jms module (Java Messaging Service) περιέχει χαρακτηριστικά για παραγωγή(producing) και κατανάλωση(consuming) μηνυμάτων. Από το Spring Framework 4.1 παρέχει integration με το spring-messaging module.

Web

Το Web επίπεδο αποτελείται από τα spring-web, spring-webmvc, spring-websocket και spring-webmvc-portlet modules.

Το spring-web module παρέχει βασικά χαρακτηριστικά διαδικτυακού(web) integration όπως λειτουργία ανεβάσματος(upload) πολυμερούς(multipart) αρχείου και την αρχικοποίηση του Spring container χρησιμοποιώντας Servlet listeners και ενός web application context. Επίσης περιέχει έναν HTTP client και τα διαδικτυακά μέρη της Spring απομακρυσμένης υποστήριξης(remoting support).

Το spring-webmvc module (επίσης γνωστό και ως Web-Servlet module) περιέχει το model-view-controller(MVC) του Spring και την υλοποίηση των REST Web υπηρεσιών(services) για τις web εφαρμογές. Το Spring MVC framework παρέχει έναν καθαρό διαχωρισμό ανάμεσα στον domain model κώδικα και τις web forms και ενσωματώνεται με όλα τα άλλα χαρακτηριστικά του Spring Framework.

Το spring-webmvc-portlet module (επίσης γνωστό και ως Web-Portlet module) παρέχει την MVC υλοποίηση η οποία θα χρησιμοποιηθεί σε ένα Portlet περιβάλλον και αντιγράφει την λειτουργικότητα του βασιζόμενου σε Servlet, spring-webmvc module.

Test

Το spring-test module υποστηρίζει την εξέταση μονάδας(unit testing) και την εξέταση ενσωμάτωσης(integration testing) των συστατικών μερών του Spring χρησιμοποιώντας JUnit ή TestNG frameworks. Παρέχει συνεπή φόρτωση των Spring ApplicationContexts και προσωρινή αποθήκευση(caching) αυτών των contexts. Επίσης παρέχει ψεύτικα αντικείμενα(mock objects) τα οποία μπορούν να χρησιμοποιηθούν για την εξέταση του κώδικα σε περιβάλλον απομόνωσης.

Τέλος ακολουθούν περιληπτικά οι στόχοι του Spring Framework :

- Απλή ανάπτυξη χρησιμοποιώντας Java POJOs(Plain Old Java Objects)
- Dependency Injection για την επίτευξη “χαλαρής διασύνδεσης”(loose coupling) μεταξύ των τμημάτων μιας εφαρμογής
- Χρήση δηλωτικού προγραμματισμού(declarative programming) με την βοήθεια του πτυχοστρεφούς/θεματοστρεφούς προγραμματισμού(Aspect Oriented Programming-AOP)
- Ελαχιστοποίηση του επαναλαμβανόμενου Java κώδικα(boilerplate code)

Spring Boot Framework

Το Spring Boot αποτελεί μια επέκταση του Spring framework η οποία εξαλείφει τις επαναλαμβανόμενες και πολύπλοκες ρυθμίσεις οι οποίες απαιτούνται για την δημιουργία μιας Spring εφαρμογής. Ουσιαστικά αποτελεί μια δογματική(opinionated) άποψη της Spring πλατφόρμας η οποία άνοιξε τον δρόμο για ένα ταχύτερο και πιο αποτελεσματικό σύστημα ανάπτυξης λογισμικού. Μερικά από τα χαρακτηριστικά του Spring Boot είναι :

- Αρχικά dependencies για την απλοποίηση της ανάπτυξης και παραμετροποίησης της εφαρμογής.
- Ενσωματωμένος διακομιστής(server) για την αποφυγή πολυπλοκότητας κατά την εγκατάσταση της εφαρμογής
- Δυνατότητα παροχής μετρήσεων(metrics), ελέγχου κατάστασης και εξωτερικής παραμετροποίησης της εφαρμογής
- Παροχή αυτόματης παραμετροποίησης για κάποια λειτουργικότητα του Spring όπου είναι αυτό δυνατό
- Καμία απαίτηση για παραμετροποίηση μέσω XML αρχείων

Στην παρούσα εφαρμογή γίνεται εκτεταμένη χρήση της τεχνικής του Dependency Injection και των annotations τα οποία υποστηρίζει το Spring Boot. Τα παραπάνω χρησιμοποιούνται και στα τρία επίπεδα της εφαρμογής, Controller Layer, Service Layer και Data Access Layer. Παρακάτω γίνεται παρουσίαση της τεχνικής Dependency Injection και των annotations τα οποία χρησιμοποιούνται στην εφαρμογή :

Dependency Injection

Ένα *Dependency Injection Framework* όπως είναι το Spring Boot, δίνει την δυνατότητα στον προγραμματιστή να “περάσει” αντικείμενα μέσα στον κώδικά του αντί να τα δημιουργήσει ο ίδιος. Με αυτόν τον τρόπο μειώνεται η εξάρτηση κλάσεων από άλλες κλάσεις, δηλαδή επιτυγχάνεται η χαλαρή σχέση μεταξύ κλάσεων. Κάτι το οποίο εκφράζεται με τον όρο decoupling.

Στο Spring Boot Framework το *Spring Container* είναι υπεύθυνο για την δημιουργία, ρύθμιση και διαχείριση του κύκλου ζωής των αντικειμένων, τα οποία αποκαλούνται *Beans*. Με την βοήθεια του *Spring Container* δηλαδή επιτυγχάνεται η τεχνική του *Dependency Injection* στα προγράμματα τα οποία κάνουν χρήση του *Spring Boot*.

Στην εφαρμογή μας γίνεται χρήση της τεχνικής του *Dependency Injection* με την μορφή *Constructor Injection*, δηλαδή στον κατασκευαστή της κλάσης.

Συγκεκριμένα Dependency Injection χρησιμοποιείται στις κλάσεις ApiCallRestController :

```
ApiCallService apiCallService;
ObjectMapper mapper;

@Autowired
public ApiCallRestController(ApiCallService apiCallService, ObjectMapper mapper){
   this.apiCallService = apiCallService;
   this.mapper = mapper;
}
```

WebsiteController :

```
ApiCallService apiCallService;

@Autowired
```

```
public WebsiteController(ApiCallService apiCallService) {
    this.apiCallService = apiCallService;
}
```

ApiCallServiceImpl :

```
ApiCallDAO apiCallDAO;

@Autowired
public ApiCallServiceImpl(ApiCallDAO apiCallDAO){
    this.apiCallDAO = apiCallDAO;
}
```

ApiCallDAOImpl :

```
EntityManager entityManager;

@Autowired
public ApiCallDAOImpl(EntityManager entityManager){
    this.entityManager = entityManager;
}
```

Σε όλους τους παραπάνω κατασκευαστές όπως φαίνεται δεν χρησιμοποιείται το keyword *new* για την κατασκευή νέων αντικειμένων(objects) αλλά το annotation *@Autowired.* Με το annotation αυτό δίνουμε εντολή στο *Spring Boot* framework για να πραγματοποιηθεί *dependency injection*. Συγκεκριμένα στον κατασκευαστή της κλάσης WebsiteController το Spring Boot αναζητά ποια κλάση υλοποιεί(implements) το *Interface* ApiCallService, δημιουργεί ένα αντικείμενο από αυτήν την κλάση(ApiCallServiceImpl) και το περνάει σαν την παράμετρο apiCallService. Στον κατασκευαστή της κλάσης ApiCallServiceImpl το Spring Boot αναζητά ποια κλάση υλοποιεί το *Interface* ApiCallDAO, δημιουργεί ένα αντικείμενο από αυτήν την κλάση(ApiCallDAOImpl) και το περνάει σαν την παράμετρο apiCallDAO. Η ίδια μέθοδος ισχύει και για τις υπόλοιπες κλάσεις που αναφέρθηκαν.

