Εργασία στο μάθημα ΑΣΦΑΛΕΙΑ ΠΛΗΡΟΦΟΡΙΑΚΏΝ ΣΥΣΤΗΜΑΤΩΝ (TMD108)

Α.Μ: Π12150

Ανάλυση

Για να μπορέσω να υλοποιήσω το θέμα εργασίας που πρότεινα, διαχώρισα τα δεδομένα σε λειτουργικές οντότητες. Κάθε μία από αυτές είναι μοναδική και συγκρατεί ένα μέρος πληροφορίας αναγκαίο για την εκτέλεση διάφορων λειτουργιών. Συνεπώς, ήταν αυτονόητος ο λόγος που πρώτα σχεδιάστηκαν αυτές οι οντότητες και τα UML διαγράμματα τους και μετέπειτα η βάση με τα αντίστοιχα table τους. Αυτού του είδους τα μοντέλα μεταφράζονται σαν PoJos ή Domain Controllers.

Οι οντότητες αυτές αλληλεπιδρούν μεταξύ τους μέσω άμμεσα είτε έμμεσα και πολλές φορές σε αυτές τις αλληλεπιδράσεις λαμβάνουν μέρος παραπάνω από δύο οντότητες. Αυτό το κομμάτι της ανάλυσης, οι “αλληλεπιδράσεις” - ενέργειες, είναι το Business Logic το οποίο υλοποιείται ξεχωριστά στην εφαρμογή μας και παρουσιάζεται στο χρήστη μέσω διάφορων Views Controller.

Εγκατάσταση

Για να μπορέσει να εγκαταστήσει κάποιος την εφαρμογή μας θα πρέπει να βεβαιωθεί πρώτα πως ο host έχει κάποιες συγκεκριμένες βιβλιοθήκες ώστε να μπορούν να υποστηρικτούν όλες οι λειτουργίες αλλά και τουλάχιστον έναν Application Container και μία βάση δεδομένων.

Συγκεκριμένα, για να μπορέσουμε να εγκαταστήσουμε την εφαρμογή μας θα πρέπει να έχουμε βεβαιωθεί πως υπάρχεί:

- η Java 8.X.X και συγκεκριμένα τα πακέτα της Oracle
- ένας Apache Tomcat και συγκεκριμένα έκδοση >= 7
- ένας Mysql ή MariaDb Server

Στην αρχή δημιουργούμε ένα νέο schema με όνομα Exabit και μέσω της mysql κάνουμε import το Exabit.sql.Το αρχείο αυτό περιέχει όλα τα Tables που έχουν αντιστοιχιστεί με Java κλάσεις της εφαρμογής μας και αποτελούν τους Domain Controllers μας.. Στην συνέχεια ελέγχουμε εάν έγινε σωστά η εισαγωγή των tables και εισάγουμε δοκιμαστικά κάποιους χρήστες.

Στην συνέχεια, χρησιμοποιώντας ένα IDE ανοίγουμε το Project και το κάνουμε Clean και Rebuild. Το .war αρχείο που θα δημιουργηθεί, περιέχει όλο το κώδικα μας, από τις Java κλάσεις που λειτουργούν ως Configuration, τα .jsp μέχρι και τις βιβλιοθήκες του Spring και είναι όλα ενσωματωμένα στο .war μας αρχείο.

Τέλος, αφού κάνουμε Build το Project επιτυχώς ήρθε η ώρα να το κάνουμε Deploy. Σταματάμε τον Apache Container χρησιμοποιώντας την κατάλληλη εντολή και αντιγράφουμε το .war αρχείο στον φάκελο /webapps στο Path στο οποίο βρίσκεται εγκατεστημένος ο Apache Container μας. Όταν ξεκινήσουμε τον Apache ξανά, θα δούμε ότι θα γίνει αυτόματα Deploy το .war μας. Η εφαρμογή, εφόσον γίνει επιτυχημένα deploy, θα είναι διαθέσιμη στο χρήστη στο url → https://localhost:8443/Exabit

Βιβλιοθήκες

Για να μπορέσει να υλοποιηθεί η εφαρμογή χρησιμοποιήθηκαν οι εξής βιβλιοθήκες:

org.hibernate – hibernate-core

Η συγκεκριμένη βιβλιοθήκη είναι εκείνη η οποία υλοποιεί το Java Persistence API. Συγκεκριμένα, όπως ανέφερα και στην ανάλυση της εφαρμογής, επειδή οι οντότητες μου αντιστοιχίζονται με διάφορα tables στην βάση δεδομένων μου, μέσω του hibernate μπορώ να κάνω την αντιστοίχιση κώδικα Java στον αντίστοιχο της βάσης που χρησιμοποιώ.

org.hibernate hibernate-entitymanager

Η συγκεκριμένη βιβλιοθήκη μας βοηθά στην διαχείριση των persistent οντοτήτων (Αυτές που έχουν οριστεί με βάση την βιβλιοθήκη hibernate-core). Μας δίνει την δυνατότητα των βασικών CRUD λειτουργιών αλλά και τα μέσα ώστε να επεκτείνουμε και να γράψουμε τις δικές μας.

org.springframework – spring-core

Βιβλιοθήκη με τις βασικές λειτουργίες του Spring, πάνω στην οποία βασίζονται όλα τα άλλα πακέτα. Υποχρεωτικά παρούσα σε κάθε web εφαρμογή που αναπτύσεται σε Spring.

org.springframework – spring-webmvc

Επίσης μία από τις βασικές βιβλιοθήκες του Spring. Αναγκαία ώστε να μπορέσουμε να έχουμε μία ιεράρχηση στις σελίδες του ιστότοπου μας. Μας δίνει τα κατάλληλα εργαλεία ώστε να αναπτύξουμε βάση της αρχιτεκτονικής MVC, αλλά και ρυθμίσουμε την βάση του server μας (Dispatcher) ώστε να κάνει κατανομή των request βάση των Annotation που έχουμε εμείς ορίσει.

