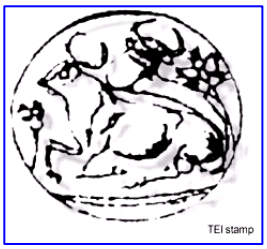


ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Θέμα: *Λογισμικό Ελληνικής Νομοθεσίας*

Ονοματεπώνυμο: *Λυρώνη Χρυσούλα*
Α.Μ: 552
Ημερομηνία:

Επιβλέπουσα: Κορτσιδάκη Αγνή

Περιεχόμενα

Σκοπός πτυχιακής

Σκοπός του συγκεκριμένου έργου είναι να δημιουργηθούν κάποιες υπηρεσίες που να διευκολύνουν τους δικηγόρους στην οργάνωση του νομικού υλικού που χρειάζονται σε κάθε νομική τους υπόθεση. Ανάμεσα στις υπηρεσίες αυτές θα συμπεριλαμβάνεται η αναζήτηση νόμων, η δημιουργία ηλεκτρονικών χαρτοφυλάκων όπου θα διατηρούνται συλλογές νόμων και λοιπόν εγγράφων που χρειάζεται ένας δικηγόρος για μια δίκη, καθώς και η συνεχής ενημέρωση με ανακοινώσεις. Συγκεκριμένα λέγοντας αναζήτηση νόμων αναφερόμαστε σε μία υπηρεσία, όπου ο δικηγόρος θα μπορεί ορίζοντας κάποια κριτήρια να βρίσκει νόμους που σχετίζονται με τα συγκεκριμένα κριτήρια. Αυτό για να γίνει απαιτεί την ύπαρξη μιας οργανωμένης βάσης δεδομένων, όπου θα υπάρχουν οι νόμοι και ενός αλγόριθμου που θα μπορεί να κάνει αναζήτηση χρησιμοποιώντας τα κριτήρια που θα ορίζει ο δικηγόρος. Επιπροσθέτως προκειμένου να διατηρεί το υλικό που συγκεντρώνει ένας δικηγόρος θα υπάρχει η δυνατότητα να δημιουργεί και να διατηρεί ηλεκτρονικούς χαρτοφύλακες. Οι ηλεκτρονικοί χαρτοφύλακες θα είναι η μεταφορά των χαρτοφυλάκων που υπάρχουν στον πραγματικό κόσμο σε μια ψηφιακή προσέγγιση.

Επιπλέον αυτό που είναι σημαντικό στην ανάπτυξη του συστήματος, είναι ότι έχει βασιστεί σε ένα πυρήνα με τέτοιο τρόπο ώστε μελλοντικά να μπορεί να επεκταθεί με την προσθήκη της οποιαδήποτε επιπλέον υπηρεσίας που θα κριθεί απαραίτητη και η οποία θα μπορέσει να διευκολύνει ακόμα περισσότερο τον εκάστοτε χρήστη-δικηγόρο.

Εισαγωγή

Περιγραφή

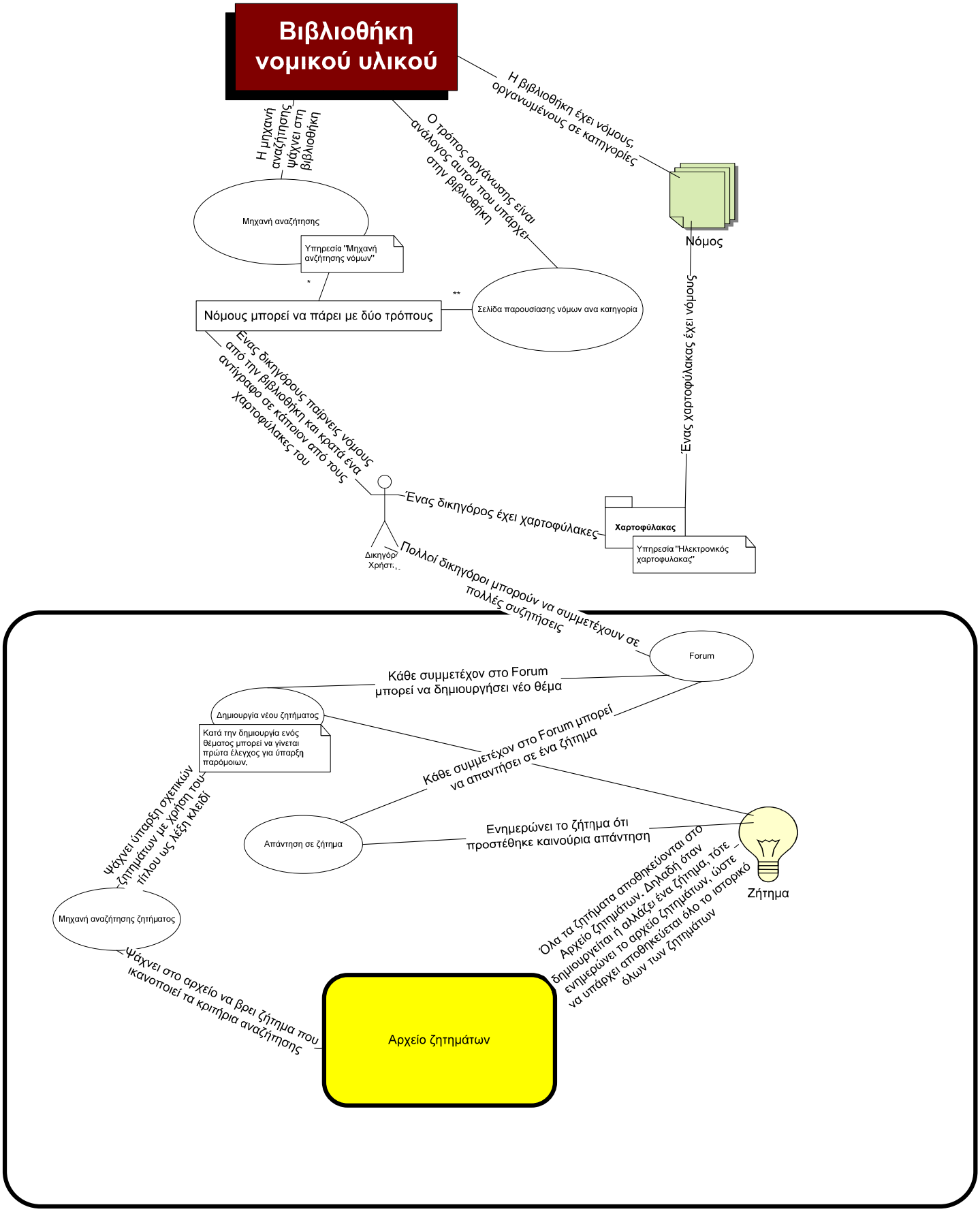
Συνοπτική περιγραφή του τι χρειάζεται για να υλοποιηθεί το συγκεκριμένο έργο

Προκειμένου να επιτευχθεί το eLaw και να παρέχει τις υπηρεσίες που περιγράφονται στον Σκοπό του έργου θα πρέπει να έχουμε κάποια σχεσιακή βάση δεδομένων όπου να τηρούνται πληροφορίες για τα εξής:

1) **Νομικό υλικό**, ώστε να μπορεί να ανακτηθεί το υλικό αυτό για την οποιαδήποτε χρήση, όπως την χρήση για την υπηρεσία «Αναζήτηση νόμων»
2) **Χαρτοφύλακες**, ώστε να διατηρούνται όλοι οι χαρτοφύλακες, όλων των δικηγόρων
3) **Δικηγόροι**, όπου να τηρούμε πλήρη στοιχεία του κάθε δικηγόρου και κάποιο username και password με τα οποία θα συνδέονται.

Ενώ όσο αφορά τις **σελίδες** που πρέπει να υπάρχουν, οι οποίες θα πρέπει να είναι δυναμικές, είναι οι εξής:

1) Κεντρική, από όπου θα γίνεται είσοδος εγγεγραμμένου δικηγόρου ή εγγραφή νέου
2) Σελίδα εγγραφής νέου, όπου θα συμπληρώνει κάποια στοιχεία και αυτά θα χρησιμοποιούνται για να δημιουργηθεί ένας καινούριος χρήστης του συστήματος-δικηγόρος.
3) Σελίδα όπου ο κάθε δικηγόρος βλέπει τα προσωπικά του στοιχεία και το ποιους χαρτοφύλακες έχει δημιουργήσει, εδώ θα μπορεί να κάνει τα εξής:
    a. Δημιουργία νέου χαρτοφύλακα
    b. Διαγραφή υπάρχοντος χαρτοφύλακα
    c. Επισκόπηση υπάρχοντος χαρτοφύλακα
    d. Αλλαγή προσωπικών στοιχείων
    e. Άλλες υπηρεσίες που μπορεί μελλοντικά να δημιουργηθούν
4) Σελίδα δημιουργίας ηλεκτρονικού χαρτοφύλακα, όπου θα δίνεται κάποιος τίτλος και μια μικρή περιγραφή για το νέο χαρτοφύλακα. *Ίσως να υπάρχει και μια επιλογή για το αν θέλει να είναι δημόσιος ή όχι, δηλαδή αν θέλει να δώσει το δικαίωμα σε τρίτους να μάθουν τι έχει στο συγκεκριμένο χαρτοφύλακα.
5) Σελίδα διαχείρισης υλικού του ηλεκτρονικού χαρτοφύλακα, όπου θα μπορεί να προσθέτει, να αφαιρεί ή να κάνει επισκόπηση στο υλικό που έχει στο χαρτοφύλακα του.
6) Σελίδα αναζήτησης νόμων, όπου θα εμφανίζει κάποιες επιλογές-κριτήρια που θα χρησιμοποιούνται για να γίνει η αναζήτηση και θα εμφανίζει αποτελέσματα, τα οποία θα έχουν επιλογή προσθήκης στο χαρτοφύλακα.
7) Σελίδα παρουσίασης νομικού υλικού σε κατηγορίες, όπου δημιουργεί ένα δέντρο πλοήγησης χρησιμοποιώντας την οργάνωση που υπάρχει σε κατηγορίες και υποκατηγορίες των νόμων.



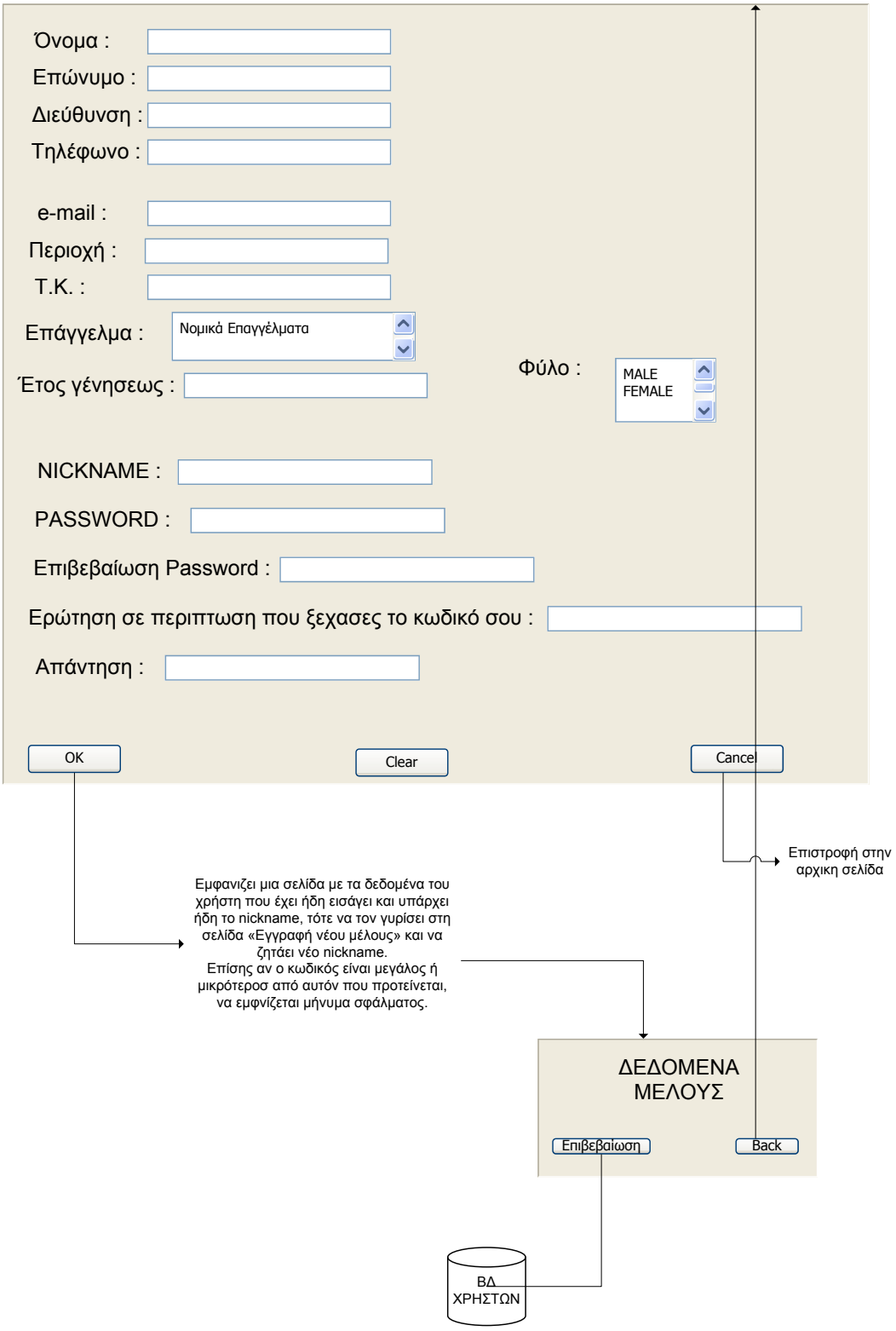
Εικόνα 1 Προσπάθεια για γραφική αναπαράσταση της βασικής δομής

Δημιουργία προτύπων χαμηλής πιστότητας

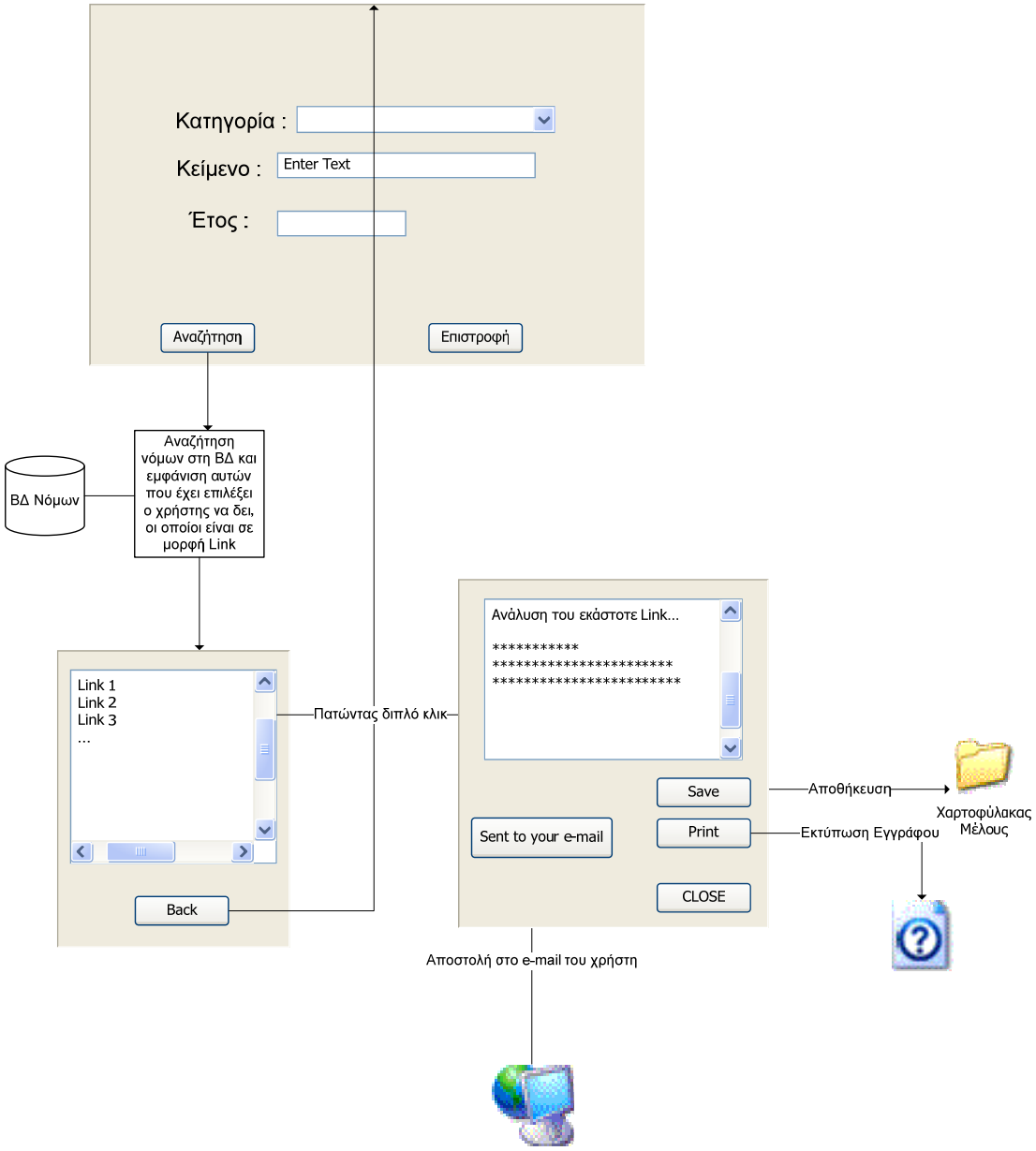
Με τη παράθεση των προτύπων χαμηλής πιστότητας έχουμε τη δυνατότητα μιας γενικότερης ματιάς στην ιδέα της αρχικά γενικότερης ανάπτυξης του συστήματός μας, μιας και μπορούμε με τη δημιουργία αυτών των προτύπων, να έχουμε την περιγραφή των υπηρεσιών που παρέχει το σύστημά, με τη χρήση του εργαλείου της Microsoft, Microsoft office Visio 2003.

