

24 ΙΑΝΟΥΑΡΙΟΥ 2023

ΑΣΦΑΛΕΙΑ ΠΛΗΡΟΦΟΡΙΑΚΩΝ
ΣΥΣΤΗΜΑΤΩΝ
ΤΕΛΙΚΗ ΕΡΓΑΣΙΑ

Συντελεστές εργασίας

Χριστοφορίδης Χαράλαμπος – Π19188

Γεωργιάδης Νικόλαος – Π19032

Καρκάνης Ευστράτιος – Π19064

Περιεχόμενα

0. Εισαγωγή	2
1. Μελέτη ασφάλειας ΠΣ.....	3
2. Κρυπτογράφηση ssl στον server της εφαρμογής	5
2.1 Εγκατάσταση πιστοποιητικού server	5
2.2 Μόνιμη σύνδεση σε HTTPS	8
3. Μηχανισμός αυθεντικοποίησης και ελέγχου πρόσβασης	10
3.1 Μηχανισμοί αυθεντικοποίησης	10
3.2 Μηχανισμός ελέγχου πρόσβασης	17
4. Input filtering και validation	21
5. Αυτοματοποιημένος έλεγχος για την εύρεση ευπαθειών ασφάλειας	30

0. 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 4ο 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**).

1. Μελέτη ασφάλειας ΠΣ

Κατά την υλοποίηση της 1^{ης} **εργασίας** του εξαμήνου μας ζητήθηκε να κάνουμε μελέτη ασφάλειας ενός πληροφοριακού συστήματος. Αναλυτικότερα, μέσα από τα κομμάτια της εργασίας, πραγματοποιήσαμε (ερωτήματα 1-6):

- Καταγραφή του υπό μελέτη συστήματος.
- Δημιουργία μοντέλου αγαθών (asset model).
- Αντιστοίχιση υπηρεσιών και υπολογιστικών συστημάτων.
- Αποτίμηση συνεπειών ή επιπτώσεων ασφάλειας (impact assessment).
- Αποτίμηση απειλών (threat assessment).
- Αποτίμηση αδυναμιών (vulnerability assessment)

Από τα παραπάνω λείπουν τα ερωτήματα 7 και 8, τα οποία σύμφωνα με την υπόδειξη της εκφώνησης τα αφήσαμε για την παράδοση της τελικής εργασίας. Για τα παραπάνω υπάρχει η σχετική τεκμηρίωση και ανάλυση στην εργασία που είχαμε παραδώσει στο gunet(Εργασία 1).

Όσον αφορά την τελική μορφή της πρώτης εργασίας έχουν προστεθεί τα ερωτήματα 7 και 8 της εκφώνησης(το αρχείο είναι Τελική Μορφής Εργασία 1). Αναλυτικότερα:

- **Αποτίμηση κινδύνων (risk assessment)**

Για το συγκεκριμένο ερώτημα χρησιμοποιήσαμε τα αποτελέσματα από προηγούμενα ερωτήματα (4,5,6) και μπορέσαμε να αξιολογήσουμε κινδύνους με βάση την πιθανότητα πραγματοποίησής τους, αλλά και την σοβαρότητα των επιπτώσεων που θα είχαν στην εφαρμογή μας.

- **Επανεκτίμηση αδυναμιών μετά την υλοποίηση των μέτρων ασφάλειας (vulnerability post assessment)**

Για το συγκεκριμένο ερώτημα χρησιμοποιήσαμε τις αδυναμίες που βρήκαμε στο ερώτημα 6 αλλά και το CVSS (Version 3.1) Calculator. Ουσιαστικά καταφέραμε να αναβαθμολογήσουμε αυτές τις αδυναμίες λαμβάνοντας υπόψη τα μέτρα ασφαλείας που έχουμε πάρει αλλά και την ίδια τη φύση της εφαρμογής μας.

Τέλος ως τελευταίο κομμάτι της μελέτης ασφάλειας και αφού είχε τελειώσει η 1^η Εργασία είναι η δημιουργία μίας **πολιτικής ασφάλειας**. Για την δημιουργία της ονομάσαμε την εταιρία μας ΧΑΝΙΣΤ Α.Ε και θεωρήσαμε πως παρέχουμε αυτήν την εφαρμογή σε πληθώρα νοσοκομείων ανά τον κόσμο. Ακολουθήθηκε πιστά το πρότυπο ISO 27001, όπως μας είχε υποδειχθεί και στο μάθημα. Το σχετικό έγγραφο είναι στο παραδοτέο της εργασίας.

(Προσθέσαμε και κάποιες λεπτομέρειες σε προηγούμενα ερωτήματα της εργασίας 1, που δεν είχαμε υπολογίσει πριν. Για παράδειγμα στο ερώτημα 2 προσθέσαμε το πρωτόκολλο ασφάλειας SSL και το υλοποιήσαμε με JSSE API)

Αναλυτικά τα περιεχόμενα της **Μελέτης Ασφάλειας του Π.Σ** είναι:

1. Καταγραφή του υπό μελέτη συστήματος.
2. Δημιουργία μοντέλου αγαθών (asset model).
3. Αντιστοίχιση υπηρεσιών και υπολογιστικών συστημάτων.
4. Αποτίμηση συνεπειών ή επιπτώσεων ασφάλειας (impact assessment).
5. Αποτίμηση απειλών (threat assessment).
6. Αποτίμηση αδυναμιών (vulnerability assessment)
7. Αποτίμηση κινδύνων (risk assessment)
8. Επανεκτίμηση αδυναμιών μετά την υλοποίηση των μέτρων ασφάλειας (vulnerability post assessment)
9. Πολιτική Ασφάλειας

(Τα πρώτα 8 αποτελούν την πρώτη εργασία και το ερώτημα 1α της τελικής εργασίας. Η πολιτική ασφάλειας αποτελεί το ερώτημα 1β της τελικής εργασίας)