Από αυτή την βιβλιοθήκη, εκμεταλλευτήκαμε την δυνατότητα που έχει να κάνουμε configure τον server μας, όχι μέσω του web.xml αλλά πλέον μέσω Java κώδικα, ώστε να έχουμε μεγαλύτερη ευελιξία και έλεγχο στην εφαρμογή μας. Χρησιμοποιήσαμε λοιπόν έντονα τα Annotations @Controller, @RequestMapping(value = "?", method = ?) ώστε να υλοποιήσουμε την πλοήγηση μας στον ιστότοπο βάση των σελίδων που είχαμε υλοποιήσει σε .jsp.

org.springframework.data - spring-data-jpa

Μία από τις βασικές βιβλιοθήκες στην ανάπτυξη αυτού της εφαρμογής. Έχοντας ορίσει πιο πάνω την έννοια των persistence object (οντοτήτων) και γνωρίζοντας πως αυτά αλληλεπιδρούν μεταξύ τους, είναι φυσικό πως θα υπήρχε το ερώτημα πως διαχειριζόμαστε και αξιοποιούμε αυτά τα δεδομένα. Εδώ έρχεται η έννοια του DAL. Η βιβλιοθήκη Spring Data Jpa, είναι μία από τις πολλές της οικογένειας Spring Data, που σκοπό έχει μέσω των “Repositories” να μας δώσει πρόσβαση σε διάφορες λειτουργίες των οντοτήτων (Ανάκτηση, Αποθήκευση, Διαγραφή).

org.springframework.security – spring-security-config

Η βασική βιβλιοθήκη μέσω της οποίας ρυθμίζουμε τις πολιτικές ασφαλείας αλλά και οτιδήποτε έχει σχέση με την ασφάλεια της εφαρμογής μας. Από την εφαρμογή permission σε συγκεκριμένες σελίδες (Authorization) μέχρι τον τρόπο που επιτυγχάνεται η ταυτοποίηση του χρήστη στο σύστημα (Authentication), η βιβλιοθήκη αυτή μας δίνει έναν έξυπνο τρόπο ώστε μέσω κώδικα Java να φτιάξουμε το δικό μας Configuration. Καλύπτει ένα μεγάλο εύρος από την ασφάλεια σε Http επίπεδο μέχρι την ασφάλεια εσωτερικά στον server μας.

org.springframework.security – spring-security-web

Η συγκεκριμένη βιβλιοθήκη προσφέρει κάποια εξιδεικευμένα στοιχεία προς configuration που αφορούν την ασφάλεια σε Http επίπεδο και τα οποία δεν εμπεριέχονται στην spring-security-config.

org.springframework.security - spring-security-taglibs

Η συγκεκριμένη βιβλιοθήκη αφορά κυρίως τα .jsp και συγκεκριμένα διάφορα tags που μπορούν να χρησιμοποιηθούν, ώστε βάση των δικαιωμάτων αλλά και του τύπου χρήστη να είναι ορατά/αόρατα διάφορα κομμάτια html.

org.springframework.security – spring-security-ldap

Η βιβλιοθήκη αυτή είναι υπεύθυνη για το Authentication των χρηστών μέσω ενός LDAP. Συγκεγκριμένα, μέσω της βιβλιοθήκης αυτής μπορεί κάποιος να κάνει configure όλα τα απαραίτητα στοιχεία ώστε να κάνουν οι χρήστες της εφαρμογής authenticate σε έναν LDAP server.

Mysql – mysql-connector-java

Η βιβλιοθήκη αυτή επιτρέπει την σύνδεση σε μία βάση δεδομένων τύπου Mysql μέσω της γλώσσας προγραμματισμού Java.

javax.servlet – javax.servlet-api

Η βιβλιοθήκη αυτή μας επιτρέπει να διαχειριστούμε τα request σε επίπεδο Servlet. Την είχα επιλέξει διότι σκεπτόμουν να υλοποιήσω uploading αρχείων και την χρειαζόμουν αλλά τελικά το απέρριψα. Την αναφέρω απλά για μελλοντική χρήση.

Commons-fileupload – commons-fileupload

Η βιβλιοθήκη αυτή είναι χρήσιμη στην διαχείριση αρχείων που έχουν γίνει upload για χρήστες. Δεν την χρησιμοποιήσα τελικά θα μπορούσα να έχω διαγράψει και το dependency της.

com.zaxxer – HikariCP

Η βιβλιοθήκη αυτή ουσιαστικά διαχειρίζεται τις συνδέσεις που κάνει η εφαρμογή μας στην βάση για να εκτελέσει τα διάφορα query. Είναι τύπου memory pool (ίδιου τύπου με την native του Tomcat) απλά φημίζεται για το perfomance της.

