ΑΣΦΑΛΕΙΑ ΠΛΗΡΟΦΟΡΙΑΚΩΝ  
ΣΥΣΤΗΜΑΤΩΝ

Τελικη ΕΡΓΑΣΙΑ

**Συντελεστές εργασίας**

Χριστοφορίδης Χαράλαμπος – Π19188

Γεωργιάδης Νικόλαος – π19032

Καρκάνης Ευστράτιος – Π19064

20 ιανουαριου 2023

**Περιεχόμενα**

[**0.** **Εισαγωγή** 2](#_Toc125179580)

[**1.** **Μελέτη ασφάλειας ΠΣ** 3](#_Toc125179581)

[**2.** **Κρυπτογράφηση ssl στον server της εφαρμογης** 3](#_Toc125179582)

[**2.1** **Εγκατάσταση πιστοποιητικού server** 3](#_Toc125179583)

[**2.2** **Μόνιμη σύνδεση σε HTTPS** 6](#_Toc125179584)

[**3.** **Μηχανισμός αυθεντικοποίησης και ελέγχου πρόσβασης** 8](#_Toc125179585)

[**3.1** **Μηχανισμοί αυθεντικοποίησης** 8](#_Toc125179586)

[**3.2** **Μηχανισμός ελέγχου πρόσβασης** 15](#_Toc125179587)

[**4.** **Input filtering και validation** 19](#_Toc125179588)

[**5.** **Αυτοματοποιημένος έλεγχος για την εύρεση ευπαθειών ασφάλειας** 19](#_Toc125179589)

# **Εισαγωγή**

Η διαδικτυακή εφαρμογή που έχουμε αναπτύξει, αποτελεί ένα σύστημα διαχείρισης ραντεβού ασθενών και ιατρών. Στο σύστημα υπάρχουν και οι διαχειριστές, οι οποίοι μπορούν να προσθέτουν και να αφαιρούν κάθε κατηγορία χρήστη από την στιγμή που θα συνδεθούν στο σύστημα. Αναλυτικές πληροφορίες για τις λειτουργίες και για το περιεχόμενο της εφαρμογής μπορούν να βρεθούν στο αρχείο “**Παρουσίαση παλιάς εργασίας.pdf**” το οποίο αποτελεί την παρουσίαση της εφαρμογής που αναπτύχθηκε στο 4o εξάμηνο κατά τη διάρκεια του μαθήματος “**Προγραμματισμός στο διαδίκτυο και στον παγκόσμιο ιστό**”.

Η εφαρμογή έχει αναπτυχθεί μέσω του **IDE** **Intellij IDEA v.2021.2.2** σε **Java**, χρησιμοποιώντας **servlets, HTML και JSP σελίδες**. Ο Application-Web server που χρησιμοποιήθηκε είναι ο **Tomcat v.8.5.6.** Η ανάπτυξη της βάσης δεδομένων έγινε μέσω του εργαλείου **MySQL WorkBench v.8.0.**

Στον φάκελο “**Web App**” βρίσκεται ο αναλυτικός κώδικας σε μορφή **Intellij IDEA** project(**Ergasia 3**), το μοντέλο της βάσης δεδομένων(**app\_db.mwb**), ο φάκελος “**sql\_statements**” που περιέχει τα queries τα οποία εισάγουν τα δεδομένα στη βάση και ένα αρχείο “**passwords memo.txt**” για να μπορούμε να έχουμε πρόσβαση σε κάποια passwords χρηστών προκειμένου να γίνει δοκιμή της εφαρμογής(τα **passwords** βρίσκονται σε **hashed + salted** μορφή στη βάση δεδομένων οπότε **δεν μπορούμε να τα ανακτήσουμε**).

# **Μελέτη ασφάλειας ΠΣ**

# **Κρυπτογράφηση ssl στον server της εφαρμογης**

## **Εγκατάσταση πιστοποιητικού server**

Κατά την υλοποίηση της **3ης εργασίας** του εξαμήνου, μας ζητήθηκε να δημιουργήσουμε ένα **αυτοϋπογεγραμμένο πιστοποιητικό αρχής πιστοποίησης**. Στη συνέχεια, αυτό το χρησιμοποιήσαμε για να εκδώσουμε ένα **υπογεγραμμένο πιστοποιητικό για τον** **server** μας. Μαζί με το **ιδιωτικό κλειδί του server** και τα δύο παραπάνω πιστοποιητικά, δημιουργούμε ένα ενιαίο αρχείο με όνομα “**serverkeystore.jks**” το οποίο εγκαθιστάμε στον Tomcat server και είναι υπεύθυνο για την **αυθεντικοποίηση** **του server στον client**.