Παρακάτω γίνεται μια παρουσίαση των *Spring Boot* annotations τα οποία χρησιμοποιήθηκαν στην εφαρμογή :

Spring Boot Annotations

@Controller : Χρησιμοποιείται στην αρχιτεκτονική MVC(Model View Controller) για κλάσεις οι οποίες έχουν τον ρόλο του Controller σε μια εφαρμογή, δηλαδή διαθέτουν μεθόδους οι οποίες επιστρέφουν HTML Views. Στην εφαρμογή μας αυτά τα Views είναι τα index.html και details.html. Επίσης με αυτό το annotation η κλάση δηλώνεται στο *Spring Container* και έτσι μπορούν να δημιουργηθούν Beans εάν αυτό ζητηθεί σε κάποιο άλλο μέρος της εφαρμογής.

@RequestMapping : Χρησιμοποιείται τόσο σε επίπεδο κλάσης όσο και σε επίπεδο μεθόδου και πραγματοποιεί την αντιστοίχιση(mapping) Web Requests και μεθόδων ή κλάσεων οι οποίες χειρίζονται τα αιτήματα αυτά. Δημιουργεί ένα URL για το οποίο ο συγκεκριμένος Controller θα χρησιμοποιηθεί.

@Autowired : Χρησιμοποιείται για να πραγματοποιήσει *Dependency Injection* κάποιου αντικειμένου(object) σε επίπεδο class Property(μεταβλητής μέλους κλάσης), Setter

method(μεθόδου η οποία χρησιμοποιείται για να αλλάξει την τιμή κάποιας μεταβλητής μέλους κλάσης) και Constructor(κατασκευαστή).

@RestController : Χρησιμοποιείται σε επίπεδο κλάσης. Δίνει στην κλάση τον ρόλο του Controller αλλά οι μέθοδοι τις οποίες διαθέτει η κλάση αυτή επιστρέφουν αντικείμενα και όχι HTML Views. Χρησιμοποιείται η αρχιτεκτονική REST(Representational State Transfer), δηλαδή η επικοινωνία μεταξύ μηχανημάτων(client, server) γίνεται μέσω πρωτοκόλλου HTTP. Έτσι η επικοινωνία με οποιοδήποτε API επιτυγχάνεται χωρίς να λαμβάνεται υπόψη η αρχιτεκτονική του κάθε συστήματος το οποίο συμμετέχει σε αυτήν.

@GetMapping : Χρησιμοποιείται σε επίπεδο μεθόδου για την αντιστοίχιση(mapping) HTTP GET Requests και μεθόδων οι οποίες χειρίζονται αυτά τα αιτήματα. Αυτό το annotation αποτελεί την συντομογραφία του *@RequestMapping(method = RequestMethod.GET) .*

@DeleteMapping : Χρησιμοποιείται σε επίπεδο μεθόδου για την αντιστοίχιση(mapping) HTTP DELETE Requests και μεθόδων οι οποίες χειρίζονται αυτά τα αιτήματα. Αυτό το annotation αποτελεί την συντομογραφία του *@RequestMapping(method = RequestMethod.DELETE) .*

@PutMapping : Χρησιμοποιείται σε επίπεδο μεθόδου για την αντιστοίχιση(mapping) HTTP PUT Requests και μεθόδων οι οποίες χειρίζονται αυτά τα αιτήματα. Αυτό το annotation αποτελεί την συντομογραφία του *@RequestMapping(method = RequestMethod.PUT) .*

@PostMapping : Χρησιμοποιείται σε επίπεδο μεθόδου για την αντιστοίχιση(mapping) HTTP POST Requests και μεθόδων οι οποίες χειρίζονται αυτά τα αιτήματα. Αυτό το annotation αποτελεί την συντομογραφία του *@RequestMapping(method = RequestMethod.POST) .*

@Service : Χρησιμοποιείται σε επίπεδο κλάσης. Χαρακτηρίζει την κλάση σαν Service το οποίο εκτελεί κάποια business logic, κάποιους υπολογισμούς ή να καλέσει μεθόδους API επιπέδου. Επίσης το *@Service* annotation δηλώνει την κλάση σαν *Bean* στο *Spring Container* και έτσι αντικείμενα(objects) της κλάσης αυτής μπορούν να χρησιμοποιηθούν μέσω Dependency Injection σε άλλα μέρη του προγράμματος.

@Transactional : Χρησιμοποιείται σε επίπεδο κλάσης, μεθόδου και interface. Οι μέθοδοι της κλάσης ή του interface ή οι μεμονωμένες μέθοδοι δηλώνονται ότι θα πραγματοποιήσουν Transactions σε μια βάση δεδομένων. Έτσι παρέχονται διευκολύνσεις όπως αυτόματο commit, δυνατότητα Rollback κ.α.

@Repository : Χρησιμοποιείται σε επίπεδο κλάσης. Δηλώνει μια κλάση η οποία έχει απευθείας πρόσβαση σε βάση δεδομένων. Επίσης αυτό το annotation δηλώνει την κλάση σαν *Bean* στο *Spring Container* και έτσι αντικείμενα(objects) της κλάσης αυτής μπορούν να χρησιμοποιηθούν μέσω Dependency Injection σε άλλα μέρη του προγράμματος.

@SpringBootApplication : Χρησιμοποιείται στην αρχική κλάση της Spring Boot εφαρμογής. Αυτό που κάνει είναι εξέταση της εφαρμογής για αναζήτηση κλάσεων οι οποίες θα προστεθούν στο *Spring Container* , έτσι ώστε να μπορούν να δημιουργηθούν Beans από αυτές. Η αναζήτηση όμως αυτή γίνεται μόνο στα πακέτα χαμηλότερου επιπέδου(sub-packages) της κλάσης αυτής. Το συγκεκριμένο annotation εκτελεί τις εργασίες annotations :

- *@Configuration*
- *@EnableAutoConfiguration*
- *@ComponentScan*

@Configuration : Χρησιμοποιείται σε επίπεδο κλάσης. Δηλώνει μια κλάση η οποία διαθέτει μεθόδους για την δημιουργία Beans. Οι μέθοδοι αυτές δηλώνονται με το annotation *@Bean*.

@EnableAutoConfiguration : Χρησιμοποιείται σε επίπεδο κλάσης. Συνήθως τοποθετείται στην κύρια κλάση της εφαρμογής και ορίζει ένα package για αναζήτηση Beans.

@ComponentScan : Χρησιμοποιείται σε επίπεδο κλάσης. Δηλώνει τα packages τα οποία θα χρησιμοποιηθούν για αναζήτηση Beans. Χρησιμοποιείται σε συνδυασμό με το annotation *@Configuration*.

@Component : Χρησιμοποιείται σε επίπεδο κλάσης. Δηλώνει την συγκεκριμένη κλάση σαν Bean έτσι ώστε ο μηχανισμός ανίχνευσης του Spring να το προσθέσει στο Spring Container.

Jackson

Το εργαλείο αυτό είναι ένα API το οποίο χρησιμοποιείται για την επεξεργασία JSON αρχείων σε προγράμματα γραμμένα σε γλώσσα προγραμματισμού Java. Η βιβλιοθήκη αυτή έχει γίνει γνωστή στον χώρο του προγραμματισμού λόγω χαρακτηριστικών όπως ταχύτητα, εργονομία και μικρή απαίτηση σε υπολογιστικούς πόρους.

Προσφέρει πολλαπλές προσεγγίσεις όσον αφορά την επεξεργασία των JSON αρχείων, όπως σχόλια(annotations) σε POJO(Plain Old Java Objects) κλάσεις για απλές περιπτώσεις χρήσης του εργαλείου, ή χρήση της κλάσης *ObjectMapper*.