2. Κρυπτογράφηση ssl στον server της εφαρμογής

2.1 Εγκατάσταση πιστοποιητικού server

Κατά την υλοποίηση της **3^{ης} 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 3^{ης} 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**):

```
<!-- Define an SSL/TLS HTTP/1.1 Connector on port 8443
This connector uses the NIO implementation. The default
SSLImplementation will depend on the presence of the APR/native
library and the useOpenSSL attribute of the
AprLifecycleListener.
Either JSSE or OpenSSL style configuration may be used regardless of
the SSLImplementation selected. JSSE style configuration is used below.
-->

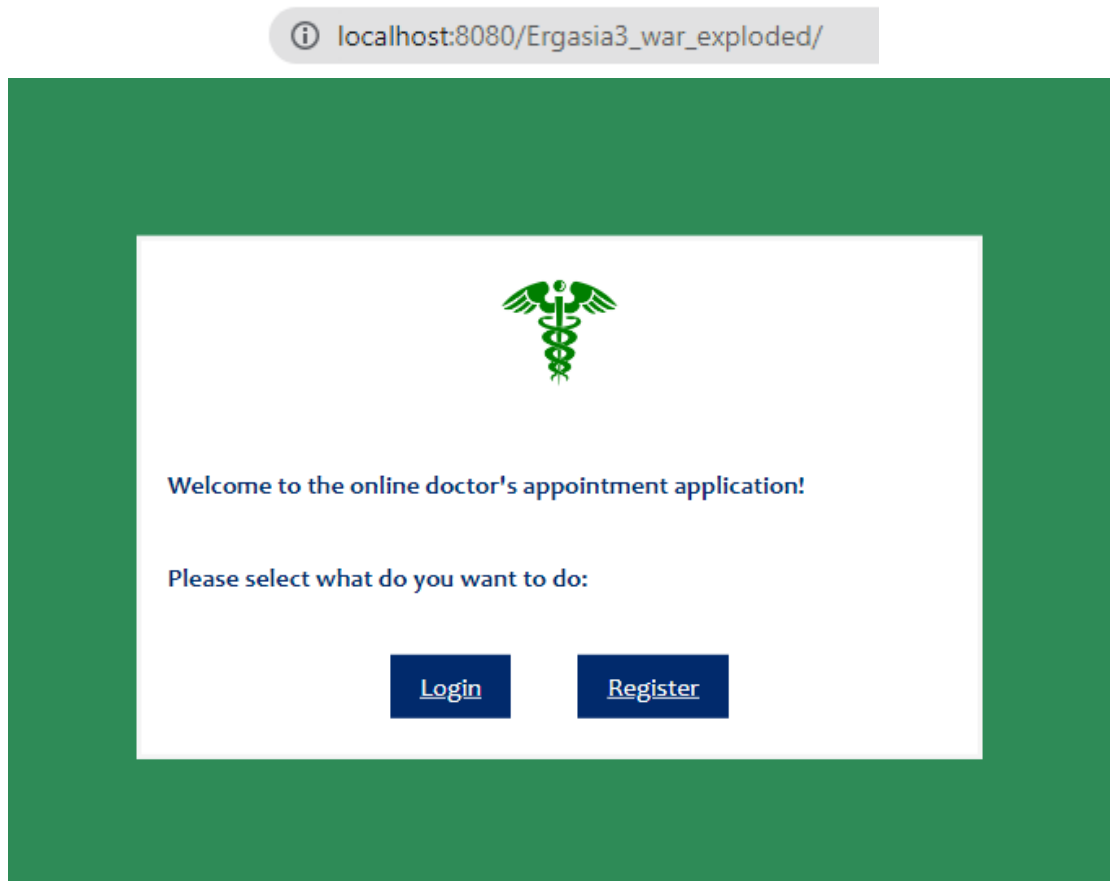
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
maxThreads="150" scheme="https" secure="true"
keystoreFile="conf\serverkeystore.jks"
keystorePass="123456"
clientAuth="false"/>
```

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:



Η εφαρμογή τρέχει σε http

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

⚠ Not secure | https://localhost:8443/Ergasia3_war_explored/



Your connection is not private

Attackers might be trying to steal your information from **localhost** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR_CERT_COMMON_NAME_INVALID



To get Chrome's highest level of security, [turn on enhanced protection](#)

Hide advanced

Back to safety

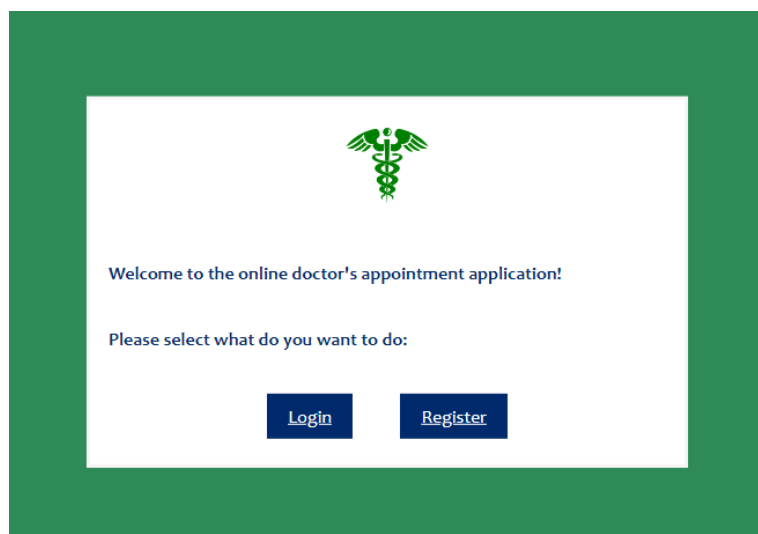
This server could not prove that it is **localhost**; its security certificate does not specify Subject Alternative Names. This may be caused by a misconfiguration or an attacker intercepting your connection.