Παρακάτω θα δούμε τις υπηρεσίες του συστήματος καθώς και τα πρότυπα με τη χρήση του Visio.

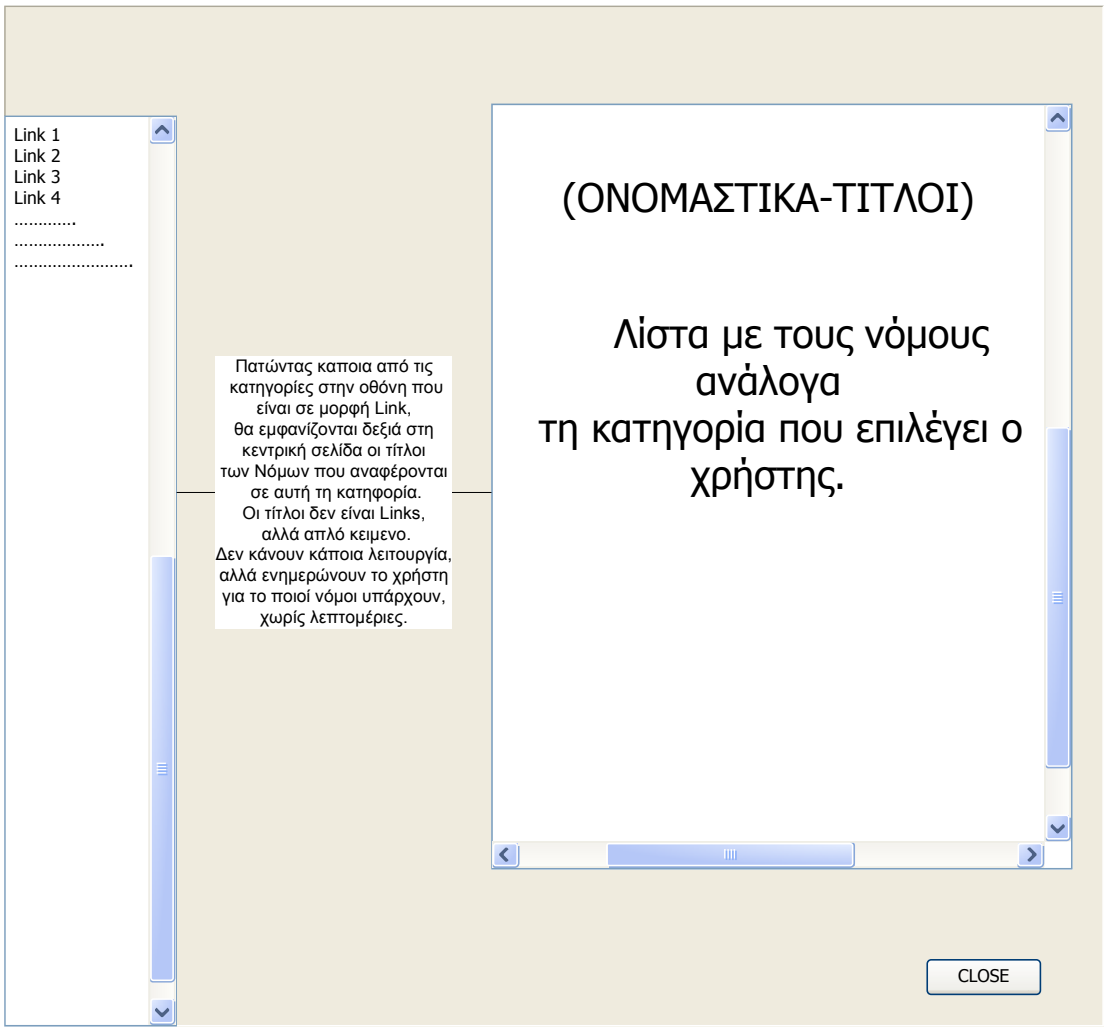
1) Εγγραφή νέου μέλους



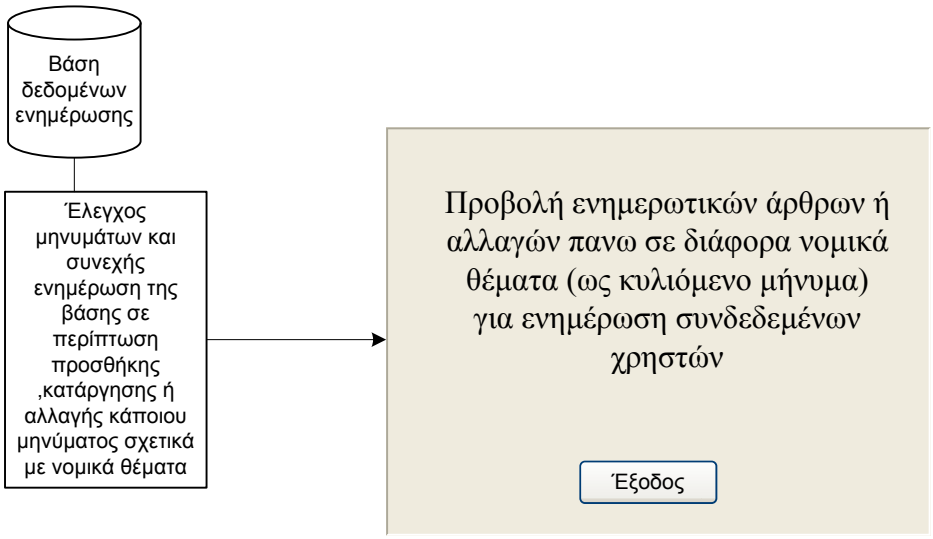
2) Αναζήτηση νόμων



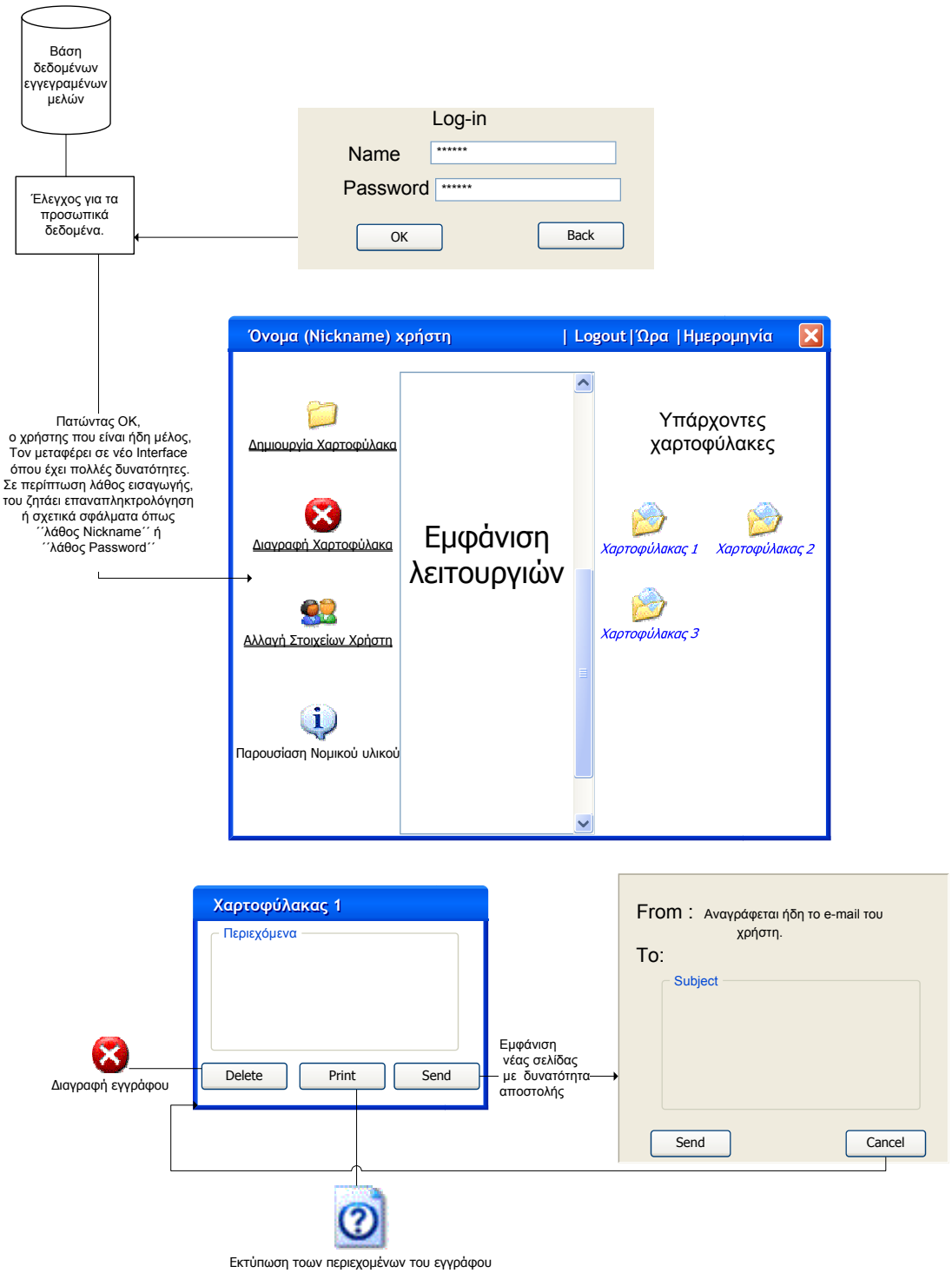
3) Παρουσίαση νομικού υλικού



4) Ανακοινώσεις γεγονότων



5) Log in / Interface μελών



Δημιουργία Διαγραμμάτων Περιπτώσεων Χρήσης (Use Case Diagrams)

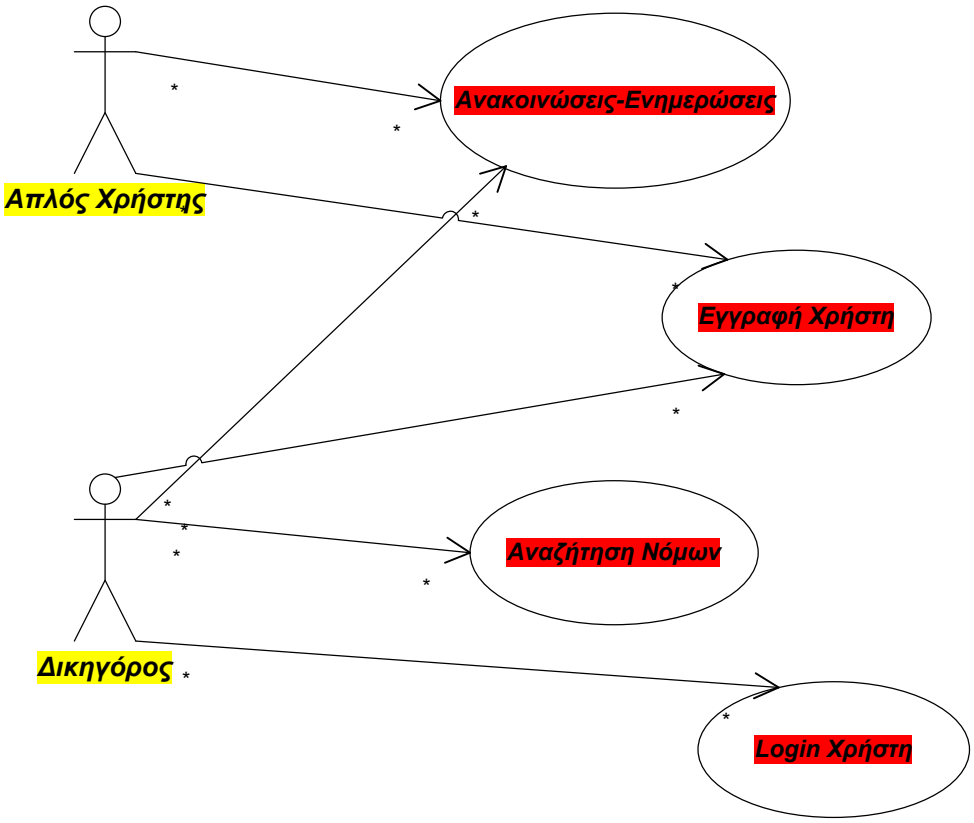
Στο διάγραμμα που ακολουθεί παρουσιάζεται το διάγραμμα περιπτώσεων χρήσης του Δικηγορικού Συλλόγου Αθηνών. Σε αυτό το διάγραμμα έχουμε τη δυνατότητα να δούμε ποιοι είναι οι χρήστες (actors) του συστήματος καθώς και τι υπηρεσίες προσφέρει στους χρήστες που το επισκέπτονται. Οι χρήστες του συστήματος, είναι :

- **Ο απλός χρήστης**
- **Ο χρήστης δικηγόρος**

Ο απλός χρήστης στο σύστημα, εμφανίζεται χωρίς καμία συγκεκριμένη ιδιότητα και έχει πρόσβαση σε ορισμένες από τις υπηρεσίες που του παρέχονται, ενώ ο χρήστης δικηγόρος εμφανίζεται με την ιδιότητα του δικηγόρου και του επιτρέπονται όλες οι υπηρεσίες του συστήματος.

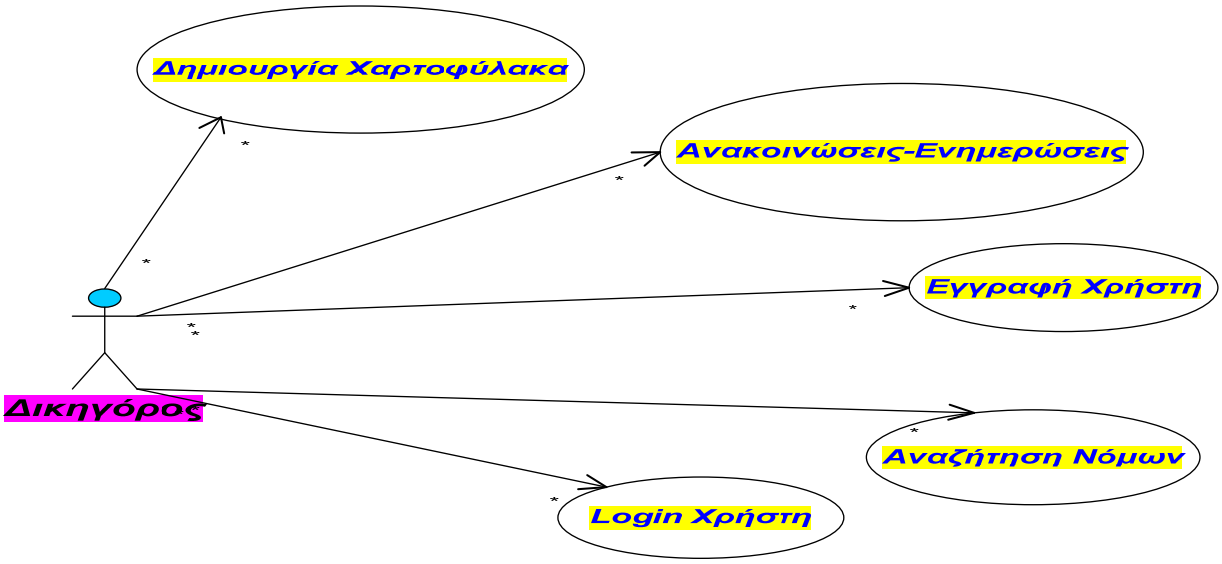
Σε ότι αφορά τις υπηρεσίες του συστήματος, εμφανίζονται παρακάτω:

- **Ανακοινώσεις – Ενημερώσεις**
- **Εγγραφή Χρήστη**
- **Αναζήτηση Νόμων**
- **Login Χρήστη**



Εικόνα 2 Use Case Diagram του Δ.Σ.Α

Το επόμενο διάγραμμα περιπτώσεων χρήσης, αναφέρεται στο διαδικτυακό χώρο του eLAW. Σε αυτό το σύστημα το ρόλο του χρήστη-actor αναλαμβάνει μόνο ο **χρήστης δικηγόρος**. Οι υπηρεσίες που παρέχονται σε αυτό το σύστημα, είναι και οι υπηρεσίες που προαναφέρθηκαν στο site του Δικηγορικού Συλλόγου Αθηνών, με τη διαφορά ότι στο site του eLAW δίδεται η δυνατότητα στο χρήστη-δικηγόρο να δημιουργήσει το δικό του προσωπικό χαρτοφύλακα στο σύστημα και να κρατάει ότι πληροφορίες τον αφορούν.