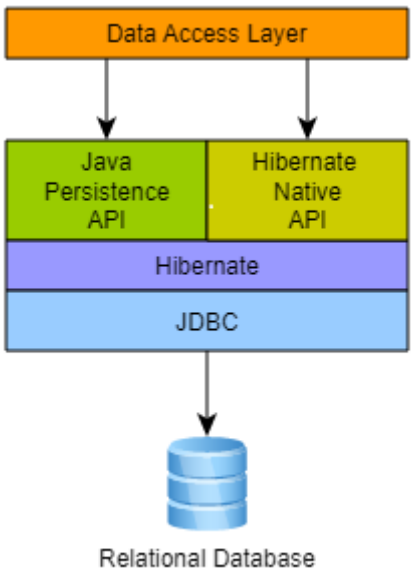
Στην εφαρμογή χρησιμοποιήθηκε η κλάση ObjectMapper και η μέθοδος *readValue()* η οποία διαβάζει κάποιο JSON αρχείο, είτε τοπικό είτε προερχόμενο από κάποιο απομακρυσμένο URL, το αντιστοιχίζει με κάποια POJO κλάση και επιστρέφει σαν τιμή ένα νέο αντικείμενο προερχόμενο από αυτήν την POJO κλάση. Για την αντίστροφη λειτουργία της μετατροπής κάποιου αντικειμένου προερχόμενο από POJO κλάση σε μορφή JSON, χρησιμοποιείται η μέθοδος *writeValue()*.

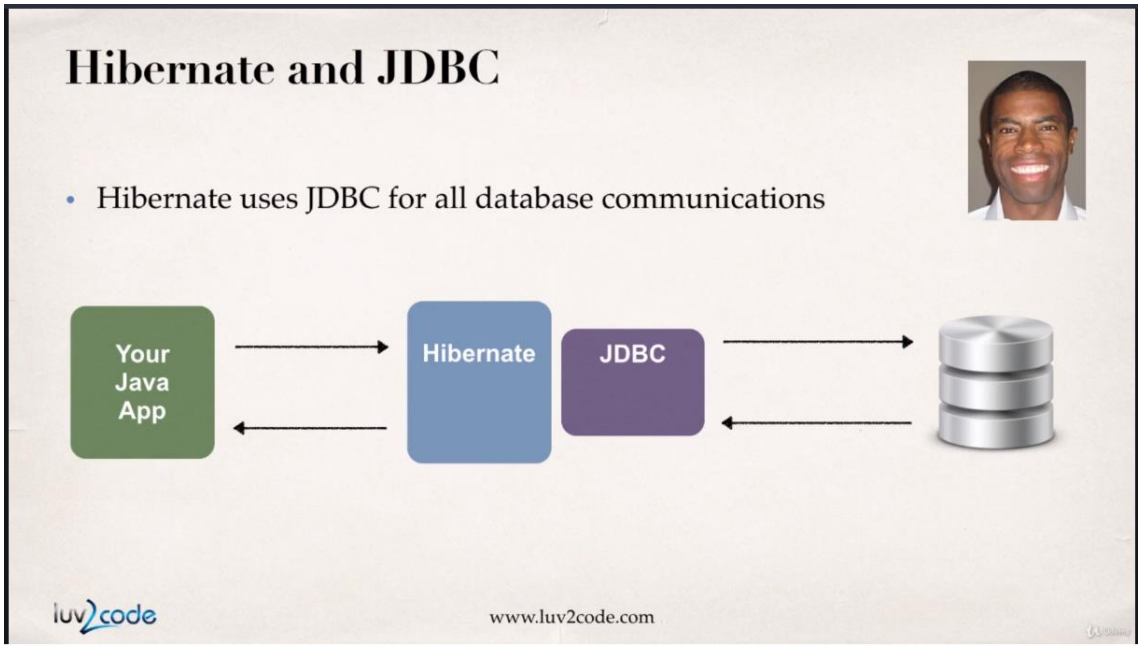
Hibernate

Είναι ένα ORM(Object Relational Mapping) εργαλείο για την γλώσσα προγραμματισμού Java και αποτελεί υλοποίηση του Java Persistence API(JPA), ακολουθώντας τα κοινά πρότυπα τα οποία παρέχονται από αυτό. Παρέχει ένα framework για την αντιστοίχιση-απεικόνιση ενός μοντέλου αντικειμενοστραφούς τομέα σε μια σχεσιακή βάση δεδομένων. Το Hibernate χειρίζεται τα προβλήματα αντιστοίχισης-απεικόνισης μεταξύ αντικειμένων και βάσης δεδομένων αντικαθιστώντας τις απευθείας προσβάσεις στην βάση δ. οι οποίες αφορούν την αποθήκευση(persistence), με μεθόδους οι οποίες χειρίζονται αντικείμενα.

Εσωτερικά χρησιμοποιείται το JDBC(Java Database Connectivity) API για την επικοινωνία με την βάση δεδομένων. Ουσιαστικά αποτελεί ένα επιπλέον επίπεδο αφηρημένης έννοιας(abstraction) πάνω από το JDBC, επιτρέποντας στον προγραμματιστή να μην χρησιμοποιεί χαμηλού επιπέδου(low level) JDBC κλήσεις.

Τα παραπάνω απεικονίζονται στα σχήματα που ακολουθούν :





Το Hibernate είναι ελεύθερο λογισμικό το οποίο διανέμεται με την άδεια του οργανισμού GNU.

Το κύριο χαρακτηριστικό του Hibernate είναι η αντιστοίχιση-απεικόνιση Java κλάσεων σε πίνακες σχεσιακών βάσεων δεδομένων και η αντιστοίχιση-απεικόνιση τύπων δεδομένων της Java σε τύπους δεδομένων SQL(Structured Query Language). Επίσης παρέχει διευκολύνσεις τόσο για ερωτήσεις(queries) προς την βάση δ., όσο και για την ανάκτηση δεδομένων τα οποία προέρχονται από αυτά τα ερωτήματα. Παράγει κλήσεις SQL και απαλλάσσει τον προγραμματιστή από την ανάγκη χειροκίνητου χειρισμού και μετατροπής του εκάστοτε αποτελέσματος από την βάση δ. σε Java αντικείμενο.

Η αντιστοίχιση-απεικόνιση Java κλάσεων σε πίνακες βάσης δ. υλοποιείται με την παραμετροποίηση ενός XML αρχείου ή χρησιμοποιώντας Java annotations. Όταν χρησιμοποιείται αρχείο XML, το Hibernate μπορεί να παράγει έναν σκελετό πηγαίου κώδικα για τις κλάσεις που αφορούν την αποθήκευση δεδομένων, κάτι το οποίο είναι βοηθητικό όταν χρησιμοποιούνται annotations. Επίσης το Hibernate μπορεί να χρησιμοποιήσει το XML αρχείο ή τα Java annotations για την συντήρηση του σχήματος(schema) της βάσης δ. Τέλος υποστηρίζει λειτουργίες για την διευθέτηση σχέσεων one-to-many(1:N) και many-to-many(M:N) μεταξύ κλάσεων.

Το Hibernate παρέχει μια γλώσσα εμπνευσμένη από την SQL, η οποία καλείται Hibernate Query Language(HQL), για την συγγραφή ερωτήσεων τύπου SQL προς τα αντικείμενα δεδομένων(data objects) του Hibernate. Η HQL αποτελεί την αντικειμενοστραφή έκδοση της γλώσσας SQL. Παράγει ερωτήσεις προς την βάση δ. ανεξάρτητες από την τεχνολογία-έκδοση της βάσης δ. Χωρίς αυτήν την δυνατότητα, οποιαδήποτε αλλαγή στην βάση δ. θα απαιτούσε και την αλλαγή των μεμονωμένων ερωτήσεων SQL, οδηγώντας σε προβλήματα συντήρησης. Μια αντικειμενοστραφής εναλλακτική σε σχέση με την γλώσσα HQL αποτελεί το ερώτημα βάσει κριτηρίων(Criteria Query). Το ερώτημα βάσει κριτηρίων χρησιμοποιείται για την τροποποίηση και τον παροχή περιορισμών σε αντικείμενα.