[Proceed to localhost \(unsafe\)](#)

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 υπογεγραμμένο από μία **αυτοϋπογεγραμμένη αρχή πιστοποίησης**):

⚠ Not secure | https://localhost:8443/Ergasia3_war_explored/



Η εφαρμογή τρέχει σε https

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

2.2 Μόνιμη σύνδεση σε HTTPS

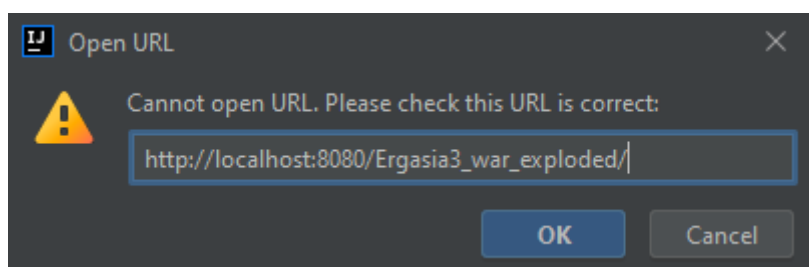
Για να επιτύχουμε μόνιμη σύνδεση σε HTTPS μεταβαίνουμε στον φάκελο του Tomcat, μετά στον υποφάκελο conf και προσθέτουμε το παρακάτω κομμάτι στο αρχείο web.xml:

```
<!-- Require HTTPS for everything except /img (favicon) and /css. -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>HTTPSOnly</web-resource-name>
    <url-pattern>*/</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>HTTPSOrHTTP</web-resource-name>
    <url-pattern>*.ico</url-pattern>
    <url-pattern>/img/*</url-pattern>
    <url-pattern>/css/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Αλλαγές στο αρχείο web.xml (πηγή:

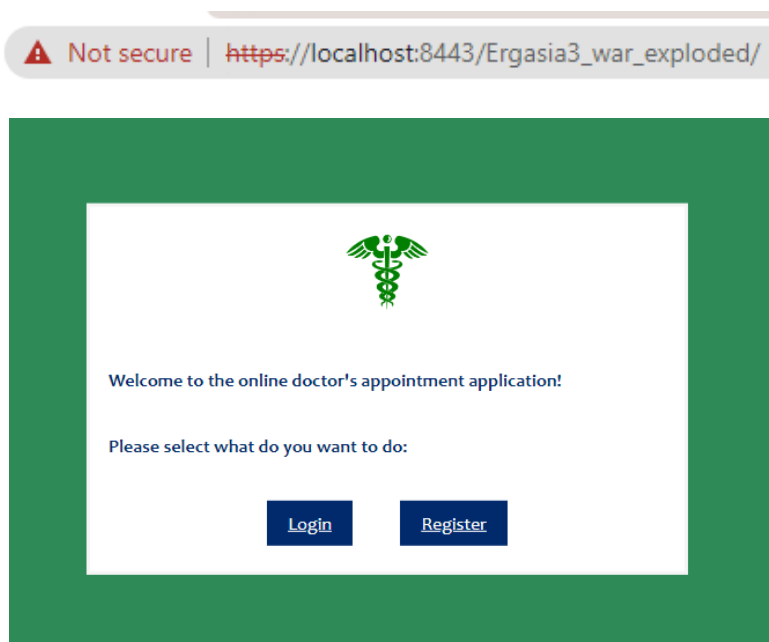
<https://www.infoworld.com/article/3304289/how-to-configure-tomcat-to-always-require-https.html>)

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



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

Τροποποιώντας τον σύνδεσμο ως εξής: https://localhost:8443/Ergasia3_war_exploded/ και πατώντας “OK”, η εφαρμογή τρέχει σε **https**:



Η εφαρμογή τρέχει σε https

Επιπλέον, 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**:



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

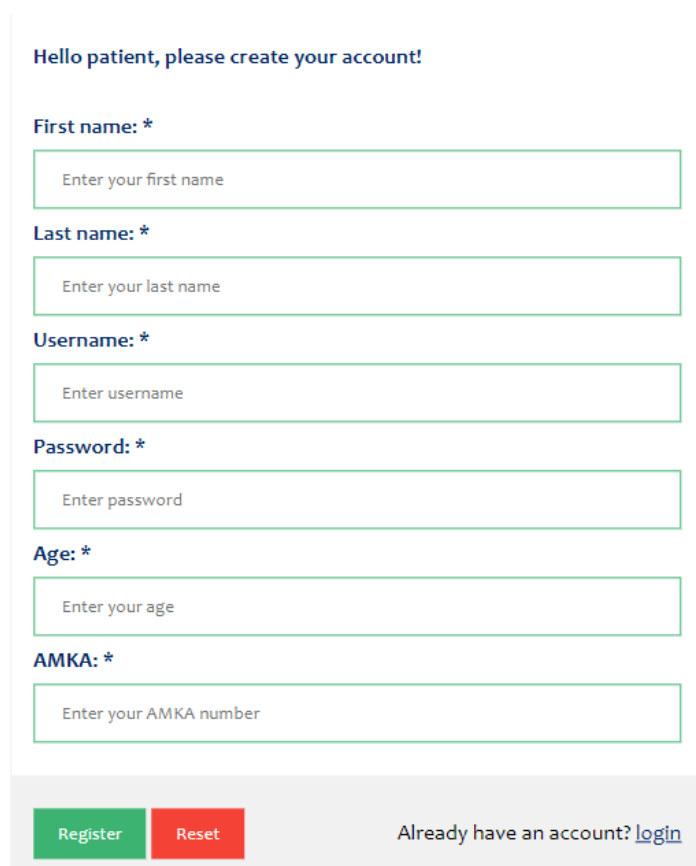
Να σημειωθεί πως το **τροποποιημένο αρχείο “web.xml”** βρίσκεται στον φάκελο **“Ερώτημα 2/tomcat”**.

3. Μηχανισμός αυθεντικοποίησης και ελέγχου πρόσβασης

3.1 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**.