Εικόνα 3 Use Case Diagram του eLAW

Παρουσίαση υπαρχόντων βάσεων νόμων

Ένα από τα πιο σπουδαία βήματα στην εξέλιξη του διαδικτυακού χώρου, είναι η ανάπτυξη των ιστοχώρων για δικηγόρους. Υπάρχουν σπουδαία sites που έχουν αναπτυχθεί, όπως το site του Δικηγορικού Συλλόγου Αθηνών, το site του lawdB, το law Net κ.α, τα οποία προσφέρουν πολλές δυνατότητες στους χρηστές.

Από τις πιο σημαντικές λειτουργίες τους είναι η αναζήτηση νόμων, η ύπαρξη πρόσφατων ανακοινώσεων, η επικοινωνία μέσω e-mail, η περιήγηση σε άλλα sites, η αναζήτηση επιπλέον γνώσης μέσα από βιβλιοθήκες, καθώς και άλλες δυνατότητες που θα τις συναντήσουμε στην παρουσίαση που ακολουθεί.

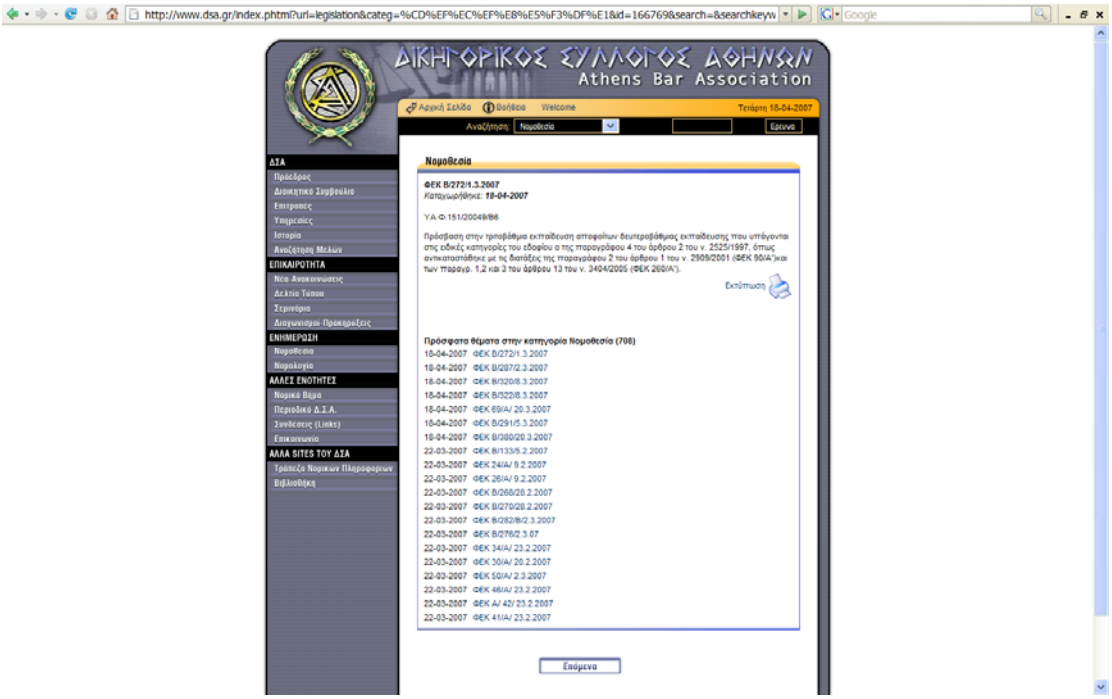
Στον διαδικτυακό χώρο του ***Δικηγορικού Συλλόγου Αθηνών***, μπορεί ο κάθε χρήστης-δικηγόρος να ενημερωθεί σχετικά με νέα γεγονότα, ανακοινώσεις που αφορούν δικαστικά θέματα, να αντλήσει σημαντικές πληροφορίες για υποθέσεις που τον αφορούν καθώς και για διάφορες εξελίξεις στον τομέα.

Η διαδικασία περιήγησης του site είναι εύκολη, καθώς ο τρόπος με τον οποίο έχει δομηθεί, είναι απλός και εύχρηστος. Αριστερά στη σελίδα υπάρχει το μενού πλοήγησης, το οποίο προσφέρει αρκετές ενότητες και υποενότητες. Κάνοντας ένα απλό κλικ σε κάθε μια από αυτές τις ενότητες και υποενότητες, μεταφερόμαστε στη δεξιά πλευρά του παραθύρου όπου παρουσιάζεται αναλυτικά η κάθε μας επιλογή.

Κάνοντας κλικ στην επικεφαλίδα κάποιας ανακοίνωσης, μπορεί ο κάθε χρήστης να δει επιπλέον πληροφορίες που αφορούν την εκάστοτε ανακοίνωση όπως και περισσότερες πληροφορίες σχετικά με την εκάστοτε ανακοίνωση. Επιπροσθέτως, αριστερά από την επικεφαλίδα της ανακοίνωσης, υπάρχει ημερομηνία προβολής της στο site, έτσι ώστε να μπορεί ο χρήστης να γνωρίζει τις πιο πρόσφατες ανακοινώσεις καθώς και τη δυνατότητα εκτύπωσής της.

Αριστερά στη λίστα παρατηρούμε ότι υπάρχει το μενού **«Ενημέρωση»**. Η *Νομοθεσία* και η *Νομολογία* αποτελούν υπομενού αυτής.

Κάνοντας κλικ στη *Νομοθεσία*, ο χρήστης μπορεί να ενημερωθεί σχετικά με πρόσφατα θέματα που αφορούν τη *Νομοθεσία*, τα οποία περιγράφονται επιγραμματικά με αντίστοιχη ημερομηνία. Πατώντας το «νόμο» που μας ενδιαφέρει, εμφανίζεται μια αναλυτική παρουσίαση του νόμου, όπως φαίνεται στο επόμενο παράθυρο καθώς και η δυνατότητα εκτύπωσής του.



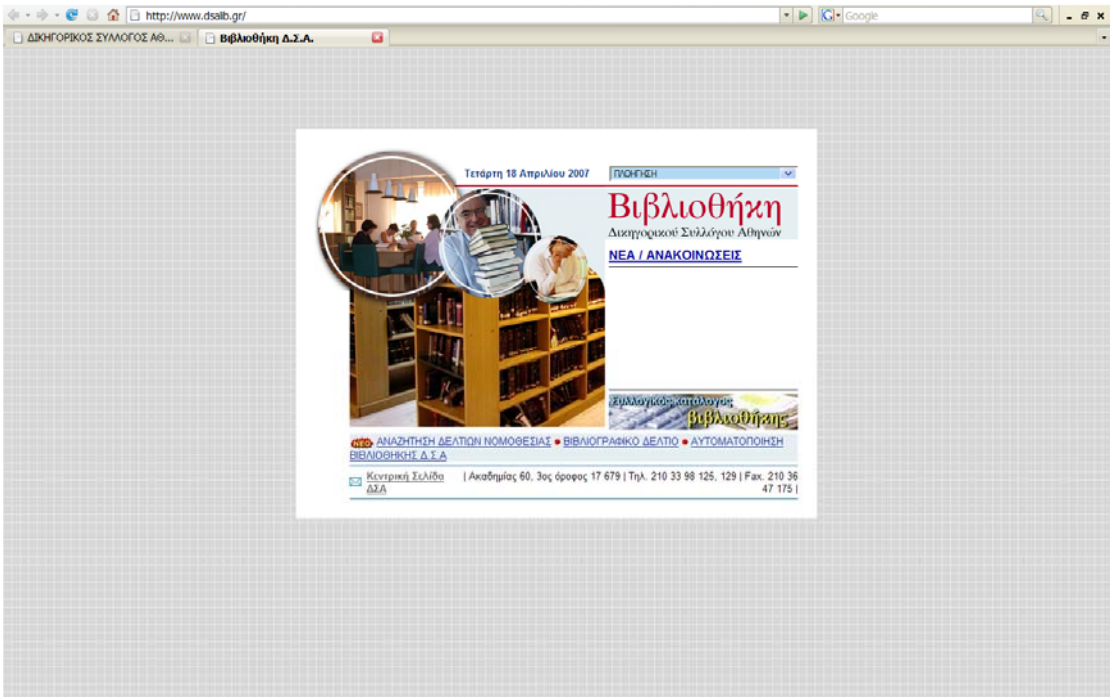
Το μενού πλοήγησης περιλαμβάνει επιπροσθέτως τη γενικού περιεχομένου ενότητα «**Άλλες ενότητες»,** η οποία περιλαμβάνει τα υπομενού *Νομικό Βήμα*, το *Περιοδικό του Δ.Σ.Α*, τις συνδέσεις (*Links*) και την *Επικοινωνία*.

Η υποενότητα που παρουσιάζεται στο επόμενο παράθυρο είναι οι Συνδέσεις *Links*. Υπάρχουν διάφορα links από δικηγορικούς συλλόγους, από διεθνής οργανισμούς και από διάφορους τομείς απασχόλησης.

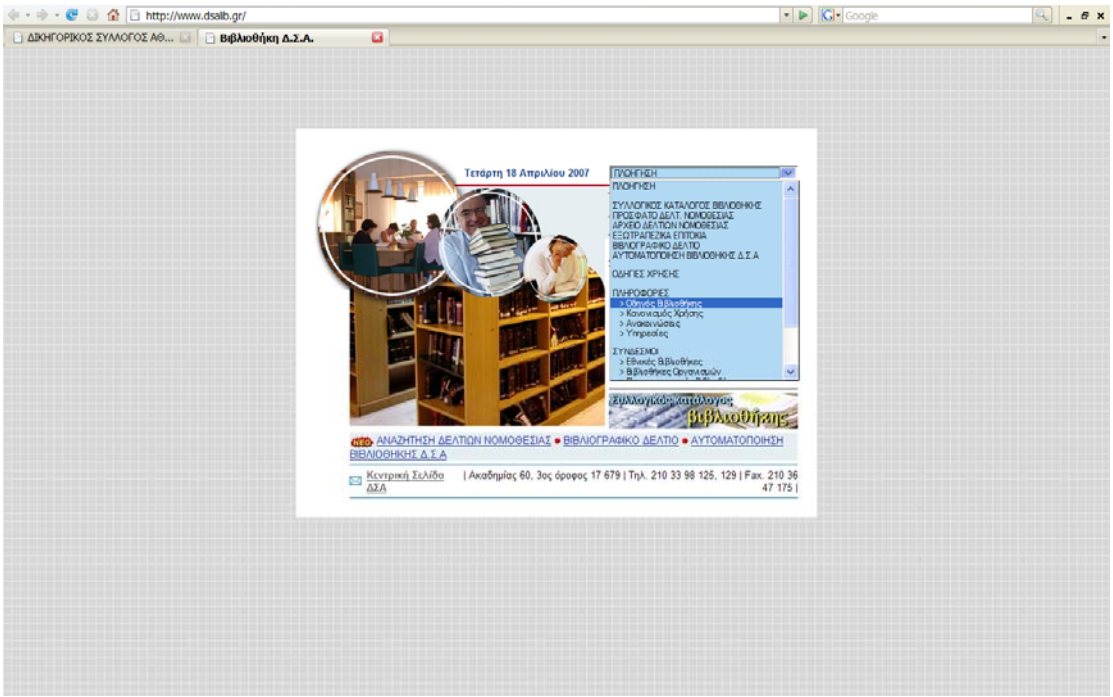
Στη συνέχεια, παρουσιάζεται η υποενότητα της *Επικοινωνίας* η οποία αποτελεί εξίσου μια από της πιο σημαντικές υποενότητες στο site του Δικηγορικού Συλλόγου Αθηνών. Κάνοντας κλικ στο μενού Επικοινωνία εμφανίζεται ένα παράθυρο με μια λίστα από πεδία, τα οποία καλείται ο χρήστης δικηγόρος να συμπληρώσει με τα απαραίτητα στοιχεία του. Πατώντας στο κουμπί της αποστολής, τα στοιχεία αποστέλλονται στο κεντρικό server του site και ο χρήστης-δικηγόρος ενημερώνεται στο mail του με τις τελευταίες εξελίξεις.

Σημαντικό κομμάτι του site του Δικηγορικού Συλλόγου Αθηνών είναι η ενότητα **«Άλλα Sites του Δ.Σ.Α»** το οποίο περιλαμβάνει τις υποενότητες *Τράπεζα Νομικών Πληροφοριών* και *Βιβλιοθήκη*.

Παρακάτω θα ξεναγηθούμε στην υποενότητα της Βιβλιοθήκης.

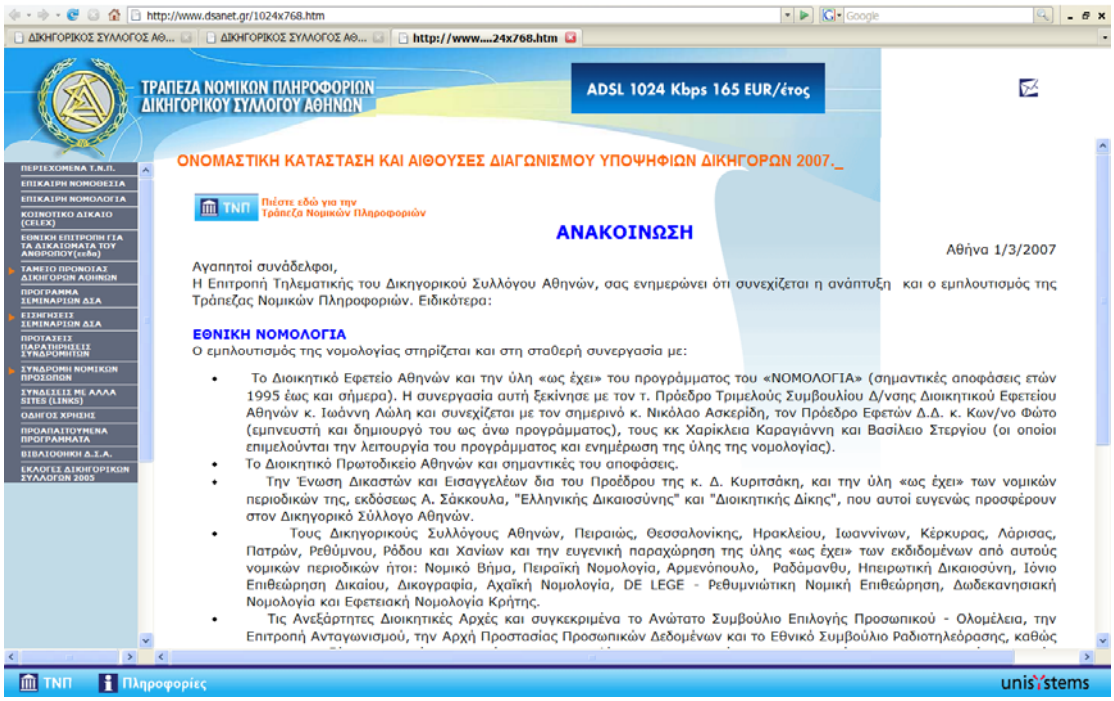


Υπάρχει μια λίστα πλοήγησης που περιλαμβάνει διάφορες πληροφορίες σχετικά με κανονισμούς χρήσης ,οδηγό βιβλιοθήκης , υπηρεσίες και οδηγίες χρήσης. Επιπλέον περιλαμβάνει συνδέσμους με εθνικές βιβλιοθήκες, πανεπιστημιακές βιβλιοθήκες , εκδοτικούς οίκους , Υπουργεία και Γενικές γραμματείες.