Το Hibernate παρέχει την δυνατότητα αποθήκευσης σε POJOs. Η μόνη προϋπόθεση για να μπορεί μία κλάση να χρησιμοποιηθεί προς αποθήκευση είναι η ύπαρξη κατασκευαστή χωρίς παραμέτρους, ο οποίος δεν είναι απαραίτητο να είναι public.

Οι συλλογές(Collections) οι οποίες αποτελούνται από αντικείμενα δεδομένων αποθηκεύονται σε Java κλάσεις συλλογών(Collection Classes), όπως οι υλοποιήσεις των Set και List interfaces. Επίσης η τεχνολογία Java generics, με την οποία δίνεται η δυνατότητα να δημιουργήσουμε κλάσεις, διεπαφές(interfaces) και μεθόδους οι οποίες θα λειτουργούν με έναν

τρόπο ανεξάρτητο από τον τύπο των δεδομένων τους(type-safe) όπως και με διάφορα είδη δεδομένων.

Eager/Lazy Loading

Όταν δουλεύουμε με ένα ORM πρόγραμμα, η φόρτωση των δεδομένων(data fetching/loading) μπορεί να κατηγοριοποιηθεί σε δύο τύπους : *Eager* και *Lazy*. Αυτοί οι δύο τύποι φόρτωσης δεδομένων στο Hibernate αποτελούν σχεδιαστικά πρότυπα(design patterns) και διαφέρουν στα εξής.

Eager Loading : Η αρχικοποίηση των δεδομένων λαμβάνει χώρα επί τόπου.

Lazy Loading : Η αρχικοποίηση δεδομένων ενός αντικειμένου καθυστερεί όσο το δυνατόν πιο πολύ.

Τα πλεονεκτήματα και μειονεκτήματα κάθε τύπου φαίνονται παρακάτω :

Lazy Loading

Πλεονεκτήματα :

- Πολύ μικρότερος αρχικός χρόνος φόρτωσης σε σχέση με τον άλλο τύπο φόρτωσης.
- Μικρότερη κατανάλωση μνήμης σε σχέση με τον άλλο τύπο.

Μειονεκτήματα :

- Η καθυστερημένη αρχικοποίηση πολλές φορές μπορεί να έχει επίπτωση στην απόδοση.
- Σε μερικές περιπτώσεις τα αντικείμενα τα οποία έχουν αρχικοποιηθεί με αυτόν τον τύπο χρειάζονται ειδική μεταχείριση αλλιώς υπάρχει ενδεχόμενο να δημιουργηθεί exception.

Eager Loading

Πλεονεκτήματα :

- Δεν υπάρχει καμία καθυστέρηση κατά την αρχικοποίηση και έτσι καμία επίπτωση στην απόδοση.

Μειονεκτήματα :

- Καθυστέρηση κατά τον αρχικό χρόνο φόρτωσης.
- Η φόρτωση πολλών αχρείαστων δεδομένων μπορεί να έχει επίπτωση στην απόδοση.

Cascade Types

Στο Hibernate, οι σχέσεις μεταξύ οντοτήτων(entities) συχνά εξαρτώνται από την ύπαρξη κάποιας άλλης οντότητας. Για παράδειγμα στην περίπτωση της σχέσης ανθρώπου-διεύθυνσης κατοικίας, όπου χωρίς την οντότητα του ανθρώπου η οντότητα της διεύθυνσης κατοικίας δεν έχει κανένα νόημα. Για αυτόν ακριβώς τον λόγο χρησιμοποιούμε την διαδικασία της αλληλουχίας με μορφή καταρράκτη(*Cascading)*. Έτσι όταν εκτελέσουμε κάποια ενέργεια σε μια συγκεκριμένη οντότητα, η ίδια ενέργεια θα εφαρμοστεί και στην οντότητα με την οποία συσχετίζεται η οντότητα αυτή.

Οι τύποι Cascade οι οποίοι υποστηρίζονται από το Hibernate ORM είναι οι εξής :

- *ALL*
- *PERSIST*
- *MERGE*
- *REMOVE*
- *REFRESH*
- *DETACH*
- *REPLICATE*
- *SAVE_UPDATE*
- *LOCK*

Για να γίνει ακριβής παρουσίαση των τύπων Cascade, θα πρέπει να παρουσιαστεί το *Persistence Context* το οποίο χρησιμοποιείται από το Hibernate για την διαχείριση του κύκλου ζωής των οντοτήτων σε μια εφαρμογή. Ουσιαστικά αποτελεί την κρυφή μνήμη(cache) πρώτου επιπέδου, όπου όλες οι οντότητες παραλαμβάνονται από την βάση δεδομένων ή αποθηκεύονται σε αυτή. Η θέση του *Persistence Context* δηλαδή βρίσκεται ανάμεσα στην εφαρμογή και την βάση δεδομένων.

Παρακάτω παρουσιάζονται αναλυτικά ο καθένας από τους τύπους Cascade :

ALL(CascadeType.ALL) : Μεταδίδει όλες τις ενέργειες από μία οντότητα γονέα σε μία οντότητα παιδί.

PERSIST(CascadeType.PERSIST) : Μεταδίδει την ενέργεια αποθήκευσης(persist) από μία οντότητα γονέα σε μία οντότητα παιδί.

MERGE(CascadeType.MERGE) : Μεταδίδει την ενέργεια συγχώνευσης(merge) από μία οντότητα γονέα σε μία οντότητα παιδί. Με την ενέργεια αυτή γίνεται ενημέρωση κάποιου αντικειμένου με νέες τιμές και στην συνέχεια ενημερώνεται η βάση δεδομένων με βάση αυτό το αντικείμενο.

REMOVE(CascadeType.REMOVE) : Μεταδίδει την ενέργεια διαγραφής(remove) από μία οντότητα γονέα σε μία οντότητα παιδί. Η ενέργεια αυτή διαγράφει την εγγραφή η οποία αντιστοιχεί στην συγκεκριμένη οντότητα, από την βάση δεδομένων όπως επίσης και από το Persistence Context.

DETACH(CascadeType.DETACH) : Μεταδίδει την ενέργεια απόσπασης(detach) από μία οντότητα γονέα σε μία οντότητα παιδί. Αυτή η ενέργεια αφαιρεί την οντότητα από το Persistent Context.

LOCK(CascadeType.LOCK) : Μεταδίδει την ενέργεια κλειδώματος(lock) από μία οντότητα γονέα σε μία οντότητα παιδί. Με την ενέργεια αυτή η οντότητα γονέας και η οντότητα παιδί επανέρχονται ξανά στο Persistence Context.

REFRESH(CascadeType.REFRESH) : Μεταδίδει την ενέργεια ανανέωσης(refresh) από μία οντότητα γονέα σε μία οντότητα παιδί. Με την ενέργεια αυτή γίνεται ξανά ανάγνωση των τιμών μιας οντότητας από την βάση δεδομένων.

REPLICATE(CascadeType.REPLICATE) : Μεταδίδει την ενέργεια αντιγραφής(replicate) από μία οντότητα γονέα σε μία οντότητα παιδί. Η ενέργεια αυτή χρησιμοποιείται όταν υπάρχουν παραπάνω από μία πηγές δεδομένων και χρειαζόμαστε συγχρονισμό των δεδομένων αυτών.