A registration form for patients. At the top, it says "Hello patient, please create your account!". Below this are six input fields, each with a label and an asterisk indicating it is required: "First name: *", "Last name: *", "Username: *", "Password: *", "Age: *", and "AMKA: *". Each field has a placeholder text: "Enter your first name", "Enter your last name", "Enter username", "Enter password", "Enter your age", and "Enter your AMKA number". At the bottom, there are two buttons: a green "Register" button and a red "Reset" button. To the right of these buttons, it says "Already have an account? [login](#)".

Εγγραφή ασθενών

register'." data-bbox="269 83 720 337"/>

Σύνδεση χρηστών

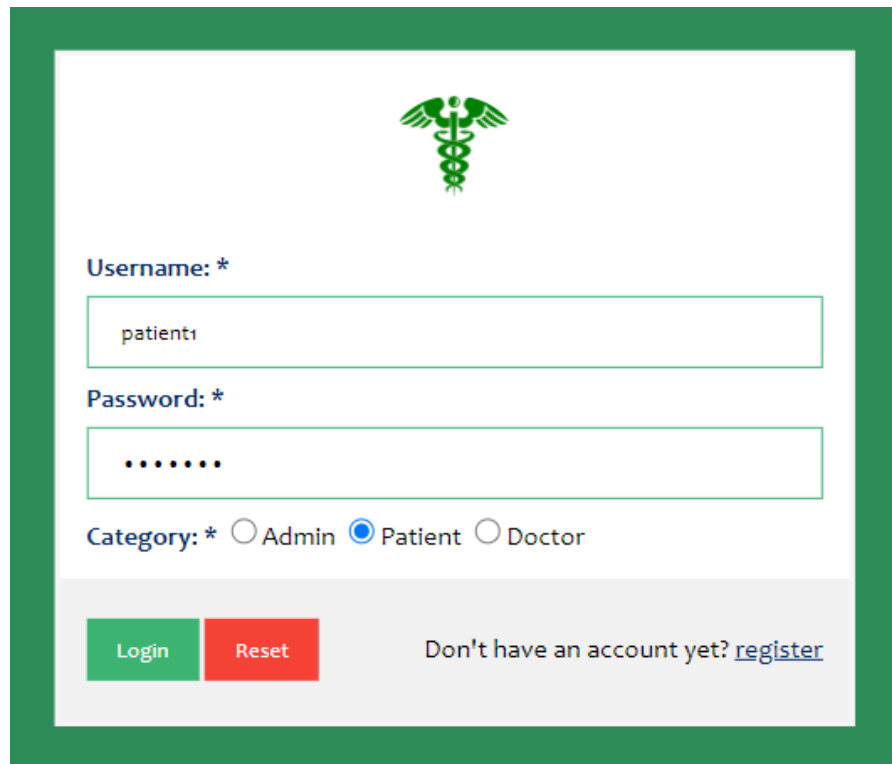
Κατά τη διάρκεια **αποθήκευσης** των στοιχείων ενός χρήστη στη **βάση δεδομένων**, προστίθεται μία **τυχαία συμβολοσειρά** γνωστή και ως “**salt**” στο **password** και το αποτέλεσμα γίνεται **hash** με τον αλγόριθμο **MD5**.

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):



Username: *

patient1

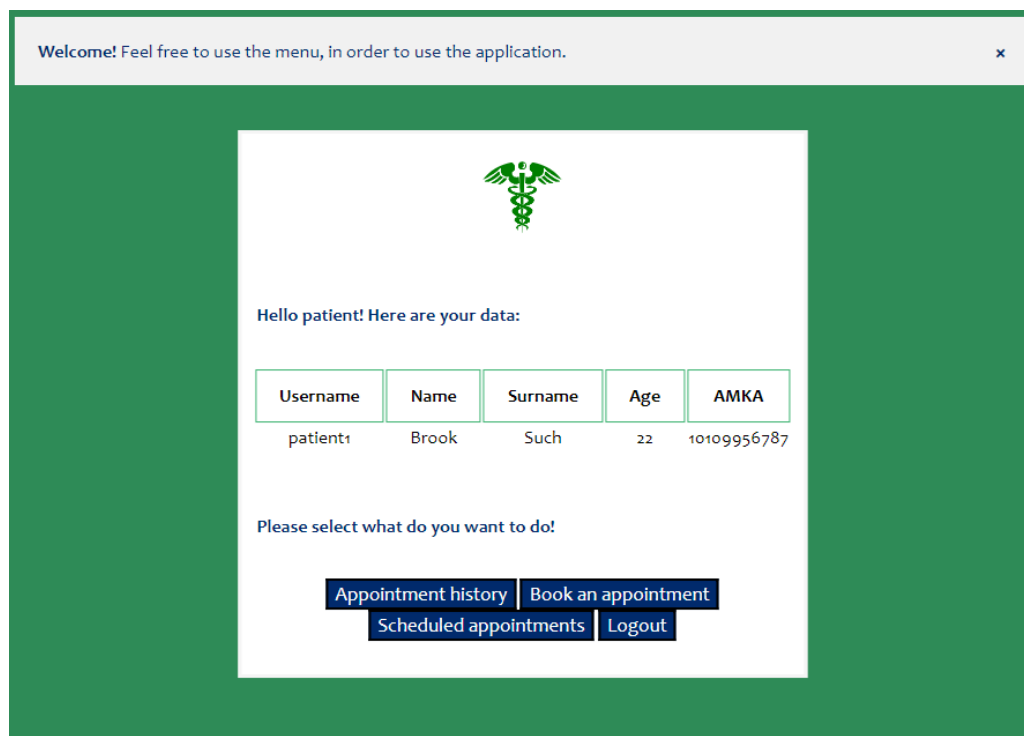
Password: *

.....

Category: * ☐ Admin ☒ Patient ☐ Doctor

Login Reset Don't have an account yet? [register](#)

Welcome! Feel free to use the menu, in order to use the application. x



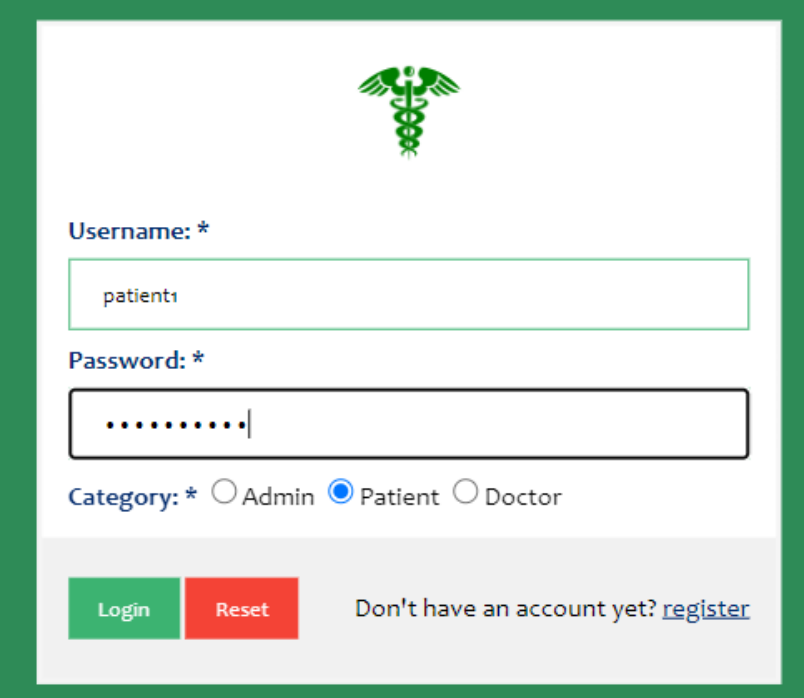
Hello patient! Here are your data:

Username	Name	Surname	Age	AMKA
patient1	Brook	Such	22	10109956787

Please select what do you want to do!

Appointment history Book an appointment
Scheduled appointments Logout

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



Medical login form interface. At the top is a green caduceus icon. Below it are three input fields: 'Username: *' containing 'patient1', 'Password: *' with masked dots, and 'Category: *' with radio buttons for 'Admin', 'Patient' (selected), and 'Doctor'. At the bottom are two buttons: a green 'Login' button and a red 'Reset' button. To the right of the buttons is a link: 'Don't have an account yet? [register](#)'.



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 3^{ης} 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**:


```

<!-- Define an SSL/TLS HTTP/1.1 Connector on port 8443
      This connector uses the NIO implementation. The default
      SSLImplementation will depend on the presence of the APR/native
      library and the useOpenSSL attribute of the
      AprLifecycleListener.
      Either JSSE or OpenSSL style configuration may be used regardless of
      the SSLImplementation selected. JSSE style configuration is used below.
-->

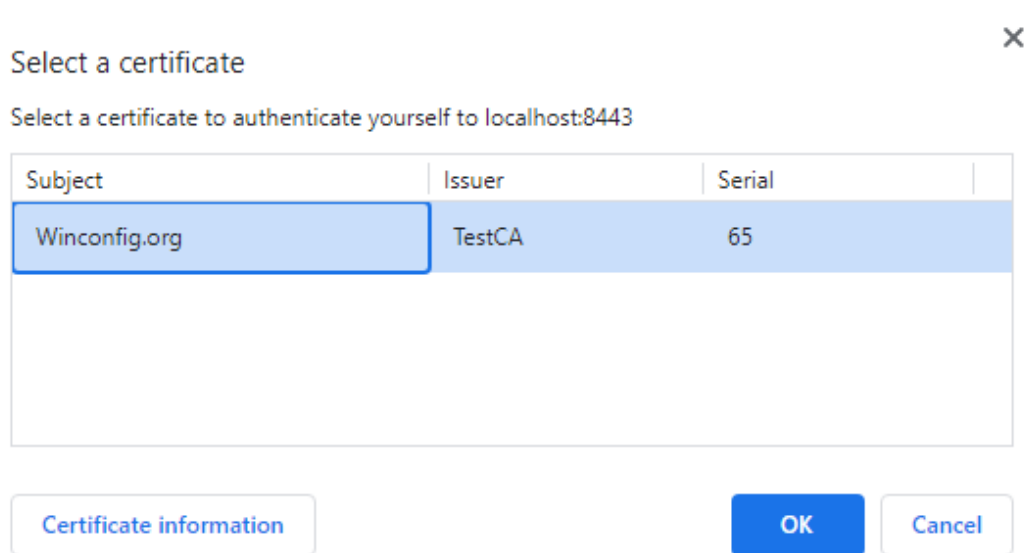
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
      SSLEnabled="true"
      maxThreads="150" scheme="https" secure="true"
      keystoreFile="conf\serverkeystore.jks"
      keystorePass="123456"
      truststoreFile="conf\catrustore.jks"
      truststorePass="123456"
      clientAuth="true"
      sslProtocol="TLS"
/>

```

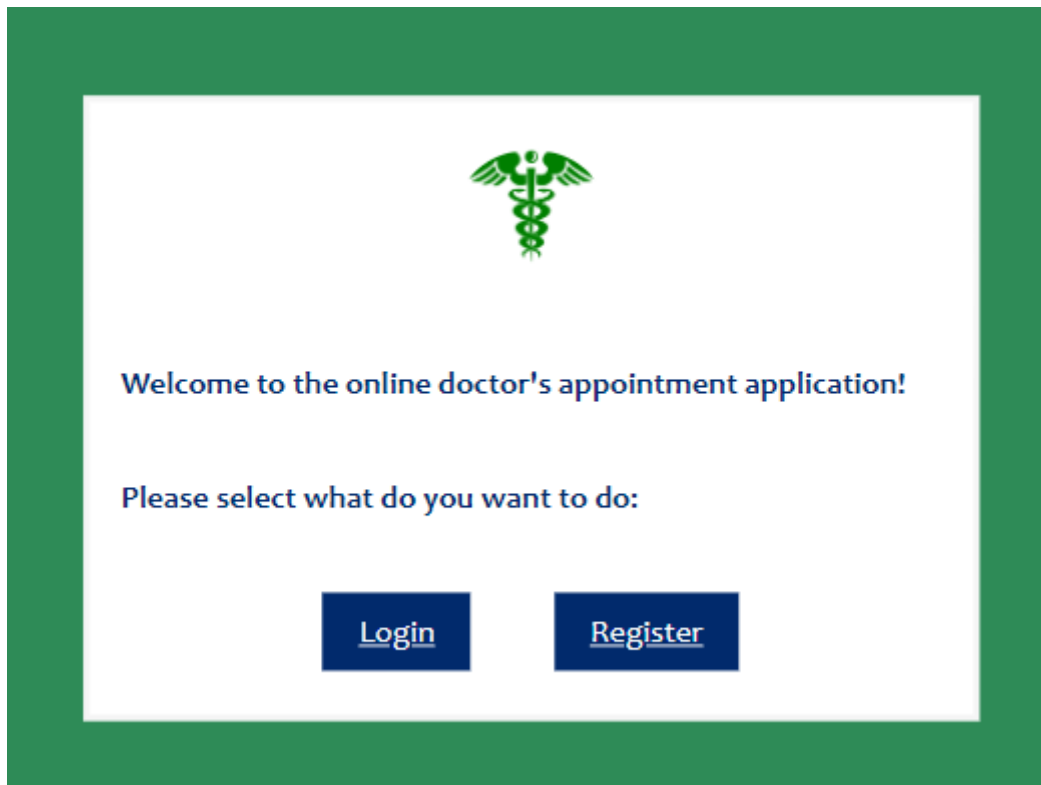
Αλλαγή του server.xml. Πλέον έχουμε αυθεντικοποίηση χρήστη ορίζοντας `clientAuth="true"`.

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**:



Επιλογή πιστοποιητικού χρήστη κατά την εκκίνηση της εφαρμογής



Επιτυχής αυθεντικοποίηση χρήστη

3.2 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.

Category: * ☐ Admin ☐ Patient ☐ Doctor

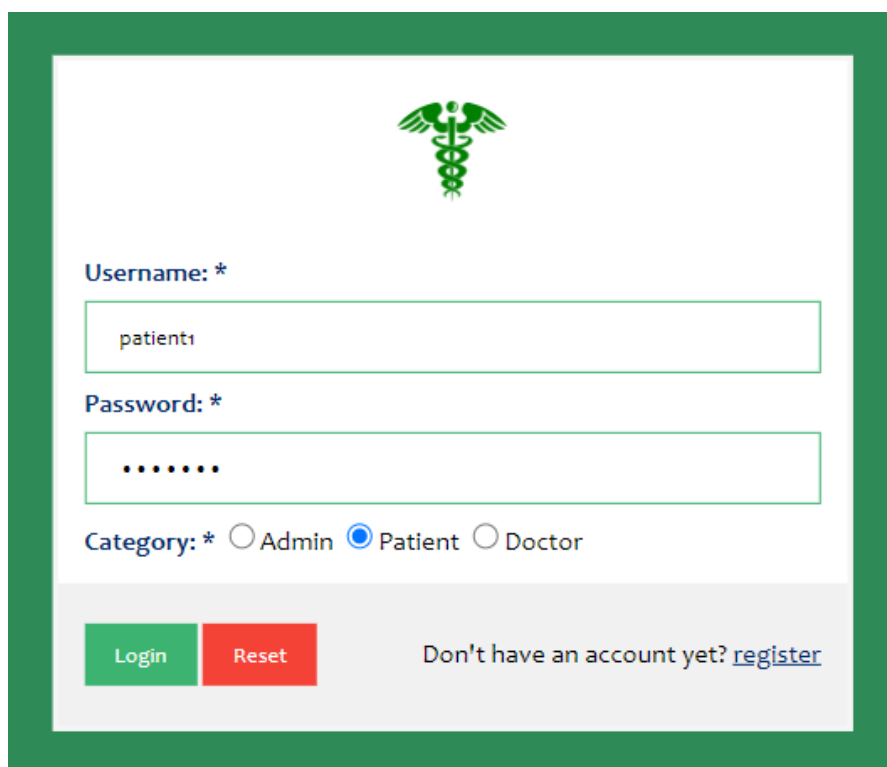
Radio buttons για την επιλογή ρόλου του χρήστη(σελίδα login.jsp)

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:



The screenshot shows a login interface with a green border. At the top center is a green caduceus icon. Below it, the form contains the following fields and controls:

- Username: *** with a text input field containing "patient1".
- Password: *** with a text input field containing seven dots.
- Category: *** with three radio button options: "Admin", "Patient" (which is selected), and "Doctor".
- At the bottom, there are two buttons: a green "Login" button and a red "Reset" button.
- To the right of the buttons, the text "Don't have an account yet? [register](#)" is displayed.

Welcome! Feel free to use the menu, in order to use the application. x




Hello patient! Here are your data:

Username	Name	Surname	Age	AMKA
patient1	Brook	Such	22	10109956787

Please select what do you want to do!

[Appointment history](#) | [Book an appointment](#)
[Scheduled appointments](#) | [Logout](#)

Λάθος επιλογή ρόλου:



Username: *

Password: *

Category: * ☒ Admin ☐ Patient ☐ Doctor

[Login](#) [Reset](#)

Don't have an account yet? [register](#)



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")))
{
```