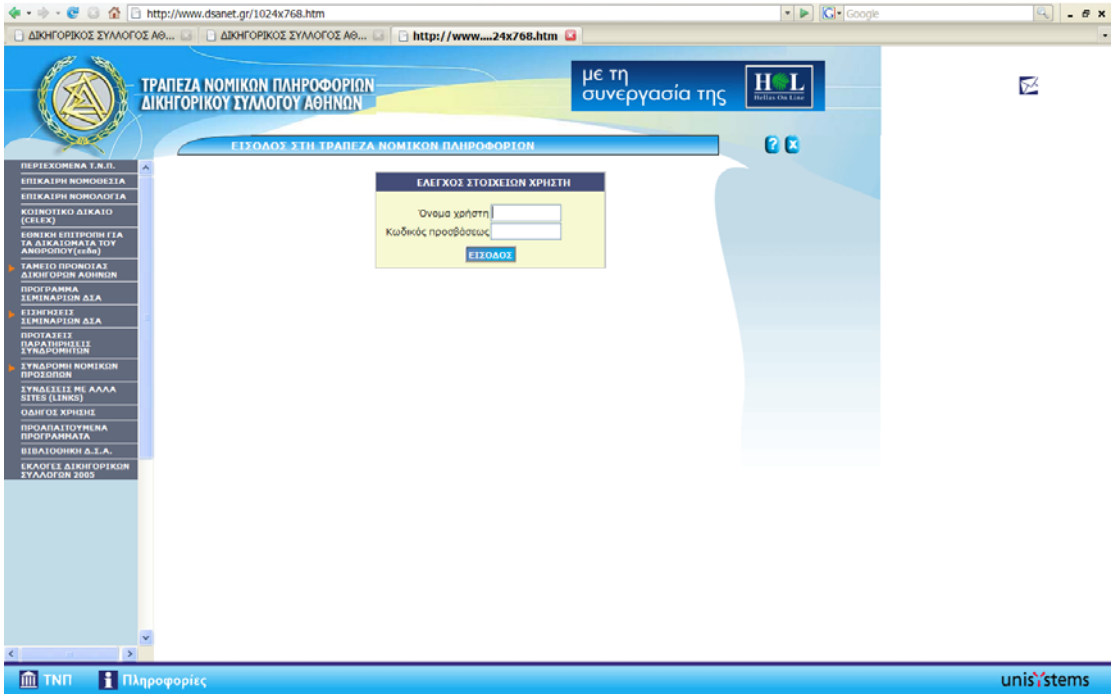


Με αυτό το site , ο χρήστης-δικηγόρος μπορεί εύκολα να έχει πρόσβαση σε σημαντικές πληροφορίες που αφορούν διάφορες υποθέσεις και να αντλεί πληροφορίες γι ’αυτές.

Άλλο αξιοσημείωτο κομμάτι του site του Δικηγορικού Συλλόγου Αθηνών είναι η υποενότητα *Τράπεζα Νομικών πληροφοριών* του Δικηγορικού Συλλόγου Αθηνών .



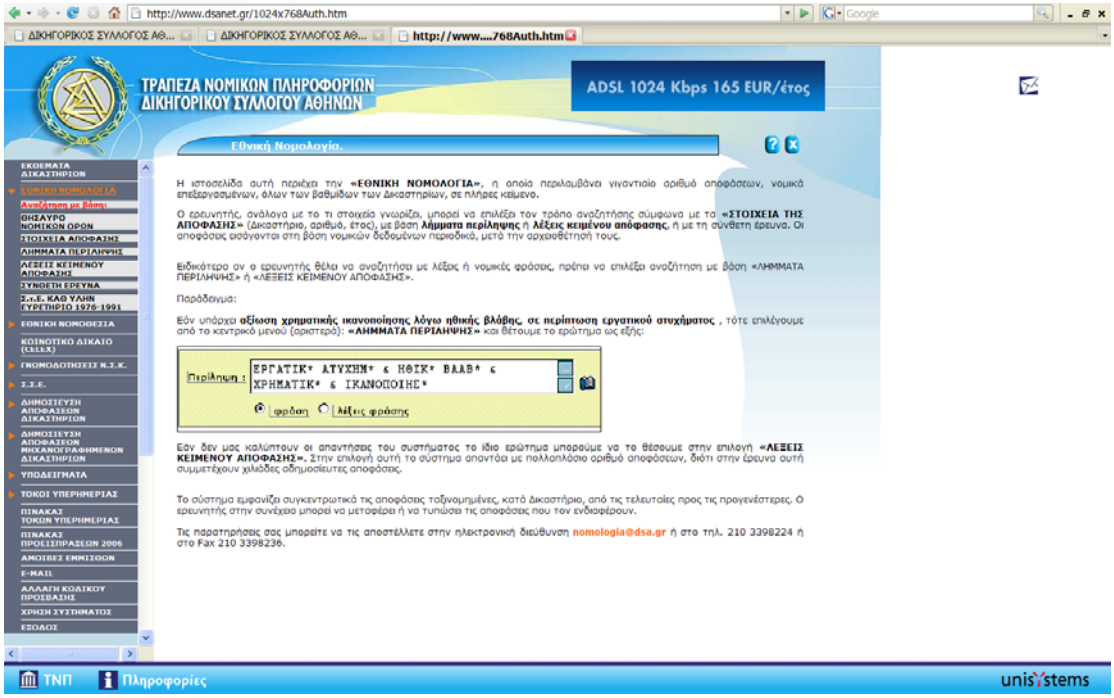
Στη Τράπεζα Νομικών πληροφοριών έχουν πρόσβαση μόνο οι χρήστες – δικηγόροι και όχι κάποιος τυχαίος επισκέπτης του site. Επιλέγοντας το link της Τράπεζας Νομικών πληροφοριών εμφανίζεται το επόμενο παράθυρο .

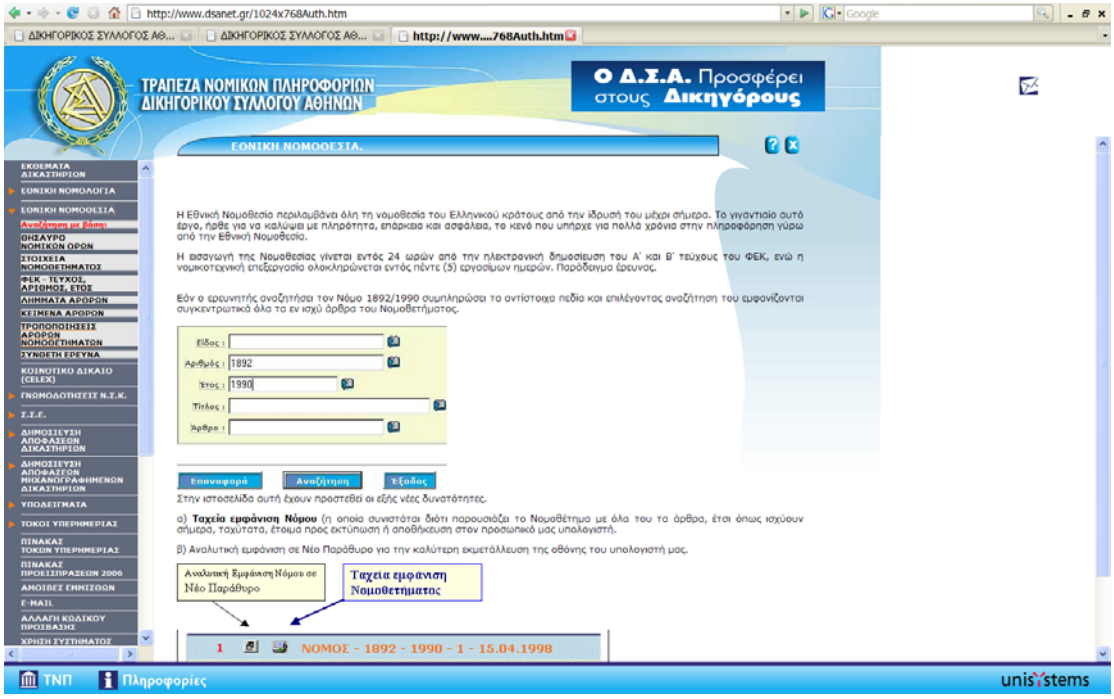


Αριστερά έχουμε τη δυνατότητα να δούμε το μενού το οποίο αναφέρεται σε περιεχόμενα της Τράπεζας Νομικών πληροφοριών, επίκαιρη νομοθεσία και νομολογία ,προγράμματα σεμιναρίων του Δικηγορικού Συλλόγου Αθηνών κ.α. Με την είσοδό μας στην Τράπεζας Νομικών πληροφοριών το πρόγραμμα ζητάει "**Όνομα Χρήστη**" και "**Κωδικό Πρόσβασης**". Συμπληρώνοντας τα στοιχεία και πατώντας **Είσοδος** γίνεται έλεγχος των στοιχείων και μας επιτρέπεται η είσοδος στο site .

Αφού τα στοιχεία μας είναι σωστά, το πρώτο πράγμα που βλέπουμε στην οθόνη μας μετά την είσοδο, είναι μια σύντομη εισαγωγή για την Τράπεζας Νομικών Πληροφοριών.

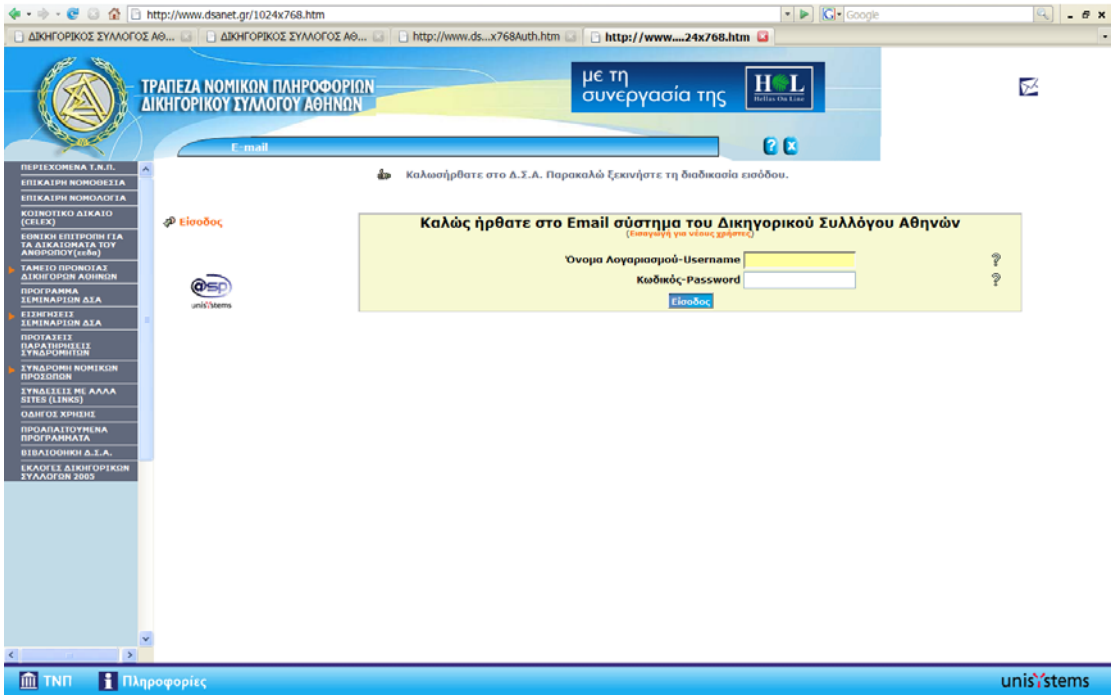
Αριστερά στο μενού επιλογών τόσο στην *«Εθνική Νομολογία»* όσο και στην *«Εθνική Νομοθεσία»* ανοίγει ένα υπομενού για αναζήτηση .Βλέπουμε ότι η αναζήτηση κάποιου νόμου , μπορεί να γίνει με βάση τα παρακάτω όπως φαίνονται στα αντίστοιχα παράθυρα.





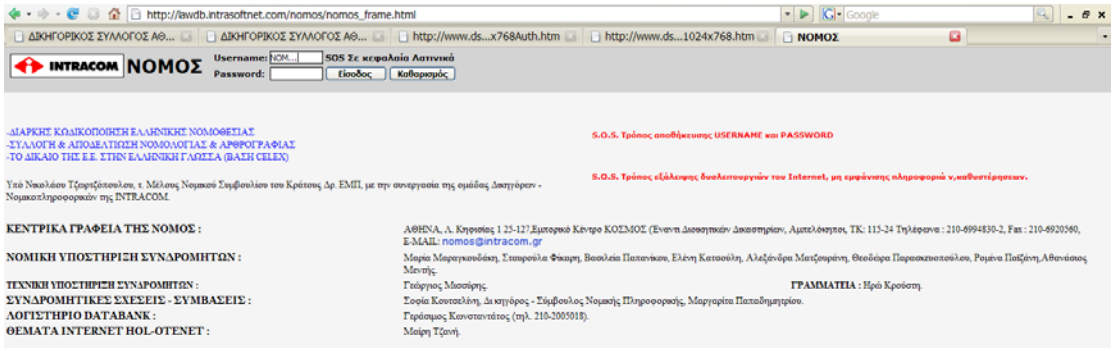
Στα παραπάνω παράθυρα μπορούμε να δούμε πιέζοντας κάθε φορά διαφορετική επιλογή για αναζήτηση , τον τρόπο με τον οποίο γίνεται η αναζήτηση ,καθώς και τα στοιχεία που απαιτούνται για την αναζήτηση.
Σε κάθε παράθυρο υπάρχουν buttons Αναζήτηση Επαναφορά Έξοδος για τις ανάλογες ενέργειες. Στα παρακάτω screen shots γίνεται μια ξενάγηση στις υπόλοιπες επιλογές του site.

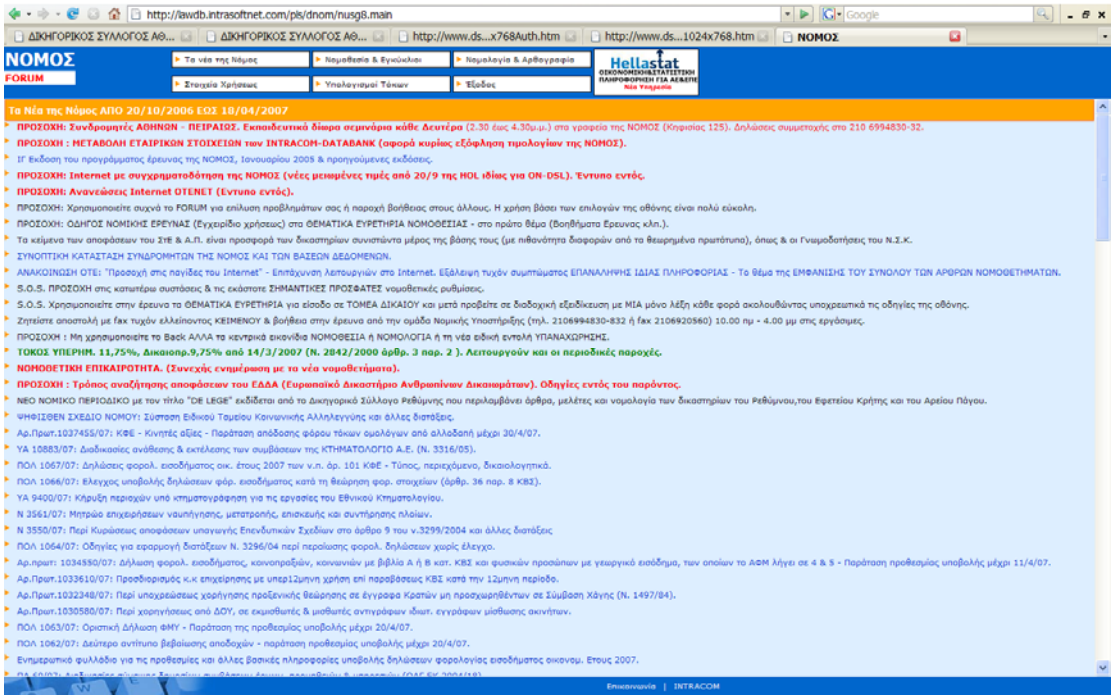
Σημαντική παρατήρηση σ' αυτό το site είναι η επιλογή του ***e-mail*** που δίνει τη δυνατότητα της επικοινωνίας με άλλους δικηγόρους και με το σύστημα του Δικηγορικού Συλλόγου Αθηνών.



Εξίσου σημαντικό όπως έχουμε προαναφέρει είναι και το site του Law dB.
Ξεκινώντας τη περιήγηση στο site του LAW DB μπορούμε να δούμε το κεντρικό μενού που αποτελείται από τις παρακάτω ενότητες «**Νόμος**», «**Φορολογική επιθεώρηση**», «**Τράπεζα ΤΕΕ**», «**Συμβολαιογραφική επιθεώρηση**» και διάφορες πληροφορίες και ενημερώσεις.