SAVE_UPDATE(CascadeType.SAVE_UPDATE) : Μεταδίδει τις ενέργειες save, update και saveOrUpdate από μία οντότητα γονέα σε μία οντότητα παιδί. Οι παραπάνω ενέργειες δεν αποτελούν μέρος του JPA και χρησιμοποιούνται αποκλειστικά στο Hibernate.

Maven

Η κατασκευή ενός έργου(project) λογισμικού συνήθως αποτελείται από εργασίες όπως το κατέβασμα(downloading) διάφορων εξαρτήσεων(dependencies), την προσθήκη JAR(Java ARchives) αρχεία στην διαδρομή του project(classpath), την μεταγλώττιση(compiling) πηγαίου(source) κώδικα σε δυαδικό(binary) κώδικα, την εκτέλεση δοκιμών(testing), το πακετάρισμα(packaging) μεταγλωττισμένου κώδικα σε αρχεία τύπου JAR, WAR ή ZIP και την ανάπτυξη αυτών των αρχείων σε διακομιστές εφαρμογών(application server) ή αποθήκες δεδομένων(repository).

Το Apache Maven αυτοματοποιεί τέτοιου είδους εργασίες, ελαχιστοποιώντας τον κίνδυνο ανθρώπινων λαθών κατά την διαδικασία κατασκευής λογισμικού χειροκίνητα και διαχωρίζει την διαδικασία μεταγλώττισης και πακεταρίσματος του κώδικα από αυτήν της κατασκευής κώδικα.

Η διαχείριση των dependencies όπως αναφέρθηκε, αποτελεί ένα σημαντικό χαρακτηριστικό του Maven. Τα dependencies είναι αρχεία όπως JAR, ZIP κ.τ.λ. τα οποία είναι απαραίτητα στο project έτσι ώστε να μπορεί να μεταγλωττιστεί, να γίνει build, να τρέξει τεστ και να εκτελεστεί. Με λίγα λόγια τα dependencies είναι εξωτερικές βιβλιοθήκες(libraries) τις οποίες χρησιμοποιεί το project. Όταν εκτελούμε ένα build για παράδειγμα, τα dependencies αναλύονται(resolved) και στην συνέχεια φορτώνονται(loaded) από κάποιο repository, τοπικό ή απομακρυσμένο.

Τα κύρια χαρακτηριστικά του Maven είναι :

- Απλή διαδικασία εγκατάστασης του project η οποία ακολουθεί δοκιμασμένες πρακτικές(best practices) : Το Maven αποφεύγει όσο το δυνατόν περισσότερες παραμετροποιήσεις, παρέχοντας έτοιμα project templates τα οποία ονομάζονται archetypes.
- Διαχείριση dependencies : Περιλαμβάνει αυτόματη ενημέρωση, κατέβασμα και έλεγχο συμβατότητας των dependencies όπως επίσης και την αναφορά του κλεισίματος του dependency.
- Απομόνωση των dependencies και των plugins ενός project : Με το Maven τα project dependencies ανακτώνται από τα dependency repositories ενώ τα dependencies των plugins ανακτώνται από τα plugin repositories. Αυτό έχει σαν αποτέλεσμα λιγότερες συγκρούσεις όταν τα plugins κατεβάζουν επιπρόσθετα dependencies.
- Κεντρικό σύστημα repository : Τα dependencies του project μπορούν να φορτωθούν από το τοπικό σύστημα αρχείων(file system) από δημόσια repositories, όπως το Maven central.

Η παραμετροποίηση ενός Maven project πραγματοποιείται μέσω ενός *Project Object Model(POM)*, το οποίο αναπαρίσταται από το αρχείο pom.xml. Το *POM* περιγράφει το project, διαχειρίζεται τα dependencies και παραμετροποιεί τα plugins για την διαδικασία build του λογισμικού. Το POM επίσης ορίζει τις σχέσεις μεταξύ των modules σε project τα οποία περιλαμβάνουν πολλαπλά modules.

Η βασική δομή ενός POM αρχείου είναι η ακόλουθη :

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.baeldung</groupId>
  <artifactId>org.baeldung</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>org.baeldung</name>
```