4. Input filtering και validation

Μετά από την υποβολή **οποιασδήποτε** φόρμας όπου ο χρήστης οφείλει να συμπληρώσει δεδομένα(**username, ΑΜΚΑ, ηλικία, κλπ**) πρέπει να γίνεται **έλεγχος** για το αν τα δεδομένα που εισήγαγε είναι σε **σωστή και ασφαλή μορφή**, για να αποφευχθούν **επιθέσεις τύπου XSS ή SQL Injection** οι οποίες στοχεύουν στην εκτέλεση **μη εξουσιοδοτημένου, κακόβουλου κώδικα**.

Το **Input filtering** στοχεύει στην **αφαίρεση χαρακτήρων(control characters)** οι οποίοι μπορούν να ερμηνευτούν μαζί με το υπόλοιπο κείμενο, ως κάποιου είδους **κώδικα προς εκτέλεση**(πχ ως **XML tag**).

Το **Input validation** στοχεύει στον **έλεγχο** των δεδομένων με **regular expressions** ή με κάποιες **λίστες που περιέχουν τις επιτρεπτές τιμές**, για να **επιβεβαιωθεί η σωστή μορφή** τους πριν περάσουν από **επεξεργασία**(πηγή:

<https://security.stackexchange.com/questions/143923/whats-the-difference-between-escaping-filtering-validating-and-sanitizing>).

Στη συγκεκριμένη εφαρμογή, έχουμε υλοποιήσει μηχανισμούς **input validation** με **regular expressions**, σε **κάθε input πεδίο** κάθε φόρμας, **πριν** εκτελέσουμε την **οποιαδήποτε ενέργεια**. Επίσης, έχουμε συμπεριλάβει πεδία τα οποία περιέχουν **προκαθορισμένες τιμές προς επιλογή** (πχ **ημερολόγιο, ή select boxes**) για να εγγυάται η υποβολή **συγκεκριμένων τύπων δεδομένων**(**ημερομηνίες και συγκεκριμένα strings** αντίστοιχα). Οι έλεγχοι των δεδομένων με **regular expressions**, γίνονται **πάντα από τη μεριά του εξυπηρετητή(Java)** και κατά την **εγγραφή ασθενών** γίνεται επίσης και στη **μεριά του πελάτη (Javascript)**.