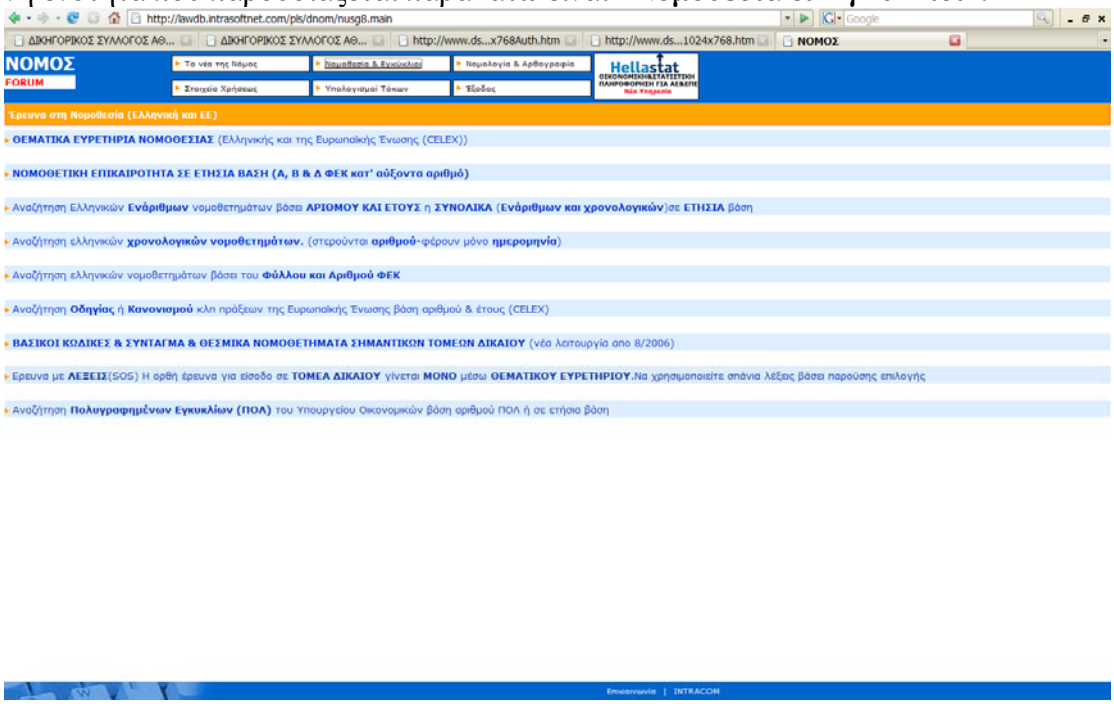
Εκεί που θα εστιάσουμε την προσοχή μας είναι στην επιλογή «**Νόμος**», χωρίς όμως να παραβλέψουμε τις λοιπές επιλογές οι οποίες φαίνονται συνοπτικά παρακάτω.
Πιέζοντας την επιλογή, ζητείται το «**User Name**» και το «**Password**» του εγγεγραμμένο χρήστη. Βάζοντας τα κατάλληλα στοιχεία και πατώντας **Είσοδος**, εισερχόμαστε στο μενού του «**Νόμος**».



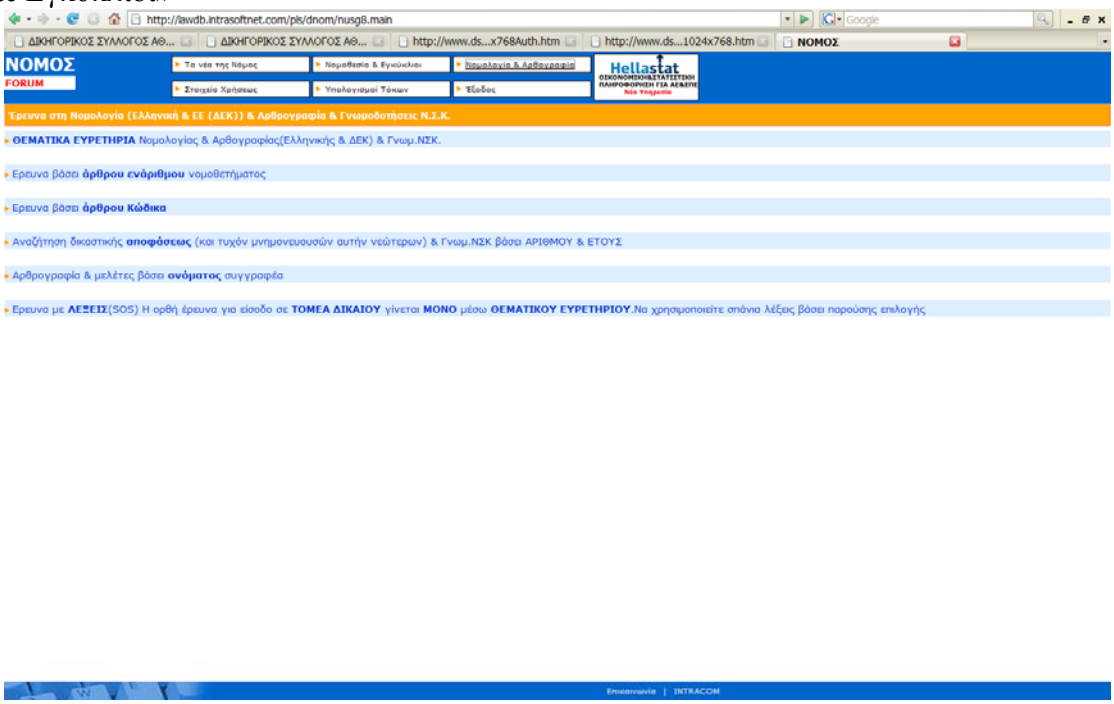


Με παρόμοιο τρόπο γίνεται και σ' αυτό το site η αναζήτηση των «νόμο», καθώς επίσης υπάρχει και δυνατότητα περισσότερων πληροφοριών λειτουργιών και δυνατοτήτων όπως φαίνεται παρακάτω.

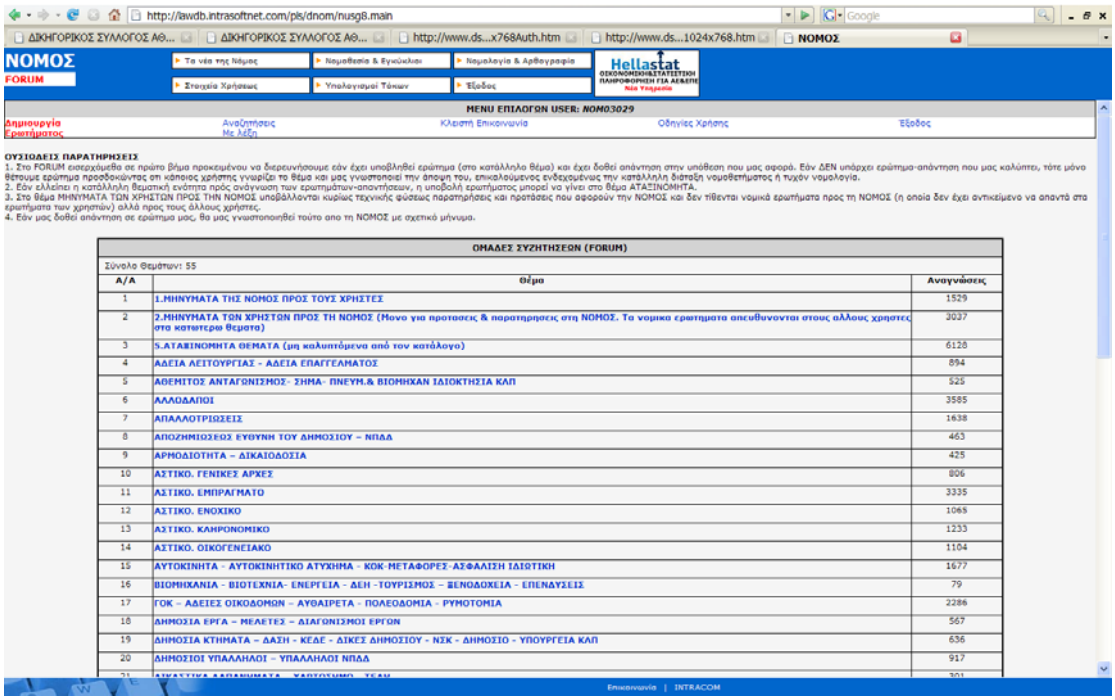
Επόμενη ενότητα που παρουσιάζεται παρακάτω είναι «**Νομοθεσία & Εγκύκλιοι».**



«Νομολογία & Εγκύκλιοι»



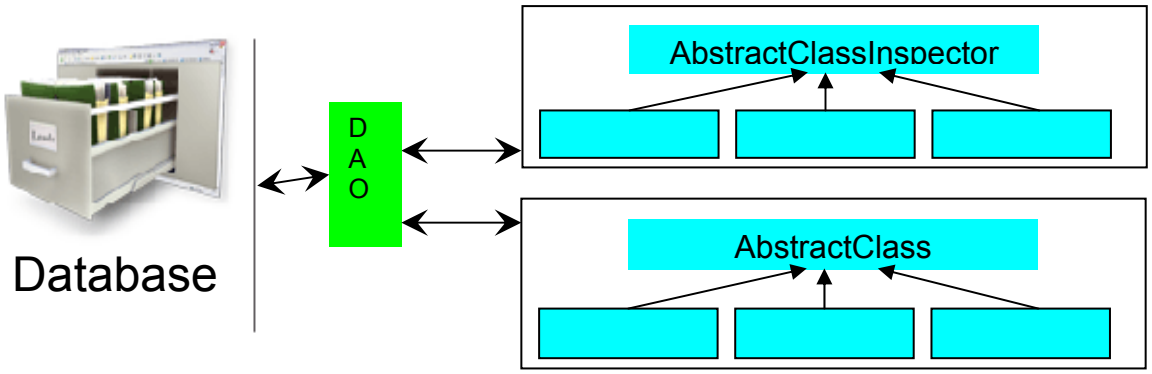
Ακόμα μία αξιοσημείωτη υπηρεσία, είναι και το forum όπως παρουσιάζεται παρακάτω.



Ο ιστοχώρος eLAW

Αρχιτεκτονική

Σημαντικό τμήμα της εργασίας είναι η χρήση του DAO (Database Access Object).Σε αυτό το σημείο, παρατίθεται η ήδη υπάρχουσα, υλοποιημένη αρχιτεκτονική του συστήματος που θα προτείνουμε να χρησιμοποιηθεί στη πτυχιακή εργασία.



Τι είναι η Database

- Η Database είναι η «περιοχή» όπου αποθηκεύεται η οποιαδήποτε απαραίτητη πληροφορία χρειάζεται στο σύστημα. Όπως για παράδειγμα τα στοιχεία των χρηστών της εφαρμογής.

Τι είναι DAO

- Το DAO (Database Access Object) είναι μια διεπαφή που αναλαμβάνει την επικοινωνία μεταξύ της Java και της Database. Δηλαδή μετατρέπει «εντολές» Java σε SQL, που είναι η γλώσσα που «καταλαβαίνει» η Database.
- Το DAO είναι φτιαγμένο έτσι ώστε να μπορεί να λειτουργήσει με οποιαδήποτε Database έχει οδηγό JDBC. (πχ JavaDB, ODBC (άρα MS-Access, MS-SQL), mySQL κ.α)
- Είναι έτσι γραμμένο ώστε να μην αναγκάζει τον προγραμματιστή να χρησιμοποιεί περίπλοκες τυποποιήσεις.
- Γενικά το DAO έχει τις έξεις δυνατότητες:

    - ✓ Αυτόματα μπορεί και δημιουργεί αρχικά το σχήμα της Database (δηλαδή όλους τους απαραίτητους πίνακες), που χρειάζεται για την εφαρμογή. Για να το επιτύχει αυτό χρησιμοποιείται μια απλή τυποποίηση στο πως γράφονται οι κλάσεις της Java που θα χρησιμοποιηθούν στο σύστημα.
    - ✓ Μπορεί να κάνει απλά με κλήση μεθόδων load, save και delete τις ανάλογες λειτουργίες, χωρίς να χρειάζεται να γνωρίζει ο προγραμματιστής SQL, καθώς τα κατάλληλα SQL statements δημιουργούνται αυτόματα, από το DAO

Παράδειγμα δομής στο DAO

Με το ξεκίνημα του προγράμματος η εντολή ***static*** δηλώνει ότι με την έναρξη του προγράμματος θα δημιουργηθεί το κομμάτι του κώδικα που περιλαμβάνεται στο συγκεκριμένο αυτό το τμήμα του κώδικα.

Σημαντικό κομμάτι του κώδικα της δημιουργίας του DAO είναι οι παρακάτω μέθοδοι:

- ***fillProperties*** → Με αυτή τη μέθοδο, δίνεται η δυνατότητα να συμπληρωθεί ένας πίνακας με τις οδηγίες για το πώς θα συνδεθεί με τη DataBase.
- ***loadDBdriver*** → Αυτή η μέθοδος, έχει τη δυνατότητα να φορτώνει τον driver για java για να ξέρει η java πως θα συνδεθεί με τη βάση δεδομένων.

Η δημιουργία της βάσης δεδομένων γίνεται με τη μέθοδο ***createDatabase***. Εξίσου σημαντική είναι και η μέθοδος δημιουργίας πινάκων στη βάση. Αντίστοιχα αυτό γίνεται με τη μέθοδο ***createTables.*** Η μέθοδος παίρνει sql[Array] με τις εντολές για το πώς θα δημιουργηθούν οι πίνακες και τα εκτελεί. Είναι σημαντικό να αναφέρουμε ότι μπορούμε να έχουμε πολλές κλάσεις με ένα ή περισσότερους πίνακες.