```
    <url>http://maven.apache.org</url>
    <dependencies>
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
      </dependency>
    </dependencies>
    <build>
      <plugins>
        <plugin>
        //...
        </plugin>
      </plugins>
    </build>
  </project>
```

Το Maven χρησιμοποιεί μια ομάδα από αναγνωριστικά(identifiers) τα οποία καλούνται και συντεταγμένες(coordinates), για αναγνωρίσει μοναδικά ένα project και να προσδιορίσει πως αυτό το project θα μετατραπεί σε πακέτο(package). Ακολουθεί η επεξήγηση των identifiers :

- groupId : Ένα μοναδικό όνομα της εταιρίας ή ομάδας η οποία δημιούργησε το project
- artifactId : Ένα μοναδικό όνομα για το συγκεκριμένο project
- version : Μία έκδοση του project
- packaging : Μία μέθοδος δημιουργίας package, όπως WAR, JAR, ZIP κ.τ.λ.

Τα πρώτα τρία αναγνωριστικά(groupId:artifactId:version) συνδυάζονται για να δημιουργηθεί το μοναδικό identifier και είναι ο μηχανισμός με τον οποίο προσδιορίζουμε ποιες εκδόσεις εξωτερικών βιβλιοθηκών θα χρησιμοποιήσει κάποιο project.

Για την δήλωση των dependencies εξωτερικών βιβλιοθηκών χρειάζεται να παρέχουμε το groupId, το artifactId και το version της βιβλιοθήκης, όπως φαίνεται και στο παραπάνω παράδειγμα. Κατά την διάρκεια της επεξεργασίας των dependencies, το Maven θα κατεβάσει την Spring βιβλιοθήκη στο τοπικό Maven repository.

Το τμήμα build είναι επίσης πολύ σημαντικό τμήμα στο Maven POM. Παρέχει πληροφορίες σχετικά με το προεπιλεγμένο σκοπό(goal) του Maven, τον φάκελο για το μεταγλωττισμένο project και το τελικό όνομα της εφαρμογής.

Κάθε Maven build ακολουθεί έναν συγκεκριμένο κύκλο ζωής(lifecycle). Μπορούν να εκτελεστούν πολλαπλά build lifecycle goals όπως αυτό της μεταγλώττισης του κώδικα του project, της δημιουργίας ενός package και της εγκατάστασης κάποιου αρχείου στο τοπικό dependency repository. Η ακόλουθη λίστα παρουσιάζει τις πιο σημαντικές φάσεις του Maven κύκλου ζωής :

- validate : Ελέγχει την ορθότητα του project
- compile : Μεταγλωττίζει τον πηγαίο κώδικα σε δυαδικά αντικείμενα(artifacts)
- test : Εκτελεί τα Unit tests
- package : Συσκευάζει(packages) τον μεταγλωττισμένο κώδικα σε αρχείο τύπου archive
- integration test : Εκτελεί επιπλέον τεστ, τα οποία απαιτούν την συσκευασία(packaging)
- verify : Ελέγχει εάν το πακέτο(package) είναι έγκυρο(valid)
- install : Εγκαθιστά το πακέτο στο τοπικό Maven repository

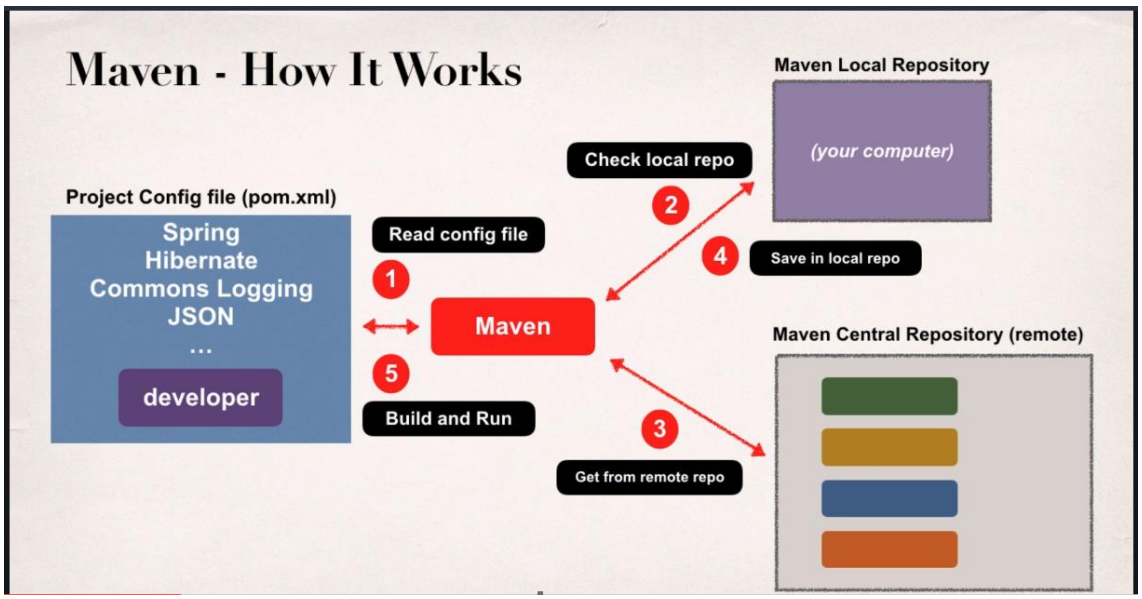
- deploy : Αναπτύσσει(deploy) το πακέτο σε έναν απομακρυσμένο server ή repository

Όσον αφορά τα Maven plugins τα οποία αναφέρονται στο POM, αυτά αποτελούν μια συλλογή από ένα ή περισσότερους στόχους(goals). Τα goals εκτελούνται σε φάσεις, κάτι το οποίο βοηθά να στο να καθοριστεί η σειρά με την οποία αυτά τα goals θα εκτελεστούν.

Τέλος παρουσιάζεται η λειτουργία των repositories στο Maven. Η διαδικασία έχει ως εξής :

1. Τα dependencies του project, τα οποία αναφέρονται στο αρχείο παραμετροποίησης pom.xml, διαβάζονται από το Maven
2. Τα dependencies τα οποία περιέχονται στο POM αρχείο αναζητούνται στο τοπικό repository
3. Εάν τα dependencies δεν βρεθούν στο τοπικό repository, τότε αναζητούνται στο απομακρυσμένο Maven central repository
4. Τα dependencies τα οποία βρέθηκαν στο Maven central repository αποθηκεύονται τοπικά στο τοπικό repository
5. Τα dependencies τα οποία βρίσκονται τοπικά πλέον, χρησιμοποιούνται για να γίνει build και να εκτελεστεί η εφαρμογή

Η λειτουργία των Maven repositories παρουσιάζεται σχηματικά παρακάτω :



Γενικά υπάρχουν δύο ειδών repositories, 1)το τοπικό(local) repository και 2)το κεντρικό(central) repository. Το local repository εγκαθίσταται στον Η/Υ στον οποίο γίνεται η ανάπτυξη της εφαρμογής. Σε λειτουργικά συστήματα MS Windows το local repository βρίσκεται στην διαδρομή C:\users\<user-home-dir>\.m2\repository, ενώ σε λειτουργικά συστήματα Mac και Linux βρίσκεται στην διαδρομή ~/.m2/repository (όπου ~ είναι το home directory). Το Maven θα αναζητήσει τα dependencies πρώτα στο local repository πριν στραφεί στο central repository. Με λίγα λόγια το local repository παίζει τον ρόλο της μνήμης cache όσον αφορά τα dependencies μιας εφαρμογής. Το Maven θα αναζητήσει το απομακρυσμένο central repository στην προεπιλεγμένη διεύθυνση https://repo.maven.apache.org/maven2/ . Όταν τα αρχεία τα οποία αποτελούν τα dependencies κατέβουν από το διαδίκτυο αποθηκεύονται στο local repository. Κατ' αυτόν τον τρόπο όταν το Maven χρειαστεί κάποιο από αυτά τα dependencies ξανά, θα το εντοπίσει αποθηκευμένο στο local repository και έτσι δεν θα χρειαστεί να γίνει ξανά αναζήτηση στο central repository.

Thymeleaf

Μία μηχανή υποδειγμάτων(template engine) επαναχρησιμοποιεί στατικά στοιχεία ιστοσελίδων ενώ ορίζει δυναμικά στοιχεία τα οποία βασίζονται σε παραμέτρους οι οποίες προέρχονται από web requests. Οι προγραμματιστές μπορούν να υλοποιήσουν templates με την βοήθεια Content management Systems(CMS), Web application frameworks και HTML editors.

Το Thymeleaf είναι μία Java XML/XHTML/HTML5 template engine, η οποία μπορεί να χρησιμοποιηθεί τόσο σε web(βασισμένα σε servlets) όσο και σε μη web περιβάλλοντα. Είναι κατάλληλο για να εξυπηρετεί XHTML/HTML5 σελίδες στο επίπεδο View μιας MVC(Model-View-Controller) web εφαρμογής. Επίσης μπορεί να επεξεργαστεί οποιοδήποτε XML αρχείο ακόμα και σε περιβάλλοντα εκτός διαδικτύου.

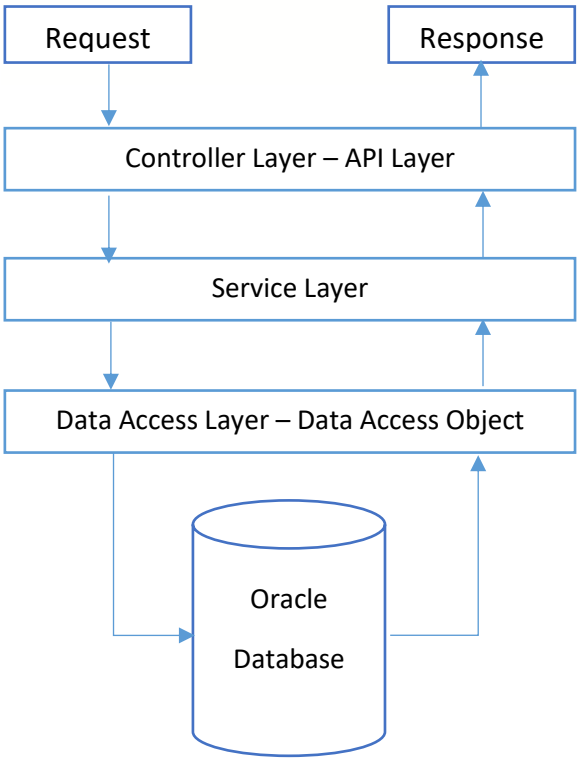
Σε web εφαρμογές το Thymeleaf στοχεύει να αποτελέσει τον αντικαταστάτη των java Server Pages(JSP) και υλοποιεί την έννοια των Natural Templates. Template αρχεία τα οποία μπορεί να ανοίξει κανείς απευθείας σε φυλλομετρητές(browsers) και τα οποία λειτουργούν σωστά σαν ιστοσελίδες.

Το Thymeleaf αποτελεί λογισμικό ανοιχτού κώδικα, με άδεια χρήσης σύμφωνη με την άδεια Apache License 2.0 .

Τα κύρια χαρακτηριστικά του Thymeleaf αναφέρονται παρακάτω :

- Java template engine για XML, XHTML και HTML5
- Χρησιμοποιείται σε web και μη web(offline) περιβάλλοντα. Δεν υπάρχει μεγάλη εξάρτηση με το Servlet Application Programming Interface(API)
- Βασίζεται σε αρθρωτά(modular) σύνολα χαρακτηριστικών τα οποία ονομάζονται διάλεκτοι(dialects)
    - Τα χαρακτηριστικά ενός dialect, όπως είναι η αξιολόγηση, η επανάληψη κ.α., εφαρμόζονται συνδέοντάς τα με tags ή attributes ενός template
    - Δύο dialects είναι άμεσα διαθέσιμοι : Standard και SpringStandard(για Spring MVC εφαρμογές)
    - Οι προγραμματιστές έχουν την δυνατότητα να επεκτείνουν και να δημιουργήσουν δικές τους διαλέκτους(dialects)
- Πολλαπλές template λειτουργίες
    - XML
    - XHTML
    - HTML5 : XML κώδικας και παλαιότερος κώδικας HTML5
- Πλήρης υποστήριξη για διεθνοποίηση(internationalization)
- Παραμετροποιήσιμη, υψηλής απόδοσης template cache, η οποία μειώνει την είσοδο(input)/έξοδο(output) στο ελάχιστο
- Αυτόματες μεταφράσεις DOCTYPE για επαλήθευση τόσο του template όσο και του εξαγόμενου κώδικα
- Πάρα πολύ επεκτάσιμο : μπορεί να χρησιμοποιηθεί σαν template engine framework εάν χρειαστεί
- Πλήρης τεκμηρίωση(documentation) συμπεριλαμβανομένων πολλών εφαρμογών παραδειγμάτων

Η αρχιτεκτονική της εφαρμογής αποτελείται από τρία επίπεδα όπως φαίνεται παρακάτω :