Οι εντολές που τρέξαμε για την υλοποίηση των παραπάνω και τα αναλυτικά βήματα, βρίσκονται στην **παρουσίαση της 3ης εργασίας** που έχει αναρτηθεί στο **gunet2**(χρησιμοποιήθηκε η βιβλιοθήκη **openssl** και **keytool** για **windows**).

Στο φάκελο όπου έχουμε εγκαταστήσει τον Tomcat, μεταβαίνουμε στον **υποφάκελο conf** και εισάγουμε εκεί το αρχείο “**serverkeystore.jks**”. Στη συνέχεια κάνουμε την παρακάτω **τροποποίηση** στο αρχείο **server.xml**(στη συγκεκριμένη περίπτωση, η χρήση του πρωτοκόλλου **SSL** υλοποιείται με τη βοήθεια του **JSSE API**.):



Αλλαγές στο server.xml (πηγή: <https://www.tencentcloud.com/document/product/1007/43804> )

Να σημειωθεί πως έχουμε υλοποιήσει και **σύστημα αυθεντικοποίησης του χρήστη** με χρήση ενός **πιστοποιητικού χρήστη**(Η διαδικασία αυτή εξηγείται στην επόμενη ενότητα).

Αφού ανοίξουμε το **Intellij IDE** και τρέξουμε τον server, το αποτέλεσμα στον browser μας είναι:

Graphical user interface, application, Teams

Description automatically generated

Η εφαρμογή τρέχει σε http

Προσθέτοντας το **πρόθεμα** “**https://**” και αλλάζοντας τη **θύρα** σε “**8443**” μπαίνουμε σε **https σύνδεση**:

Graphical user interface, application, Teams

Description automatically generated

Απόπειρα σύνδεσης σε https

Στη συνέχεια πατάμε **advanced** και **Proceed to localhost(unsafe)** για να **αγνοήσουμε** το κίνδυνο του μη έμπιστου πιστοποιητικού(το πιστοποιητικό του server είναι **αναγκαστικά μη έμπιστο** διότι είναι υπογεγραμμένοαπό μία **αυτοϋπογεγραμμενη αρχή πιστοποίησης**):



Graphical user interface, application, Teams

Description automatically generated

Η εφαρμογή τρέχει σε https

Να σημειωθεί πως το αρχείο “**serverkeystore.jks**” και το **τροποποιημένο** **αρχείο “server.xml”** βρίσκονται στον φάκελο “**Ερώτημα 2/tomcat”**.

## **Μόνιμη σύνδεση σε HTTPS**

Για να επιτύχουμε μόνιμη σύνδεση σε HTTPS μεταβαίνουμε στον φάκελο του Tomcat,μετά στον υποφάκελο conf και προσθέτουμε το παρακάτω κομμάτι στο αρχείο web.xml:

A screenshot of a computer

Description automatically generated with medium confidence

Αλλαγές στο αρχείο web.xml (πηγή: <https://www.infoworld.com/article/3304289/how-to-configure-tomcat-to-always-require-https.html>)

Εάν προσπαθήσουμε να τρέξουμε τον server τώρα, το **Intellij IDE** θα μας δείξει την παρακάτω **ιδιοποίηση**:

Graphical user interface, text, application

Description automatically generated

Αυτό συμβαίνει διότι πλέον η εφαρμογή μπορεί να τρέχει **μόνο** σε **HTTPS σύνδεση**.