Παράδειγμα εφαρμογής

```
import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import com.gorbas.cache.Cache;
import com.gorbas.communication.methods.Multicast;
import com.gorbas.db.events.DataBaseEvent;
import com.gorbas.db.events.DataBaseListener;
import com.gorbas.db.methods.DataTypeMethods;
import com.gorbas.general.error.ApplicationError;
import com.gorbas.general.error.ErrorHandler;
import com.gorbas.general.exceptions.ApplicationException;
import com.gorbas.internationalization.ApplicationMessageMethods;
import com.gorbas.internationalization.model.ApplicationMessage;
import com.gorbas.model.AbstractClass;
import com.gorbas.model.AbstractClassInspector;
import com.gorbas.model.AbstractClassMember;
```

```
/**
 * <H1>DAO (Database Access Object)</H1>
 * <BR>
 * <H2>What is DAO?</H2>
 * <p>
 * Dao is is the Class via to which the Application can access the DataBase <BR>
 * <H2>What can DAO do?</H2>
 * <p>
 * Dao can load/save/delete AbstractClass instances from the DB. And can also
 * create the appropriate Schema that is needed to store those kind of
 * AbstractClass
 *
 * @version 1.0
 * @author Gorbas
 *
 */
public class DAO {

	static {
		fillDBProperties();
		setDBSystemDir();
		loadDatabaseDriver();
		if (!dbExists()) {
			System.out.println("CREATING DATABASE");
			createDatabase();
		} else {
			System.out.println("DATABASE ALREADY EXISTS");
		}
		try {
			Connection dbConnection = DriverManager.getConnection(
					getDatabaseUrl(), getDBProperties());
			createTables(dbConnection);
			Cache.initialize();
		} catch (SQLException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}

	private static final String SCHEMA = "db";

	// private static final String DB_DRIVER =
	// "org.apache.derby.jdbc.EmbeddedDriver";

	// private static final String DB_DRIVER = "com.mysql.jdbc.Driver";
	private static final String DB_DRIVER = "org.postgresql.Driver";

	private static final String USER_NAME = "ROOT";
```

```
	private static final String PASSWORD = "";// "2602";

	private static final String DATABASE = "new_db";

	// private static final String DB_URL = "jdbc:derby";

	// private static final String DB_URL = "jdbc:mysql";

	private static final String DB_URL = "jdbc:postgresql";

	private static final String ROOT_DATABASE = "postgres";

	// private static final String ROOT_DATABASE = "mysql";

	// private static final String DB = "";

	private static final String DB = "//127.0.0.1:5432/";

	/*
	 * + "&user=" + USER_NAME // + "&password=" + // PASSWORD;
	 */
	// the root directory for the database
	private static final String userHomeDir = ".././db";

	private static Connection dbConnection;

	private static boolean isConnected;

	/**
	 * @param name
	 * @throws ApplicationException
	 */
	public static void drop(Class<AbstractClass> clazz) throws
ApplicationException {
		try {
			AbstractClass tmp = clazz.newInstance();
			String dropSql = "DROP TABLE " + tmp.TableName();
			if (dbConnection == null) {
				connect();
			}
			Statement stmt = dbConnection.createStatement();
			stmt.execute(dropSql);
			String deleteFromAbstractClassMember="DELETE FROM
ABSTRACTCLASSMEMBER WHERE ParentClass='"+clazz.getName()+"'";
			stmt.execute(deleteFromAbstractClassMember);
		} catch (Throwable thr) {
			throw new ApplicationException("DAO.drop", thr);
		}
	}
```

```
	/**
	 * Possible inner exceptions <table>
	 * <tr>
	 * <td>code</td>
	 * <td>meaning</td>
	 * <td>IS_NULL</td>
	 * <td>AbstractClass or AbstractClass.Id is null</td>
	 * <td>ANY_OR_NULL</td>
	 * <td>AbstractClass.Id is ANY or NON</td>
	 * </tr>
	 * </table>
	 *
	 * @param object
	 * @return
	 * @throws ApplicationException
	 */
	public static Boolean save(AbstractClass object)
			throws ApplicationException {
		System.out.println(object.getDebugString());
		try {
			if (object == null || object.getId() == null) {
				throw new Exception("IS_NULL");
			} else if (object.getId() ==
AbstractClass.ID_OF_ANY_ABSTRACT_CLASS
					|| object.getId() ==
AbstractClass.ID_OF_NON_ABSTRACT_CLASS) {
				throw new Exception("ANY_OR_NON");
			}
			boolean isInsert = true;
			if (dbConnection == null) {
				connect();
			}
			PreparedStatement statement;
			if (isInsert = (load(object.getClass(), object.getId()) == null)) {
				System.out.println(object.InsertSQL());
				statement =
dbConnection.prepareStatement(object.InsertSQL());
				DatabaseChanged(BEFORE_INSERT, object);
			} else {
				System.out.println(object.UpdateSQL());
				String updateSQL = object.UpdateSQL();
				statement = null;
				if (updateSQL != null)
					statement =
dbConnection.prepareStatement(object
							.UpdateSQL());
				DatabaseChanged(BEFORE_UPDATE, object);
			}
			object.applyInsertUpdateParameters(statement);
```

```
                boolean success = statement == null
                        || (statement.executeUpdate() > 0);
                if (success) {
                    AbstractClassMember[] listMembers =
object.getListMembers();
                    for (int k = 0; k < listMembers.length; k++) {
                        boolean suc = DAO.saveListMember(object,
listMembers[k]);
                        success = success && suc;
                    }
                    if (success) {
                        if (isInsert) {
                            DatabaseChanged(AFTER_INSERT,
object);
                            try {
                                fireDataChanged(new
DataBaseEvent(

        DataBaseEvent.INSERT,
                                                    new
AbstractClass[] { object }));
                            } catch (ApplicationException exc) {
                                ErrorHandler.AddError(new
ApplicationError(exc,

        "DAO.save(AbstractClass object)"));
                            }
                        } else {
                            DatabaseChanged(AFTER_UPDATE,
object);
                            try {
                                fireDataChanged(new
DataBaseEvent(

        DataBaseEvent.UPDATE,
                                                    new
AbstractClass[] { object }));
                            } catch (ApplicationException exc) {
                                ErrorHandler.AddError(new
ApplicationError(exc,

        "DAO.save(AbstractClass object)"));
                            }
                        }
                    }
                    return success;

                } catch (Exception e) {
                    if (e.getMessage() == null) {
```

```
                        e.printStackTrace();
                    } else if (e.getMessage().equals("IS_NULL")) {
                        throw new ApplicationException(
                                "AbstractClass or AbstractClass.Id is
null");
                    } else if (e.getMessage().equals("ANY_OR_NON")) {
                        throw new ApplicationException("AbstractClass.Id is
ANY or NON");
                    } else {
                        if(!ApplicationMessage.class.isInstance(object)){
                        ApplicationException exc = new
ApplicationException(e
                                .getMessage());
                        exc.initCause(e);
                        throw exc;
                        }
                        e.printStackTrace();
                    }
                }
                return false;
        }

        /**
         * Loads the instance of AbstractObject that has the given id
         *
         * @param clazz
         * @param id
         * @return
         */
        public static AbstractClass load(Class<? extends AbstractClass> clazz,
                    String id) {
                if (clazz == null || id == null
                        || id ==
AbstractClass.ID_OF_ANY_ABSTRACT_CLASS
                        || id ==
AbstractClass.ID_OF_NON_ABSTRACT_CLASS)
                    return null;
                AbstractClass cache = Cache.get(id);
                if (cache != null)
                    return cache;

                if (dbConnection == null) {
                    connect();
                }
                try {
                    AbstractClass object = clazz.newInstance();
                    PreparedStatement statement =
dbConnection.prepareStatement(object
                            .getSelectByIdSQL());
                    statement.setString(1, id);
```

```
                        System.out.println(object.getSelectByIdSQL());
                        System.out.println("id=" + id);
                        ResultSet rs = statement.executeQuery();
                        object = object.ResultSet2AbstractClass(rs);
                        if (object != null) {
                                AbstractClassMember[] listMembers =
object.getListMembers();
                                if (listMembers != null) {
                                        for (int k = 0; k < listMembers.length; k++) {

        object.setMemberValue(listMembers[k].getMemberName(),
                                                        loadListMember(object,
listMembers[k]));
                                        }
                                }
                        }
                        if (rs != null) {
                                if (object != null)
                                        DatabaseChanged(AFTER_LOAD, object);
                                return object;
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                } catch (InstantiationException e) {
                        e.printStackTrace();
                } catch (IllegalAccessException e) {
                        e.printStackTrace();
                } catch (ApplicationException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return null;
        }

        /**
         * @param sql
         * @param parameters
         * @return
         * @throws ApplicationException
         */
        public static ResultSet query(String sql, Object[] parameters)
                        throws ApplicationException {
                if (dbConnection == null) {
                        connect();
                }
                try {
                        PreparedStatement statement =
dbConnection.prepareStatement(sql);
                        if (parameters != null)
                                for (int i = 1; i <= parameters.length; i++)
```

```
                                        statement.setObject(i, parameters);
                        return statement.executeQuery();
                } catch (Throwable e) {
                        throw new ApplicationException("query", e);
                }
        }

        public static LoadListQuery<AbstractClass> search(
                        AbstractClass abstractClass, int maxResults,
                        List<Integer> notSelectedCriteria) {
                return search(abstractClass, maxResults, notSelectedCriteria, false);
        }

        /**
         * Returns a list of AbstractClass based on the given criteria
         *
         * @param abstractClass
         * @param maxResults
         * @param notSelectedCriteria
         */
        public static LoadListQuery<AbstractClass> search(
                        AbstractClass abstractClass, int maxResults,
                        List<Integer> notSelectedCriteria, boolean isExact) {
                if (abstractClass == null)
                        return null;
                StringBuffer whereClause = null;
                List whereClauseParameters = null;

                for (int i = 0; i < abstractClass.MemberCount(); i++) {
                        Class memberClass = abstractClass.MemberClass(i);
                        boolean isSearchCriterion = (!notSelectedCriteria
                                        .contains(new Integer(i)))
                                        && abstractClass.isSearchCriterion(i);
                        boolean isNull = abstractClass.MemberValue(i) == null;
                        boolean isBoolean =
Boolean.class.isAssignableFrom(memberClass);
                        boolean isNumber =
Number.class.isAssignableFrom(memberClass);
                        boolean isAbstractClass = AbstractClass.class
                                        .isAssignableFrom(memberClass);
                        boolean isAnyAbstractClass = isAbstractClass
                                        && !isNull
                                        && ((AbstractClass)
abstractClass.MemberValue(i)).getId() != null
                                        && ((AbstractClass)
abstractClass.MemberValue(i)).getId()

        .equals(AbstractClass.ID_OF_ANY_ABSTRACT_CLASS);
                        if (isSearchCriterion
```

```
                                        && (!((isNull && (isBoolean || isNumber ||
isAbstractClass)) || isAnyAbstractClass))) {
                                if (whereClauseParameters == null)
                                        whereClauseParameters = new ArrayList();
                                if (whereClause == null)
                                        whereClause = new StringBuffer();
                                else
                                        whereClause.append(" and ");
                                boolean isString =
String.class.isAssignableFrom(abstractClass
                                                .MemberClass(i));

                                whereClause.append(isString ? " UPPER(" : "");
                                whereClause.append(abstractClass.MemberName(i));
                                whereClause.append(isString ? ") " : "");
                                whereClause.append(isString && !isExact ? " like
UPPER(?)"
                                                : "= ? ");

                                whereClauseParameters.add(isString && !isExact ?
"%"
                                                +
AbstractClass.MemberValueForField(abstractClass, i)
                                                + "%" :
AbstractClass.MemberValueForField(
                                                abstractClass, i));

                        }
                }
                return new LoadListQuery<AbstractClass>(abstractClass, -1,
maxResults,
                                whereClause == null ? null : whereClause.toString(),
                                whereClauseParameters == null ? null :
whereClauseParameters
                                                .toArray(), null);
        }

        /**
         * Loads all the instances of the clazz
         *
         * @param clazz
         * @return
         * @throws ApplicationException
         */
        public static LoadListQuery<AbstractClass> loadList(
                        Class<? extends AbstractClass> clazz) {
                return loadList(clazz, -1, -1);
        }

        public static LoadListQuery<AbstractClass> loadList(
```

```
                        Class<? extends AbstractClass> clazz, int firstRow, int
maxRows) {
                return loadList(clazz, firstRow, maxRows, null, null, null);
        }

        public static LoadListQuery<AbstractClass> loadList(
                        Class<? extends AbstractClass> clazz, int firstRow, int
maxRows,
                        String customWhereClause, Object[]
customWhereClauseParameters)
                        throws ApplicationException {
                return loadList(clazz, firstRow, maxRows, customWhereClause,
                                customWhereClauseParameters, null);
        }

        /**
         *
         * @param clazz
         * @param firstRow
         * @param maxRows
         * @param customWhereClause
         * @param customWhereClauseParameters
         * @param orderClause
         * @return
         */
        public static LoadListQuery<AbstractClass> loadList(
                        Class<? extends AbstractClass> clazz, int firstRow, int
maxRows,
                        String customWhereClause, Object[]
customWhereClauseParameters,
                        String orderClause) {
                try {
                        AbstractClass obj = (AbstractClass) clazz.newInstance();
                        return (new LoadListQuery<AbstractClass>(obj, firstRow,
maxRows,
                                        customWhereClause,
customWhereClauseParameters, orderClause));
                } catch (Exception e) {
                        ErrorHandler.AddError(new ApplicationError(e,

        "DAO.loadList(Class,int,int,String,Object[],String)"));
                }
                return null;
        }

        /**
         * Loads all the instances of AbstractClass as a List
         *
         * @param object
         * @param firstRow
```

```
	 *		if firstRow<=0 then it doesn't matter!
	 * @param maxResults
	 *		if
	 * @param customWhereClause
	 *		is any additional whereClause that you would like to have (can
	 *		also be null)
	 * @param customWhereClauseParameters
	 *		is an object array of the appropriate parameters for the
	 *		customWhereClause (can also be null)
	 * @param orderClause
	 *		for instance "name asc, phone desc" (can also be null)
	 * @return List of AbstractClass instances that were loaded
	 */
	public static List<AbstractClass> loadList(AbstractClass object,
			int firstRow, int maxRows, String customWhereClause,
			Object[] customWhereClauseParameters, String orderClause)
{
		if (dbConnection == null) {
			connect();
		}
		try {
			boolean hasWhereClause = false;
			int parameterCounter = 0;
			String sql = object.getSelectAllSQL();
			if (customWhereClause != null) {
				if (hasWhereClause) {
					sql += " and " + customWhereClause;
					hasWhereClause = true;
				} else
					sql += " WHERE " + customWhereClause;
			}
			if ((orderClause != null) && (!orderClause.equals(""))) {
				sql += " order by " + orderClause;
			} else {
				sql += " order by id ";
			}
			if (firstRow > 0) {
				/*
				 * String rownum="rownum > ?";//In case of Oracle
style if
				 * (!hasWhereClause) { sql += " WHERE "+rownum ;
hasWhereClause =
				 * true; } else { sql += " and "+rownum; }
				 */
				String rownum = "\n OFFSET  ?";// In case of postgres
				sql += "\n" + rownum;
			}
			System.out.println(sql);
			PreparedStatement statement =
dbConnection.prepareStatement(sql);
```

```
                    if (maxRows > 0)
                            statement.setMaxRows(maxRows);
                    if ((customWhereClause != null)
                                    && (customWhereClauseParameters != null)) {
                            for (int i = 0; i <
customWhereClauseParameters.length; i++) {
                                    statement.setObject(++parameterCounter,

        customWhereClauseParameters[i]);
                            }
                    }
                    if (firstRow > 0)
                            statement.setInt(++parameterCounter, firstRow);

                    ResultSet rs = statement.executeQuery();
                    if (rs != null) {
                            return object.ResultSet2ListOfAbstractClass(rs);
                    }
            } catch (SQLException e) {
                    e.printStackTrace();
            } catch (InstantiationException e) {
                    e.printStackTrace();
            } catch (IllegalAccessException e) {
                    e.printStackTrace();
            }
            return null;
    }

    public static int countList(AbstractClass object, String customWhereClause,
                    Object[] customWhereClauseParameters) {
            if (dbConnection == null) {
                    connect();
            }
            String sql = "";
            try {
                    boolean hasWhereClause = false;
                    int parameterCounter = 0;
                    sql = object.SelectCountSQL();
                    if (customWhereClause != null) {
                            if (hasWhereClause) {
                                    sql += " and " + customWhereClause;
                                    hasWhereClause = true;
                            } else
                                    sql += " WHERE " + customWhereClause;
                    }

                    PreparedStatement statement =
dbConnection.prepareStatement(sql);
                    if ((customWhereClause != null)
                                    && (customWhereClauseParameters != null)) {
```

```
                        for (int i = 0; i <
customWhereClauseParameters.length; i++) {
                            statement.setObject(++parameterCounter,

        customWhereClauseParameters[i]);
                        }
                    }

                    ResultSet rs = statement.executeQuery();
                    if (rs != null) {
                        rs.next();
                        return rs.getInt(1);
                    }
                } catch (SQLException e) {
                    System.err.println("countList(AbstractClass " + object
                            + ", String " + customWhereClause
                            + ",Object[] customWhereClauseParameters)"
                            + "\nCustomWhereClauseParameters are");
                    if (customWhereClauseParameters != null) {
                        for (int a = 0; a <
customWhereClauseParameters.length; a++) {
                            System.err
                                    .println(a + ")" +
customWhereClauseParameters[a]);
                        }
                    }
                    System.err.println("SQL=" + sql);
                    e.printStackTrace();
                }
                return -1;
        }

        /**
         * Deletes the given instance of AbstractClass
         *
         * @param object
         * @return
         */
        public static Boolean delete(AbstractClass object) {
                if (object == null || object.getId() == null
                        || object.getId() ==
AbstractClass.ID_OF_ANY_ABSTRACT_CLASS
                        || object.getId() ==
AbstractClass.ID_OF_NON_ABSTRACT_CLASS)
                    return false;
                if (dbConnection == null) {
                    connect();
                }
                try {
                    DatabaseChanged(BEFORE_DELETE, object);
```

```
                        AbstractClassMember[] listMembers =
object.getListMembers();
                        for (int k = 0; k < listMembers.length; k++) {
                                DAO.deleteListMember(object, listMembers[k]);
                        }
                        PreparedStatement statement =
dbConnection.prepareStatement(object
                                        .getDeleteByIdSQL());
                        statement.setString(1, object.getId());
                        boolean success = (statement.executeUpdate() > 0);
                        if (success) {
                                DatabaseChanged(AFTER_DELETE, object);
                                try {
                                        fireDataChanged(new
DataBaseEvent(DataBaseEvent.DELETE,
                                                        new AbstractClass[] { object }));
                                } catch (ApplicationException exc) {
                                        ErrorHandler.AddError(new
ApplicationError(exc,
                                                        "DAO.delete(AbstractClass
object)"));
                                }
                        }
                        return success;
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (ApplicationException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return false;
        }

        /**
         * @deprecated Please don't use this method if you can use the default
         * @return Returns a Statement that will be used for custom uses
         * @throws SQLException
         */
        public static Statement createStatement() throws SQLException {
                if (dbConnection == null)
                        connect();
                return dbConnection.createStatement();
        }

        /**
         *
         * @param sql
         *          The SQL Update Statement that you want to execute
         * @param parameters
```

```
 *          the parameters for SQL Update statement
 * @return If the update was successfully done or not
 * @throws SQLException
 */
public static Boolean customUpdate(String sql, Object[] parameters)
                throws SQLException {
        if (dbConnection == null)
                connect();
        PreparedStatement stmt = dbConnection.prepareStatement(sql);
        for (int i = 0; i < parameters.length; i++) {
                stmt.setObject(i + 1, parameters[i]);
        }
        return (stmt.execute());
}

private static Properties dbProperties;

public static Properties getDBProperties() {
        return dbProperties;
}

private static void fillDBProperties() {
        dbProperties = new Properties();
        dbProperties.setProperty("db.driver", DB_DRIVER);
        System.err.println("db.drive=" + DB_DRIVER);
        dbProperties.setProperty("db", DB);
        dbProperties.setProperty("database", DATABASE);
        dbProperties.setProperty("db.url", DB_URL);
        if (ROOT_DATABASE != null)
                dbProperties.setProperty("db.root", ROOT_DATABASE);
        dbProperties.setProperty("SCHEMA", SCHEMA);
        dbProperties.setProperty("user", USER_NAME);
        dbProperties.setProperty("password", PASSWORD);
}

private static String getDatabaseLocation() {
        String dbLocation = System.getProperty("derby.system.home") + "/"
                        + dbProperties.getProperty("db") + "/"
                        + dbProperties.get("database");
        return dbLocation;
}

private static boolean dbExists() {
        String dbLocation = getDatabaseLocation();
        File dbFileDir = new File(dbLocation);
        return dbFileDir.exists();
}

private static void setDBSystemDir() {
        // decide on the db system directory
```

```
            String systemDir = userHomeDir + "/."
                        + dbProperties.getProperty("SCHEMA");
            System.setProperty("derby.system.home", systemDir);
            // create the db system directory
            File fileSystemDir = new File(systemDir);
            fileSystemDir.mkdir();
        }

        private static void loadDatabaseDriver() {
            // load Derby driver
            try {
                Class.forName(dbProperties.getProperty("db.driver"));
            } catch (ClassNotFoundException ex) {
                ex.printStackTrace();
            }
        }

        private static boolean createDatabase() {
            boolean bCreated = false;
            Connection dbConnection = null;

            String dbUrl = getDatabaseUrl();
            dbProperties.put("create", "true");

            try {
                if (getRootDatabaseUrl() != null) {
                    dbConnection = DriverManager.getConnection(
                                getRootDatabaseUrl(), dbProperties);
                    Statement stmt = dbConnection.createStatement();
                    stmt.execute("CREATE DATABASE " +
dbProperties.get("database"));
                    dbConnection.close();
                }
                dbConnection = DriverManager.getConnection(dbUrl,
dbProperties);
            } catch (SQLException ex) {
            }
            dbProperties.remove("create");
            return bCreated;
        }

        private static boolean createTables(Connection dbConnection) {
            boolean bCreatedTables = false;
            Statement statement = null;
            try {
                statement = dbConnection.createStatement();
                for (int i = 0; i < AbstractClass.countAbstractClasses(); i++) {
                    try {
                        String sql[] = AbstractClass.CreateTableSQL(
```

```
        AbstractClass.getAbstractClass(i)).split("/");
                                        for (int k = 0; k < sql.length; k++) {
                                                System.out.println(sql[k]);
                                                try {
                                                        if (statement.execute(sql[k]))
                                                                System.out.println("Table
created for "
                                                                                + sql[k]);
                                                } catch (Throwable e) {

                                                }
                                        }
                                } catch (InstantiationException e) {
                                        e.printStackTrace();
                                } catch (IllegalAccessException e) {
                                        e.printStackTrace();
                                } catch (ClassNotFoundException e) {
                                        e.printStackTrace();
                                }
                        }
                        bCreatedTables = true;
                } catch (SQLException ex) {
                        ex.printStackTrace();
                }
                return bCreatedTables;
        }

        private static boolean connect() {
                String dbUrl = getDatabaseUrl();
                try {
                        System.err.println("I am trying to connect to " + dbUrl);
                        loadDatabaseDriver();
                        dbConnection = DriverManager.getConnection(dbUrl,
dbProperties);// dbProperties);
                        System.err.println("Connected?" + (dbConnection == null));
                        isConnected = dbConnection != null;
                } catch (SQLException ex) {
                        isConnected = false;
                        ex.printStackTrace();
                        System.err.println(ex.toString());
                }
                System.err.println("Connected?" + isConnected);
                return isConnected;
        }

        public static void disconnect() {
                if (isConnected) {
                        String dbUrl = getDatabaseUrl();
                        dbProperties.put("shutdown", "true");
```

```
                        try {
                                DriverManager.getConnection(dbUrl, dbProperties);
                        } catch (SQLException ex) {
                        }
                        isConnected = false;
                }
        }

        private static String getRootDatabaseUrl() {
                if (dbProperties.get("db.root") == null)
                        return null;
                String dbUrl = dbProperties.get("db.url") + ":"
                                + dbProperties.get("db") +
dbProperties.get("db.root");
                return dbUrl;
        }

        private static String getDatabaseUrl() {
                String dbUrl = dbProperties.get("db.url") + ":"
                                + dbProperties.get("db") +
dbProperties.get("database");
                return dbUrl;
        }

        /**
         * Inspector related methods
         */
        private static final int BEFORE_INSERT = 1;

        private static final int BEFORE_UPDATE = 2;

        private static final int BEFORE_DELETE = 3;

        private static final int AFTER_INSERT = 4;

        private static final int AFTER_UPDATE = 5;

        private static final int AFTER_DELETE = 6;

        private static final int AFTER_LOAD = 7;

        private static void DatabaseChanged(int CHANGE_TYPE, AbstractClass
object)
                        throws ApplicationException {
                String InspectorClassName = object.getClass().getName() +
"Inspector";
                boolean possiblyCriticalChange = false;
                try {
                        AbstractClassInspector inspector = (AbstractClassInspector)
Class
```

```
                        .forName(InspectorClassName).newInstance();
                inspector.setClazz(object);
                switch (CHANGE_TYPE) {
                case BEFORE_INSERT:
                        inspector.beforeInsert();
                        inspector.beforeInsertUpdate();
                        break;
                case BEFORE_UPDATE:
                        inspector.beforeUpdate();
                        inspector.beforeInsertUpdate();
                        break;
                case BEFORE_DELETE:
                        inspector.beforeDelete();
                        break;
                case AFTER_INSERT:
                        inspector.afterInsert();
                        inspector.afterInsertUpdate();
                        break;
                case AFTER_UPDATE:
                        inspector.afterUpdate();
                        inspector.afterInsertUpdate();
                        possiblyCriticalChange = true;
                        break;
                case AFTER_DELETE:
                        inspector.afterDelete();
                        possiblyCriticalChange = true;
                        break;
                case AFTER_LOAD:
                        inspector.afterLoad();
                        break;
                }
        } catch (InstantiationException e) {
                ErrorHandler
                        .AddError(new ApplicationError(
                                e,
                                ApplicationMessageMethods
                                        .get("Please chech
the AbstractClassInspector's contructor:")
                                        +
InspectorClassName));
        } catch (IllegalAccessException e) {
                ErrorHandler.AddError(new ApplicationError(e, ""));
        } catch (ClassNotFoundException e) {
                ErrorHandler
                        .AddError(new ApplicationError(
                                e,
                                ApplicationMessageMethods
                                        .get("There is not
such a AbstractClassInspector, please create the class:")
```

```
                                                                +
InspectorClassName));
                }
                if (!object.getClass().equals(AbstractClassMember.class)) {//
&&possiblyCriticalChange){//
                        // &&possiblyCriticalChange){
                        try {
                                if(false){
                                System.out.println("SEND MESSAGE");
                                Cache.getMulticast().sendMessage(
                                                new
Multicast.MulticastMessage(object.getId()));
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                                throw new RuntimeException(
                                                "Other Nodes are not informed about a
possibly critical change");
                        }
                }
        }

        // ListMember's related methods
        /**
         * ListMember is a special type of members that is java.util.List, so it has
         * many records that represents one member!
         */
        /**
         * Loads a List and sets it to the given member
         *
         * @param clazz
         * @param member
         */
        public static List loadListMember(AbstractClass clazz,
                        AbstractClassMember member) {
                if ((clazz == null) ||
(!clazz.isMemberList(member.getMemberName())))
                        return null;
                List listMember = new ArrayList();
                if (dbConnection == null) {
                        connect();
                }
                try {
                        StringBuffer sql = new StringBuffer();
                        sql.append("SELECT *\n FROM ");
                        sql.append(AbstractClass.TableNameForMemberList(clazz,
member));
                        sql.append("\nWHERE " + clazz.TableName() + "=?");
                        PreparedStatement statement =
dbConnection.prepareStatement(sql
```

```
                                        .toString());
                        statement.setString(1, clazz.getId());
                        ResultSet rs = statement.executeQuery();
                        while (rs.next()) {

        listMember.add(DataTypeMethods.DataBase2Java(rs.getObject(2),
                                                member.getComponentClass()));
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                        return null;
                }
                clazz.setMemberValue(member.getMemberName(), listMember);
                return listMember;
        }

        private static boolean deleteListMember(AbstractClass clazz,
                        AbstractClassMember member) {
                if ((clazz == null) ||
(!clazz.isMemberList(member.getMemberName())))
                        return false;
                boolean success = false;
                if (dbConnection == null) {
                        connect();
                }
                try {
                        StringBuffer sql = new StringBuffer();
                        sql.append("DELETE\n FROM ");
                        sql.append(AbstractClass.TableNameForMemberList(clazz,
member));
                        sql.append("\nWHERE " + clazz.TableName() + "=?");
                        PreparedStatement statement =
dbConnection.prepareStatement(sql
                                        .toString());
                        statement.setString(1, clazz.getId());
                        success = (statement.executeUpdate() > 0);
                } catch (SQLException e) {
                        e.printStackTrace();
                        success = false;
                }
                return success;
        }

        /**
         * Proccess<BR>
         * 1.delete the existed records<BR>
         * 2.insert the new records
         *
         * @param clazz
         * @param member
```

```
         * @return Returns whether the listMember was saved or not
         */
        private static boolean saveListMember(AbstractClass clazz,
                        AbstractClassMember member) {
                if ((clazz == null) ||
(!clazz.isMemberList(member.getMemberName())))
                        return false;
                deleteListMember(clazz, member);
                if (clazz.MemberValue(member.getMemberName()) == null)
                        return true;

                boolean success = false;
                if (dbConnection == null) {
                        connect();
                }
                try {
                        List list = (List)
clazz.MemberValue(member.getMemberName());
                        StringBuffer sql = new StringBuffer();
                        sql.append("INSERT INTO\n ");
                        sql.append(AbstractClass.TableNameForMemberList(clazz,
member));
                        sql.append("( " + clazz.TableName() + ", " +
member.getMemberName()
                                        + " )");
                        sql.append("VALUES (? ,?)");
                        PreparedStatement statement =
dbConnection.prepareStatement(sql
                                        .toString());
                        statement.setString(1, clazz.getId());
                        success = true;
                        for (int i = 0; i < list.size(); i++) {
                                if (list.get(i) == null)
                                        continue;
                                statement.setObject(2,
DataTypeMethods.Java2DataBase(list
                                                .get(i), list.get(i).getClass()));
                                success = success && (statement.executeUpdate() > 0);
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                        success = false;
                }
                return success;
        }

        private static void fireDataChanged(DataBaseEvent dataBaseEvent) {
                if (dbListeners == null)
                        dbListeners = new ArrayList<DataBaseListener>();
                for (int i = 0; i < dbListeners.size(); i++) {
```

```
                        if
(dataBaseEvent.getClazz().equals(dbListeners.get(i).getClazz())) {
                                switch (dataBaseEvent.getEventCode()) {
                                case DataBaseEvent.INSERT:
                                        dbListeners.get(i).dbInsert(dataBaseEvent);
                                        break;
                                case DataBaseEvent.UPDATE:
                                        dbListeners.get(i).dbUpdate(dataBaseEvent);
                                        break;
                                case DataBaseEvent.DELETE:
                                        dbListeners.get(i).dbDelete(dataBaseEvent);
                                        break;
                                }
                        }
                }
        }

        private static List<DataBaseListener> dbListeners = new
ArrayList<DataBaseListener>();

        public static void removeDataBaseListener(DataBaseListener
dataBaseListener) {
                dbListeners.remove(dataBaseListener);
        }

        public static void addDataBaseListener(DataBaseListener dataBaseListener)
{
                dbListeners.add(dataBaseListener);
        }

        /**
         * @param member
         * @return
         */
        public static List loadAllMemberValues(AbstractClassMember member) {
                if (member.isListMember()) {
                        new Throwable("It is not implemented for list members");
                        return null;
                }
                try {
                        if (dbConnection == null) {
                                connect();
                        }
                        List objectValues = new ArrayList();
                        StringBuffer sql = new StringBuffer();
                        sql.append("SELECT DISTINCT ");
                        sql.append(member.getMemberName());
                        sql.append(" \nFROM ");

        sql.append(member.getParentClass().newInstance().TableName());
```

```
                        System.out.println(sql.toString());
                        PreparedStatement statement =
dbConnection.prepareStatement(sql
                                        .toString());
                        ResultSet rs = statement.executeQuery();
                        while (rs.next()) {

        objectValues.add(DataTypeMethods.DataBase2Java(rs.getObject(1),
                                                member.getMemberClass()));
                        }
                        System.out.println("ObjectValues size=" +
objectValues.size());
                        return objectValues;
                } catch (InstantiationException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (IllegalAccessException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                return null;
        }

}
```