Controller Layer – API Layer

Χειρίζεται τα εξωτερικά HTTP αιτήματα των χρηστών.

Βρίσκεται στο package **api** του project και αποτελείται από δύο κλάσεις, την *ApiCallRestController* και την *WebsiteController*.

Η κλάση *ApiCallRestController* διαθέτει τις μεθόδους *save(city, state, country), localFindAll(), localFindById(apiCallId), localDeleteById(apiCallId), localUpdateApiCall(apiCall), localAddApiCall(apiCall)*. Κάθε μία από αυτές τις μεθόδους ανταποκρίνεται σε κάποιο HTTP αίτημα(GET, POST, PUT, DELETE) και σε κάποιο URL(http://localhost:8089/api/localdata/apicalls, http://localhost:8089/api/{city}/{state}/{country} κ.α.) και επιστρέφει στον χρήστη δεδομένα με την μορφή JSON.

Στην μέθοδο save(*city, state, country*) γίνεται χρήση του εργαλείου *Jackson* για την εργασία Data Binding μεταξύ του αρχείου JSON το οποίο προέρχεται από το WebAPI της iqair.com και ενός αντικειμένου το οποίο προέρχεται από την κλάση City.

Όταν χρειάζεται κάποια μέθοδος να διαβάσει ή να γράψει στην βάση δεδομένων, τότε γίνεται χρήση κάποιας μεθόδου της κλάσης *ApiCallServiceImpl* η οποία αποτελεί το Service Layer.

Η κλάση *WebsiteController* διαθέτει τις μεθόδους *showList(model)* και *showDetails(id, model)* οι οποίες ανταποκρίνονται σε HTTP αιτήματα τύπου GET και στα URL http://localhost:8089/website/list και http://localhost:8089/website/details/{id} αντίστοιχα. Οι μέθοδοι αυτές επιστρέφουν τις HTML σελίδες index.html και details.html αντίστοιχα, στις οποίες γίνεται χρήση του *Thymeleaf* template engine για την παρουσίαση των δεδομένων τα οποία προέρχονται από την βάση δεδομένων.

Και σε αυτές τις μεθόδους όταν χρειαστεί να ανακτηθούν ή να γραφούν δεδομένα από και προς την βάση δεδομένων, χρησιμοποιούνται μέθοδοι του Service Layer.

Service Layer

Βρίσκεται στο package **service** του project και αποτελεί το ενδιάμεσο επίπεδο ανάμεσα στο Controller και στο Data Access.Το επίπεδο αυτό αποτελείται από το interface ApiCallService και την κλάση η οποία το υλοποιεί, ApiCallServiceImpl. Περιέχει την επιχειρηματική λογική(business logic) της εφαρμογής. Οι μέθοδοι αυτού του επιπέδου κάνουν χρήση της κλάσης ApiCallDAOImpl του Data Access Layer όταν χρειάζεται να έχουν πρόσβαση στην βάση δεδομένων για εγγραφή ή ανάγνωση. Επίσης όσες μέθοδοι του Service Layer επιστρέφουν τιμές , αυτές είναι αντικείμενα τα οποία προέρχονται από την κλάση ApiCall.

Data Access Layer – Data Access Object

Βρίσκεται στο package **dao** του project και αποτελεί το επίπεδο το οποίο επικοινωνεί απευθείας με την βάση δεδομένων. Το επίπεδο αυτό αποτελείται από το interface ApiCallDAO και την κλάση η οποία το υλοποιεί, ApiCallDAOImpl. Οι μέθοδοι αυτού του επιπέδου χρησιμοποιούν το ORM εργαλείο *Hibernate* έτσι ώστε να δημιουργήσουν, διαβάσουν, ενημερώσουν ή να διαγράψουν εγγραφές στην βάση δεδομένων. Το *Hibernate* αντιστοιχίζει πίνακες και τα πεδία αυτών από την βάση δ., με αντικείμενα και τα πεδία αυτών, τα οποία δημιουργούνται από κλάσεις. Συγκεκριμένα στην εφαρμογή μας αντιστοιχίζονται οι πίνακες της σχεσιακής βάσης δεδομένων apicall και city_coordinates, με αντικείμενα της κλάσης ApiCall. Η κλάση ApiCall όπως και οι κλάσεις τις οποίες περιλαμβάνει η ApiCall βρίσκονται στο package **model** της εφαρμογής και συμβολίζουν το μοντέλο, σύμφωνα με το οποίο έχει δημιουργηθεί η δομή των πινάκων της βάσης δεδομένων.

Η προαναφερθείσα αντιστοίχιση επιτυγχάνεται από το *Hibernate* χρησιμοποιώντας σχόλια(annotations), τα οποία ξεκινούν με το σύμβολο @, μέσα στις κλάσεις του μοντέλου.

Λίστα με τα *Hibernate Annotations* τα οποία έχουν χρησιμοποιηθεί στις κλάσεις ApiCall, City, Current, Data, Location, Pollution, Weather οι οποίες αποτελούν και το μοντέλο της εφαρμογής, δίνεται παρακάτω :

@Entity : Χαρακτηρίζει μία κλάση σαν οντότητα η οποία αντιστοιχίζεται-απεικονίζεται σε έναν πίνακα της σχεσιακής βάσης δεδομένων.

@Id : Χαρακτηρίζει μια μεταβλητή μέλος μιας κλάσης σαν κύριο κλειδί(primary key). Έτσι η μεταβλητή αυτή θα αντιστοιχισθεί σε ένα πεδίο ενός πίνακα, το οποίο θα αποτελέσει το κύριο κλειδί του πίνακα αυτού.

@GeneratedValue(strategy = GenerationType.IDENTITY) : Η συγκεκριμένη στρατηγική η οποία επιλέχθηκε για το κύριο κλειδί είναι αυτή της αυτόματης αύξησης(auto increment). Κάθε φορά που δημιουργείται καινούρια εγγραφή στην βάση δ., η τιμή του πεδίου αυτού αυξάνεται κατά ένα σε σχέση με την προηγούμενη εγγραφή.

@Embedded : Ενσωμάτωση μιας μεταβλητής μέλους της κλάσης η οποία αναφέρεται σε κάποιο αντικείμενο(object reference variable) και αντιστοίχιση-απεικόνιση αυτής της μεταβλητής στον ίδιο πίνακα στην βάση δεδομένων, με αυτόν της κλάσης στην οποία ανήκει. Με λίγα λόγια, η κλάση στην οποία αναφέρεται η *@Embedded* μεταβλητή δεν θα αποτελέσει ξεχωριστό πίνακα στην βάση δεδομένων.

@Embeddable : Χρησιμοποιείται για να δηλώσει ότι μια κλάση θα ενσωματωθεί από άλλες οντότητες(Entities) και δεν θα αποτελέσει ξεχωριστό πίνακα στην βάση δεδομένων. Χρησιμοποιείται σε συνδυασμό με το annotation *@Embedded*. Με το annotation *@Embedded* χαρακτηρίζουμε την μεταβλητή αναφοράς αντικειμένου ενώ με το annotation *@Embeddable* χαρακτηρίζουμε την κλάση από την οποία προέρχονται τα *@Embedded* αντικείμενα.

@ElementCollection : Χρησιμοποιείται για την αποθήκευση μιας λίστας τιμών σε ξεχωριστό πίνακα στην βάση δεδομένων, σαν η λίστα αυτή να αποτελούσε μια ξεχωριστή οντότητα(Entity).

@CollectionTable(name = "city_coordinates", joinColumns = @JoinColumn(name = "apicall_id")) : Ορίζεται ο πίνακας(*"city_coordinates"*) ο οποίος θα χρησιμοποιηθεί για την αποθήκευση της λίστας τιμών και το πεδίο(*"apicall_id"*) το οποίο θα αποτελέσει το ξένο κλειδί(foreign key) προς την κεντρική οντότητα του μοντέλου μας. Με λίγα λόγια το πεδίο *apicall_id* συνδέει τον πίνακα *city_coordinates* με τον κεντρικό πίνακα *apicall* και συγκεκριμένα το κύριο κλειδί του, *id*.

@Column : Ορίζει ιδιότητες για κάποια μεταβλητή μέλος μιας κλάσης τις οποίες θα έχει το αντίστοιχο πεδίο πίνακα στην βάση δεδομένων.

name = "weather_ts" : Ορίζει το όνομα του πεδίου στον πίνακα της βάσης δεδομένων. Η προεπιλογή στο *Hibernate* είναι το πεδίο του πίνακα να παίρνει το ίδιο όνομα με την μεταβλητή κλάσης με την οποία έχει αντιστοιχισθεί-απεικονισθεί. Εάν χρειάζεται να αλλάξουμε το προεπιλεγμένο όνομα πεδίου πίνακα χρησιμοποιούμε το παρόν annotation.

Oracle Database

Η βάση δεδομένων αποτελείται από τους πίνακες apicall και city_coordinates όπως προαναφέρθηκε. Στον πίνακα apicall αποθηκεύονται στοιχεία όπως χώρα, επαρχία και πόλη για τις οποίες λήφθηκαν ατμοσφαιρικές μετρήσεις όπως θερμοκρασία, υγρασία, ρύπανση κ.α. Στον πίνακα city_coordinates αποθηκεύονται οι συντεταγμένες του μέρους για το οποίο έγινε η ατμοσφαιρική μέτρηση. Για κάθε εγγραφή στον πίνακα apicall αντιστοιχούν δύο εγγραφές του πίνακα city_coordinates, μία εγγραφή για το γεωγραφικό μήκος και μία για το γεωγραφικό πλάτος της πόλης στην οποία έγινε η μέτρηση. Δηλαδή έχουμε μία σχέση 1:N μεταξύ των δύο πινάκων.

Παρακάτω δίνονται οι ορισμοί των πινάκων της βάσης δεδομένων με την μορφή εντολών SQL(Structured Query Language) :