com.fasterxml.jackson.core – jackson-core

Η βιβλιοθήκη αυτή ουσιαστικά έχει την δυνατότητα να κάνει Serialize και Deserialize πολύπλοκες κλάσεις. Θα μπορούσα να χρησιμοποιήσω και κάτι τύπου Guava, αλλά την σύστηναν commercial sites. Ουσιαστικά ρόλος της είναι να κάνει convert το body των Request/Responses όπου χρειάζεται σε JSON.

com.fasterxml.jackson.core – jackson-databind

Η βιβλιοθήκη αυτή είναι συμπληρωματίκη του core, ουσιαστικά προσφέρει αντιστοίχιση των βασικών Objects της Java σε αυτά ενός JSON και vice verca.

com.fasterxml.jackson.datatype – jackson-datatype-jsr310

Η βιβλιοθήκη αυτή είναι βοηθητική στην δήλωση Complex κλάσεων βάση (Συγκεκριμένα κάποιες από τις νέες κλάσεις της Java 8).

com.googlecode.owasp-java-html-sanitizer – owasp-java-html-sanitizer

Η βιβλιοθήκη αυτή έχεις ως σκοπό να κάνει sanitize τα διάφορα inputs που στέλνονται πίσω μέσω των Restfull Services. Παρόλο που είναι αρκετά εύκολη στο configuration του Policy της, με τα PGP κλειδικά είχα προβλήματα.
(Χρησιμοποιήσα σε client side την DomPurify, της Cure53 ώστε να κάνω Sanitize τα input.)

Documentation κώδικα.

main.java.com.exabit.Entities [1]

Σε αυτό εδώ το πακέτο ορίζω τις οντότητες μου, οι οποίες αντιστοιχίζονται σε tables στην βάση μου. Χρησιμοποιώντας τα annotations @Entity και @Table(name = "X", schema = "Y", catalog = ""), δηλώνω ότι η συγκεκριμένη Java κλάση θα αντιμετωπιστεί σαν μία οντότητα η οποία αντιστοιχίζεται σε ένα X Table στην βάση με schema Y. Έπειτα, επειδή όλα τα table έχουν σχεδιαστεί με ένα PK τύπου Int το οποίο είναι AI, αναλογικά και οι κλάσεις εμπεριέχουν μία μεταβλητή τύπου int (θα μπορούσε να ήταν long εάν είχα επιλέξει στην βάση άλλο τύπο πεδίου) και τα annotation @Id[2] και @GeneratedValue[2]. Αυτά κάθε φορά που θα γίνετια δημιουργία μίας νέας εγγραφής στο table που έχει αντιστοιχιστεί, θα γίνεται generated το id αυτόματα χωρίς να χρειάζεται να επεμβούμε εμείς.

Σε περίπτωση που υπήρχε Authentication ενός χρήστη με έναν Ldap και authorization μέσω βάσης, μπορούσαμε να ρυθμίσουμε τον τρόπο που θα γινόταν Generated το αντίστοιχο value (Θα πρέπει να αλλάξουμε και τον τύπο πεδίου που θα ήταν και PK) έτσι ώστε να ταίριαζε και με αυτό με το οποίο αναγνωρίζεται ο χρήστης στον LDAP.

Τέλος, κάθε μεταβλητή της κλάσης μας αντιστοιχίζεται με ένα πεδίο του table μέσω των @Basic και @Column()[3] annotation. Στο τελευταίο μπορούμε να εισάγουμε και τις ιδιότητες κάθε πεδίου (πχ not null, εάν είναι τύπου varchar το επιτρεπόμενο length ενός string πριν την εισαγωγή του κτλπ).

main.java.com.exabit.Repositories

Σε αυτό εδώ το πακέτο, δηλώνουμε τα Repositories μας βάση των οντοτήτων που δηλώθηκαν στο main.java.com.exabit.Entities. Αυτό επιτυγχάνεται κάνοντας extend το Interface JpaRepository[4] και δηλώνοντας την οντότητα στην οποία θα εφαρμοστούν οι λειτουργίες αυτές και τον τύπο του PK τους. Το συγκεκριμένο Repository μας προσφέρει ότι λειτουργίες προσφέρει και το Crud Repository[5] σύν κάποια Sort και Paging query.Παρόλο που στις πιο πολλές περιπτώσεις η χρήση της γίνεται για τους τελευταίους δύο λόγους, εμείς την χρησιμοποιούμε και για την δυνατότητα να δηλώσουμε τα δικά μας custom queries.

Συγκεκριμένα, από το Interface JpaRepository και όσα άλλα το κάνουν extend, ο χρήστης μπορεί με την χρήση του @Query annotation[7], να κάνει αντιστοίχιση ενός custom query σε μία συνάρτηση που εκείνος επιθυμεί.Για παράδειγμα ας εξετάσουμε το εξής Query :

```
@Query("SELECT p FROM FriendsEntity p WHERE ( p.fromUserId = :fromUserId AND
p.toUserId = :toUserId ) OR ( p.fromUserId = :toUserId AND p.toUserId = :fromUserId )")
   FriendsEntity IsFriendShipExists(@Param("fromUserId") int fromUserId,
@Param("toUserId") int toUserId );
```

Για να γράψουμε ένα Jpa Custom Query το μόνο που έχουμε να κάνουμε είναι να γράψουμε το query μας όπως θα το γράφαμε στην αντίστοιχη γλώσσα της βάσης μας απλά έχοντας υπόψην μας τους εξής περιορισμούς :