Τεχνικές Cache και Multicast

Πολύ σημαντικό κομμάτι της υλοποιημένης αρχιτεκτονικής του συστήματος του DAO είναι οι τεχνικές του **cache** και του **multicast**. Και οι δύο αυτές τεχνικές παρουσιάζονται παρακάτω και ακολουθεί το κομμάτι του κώδικά όπου τις εφαρμόζουμε.

Cache ➔ Ένας προσωρινός χώρος αποθήκευσης των instances από Abstract Classes που τις πήραμε από τη βάση δεδομένων και θα τις ξαναχρησιμοποιήσουμε. Έχουν συγκεκριμένο χρόνο ζωής που ορίζεται από μια κλάση από στο cacheInstance έτσι ώστε να μη σπαταλάμε χώρο στη μνήμη.

Multicast ➔ Στο σύστημά μου έχω δύο εφαρμογές. Η πρώτη εφαρμογή, αφορά τους χρήστες-δικηγόρους του συστήματός μας και είναι γνωστή ως webLaw ενώ η δεύτερη εφαρμογή έχει να κάνει με τη διαχείριση του συστήματος από τον system administrator και είναι το λεγόμενο BackOffice. Το σημαντικό κοινό σημείο σε αυτές τις εφαρμογές είναι ότι τόσο η πρώτη όσο και η δεύτερη εφαρμογή μας βλέπουν την ίδια Βάση Δεδομένων. Ως γνωστό όμως από την εφαρμογή μας γνωρίζουμε ότι κάθε μια έχει το δικό της χώρο αποθήκευσης των instances γνωστό ως cache όπως έχουμε προαναφέρει, με αποτέλεσμα να έρχεται στην επιφάνεια και να προκύπτει το εξής πρόβλημα.

Έστω ότι έχουμε μια εγγραφή, η οποία υπάρχει ταυτόχρονα στο cache τόσο στη μια όσο και στην άλλη εφαρμογή. Έστω ότι η εφαρμογή webLaw αλλάζει την εγγραφή αυτή. Αμέσως αντιλαμβανόμαστε ότι η εφαρμογή backOffice δε θα έχει ενημερωθεί, επομένως θα πάρει την εγγραφή αυτή από τη cache αναλλοίωτη. Το αποτέλεσμα σε αυτή τη περίπτωση είναι ότι για την ίδια εγγραφή θα βλέπουν διαφορετικό αποτέλεσμα. Οι λύσεις που προτείνουμε είναι οι εξής:

- **Λύση 1** Κατάργηση του cache. Αυτή η προτεινόμενη λύση θα μπορούσε να έχει άριστο αποτέλεσμα στο πρόβλημα που αντιμετωπίζουμε όσο αφορά το ίδιο το πρόβλημα, όμως αυτή η λύση μας οδηγεί στη δημιουργία ενός άλλου προβλήματος που δεν είναι άλλο από την αύξηση του κόστους της ταχύτητας στο σύστημά μας άρα και του χρόνου.
- **Λύση 2** Κάθε εφαρμογή να αναλαμβάνει με κάποιο τρόπο να ενημερώνει τις υπόλοιπες εφαρμογές, έτσι ώστε όποια εγγραφή έχει υποστεί κάποια αλλαγή, να τη καταργεί, και να τη παίρνει από τη Βάση Δεδομένων όπου υπάρχει πλέον με τις καινούργιες αλλαγές.

Σε αυτό το σημείο θα δούμε πως παρεμβαίνει το Multicast. Αυτό που κάνει το Multicast είναι ότι αναλαμβάνει να στέλνει και να δέχεται αυτά τα μηνύματα ώστε να επιτυγχάνεται αυτή η ενημέρωση του συστήματος μας.