Πιο συγκεκριμένα έχουμε κατά την **εγγραφή ασθενών/προσθήκη χρηστών από διαχειριστή**:

- Έλεγχο για το αν το **username** αποτελείται **μόνο από 1-12 λατινικούς ή αριθμητικούς χαρακτήρες**
- Έλεγχο για το αν το **password** αποτελείται από **4 χαρακτήρες και άνω**

- Έλεγχος για το αν το **όνομα** και το **επώνυμο** αποτελούνται **μόνο** από **λατινικούς χαρακτήρες** και **ξεκινούν με κεφαλαίο**
- Έλεγχος για το αν η **ηλικία** είναι **ακέραιος αριθμός** και βρίσκεται στο **διάστημα 1 έως 119**
- Έλεγχος για το αν το **ΑΜΚΑ** του ιατρού/ασθενή αποτελείται **μόνο** από **ακριβώς 11 αριθμητικούς χαρακτήρες**
- Έλεγχος για το αν η **ειδικότητα του ιατρού** βρίσκεται στη λίστα με τις τιμές: **Pathologist, Ophthalmologist, Orthopedist**.

Τα τμήματα κώδικα που υλοποιούν τους παραπάνω ελέγχους είναι τα εξής:

- Έλεγχος στη μεριά του πελάτη κατά την εγγραφή ασθενών(**javascript**):

```
function validation()
{
    // get the values of inputs that user gave in the form
    var first_name = document.getElementById("fn").value;
    var last_name = document.getElementById("ln").value;
    var username = document.getElementById("username").value;
    var password = document.getElementById("password").value;
    var age = document.getElementById("age").value;
    var AMKA = document.getElementById("AMKA").value;
    var error_message = document.getElementById("error_message");

    var text; // set a new variable

    // use a new variable for storing the errors during validation
    error_message.style.padding = "10px";
    error_message.style.display = "block"

    // validation check begins

    // check first name
    if (!/^[A-Z][a-z]+$/.test(first_name))
    {
        text = "First name must contain only letters(at least two)
and should begin with a capital letter. " +
        "No more capital letters are allowed.";
        error_message.innerHTML = text;
        return false;
    }
}
```

```

// check last name
if (!/^[A-Z][a-z]+$/.test(last_name))
{
    text = "Last name must contain only letters(at least two) and should begin with a capital letter. " +
        "No more capital letters are allowed.";
    error_message.innerHTML = text;
    return false;
}

// check username
if (!/^[A-Za-z0-9]{1,12}$/.test(username))
{
    text = "Username length must be between 1 and 12 characters. Only alphabetic and numeric characters are allowed";
    error_message.innerHTML = text;
    return false;
}

// check password
if (password.length < 4)
{
    text = "Password consists of at least 4 characters!";
    error_message.innerHTML = text;
    return false;
}

// check age
if (!/^[0-9]+$/.test(age) || age > 119 || age <= 0)
{
    text = "Age consist of positive integer numbers and is lower than or equal to 119!";
    error_message.innerHTML = text;
    return false;
}

// check AMKA
if (!/^\d{11}$/.test(AMKA))
{
    text = "AMKA should be a 11-digit number!";
    error_message.innerHTML = text;
    return false;
}

error_message.style.display = "none"
return true; // everything is fine, no errors occurred
}

```

Ο παραπάνω κώδικας τρέχει μετά την υποβολή της φόρμας εγγραφής(αφού πατηθεί το κουμπί register). Αν υπάρξει κάποιο σφάλμα, ο χρήστης ενημερώνεται και η φόρμα δεν υποβάλλεται.

- Έλεγχοι στη μεριά του εξυπηρετητή κατά την εγγραφή οποιουδήποτε χρήστη(**Java**):

```
//Checks for all the fields. We use Users.Fail() to provide a plain-
text HTML page to print any errors.

if (!this.getUsername().matches("[A-Za-z0-9]{1,12}"))
{
    Fail(response, "Invalid Username! The username length must be
between 1 and 12 characters. Only alphabetic and numeric characters
are allowed",register_page);
    return;
}

else if (this.getPassword().length() < 4)
{
    Fail(response, "Provide a password with at least 4
characters.",register_page);
    return;
}

else if (!this.getFirstname().matches("[A-Z][a-z]+"))
{
    Fail(response, "Invalid Firstname! All first/last names must
start with one capital letter with succeeding lowercase letters. No
other characters, other than letters, are allowed.",register_page);
    return;
}

else if (!this.getSurname().matches("[A-Z][a-z]+"))
{
    Fail(response, "Invalid Lastname! All first/last names must start
with one capital letter with succeeding lowercase letters. No other
characters, other than letters, are allowed.",register_page);
    return;
}

else if (this.getAge() > 119 || this.getAge() <= 0)
{
    Fail(response, "Invalid Age! A registered age cannot be greater
than 119 years or a non-positive number.",register_page);
    return;
}

else if (this instanceof Patient &&
!((Patient)this).getAMKA().matches("[0-9]{11}") ||
    this instanceof Doctor &&
!((Doctor)this).getAMKA().matches("[0-9]{11}"))
{
    Fail(response, "Invalid AMKA! A social security number must have
exactly 11 digits.",register_page);
    return;
}
```

```

else if (this instanceof Doctor &&
!((Doctor)this).getSpeciality().equals("Pathologist") &&
!((Doctor)this).getSpeciality().equals("Ophthalmologist") &&
!((Doctor)this).getSpeciality().equals("Orthopedist"))
{
    Fail(response, "Doctors can have only one of the following
specialities: Pathologist, Ophthalmologist and
Orthopedist", register_page);
    return;
}

//if we get to this point it means none of the fields are incorrect.
we can execute sql statements safely.
//checking for duplicates in the database

try
{...}

```

this: το αντικείμενο χρήστη (**patient**, **doctor** ή **admin**) το οποίο έχει σαν **attributes** τα **δεδομένα της εγγραφής του**. Σε περίπτωση **σφάλματος** μετά την υποβολή, καλείται η **στατική μέθοδος Fail()** η οποία κάνει **ανακατεύθυνση** τον χρήστη σε μία σελίδα που του εμφανίζεται η **πληροφορία του σφάλματος**.