Στην μεταβλητή From, αντί για το παραδοσιακό table, βάζουμε την οντότητα με την οποία έχει αντιστοιχιστεί. Έτσι στην συγκεκριμένη περίπτωση έχουμε την FriendsEntity που περιέχει τις μεταβλητές toUserId και fromUserId. Αυτές χρησινοποιούνται και στο Where κομμάτι όπου στην θέση των πιθανών τιμών, βάζουμε τις “ψευτομεταβλητές” :fromUserId και :toUserId. Το όνομα δεν παίζει τόσο ρόλο, θα μπορούσαμε να είχαμε βάλει :X και :Y, απλά για εννοιλογικούς ρόλους τις αναθέσαμε έτσι.
Στην συνέχεια δηλώνουμε το όνομα της συνάρτησης που στο κάλεσμα της θα εκτελεστεί το προαναφερθέν query και τον τύπο οντότητας που θα επιστρέφει (Εξαρτάται από το τι δηλώσαμε στο Query μας).Τέλος, αντιστοιχίζουμε τα ορίσματα της συνάρτησης μας με τις “ψευτομεταβλητές” του query μας, χρησιμοποιώντας το @Param annotation.

Αλλά η δύναμη του JpaRepository δεν φθάνει μόνο εκεί. Μπορούν να κατασκευαστούν query πάνω σε συγκεκριμένους πίνακες χρησιμοποιώντας keywords και syntax που προσφέρεται από την spring data βιβλιοθήκη. Εκείνη μέσω ενός Matcher που χρησιμοποιεί εσωτερικά, τα επαληθεύει και μέσω των keywords που έχουν χρησιμοποιηθεί δημιουργούνται τα αντίστοιχα query των βάσεων.

Έτσι για παράδειγμα, εάν πάρουμε το FriendsEntityRepository θα δούμε ότι έχει δηλωθεί μία συνάρτηση List<FriendsEntity> findByfromUserId(int fromUserId); Το Spring για να κατασκευάσει το Query[8], αυτο που θα κάνει είναι να δει ποια keywords από αυτά που προσφέρει έχουν χρησιμοποιηθεί (εδώ κάνουμε χρήση του findBy) και να σπάσει το κείμενο με βάση αυτά τα keywords[9]. Έπειτα, για κάθε λέξη ανάμεσα στα keywords, την θεωρεί είτε μεταβλητή μέσα στην οντότητα η οποία κάνει extend αυτό το Interface και την ψάχνει μέσω Reflection, είτε την θεωρεί ένα μία άλλη οντότητα και ψάχνει το όνομα της αντίστοιχης κλάσης.

Σε περίπτωση που το query έχει οριστεί σωστά από συντακτική πλευρά τότε γίνεται επαλήθευση των παραμέτρων της συνάρτησης, εάν συντακτικα είναι σωστά δηλωμένες και εάν το πλήθος τους ταιρίαζει με το πλήθος των παραμέτρων που ταυτοποιήθηκαν στην κατασκευή του query. Για όλα αυτά, το Spring έχει αρκετά επεξηγηματικά Exceptions που διευκολύνουν αρκετά την ζωή του προγραμματιστή.

main.java.com.exabit.Services;

Σε αυτό το πακέτο δηλώνονται όλες οι συναρτήσεις, οι οποίες υλοποιούν κομμάτι του Business logic. Κάθε οντότητα έχει από ένα Interface και την αντίστοιχη κλάση που υλοποιεί το Interface αυτό[10], χρησιμοποιώντας σαν Data Access Layer τα repositories που ορίσαμε πριν.Τα repositories, γίνοναται Inject με το annotation @Resource, μιας και είναι μοναδικά και δεν χρειάζεται το Spring να ψάξει να τα βρει σε multiple πακέτα. Έτσι επιτυγχάνουμε μία ελευθερία στον τρόπο που υλοποιούμε τις αλληλεπιδράσεις των οντοτήτων μεταξύ τους, αλλά και τα δομικά θεμέλια ώστε να επιτευχθούν αυτές.

Ας πάρουμε για παράδειγμα το Interface FriendsEntityService. Αυτό κάνει extend το Generic Service (το οποίο το έχω φτιάξει εγώ και περιέχει τις βασικές συναρτήσεις για CRUD λειτουργίες) αλλά και το Interaction Service το οποίο έχει συναρτήσεις που είναι αναγκαίες στην υλοποίηση του της λογικής της ανταλλαγής μηνυμάτων των χρηστών. Υπάρχει η δυνατότητα να δηλωθούν και άλλες συναρτήσεις και έτσι δηλώνουμε και την FriendsEntity IsUserFriendWith(int userId_1, int userId_2);

Στην συνέχεια, στο FriendsEntityServiceImpl γίνεται η υλοποίηση των συναρτήσεων που δηλώθηκαν στο FriendsEntityService interface. Ειδική προσοχή θέλει η χρήση των Annotation @Service και @Transactional. Το πρώτο δηλώνει ότι αυτά που υλοποιούνται είναι το Service Layer και θα μπορέσουμε μετά να τα κάνουμε Inject στους Controllers μας και το δεύτερο έχει να κάνει με την εκτέλεση των queries από το repository. Όπως ξέρουμε, για να εκτελεστεί ένα query πρέπει να γίνουν ορισμένα βήματα όπως να δηλώσουμε ότι ξεκινάμε ένα Transaction, μετέπειτα να εκτελέσουμε κάποιον κώδικα και στην συνέχεια να κάνουμε commit τις αλλαγές μας. Σε περίπτωση κάποιου λάθους, να μπορούμε να κάνουμε rollback στην βάση σε προηγούμενη κατάσταση.