Τροποποιώντας τον σύνδεσμο ως εξής: [**https://localhost:8443/Ergasia3\_war\_exploded/**](https://localhost:8443/Ergasia3_war_exploded/)και πατώντας “**OK**”, η εφαρμογή τρέχει σε **https**:



Graphical user interface, application, Teams

Description automatically generated

Η εφαρμογή τρέχει σε https

Επιπλέων, κάνουμε τις **απαραίτητες ρυθμίσεις** στα **configurations** του Tomcat στο IntelliJ έτσι ώστε η εφαρμογή να ξεκινάει **πάντα** σε **https** και σε κάθε **απόπειρα σύνδεσης** σε **http** να γίνεται **ανακατεύθυνση** σε **https**:



Graphical user interface, application

Description automatically generated

Αλλαγή της θύρας HTTPS σε 8443 και του URL σε αυτό που αντιστοιχεί στην HTTPS σύνδεση

Να σημειωθεί πως το **τροποποιημένο** **αρχείο “web.xml”** βρίσκεται στον φάκελο “**Ερώτημα 2/tomcat”**.

# **Μηχανισμός αυθεντικοποίησης και ελέγχου πρόσβασης**

## **Μηχανισμοί αυθεντικοποίησης**

Σαν μηχανισμοί αυθεντικοποίησης έχουν χρησιμοποιηθεί **δύο**. Ο ένας αποτελεί τον κλασσικό τρόπο μέσω **username** και **password** και ο άλλος αποτελεί το **πιστοποιητικό χρήστη**.

Κάθε χρήστης που έχει **εγγραφεί**(**register**) στο σύστημα, έχει την δυνατότητα να **συνδεθεί**(**login**) αργότερα στον λογαριασμό του μέσω του **username** και του **password** του. Να σημειωθεί πως **μόνο ασθενείς** μπορούν να **εγγραφούν** **μόνοι τους**, καθώς οι **ιατροί** και οι **διαχειριστές**, **εγγράφονται** στο σύστημα από **διαχειριστές**.

Graphical user interface, text, application

Description automatically generated

Εγγραφή ασθενών

Graphical user interface, application

Description automatically generated

Σύνδεση χρηστών

Κατά τη διάρκεια **αποθήκευσης** των στοιχείων ενός χρήστη στη **βάση δεδομένων**, προστίθεται μία **τυχαία συμβολοσειρά** γνωστή και ως “**salt**” στο **password** και το αποτέλεσμα γίνεται **hash** με τον αλγόριθμο **MD5**.

Ο χρήστης αφού εισάγει τα στοιχεία του, επιλέγει την **κατηγορία** στην οποία ανήκει κάνοντας κλικ το **αντίστοιχο** **radio button**, πριν πατήσει το κουμπί **Login**. Στη συνέχεια βρίσκεται η **εγγραφή** της βάσης δεδομένων που περιέχει το **username** που έδωσε ο χρήστης(**αν υπάρχει**) και από εκεί λαμβάνεται το **salt** και το **hashed password**. Αν το **password** που έδωσε ο χρήστης μαζί με το **salt** έχουν το **ίδιο hash** με αυτό της εγγραφής της βάσης, τότε ο χρήστης **συνδέεται** στο σύστημα.

Από τα προηγούμενα μπορούμε να **συμπεράνουμε** **τα εξής**:

* Ένας χρήστης μιας **συγκεκριμένης** κατηγορίας **δεν μπορεί** να συνδεθεί με ένα **username** που είναι **ανύπαρκτο** καθώς **δεν θα βρεθεί** η αντίστοιχη εγγραφή στη βάση(**ταυτοποίηση**)
* Ένας χρήστης μιας **συγκεκριμένης** κατηγορίας με ένα **σωστό** username **δεν μπορεί** να συνδεθεί στην εφαρμογή **χωρίς το σωστό** password καθώς το **hash** του **λάθος** **password** με το **salt** **δεν θα είναι** αυτό που είναι **αποθηκευμένο** στη βάση(**αυθεντικοποίηση**)

Παράδειγμα **σωστής** σύνδεσης χρήστη(**username**: patient1, **password**: patient):

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, website

Description automatically generated

Παράδειγμα **λάθος** σύνδεσης χρήστη(**username**: patient1, **password**: patient123):

Graphical user interface, application

Description automatically generated

A red and white sign

Description automatically generated with medium confidence

Να σημειωθεί πως έχουμε χρησιμοποιήσει την βιβλιοθήκη “**java.security.MessageDigest”** για την υλοποίηση του **MD5 hashing** και την βιβλιοθήκη “**java.security.SecureRandom”** για την παραγωγή **τυχαίων χαρακτήρων salt.**