```
CREATE TABLE apicall(
   id NUMBER GENERATED BY DEFAULT AS IDENTITY,
   status VARCHAR2(50) NOT NULL,
   city VARCHAR2(50) NOT NULL,
   state VARCHAR2(50) NOT NULL,
   country VARCHAR2(50) NOT NULL,
   type VARCHAR2(50) NOT NULL,
   weather_ts VARCHAR2(50) NOT NULL,
   tp NUMBER NOT NULL,
   pr NUMBER NOT NULL,
   hu NUMBER NOT NULL,
   ws NUMBER NOT NULL,
   wd NUMBER NOT NULL,
   ic VARCHAR2(50) NOT NULL,
   pollution_ts VARCHAR2(50) NOT NULL,
   aqius NUMBER NOT NULL,
   mainus VARCHAR2(50) NOT NULL,
   aqicn NUMBER NOT NULL,
   maincn VARCHAR2(50) NOT NULL,

   PRIMARY KEY(id)
);

CREATE TABLE city_coordinates(
   apicall_id NUMBER NOT NULL,
   coordinate NUMBER NOT NULL,

   FOREIGN KEY(apicall_id) REFERENCES apicall(id)
);
```

Τέλος όλες οι απαραίτητες ιδιότητες για την σύνδεση της εφαρμογής μας με την βάση δεδομένων της Oracle ορίζονται στο αρχείο *application.properties* του project. Αυτές είναι το URL(Uniform Resource Locator) για την σύνδεση με την βάση δ., το όνομα χρήστη(username) και ο κωδικός(password), ο οδηγός(driver) ο οποίος χρησιμοποιείται για την σύνδεση με την συγκεκριμένη βάση δ. Όλες αυτές οι ρυθμίσεις δίνονται παρακάτω :

```
spring.datasource.url = jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username = DIP
spring.datasource.password = DIP

spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
```

Επίσης στο αρχείο *application.properties* ορίζεται και η πόρτα την οποία χρησιμοποιεί ο τοπικός διακομιστής διαδικτύου(Localhost Web Server) :

```
server.port=8089
```

Συμπεράσματα

Η εφαρμογή η οποία παρουσιάστηκε μπορεί να χρησιμεύσει στον χρήστη να κρατά ένα ιστορικό μετρήσεων, οι οποίες αντλούνται από το συγκεκριμένο απομακρυσμένο API. Ο χρήστης έχει την δυνατότητα να ανατρέχει σε αυτά τα δεδομένα, τα οποία αφορούν ατμοσφαιρικές μετρήσεις και να διαγράφει, ενημερώνει ή να εισάγει νέα δεδομένα.

Αντίστοιχες εφαρμογές θα μπορούσαν να χρησιμοποιηθούν και για άλλου είδους Web APIs τα οποία μπορεί να αφορούν άλλου είδους δεδομένα εκτός από ατμοσφαιρικές μετρήσεις. Για παράδειγμα , δεδομένα που αφορούν αθλητικούς αγώνες, δεδομένα που αφορούν το χρηματιστήριο, δεδομένα που αφορούν το διάστημα(θέσεις αστεροειδών, πλανητών κλπ), δεδομένα που αφορούν ισοτιμίες νομισμάτων κ.α.

Με την βοήθεια του Spring Boot framework και των άλλων εργαλείων τα οποία χρησιμοποιήθηκαν όπως Hibernate ORM, Jackson, Maven, Thymeleaf δημιουργήθηκε μια εφαρμογή σύγχρονη, εύκολα συντηρήσιμη, χωρίς επαναλαμβανόμενο κώδικα και εύκολα κατανοήσιμη.

Βιβλιογραφία

Udemy Course : Spring & Hibernate for Beginners (includes Spring Boot), Chad Darby

Java The Complete Reference Eleventh Edition, Herbert Schildt

https://springframework.guru/

https://www.baeldung.com/

https://mkyong.com/

https://api-docs.airvisual.com/

https://www.oracletutorial.com/

https://www.javaguides.net/

https://docs.spring.io/spring-framework/docs/

https://en.wikipedia.org/