Επειδή όμως εμείς κάνουμε χρήση του Hibernate ORM και του entitymanager, δεν διαχειριζόμαστε τα session χειροκίνητα αλλά αφήνουμε στον Manager να τα διαχειρίζεται εκείνος. Χρησιμοποιώντας λοιπόν το annotation αυτό, ενημερώνουμε το Spring ότι το Service Layer κάνει χρήση του Data Access Layer για να υλοποιήσει το Business Logic (και ως συνέπεια έχει αλληλεπίδραση με την βάση) και τον αναγκάζουμε να διαχειριστεί αυτός τα διάφορα Sessions και Transactions states σε περίπτωση αποτυχίας, επιτυχίας κτλπ.

main.java.com.exabit.RestControllers;

Σε αυτό εδώ το πακέτο υλοποιούμε τις διευθύνσεις στις οποίες θα είναι διαθέσιμες οι CRUD[9] Λειτουργίες απαραίτητα και όσες κρίνουμε ότι πρέπει να έχουν πρόσβαση οι χρήστες. Ένας λόγος που επιλέχθηκε αυτή η αρχιτεκτονική σχεδίασης και όχι η απευθείας ανάθεση των repositories στους χρήστες, είναι διότι υπάρχουν κάποια κομμάτια της Business λογικής που θέλουμε να κρατηθούν κρυφά από το χρήστη. Χρησιμοποιώντας λοιπόν Services και RestControllers ώστε να τα “εκθέσουμε” στον χρήστη (Δηλαδή να τα κάνουμε προσβάσιμα σε αυτόν), επιλέγουμε τι κομμάτι της λογικής μας θα είναι διαθέσιμο στο χρήστη.

Εξαρχής, για να δηλώσουμε ότι μία κλάση αποτελεί έναν RestController, χρησιμοποιούμε τα annotation @RestController και @RequestMapping("Το url που επιθυμούμε").Το πρώτο έχει την ίδια λειτουργικότητα όπως και το @Controller annotation απλά ενσωματώνει και το @ResponseBody annotation. Το δεύτερο είναι το γνωστό annotation, το οποιό δηλώνει το url στο οποίο θα διατίθονται τα request στον χρήστη.

Στην συνέχεια βλέπουμε την υλοποίηση των CRUD Requests και τις αντιστοιχίσεις τους με βάση το url που δηλώθηκε στην αρχή της κλάσης. Ιδιαίτερα ενδιαφέρον έχουν τα annotation @RequestMapping(method = RequestMethod.X), @ResponseStatus(HttpStatus.Y) και @PathVariable("id"). Το πρώτο δηλώνει τον μοναδικό τύπο μεθόδου HTTP στο οποίο αντιστοιχίζεται η συνάρτηση που χρησιμοποιει το annotation αυτό. Το δεύτερο είναι ένα παράδειγμα Status Code που θα επιστραφεί σε περίπτωση επιτυχημένης εκτέλεσης του Htto Request. Το τελευταίο, είναι συνδιαστικό με το @RequestMapping και χρησιμοποιείται για να εξαχθεί μία μεταβλητή από το url στο οποίο έχει αντιστοιχηθεί το Http Request. (πχ /api/request/{id} με id = 3 → τοτε api/request/3 και το @PathVariable θα εξάγει το 3 και θα το τοποθετήσει στην μεταβλητή id.).

Τέλος ένα annotation πάρα πολύ σημαντικό είναι το @RequestBody. Αυτό δηλώνει τι θα βρίσκεται στο body κάθε Http Request και επειδή μιλάμε για Restful Services, μιλάμε για τα αντίστοιχα Entities πάνω στα οποία εκτελούνται όλες οι αλληλεπιδράσεις. Σε παλαιότερες εκδόσεις, συνοδευόταν από το annotation @Valid. Ευτυχώς, το Spring κάνει μόνο του το Validation πλέον.

main.java.com.exabit.RestOtherControllers;

Στο συγκεκριμένο πακέτο υλοποιούνται και πάλι κάποια Restful Services με την ίδια λογική και Annotations όπως και στο main.java.com.exabit.RestControllers. Τα διαχώρισα ενώ θα μπορούσα να τα έχω μαζί λόγω Abstraction και γιατί ήθελα να έχω ξεχωριστά τα CRUD endpoints από αυτά με την καθαρά Business λογική. Η χρήση μεθοδολογίας είναι ακριβώς ίδια με μόνη διαφορά την έντονη χρήση του @AuthenticationPrincipal στην οποία θα αναφερθώ λεπτομερώς αργότερα αλλά και του @RequestParam αντί του @RequestBody(μιας και δεν χρησιμοποιούμε complex objects).

main.java.com.exabit.Controllers

Στο πακέτο αυτό υλοποιούμε το navigation της εφαρμογής μας, δηλαδή την αντιστοίχιση των urls με τα υλοποιημένα jsp. Εδώ γίνεται χρήση του @Controller annotation, μιας και σαν response δεν στέλνουμε δεδομένα αλλά ενημερώνουμε το server μας σε ποια σελίδα να ανακατευθύνει το χρήστη.Απαραίτητο συστατικό για την λειτουργία αυτή είναι και οι ViewResolvers οι οποίοι υλοποιούνται στο main.java.com.exabit.Config και επιτρέπουν την αντιστοίχιση αυτή.

main.java.com.exabit.Utils

Στο συγκεκριμένο πακέτο γίνεται υλοποίηση κάποιων βασικών Utils όπως HTML sanitization κτλπ.

main.java.com.exabit.Config.Security

Στο πακέτο αυτό υλοποιούνται κάποιες από τις βασικότερες ρυθμίσες του server μας όσο αφορά την βασική λειτουργικότητα του στην διαχείριση των HTTP Request, στον τρόπο που διαχειρίζεται το Authentication αλλά και το Authorization. Συγκεκριμένα εδώ γίνεται χρήση των περισσότερων βιβλιοθηκών του Spring όσο αφορά το security (είτε web είτε webmvc).