**Κώδικας παραγωγής τυχαίου salt:**

private static String createSalt()  
{  
 try  
 {  
 SecureRandom random = new SecureRandom();  
  
 byte bytes[]= new byte[20];  
 random.nextBytes(bytes);  
  
 for(int i = 0; i< bytes.length; i++)  
 {  
 if(bytes[i] < 0)  
 bytes[i] = (byte) -bytes[i];  
 }  
  
 return new String(bytes, "UTF-8");  
  
 }  
 catch (IOException e)  
 {  
 System.*out*.println("IOException: "+e.toString());  
 return "";  
 }  
}

**Κώδικας MD5 hashing:**

private static String hashPassword(String password, String salt)  
{  
 // Hash the password.  
 final String toHash = salt + password + salt;  
 MessageDigest messageDigest = null;  
 try  
 {  
 messageDigest = MessageDigest.*getInstance*("MD5");  
  
 } catch (Exception ex)  
 {  
 System.*out*.println(ex.toString());  
 return "00000000000000000000000000000000";  
 }  
 messageDigest.update(toHash.getBytes(), 0, toHash.length());

String hashed =

new BigInteger(1,messageDigest.digest()).toString(16);

if (hashed.length() < 32)  
 {  
 hashed = "0" + hashed;  
 }  
 return hashed.toUpperCase();  
}

**Συνθήκη που ελέγχει αν υπάρχει το username και αν το password είναι σωστό:**

if( *rs*.next()

&& *hashPassword*(pass,*rs*.getString("salt")).equals(

*rs*.getString("hashedpassword")) )  
{ … }

**pass**: το password που έδωσε ο χρήστης, **rs**: το αντικείμενο που περιέχει τα δεδομένα της **εγγραφής της βάσης** με το **username** που έδωσε ο **χρήστης**. Αν **υπάρχει** το username, το **rs.next()** επιστρέφει **true** αλλιώς **false**. Αν το **hash του pass** **και** **της τιμής του πεδίου salt** μας κάνουν τη **τιμή του πεδίου hashedpassword**, τότε έχουμε **επιτυχή** **σύνδεση**.

Όσον αφορά την **αυθεντικοποίηση** του χρήστη μέσω **πιστοποιητικού**, έχουμε δημιουργήσει ένα **πιστοποιητικό χρήστη** το οποίο έχει **υπογραφεί** από την **αυτοϋπογεγραμμένη αρχή πιστοποίησης** μας(αρχείο “**client.p12**” στον φάκελο “**Ερώτημα 2**”). Επίσης, έχουμε δημιουργήσει και ένα αρχείο “**catrustore.jks**”(φάκελος “**Ερώτημα 2/tomcat**”) το οποίο είναι απαραίτητο να εγκατασταθεί στον tomcat για να λειτουργήσει η αυθεντικοποίηση χρήστη μέσω πιστοποιητικού.

Οι εντολές που τρέξαμε για την δημιουργία των παραπάνω και τα αναλυτικά βήματα, βρίσκονται στην **παρουσίαση της 3ης εργασίας** που έχει αναρτηθεί στο **gunet2**(χρησιμοποιήθηκε η βιβλιοθήκη **openssl** και **keytool** των **windows**).

Στο φάκελο όπου έχουμε εγκαταστήσει τον Tomcat, μεταβαίνουμε στον **υποφάκελο conf** και εισάγουμε εκεί το αρχείο “**catrustore.jks**”. Στη συνέχεια κάνουμε την παρακάτω **τροποποίηση** στο αρχείο **server.xml**:

Text, letter

Description automatically generated

*Αλλαγή του server.xml. Πλέον έχουμε αυθεντικοποίηση χρήστη ορίζοντας clientAuth=”true”. Πηγή:* <https://stackoverflow.com/questions/1552345/tomcat-client-authentication-using-ssl>

Στη συνέχεια εγκαθιστάμε το **πιστοποιητικό χρήστη**(“**client.p12”**) στον browser μας. Αφού τρέξουμε τον server μετά από τα παραπάνω βήματα, μας ζητείται **αυθεντικοποίηση**:

Graphical user interface, application

Description automatically generated

Επιλογή πιστοποιητικού χρήστη κατά την εκκίνηση της εφαρμογής

Graphical user interface, application

Description automatically generated

Επιτυχής αυθεντικοποίηση χρήστη

## **Μηχανισμός ελέγχου πρόσβασης**

Η εφαρμογή, υποστηρίζει **3 ρόλους**: **διαχειριστές**, **ιατρούς** και **ασθενείς**. Ο κάθε χρήστης έχει **διαφορετικά δικαιώματα**. Για παράδειγμα ένας **Ιατρός** έχει πρόσβαση στα **προγραμματισμένα ραντεβού** του ενώ ένας **διαχειριστής** **δεν μπορεί** να έχει **προγραμματισμένα ραντεβού**. Επίσης, ένας **διαχειριστής** έχει δικαίωμα να **προσθέσει** ή να **αφαιρέσει** **διαχειριστές και ιατρούς** από το σύστημα, ενώ οι **υπόλοιποι χρήστες όχι**. Ο έλεγχος πρόσβασης κάθε ρόλου επιτυγχάνεται με την χρήση **radio buttons**, ένα για κάθε ρόλο.



Radio buttons για την επιλογή ρόλου του χρήστη(σελίδα login.jsp)

Για παράδειγμα, όταν ένας **Ιατρός** θέλει να συνδεθεί, εισάγει τα στοιχεία του και στη συνέχεια επιλέγει “**Doctor**” για να αποκτήσει τα **κατάλληλα δικαιώματα**.

Η εφαρμογή είναι υλοποιημένη με τέτοιο τρόπο ώστε ένας χρήστης να **μη μπορεί να πάρει εσφαλμένα δικαιώματα σε περίπτωση που επιλέξει κάποιον ρόλο που δεν του ανήκει**. Υπάρχουν **3 διαφορετικοί** πίνακες για τους χρήστες στη βάση δεδομένων, όπου ο καθένας περιέχει τα στοιχεία των χρηστών **μόνο ενός ρόλου**. Όταν ο χρήστης ενός ρόλου προσπαθεί να συνδεθεί στην εφαρμογή, στη μεριά του εξυπηρετητή γίνεται **αναζήτηση των στοιχείων του στον αντίστοιχο πίνακα**. Οπότε εάν κάποιος χρήστης προσπαθήσει να συνδεθεί με **λάθος ρόλο**, τα στοιχεία του θα **αναζητηθούν σε λάθος πίνακα** οπότε η σύνδεση του θα **αποτύχει**. Να σημειωθεί πως η εφαρμογή **δεν επιτρέπει δύο οποιοιδήποτε χρήστες να έχουν το ίδιο όνομα χρήστη**, **με σκοπό να αποφευχθεί ο ένας να συνδεθεί με τον ρόλο του άλλου**.

Ακολουθεί παράδειγμα για τον χρήστη με **username**: patient1 και **password**: patient:

**Σωστή επιλογή ρόλου:**

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, website

Description automatically generated

**Λάθος επιλογή ρόλου:**

Graphical user interface, application

Description automatically generated

A red and white sign

Description automatically generated with medium confidence

Τμήμα κώδικα της συνάρτησης **Login()** που υποδεικνύει την **αναζήτηση στον κατάλληλο πίνακα**. **type**: ο ρόλος που επιλέχθηκε από τον χρήστη, **name**: το username που έδωσε ο χρήστης. Αν το **username** υπάρχει στο **table του ρόλου που ψάχνουμε**, συνεχίζουμε με τον έλεγχο του **password**:

*connection* = datasource.getConnection();  
  
table = type.toLowerCase();  
  
*statement* = *connection*.prepareStatement("SELECT \* FROM `"+ table +"` WHERE username=?");

*statement*.setString(1, name);  
  
*rs* = *statement*.executeQuery();  
  
  
if(*rs*.next() && *hashPassword*(pass, *rs*.getString("salt")).equals(*rs*.getString("hashedpassword")))  
{

# **Input filtering και validation**

# **Αυτοματοποιημένος έλεγχος για την εύρεση ευπαθειών ασφάλειας**