The image shows a web form for patient registration. At the top, a red banner contains the error message: "Username length must be between 1 and 12 characters. Only alphabetic and numeric characters are allowed". Below this, the text "Hello patient, please create your account!" is displayed. The form contains several input fields, each with a label and an asterisk indicating it is required: "First name: *" (containing "Nick"), "Last name: *" (containing "Geo"), "Username: *" (containing "NickGeo<2>"), "Password: *" (containing four dots), "Age: *" (containing "21"), and "AMKA: *" (containing "12345678112"). At the bottom, there are two buttons: a green "Register" button and a red "Reset" button. To the right of these buttons is a link that says "Already have an account? [login](#)".

Δοκιμή εγγραφής ασθενή με λάθος username(client side έλεγχος)

Add a new doctor to the system:

Username: *

Doctor23

Password: *

• • • •

First name: *

Elmer<123>

Last name: *

Jones

Age: *

37

Speciality: *

Ophthalmologist

Doctor's AMKA: *

23456789021

Add doctor



**Something went
wrong!**

**Invalid Firstname! All first/last names must start
with one capital letter with succeeding
lowercase letters. No other characters, other
than letters, are allowed.**

Δοκιμή προσθήκης ιατρού με λάθος First name(server side έλεγχος)

Ακολουθούν μερικά στιγμιότυπα ακόμα, τα οποία υποδεικνύουν τον **έλεγχο των πεδίων** σε διαφορετικές φόρμες της εφαρμογής:

- Έλεγχος **ΑΜΚΑ**, στην **προβολή ιστορικού ραντεβού ενός ασθενή**:



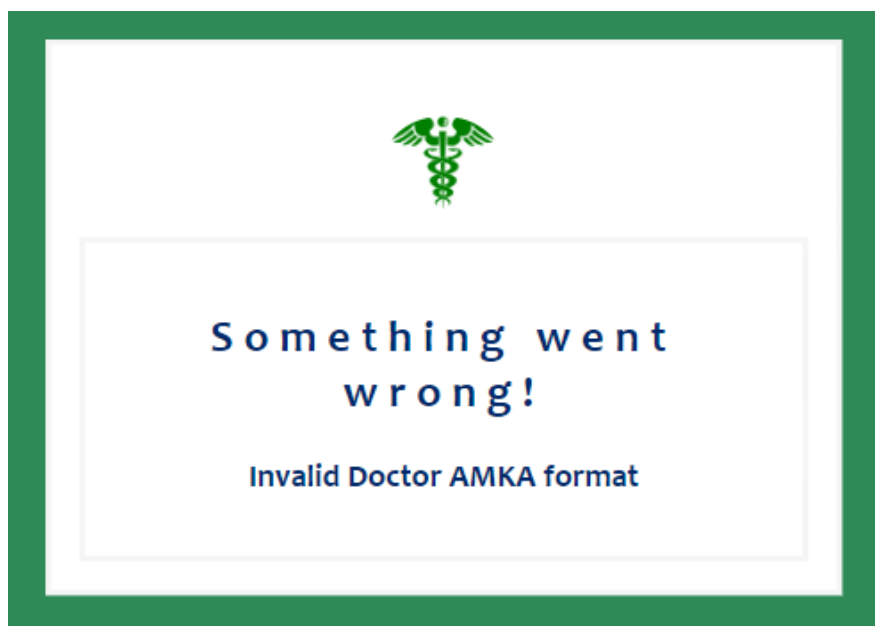
Date	Start time	End time	Doctor AMKA	Doctor name	Doctor surname	Doctor specialty
20-01-2023	17:40:00	18:10:00	19099309918	Kermie	Kegley	Pathologist
20-01-2023	22:32:00	23:02:00	19099309918	Kermie	Kegley	Pathologist

Choose a category to search appointments by: Doctor AMKA ▼

Insert the doctor's AMKA/appointment date/specialty: 123aaaaa

Search

Το ΑΜΚΑ δεν πρέπει να περιέχει άλλους χαρακτήρες πέρα από αριθμητικούς



Ανακατεύθυνση στη σελίδα σφάλματος

- Έλεγχος της ημερομηνίας, στην προβολή προγραμματισμένων ραντεβού ασθενή:



Choose a category to search appointments by:

Insert the doctor's AMKA/appointment date/speciality:

[Search](#)

Η ημερομηνία περιέχει λατινικό γράμμα στο κομμάτι του μήνα



Something went wrong!

Invalid Date (dd-MM-yyyy) format

Ανακατεύθυνση στη σελίδα σφάλματος

- Προκαθορισμένες τιμές για επιλογή(εβδομάδα και μήνας) στην προβολή προγραμματισμένων ραντεβού ιατρού:

Choose an interval to show appointments by:

Choose a week:

January 2023

Week	Su	Mo	Tu	We	Th	Fr	Sa
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21
4	22	23	24	25	26	27	28
5	29	30	31	1	2	3	4
6	5	6	7	8	9	10	11

[Clear](#) [This week](#)

Επιλογή εβδομάδας

Choose an interval to show appointments by:

Choose a month:

2023

Jan	Feb	Mar	Apr
May	Jun	Jul	Aug
Sep	Oct	Nov	Dec

[Clear](#) [This month](#)

Επιλογή μήνα

5. Αυτοματοποιημένος έλεγχος για την εύρεση ευπαθειών ασφάλειας

Στον φάκελο ερώτημα 5 της τελικής εργασίας βρίσκεται η αναφορά του αυτοματοποιημένου ελέγχου που δημιουργήθηκε από το πρόγραμμα **OWASP ZAP**. Ο έλεγχος πραγματοποιήθηκε για το domain name της web εφαρμογής σε **HTTPS** σύνδεση.

Στο συγκεκριμένο report (αρχείο με τίτλο **owasp_zap_report.html**) υπάρχουν τα ακόλουθα δεδομένα:

- a) Περιγραφή των ευπαθειών του συστήματος και βαθμολόγηση της σημαντικότητάς τους
- b) Που υπάρχουν οι ευπάθειες αυτές (σε ποιες σελίδες και σε ποιο κομμάτι)
- c) Πρόταση αντιμετώπισης των προβλημάτων της εφαρμογής
- d) Ενημέρωση για τις ευπάθειες της εφαρμογής (εξηγεί στον αναγνώστη τι είδους προβλήματα είναι αυτές οι ευπάθειες, ώστε να καταλάβει καλύτερα πως να τις αντιμετωπίσει)