SecurityConfig

Στην συγκεκριμένη κλάση υλοποιούνται οι κανόνες περί του Authentication και Authorization. Αυτό επιτυγχάνεται κάνοντας override ή προσθέτοντας κάποιες νέες συναρτήσεις και δηλώνοντας τες σαν components.

Συγκεκριμένα, με την configureGlobal(AuthenticationManagerBuilder auth) υλοποιούμε τον τρόπο με τον οποίο θα γίνονται Authenticate οι χρήστες στην εφαρμογή μας. Μπορούμε να έχουμε παραπάνω από έναν τρόπους. Για παράδειγμα, εδώ χρησιμοποιείται ένα Service που υλοποιήθηκε από εμάς. Για την ταυτοποίηση έχουν χρησιμοποιηθεί τα defaults credentials username/password που θα στέλνει η φόρμα login που έχουμε ορίσει. Για τους κωδικούς, από την στιγμή που αποθηκεύονται σαν bcrypt hashes, χρησιμοποιούμε και έναν Bcrypt Encoder ώστε να γίνεται σωστή σύγκριση των credentials. Θα μπορούσαμε να έχουμε πολλαπλούς τρόπους ταυτοποίησης χρηστών, όπως πχ και έναν LDAP authentication Manager είτε έναν επάνω στην μνήμη (InMemoryUser).

Με την χρήση της configure(WebSecurity web) θέτουμε τους κανόνες στην διαχείριση των resources του server (εδώ θα βάζαμε ως εξαίρεση σύμφωνα με την δική μας λογική, όλα τα αρχεία – resources που έχουν πρόσβαση όλοι οι χρήστες, εγγεγραμμένοι και μη για τις public σελίδες μας, πχ login,index κτλπ).

Με την χρήση της configure(HttpSecurity http) ουσιαστικά υλοποιούμε το περισσότερο σκέλος όσο αφορά την ασφάλεια των Http Request αλλά και τα δικαίωματα κάθε χρήστη στα διάφορα API/σελίδες μας. Αναλυτικά έχουμε τα εξής :

http

.authorizeRequests()

Σε όλα τα request θα πρέπει να ληφθούν υπόψην τα δικαιώματα και οι ρόλοι των χρηστών

.antMatchers(HttpMethod.POST, "/api/**").authenticated()
.antMatchers(HttpMethod.PUT, "/api/**").authenticated()
.antMatchers(HttpMethod.DELETE, "/api/**").authenticated()
.antMatchers(HttpMethod.GET, "/api/**").authenticated()

Εδώ, επειδή τα Restful Services παίζουν επιτηδευμένα όλα στο url /api/Ονομα Οντότητας ή Custom Συνάρτησης, βάζοντας τους διπλούς αστερίσκους καταφέρνουμε να αναγκάσουμε το Spring, κάθε φορά που βλέπει ένα Request τύπου POST,PUT,DELETE,GET σε ένα url τύπου /api/ονομα service, να τσεκάρει εάν ο χρήστης είναι ταυτοποιημένος από το σύστημα αλλιώς να μην του επιτρέπει την πρόσβαση.

.anyRequest().permitAll()
Ξανά, σε καθένα από τα παραπάνω Request έχουν πρόσβαση όλοι ανεξαρτήτως ρόλων. Σε περίπτωση που είχα ένα url τύπου “/admin” και ήθελα να έχει πρόσβαση MONO οι χρήστες που έχουν ταυτοποιηθεί σαν Admin, θα έβαζα αντί του permitAll το hasRole(“ADMIN”).

.antMatchers("/resources/**").permitAll()

Όλοι ανεξαρτήτως δικαιωμάτων και ρόλων έχουν πρόσβαση στα Resources
.anyRequest().authenticated()
Κάθε Request όμως στα Resources πρέπει να είναι από ταυτοποιημένους χρήστες (Για αυτό και πιο πριν είχαμε αναφέραμε ότι για κάποιες public σελίδες, που τις βλέπουν απλοί επισκέπτες, θα πρέπει να μπορεί το Spring να δώσει πρόσβαση σε συγκεκριμένα resources)

.and()
.formLogin()
.loginPage("/login").defaultSuccessUrl("/home")
.permitAll()

Εδώ ορίζουμε το url στο οποίο θα βρίσκεται το login μας (μιας και το Mapping που αναφέραμε πριν στο Controller πακέτο προστατεύεται από τον κανόνα authorizeRequests()) αλλά και την σελίδα που θα μεταφέρεται ο χρήστης σε περίπτωση επιτυχημένης ταυτοποίησης.

.and()
.logout()
.permitAll()

Λόγω τυπικότητας ξανά, αναφέρουμε ότι κάθε χρήστης ανεξαρτήτου ρόλου μπορεί να κάνει logout

and()
.headers()
.httpStrictTransportSecurity()
.and()
.xssProtection()
.and()
.frameOptions()
.and()
.addHeaderWriter(new StaticHeadersWriter("X-Content-Security-Policy","default-src 'self'"))
.addHeaderWriter(new StaticHeadersWriter("X-WebKit-CSP","default-src 'self'"))
.and()

Εδώ ορίζουμε πως σε κάθε τύπου Http Request, θα πρέπει να προστεθούν οι παραπάνω Headers ώστε να αποφύγουμε επιθέσεις τύπου XSS, Session Hijacking κτλπ, Strict Policy στον τρόπο με τον οποίο γίνονται ρε Request κτλπ. Τα πιο πολλά λόγω ονόματος είναι κατανοητά από την χρήστη και πλήρως τεκμηριωμένα στο Spring.

requiresChannel().anyRequest().requiresSecure();

Τέλος, ορίζουμε πως για κάθε Http Request θα πρέπει να γίνεται μέσω HTTPS και πότε μέσω HTTP. Ουσιαστικά μαζί με το httpStrictTransportSecurity αναγκάζουμε τον server μας να διαχειρίζεται όλα τα Request μέσω Https και να μην επιτρέπει ποτέ μέσω http.

CustomUserDetailsService

Είναι το Service με το οποίο επιτυγχάνουμε την ταυτοποίηση του χρήστη. Στην περίπτωση μου, χρησιμοποιώντας τα αντίστοιχα repository, αναζητώ τον χρήστη στην βάση μου βάσει του username του και δημιουργώ ένα νέο CustomUserEntity (Το οποίο το έχω υλοποιήσει εγώ, κάνοντας extend το original ώστε να κρατά επιπλέον πληροφορία). Ο λόγος που το Spring απαιτεί το Auth να γίνεται με τέτοια Service είναι διότι έτσι δημιουργείται ένας νέος χρήστης που περιέχει και τον κωδικό ώστε στην συνέχεια να κάνει την ταυτοποίηση αλλά και τους ρόλους σε περίπτωση που αυτή είναι επιτυχημένη.

WebMvcContext

Σε αυτή εδώ την κλάση γίνονται οι περισσότερες ρυθμίσεις όσο αφορά την MVC αρχιτεκτονική. Όπως μαρτυρά και το όνομα, το περιεχόμενο κάθε Request, ο τρόπος που διαχειριζόμαστε τα Views όλα ρυθμίζονται σε αυτή εδώ την κλάση. Για αυτό άλλωστε, είναι αναγκαίο να χρησιμοποιήσουμε τα Annotation @Configuration @EnableWebMvc.

configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer)

Συνάρτηση με την οποία αναθέτουμε στον server μας τον DefaultHandler των Request.

configureMessageConverters(List<HttpMessageConverter<?>> converters)

Αναγκαία συνάρτηση σε περίπτωση που έχουμε Restful Services. Σε αυτή εδώ θα δηλώσουμε τον τρόπο και τον τύπο δεδομένων που θα έχουν στο Body τους τα διάφορα Request/Responses. Στην συγκεκριμένη περίπτωση, επιλέξαμε JSON, συνεπώς χρησιμοποιούμε τον Jackson ώστε να κάνει Serialize τις οντότητες μας σε JSON και deserialize τα διάφορα JSON που στέλνουμε στις οντότητες μας. Θα μπορούσαμε να έχουμε περισσότερους από έναν converters (πχ και XML) αλλά το θεωρήσαμε overkill.

addViewControllers(ViewControllerRegistry registry)

Στην συνάρτηση αυτή, θα δηλώσουμε όλες τις σελίδες οι οποίες είναι προσβάσιμες από μη εγγεγραμμένους χρήστες (Κάνουμε μία παραδοχή εδώ, ότι μη εγγεγραμμένος χρήστης == χρήστης και χωρίς ρόλους). Επειδή όπως αναφερθήκαμε πριν, όλα τα HTTP request πρέπει να είναι από Authorized και Authenticated χρήστες, εδώ ορίζουμε όλες τις σελίδες που θέλουμε να είναι από όλους προσβάσιμες.(Πχ login, index κτλπ). Επίσης εδώ μπορούμε να δηλώσουμε και Mappings σε σελίδες που δεν είναι jsp αλλά Html.

addResourceHandlers(ResourceHandlerRegistry registry)

Η συνάρτηση αυτή ορίζει πως γίνεται το mapping των resources στον server(Κύρια χρήση της είναι να διευκολύνει το include των resources στα .jsp).

configureViewResolvers(ViewResolverRegistry registry)

Μία από τις σημαντικότερες συναρτήσεις. Σε αυτή, ουσιαστικά υποχρεώνουμε το Spring να κάνει mapping όλα τα strings που επιστρέφουμε από τους Controllers σε αντιστοίχιση με τα jsp που βρίσκονται στο φάκελο /WEB-INF/jsp/ και να ανακατευθύνει εκεί τον χρήστη. Λόγω αυτής της συνάρτησης ουσιαστικά ο Web Controller μας και κατά συνέπεια το μενού μας έχει υλοποιηθεί τόσο γρήγορα και εύκολα.

PersistenceConfig

Σε αυτή εδώ την κλάση γίνεται το configuration της βάσης, της αντιστοίχισης που θα κάνει το Hibernate του κώδικα Java με αυτού της βάσης αλλά και λοιπές ρυθμίσεις στις συνδέσεις με την βάση. Επειδή όμως ακριβώς λειτουργεί σαν Configuration, γίνεται χρήση του @Configuration. Το @EnableTransactionManagement δηλώνεται λόγω της τελευταίας συνάρτησης.

DataSource dataSource(Environment env)

Σε αυτήν την συνάρτηση ορίζουμε τα απαραίτητα δεδομένα ώστε να καταφέρει το Spring να δημιουργήσει μία νέα σύνδεση με την βάση μας και να τρέξει τα Query μας. Τα περισσότερα properties είναι αυτονόητα. Ίσως προκαλέσει εντύπωση η χρήση του Hikari αντί του native του Tomcat (μιας και Tomcat έχουμε για Application Container). Αυτό έγινε λόγω περιέργειας μιας και η βιβλιοθήκη φημίζεται για το perfomance της και μιας και εμείς έχουμε restful services και CRUD λειτουργίες, θεώρησα πως άξιζε η χρήση της.

LocalContainerEntityManagerFactoryBean entityManagerFactory

Στην συνάρτηση αυτή ουσιαστικά είναι που δημιουργούμε το configuration για τον Hibernate. Αρχικά δηλώνουμε κάποια properties σχετικά με την εμφάνιση του κώδικα της βάσης που εκτελεί τα Queries που γίνονται. Στην συνέχεια, ορίζουμε το πακέτο το οποίο περιέχει της οντότητες που έχουμε ορίσει και αντιστοιχίζονται σε Tables.

Στην συνέχεια δηλώνονται κάποιες προτιμήσεις σχετικά με την διαχείριση των οντοτήτων από τον EntityManager του Hibernate. Εάν και τα περισσότερα και πάλι είναι αυτονόητα, θα ήθελα να τονίσω την ιδιαίτερη προσοχή στο property hibernate.hbm2ddl.auto. Όταν έχουμε ήδη μία βάση με τα αντίστοιχα tables, τότε βάζουμε ως επιλογή το validate ώστε να επαληθεύση την ορθότητα των οντοτήτων μας.

Εάν δεν έχουμε τα tables, βάζουμε create και το Hibernate αναλαμβάνει να δημιουργήσει τα διάφορα tables βάση των οντοτήτων μας. Ιδιαίτερη προσοχή: εάν ξεχάσουμε το create, κάθε φορά που θα ξεκινάει η εφαρμογή μας τότε θα διαγράφονται τα παλιά Table και θα δημιουργούνται από την αρχή, οδηγώντας σε απώλεια των δεδομένων μας.

transactionManager(EntityManagerFactory entityManagerFactory)

Με την χρήση της συνάρτησης αυτής ενεργοποιούμε την δυνατότητα χρήσης του @Transactional annotation, αναγκάζουμε δηλαδή το Spring να αναλάβει την διαχείριση των Transactions. Επειδή βασίζεται στις δύο προηγούμενες συναρτήσεις, είναι απαραίτητες να υλοποιηθούν ώστε να μπορέσουμε να χρησιμοποιήσουμε το annotation.

WebAppConfig

Μία από τις σημαντικότερες κλάσεις και αναγκαία εάν θέλουμε να έχουμε μία λειτουργική εφαρμογή. Το configuration αυτό είναι το αντίστοιχο που κάνουμε στο web.xml αλλά κάνοντας το σε Java μας δίνεται η δυνατότητα να το κάνουμε customize ευκολότερα. Εδώ το μόνο άξιο αναφοράς είναι η χρήση default encoding σε οποιοδήποτε request που έχει ως root url την εφαρμογή μας και την δήλωση του configureSpringSecurityFilter ώστε να μπορέσουμε να δηλώσουμε την κλάση SecurityConfig και να επιβάλλει τις πολιτικές της.

ApplicationContext

Στην συγκεκριμένη κλάση, ουσιαστικά δηλώνεται δηλώνονται όλα τα περιεχόμενα και οι λειτουργίες της εφαρμογής μας. Ουσιαστικά εδώ δηλώνονται και τακτοποιούνται ιεραρχικά όλα τα Configuration που έχουμε δηλώσει πιο πριν με την χρήση annotation. Αναλυτικότερα:

@Configuration

Μιας και αυτή η κλάση έχει ως σκοπό το configurarion του server μας.

@ComponentScan("main.java.com.exabit")

Το δηλώνουμε ώστε να μπορέσει το Spring να βρει τα διάφορα Repositories, τα Services κτλπ, να εντοπίσει τα Path τους ώστε να τα κάνει αργότερα Inject μέσω των @Autowired/@Resources. Επίσης ψάχνει και για Bean τα οποία δεν μπορούσαμε να τα ομαδοποιήσουμε σε κάποια από τις κλάσεις του Config.

@EnableJpaRepositories(basePackages = "main.java.com.exabit.Repositories")

Αντί να πάμε σε κάθε Repository και να προσθέσουμε το αντίστοιχο annotation, επιλέγουμε να το κάνουμε μαζικά χρησιμοποιώντας το @EnableJpaRepositories annotation.

@Import({SecurityConfig.class, WebMvcContext.class, PersistenceConfig.class})

Εδώ κάνουμε Import όλα τα άλλα configurarions. Πολύ χρήσιμο annotation, δεν παίζει ρόλο η σειρά, όπως προείπα το Spring θα ιεραρχίσει τα Configuration μόνο του.