Παράδειγμα εφαρμογής

```
if (!object.getClass().equals(AbstractClassMember.class)) {///
&&possiblyCriticalChange){///
                        // &&possiblyCriticalChange){
                        try {
                                if(false){
                                System.out.println("SEND MESSAGE");
                                Cache.getMulticast().sendMessage(
                                                new
Multicast.MulticastMessage(object.getId()));
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                                throw new RuntimeException(
                                                "Other Nodes are not informed about a
possibly critical change");
```

Τι είναι AbstractClass

- Προκειμένου να επιτευχθούν όλοι οι στόχοι του DAO πρέπει να χρησιμοποιηθεί μια κάποια τυποποίηση. Η βασικότερη είναι ότι όλες οι κλάσεις που θέλουμε να αποθηκεύονται στη βάση δεδομένων οι πληροφορίες που θα αναδημιουργούν τα instances τους, πρέπει να επεκτείνουν την AbstractClass.
  Δηλαδή η AbstractClass είναι ο «πατέρας» όλων των κλάσεων του συστήματος.
  Η AbstractClass έχει αρκετές και περίπλοκες μεθόδους που απαλλάσσουν τον προγραμματιστή από πολλά θέματα, όπως το πως θα αποθηκεύεται ο κάθε τύπος δεδομένων, αλλά υποχρεώνει να χρησιμοποιηθεί μια τυποποίηση στον τρόπο δημιουργίας των κλάσεων.

Πως δημιουργείται μια AbstractClass

Σε μια Abstract Class ορίζω τα πεδία που χρησιμοποιώ και ορίζω ένα πίνακα AbstractClass με AbstractClass members για κάθε πεδίο και δημιουργώ τον αντίστοιχο Inspector ο οποίος έχει κάποιες μεθόδους.

Όλα τα AbstractClass έχουν τη μέθοδο ***getOriginalMembers*** η οποία αποτελεί και μια από τις πιο ουσιαστικές μεθόδους γιατί δηλώνει τι πεδία θα χρησιμοποιήσω στο πίνακα.
Μια σημαντική πληροφορία, είναι τι πεδία θα έχω στο πίνακα και τα δημιουργώ με
new AbstractClassMember("Owner").changeParentClass(Briefcase.class)

.toRequired().toSearchCriterion().toVisible().

Με προσοχή το πρώτο γράμμα να είναι κεφαλαίο πχ "Owner" και για κάθε τέτοιο υπάρχουν οι μέθοδοι getOwner και setOwner (να κάνει get και set)οι οποίες θα πρέπει να είναι και ίδιου τύπου.

Για κάθε AbstractClass πρέπει να έχω ένα Inspector όπου γράφω το κωδικό για το τι θέλω μετά κάτι που συμβαίνει στο αντικείμενο.

Όλες αυτές οι πληροφορίες και οι επισημάνσεις που έχουμε προαναφέρει, μπορούμε να τις εντοπίσουμε και στα ακόλουθα παραδείγματα.

Παράδειγμα κλάσεως 1

```
public class Briefcase extends AbstractClass {

        private static AbstractClassMember[] members = new
AbstractClassMember[] {
                        new
AbstractClassMember("Owner").changeParentClass(Briefcase.class)
                                        .toRequired().toSearchCriterion().toVisible(),
                        new
AbstractClassMember("Title").changeParentClass(Briefcase.class)
```

```
        .changeFieldSize(100).toVisible().toEditable().toRequired()
                                        .toSearchCriterion(),
                        new AbstractClassMember("Description").changeParentClass(

        Briefcase.class).changeFieldSize(500).toVisible()
                                        .toEditable().toRequired().toSearchCriterion(),
                        new
AbstractClassMember("Laws").changeParentClass(Briefcase.class)

        .changeComponentClass(Law.class).toVisible().toEditable()
                                        .toRequired(),
                        new
AbstractClassMember("Notes").changeParentClass(Briefcase.class)

        .changeComponentClass(Note.class).toVisible().toEditable()
                                        .toRequired(),
                        AbstractClassMember.generateIdAbstractClassMember() };

        /*
         * (non-Javadoc)
         *
         * @see com.gorbas.model.AbstractClass#getOriginalMembers()
         */
        @Override
        protected AbstractClassMember[] getOriginalMembers() {
                return members;
        }

        private String title;

        private String description;

        private List<Law> laws = new ArrayList<Law>();

        private List<Note> notes = new ArrayList<Note>();

        private ApplicationUser owner;

        public ApplicationUser getOwner() {
                return this.owner;
        }

        public void setOwner(ApplicationUser owner) {
                this.owner = owner;
        }

        public String getDescription() {
                return this.description;
        }
```

```
	public void setDescription(String description) {
		this.description = description;
	}

	public List<Law> getLaws() {
		return this.laws;
	}

	public void setLaws(List<Law> laws) {
		this.laws = laws;
	}

	public List<Note> getNotes() {
		if (this.notes == null)
			this.notes = new ArrayList<Note>();
		return this.notes;
	}

	public void setNotes(List<Note> notes) {
		this.notes = notes;
	}

	public String getTitle() {
		return this.title;
	}

	public void setTitle(String title) {
		this.title = title;
	}

	public String toString() {
		return getTitle();
	}

	public String getToolTip() {
		return getDescription();
	}
}
```

Παράδειγμα κλάσεως 2

```
public class Writer extends AbstractClass {
        private String name;
        private static final AbstractClassMember[] members =
         new AbstractClassMember[] {
                new AbstractClassMember("Name", new Boolean(true), new
Boolean(true), new Boolean(false), 100),
                 AbstractClassMember.generateIdAbstractClassMember()
                };
        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        @Override
       //Πρέπει να επιστέφει μια χαρακτηριστική περιγραφή
        public String toString() {
                return name;
        }

        @Override
       //Πρέπει να επιστρέφει έναν πίνακα με τα members της κλάσης
        public AbstractClassMember[] getMembers() {
                return members;
       }
```

Οπότε συνοψίζοντας, μια κλάση που κάνει extend την AbstractClass πρέπει να έχει τα εξής:

1) Έναν πίνακα με τα members, που να έχει τουλάχιστον το Id
2) Μια μέθοδο toString() που να επιστρέφει μια περιγραφή
3) Τα όποια αναγκαία members με public accessors (getters, setters).

Τι είναι AbstractClassInspector

- AbstractClassInspector είναι η κλάση πατέρας όλων των Inspectors.
  Inspector είναι μια κλάση όπου έχει μεθόδους του τύπου beforeXXX και afterXXX, οπότε γράφουμε κώδικα σχετικά με το τι πρέπει να γίνεται πριν ή μετά από συγκεκριμένες ενέργειες.
  Οι ενέργειες αυτές μπορεί να είναι Select/Insert/Update/Delete
  Οπότε το DAO καλεί τις αντίστοιχες μεθόδους σε αντίστοιχες περιστάσεις.
  Να σημειωθεί ότι πάντα μπορούν να χρειασθούν κι άλλες τέτοιες μέθοδοι, οπότε και θα επεκταθεί ο Inspector!

Πως δημιουργείται ένας Inspector

Το όνομα του είναι το ίδιο με της κλάσης συν τη κατάληξη Inspector πχ BriefCaseInspector, κάνει extends την AbstractClassInspector και έχει διάφορες μεθόδους. Με το Quick Outline μπορώ εύκολα να δω τις μεθόδους από τις οποίες απαρτίζεται. Οι πιο σημαντικές είναι:

- ***Save*** →Η οποία παίρνει όρισμα μια AbstractClass.
- ***Load*** →Η οποία παίρνει όρισμα μια κλάση τύπου AbstractClass και ένα string που είναι το id του αντικειμένου που θέλει να φορτώσει.
- ***Search*** →Η οποία παίρνει όρισμα μια κλάση τύπου AbstractClass και ένα αριθμό για το μέγιστο πλήθος αποτελεσμάτων του αντικειμένου που θέλει να σου επιστρέψει και μια λίστα με το πια κριτήρια θες να αγνοηθούν
- ***Loadlist*** →Η οποία παίρνει όρισμα μια κλάση τύπου AbstractClass και είναι η μέθοδος με την οποία βλέπω πια εγγραφή θέλουμε να επιστρέψει με μέγιστο αριθμό.
- ***Delete*** →Η οποία παίρνει όρισμα μια κλάση τύπου AbstractClass.

Παράδειγμα κλάσεως

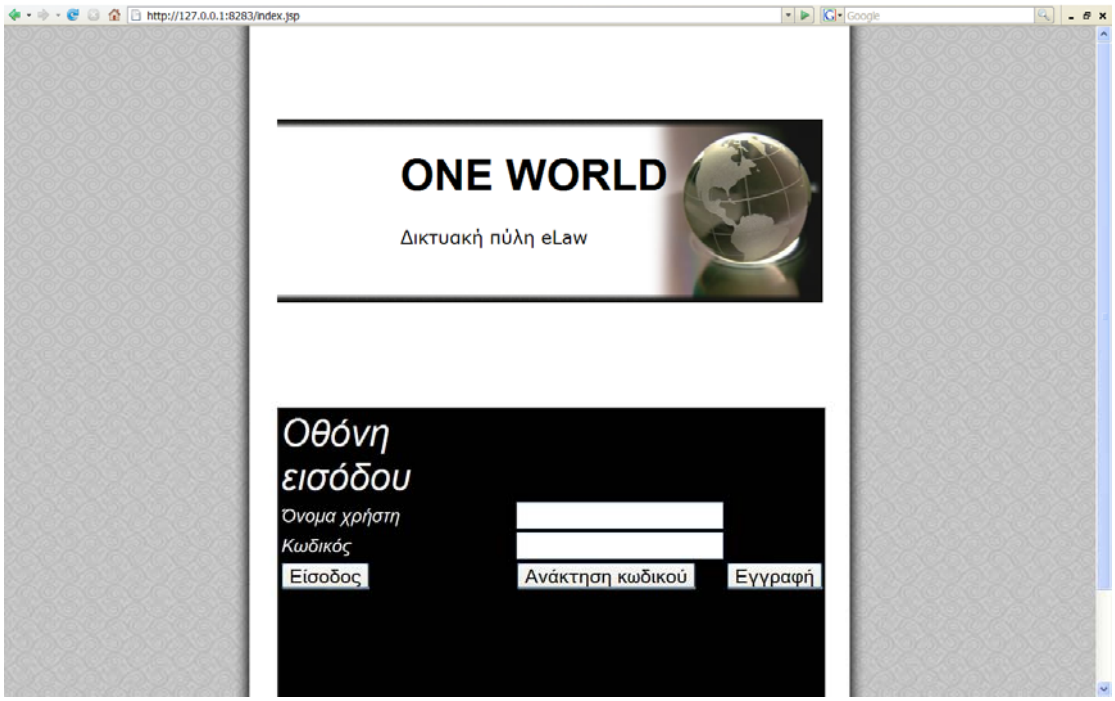
```
public class BriefcaseInspector extends AbstractClassInspector {
        public void beforeInsert() {

                super.beforeInsert();
        }

        public void beforeUpdate() {

                super.beforeUpdate();
        }

        public void beforeInsertUpdate() throws ApplicationException {
                super.beforeInsertUpdate();
        }

        public void beforeDelete() {
```

```
            super.beforeDelete();
        }

        public void afterInsert() {

            super.afterInsert();
        }

        public void afterUpdate() {

            super.afterUpdate();
        }

        public void afterInsertUpdate() {

            super.afterInsertUpdate();
        }

        public void afterDelete() {

            super.afterDelete();
        }

        public void afterLoad() {

            super.afterLoad();
        }

}
```

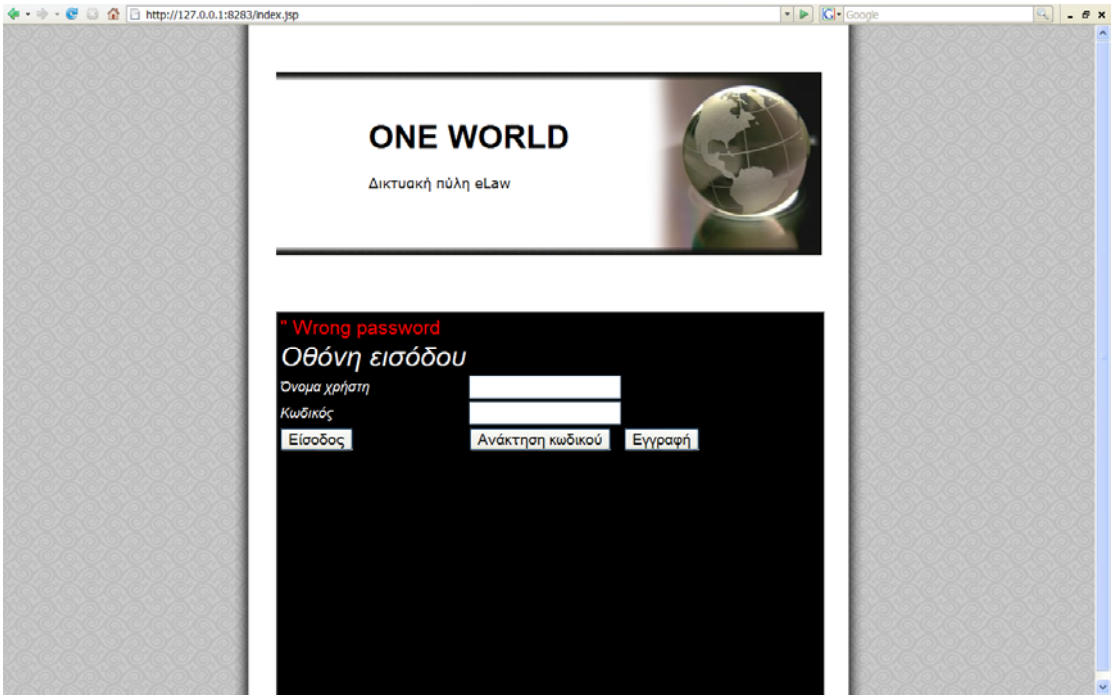
Άρα αποτέλεσμα της τυποποίησης αυτής, είναι ότι με αυτόν τον τρόπο έχουμε ξεφύγει από την ανάγκη μελέτης του πως θα είναι η βάση, καθώς τώρα θα δημιουργηθεί μόνη της!

Παράδειγμα χρήσης

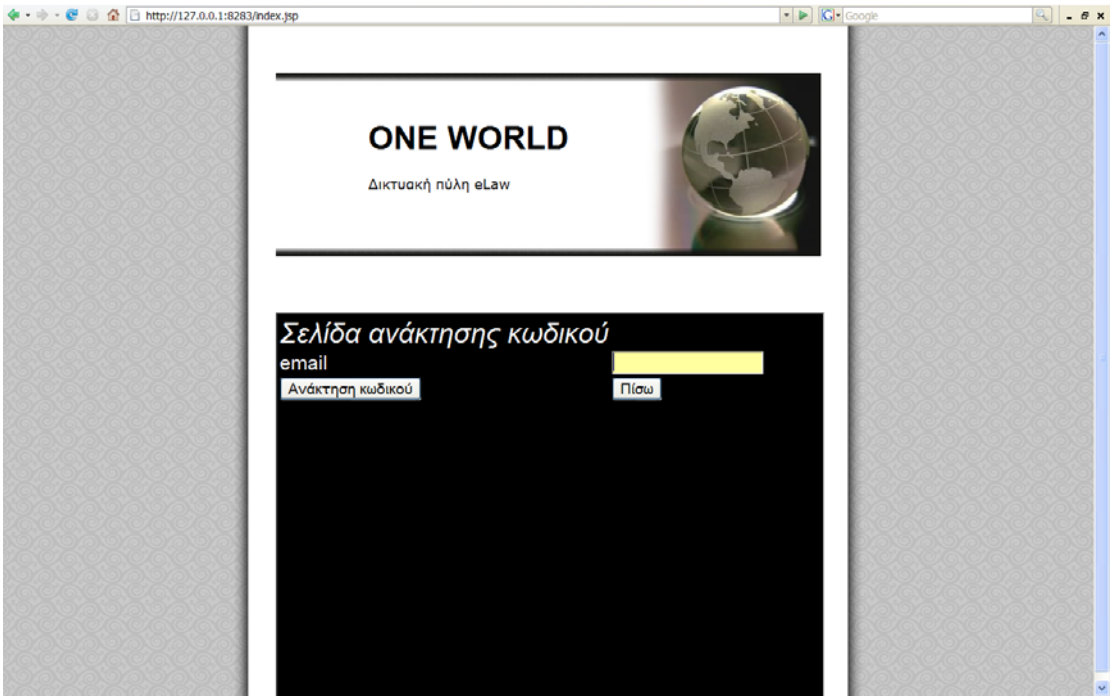
Ο διαδικτυακός χώρος eLaw που παρουσιάζουμε στην παρούσα πτυχιακή εργασία έχει πολλές δυνατότητες και προσφέρει αρκετά σημαντικές υπηρεσίες οι οποίες είναι στη διάθεση οποιουδήποτε χρηστή-δικηγόρου. Ξεκινώντας τη περιήγησή μας στο site του eLaw βλέπουμε το επόμενο παράθυρο. Αυτή είναι η εισαγωγική σελίδα, η οθόνη *εισόδου* στην οποία ο χρήστης-δικηγόρος μπορεί να κάνει login στο σύστημα απλά βάζοντας το «Όνομα χρήστη» και το «password» και πιέζοντας το κουμπί εισόδου.



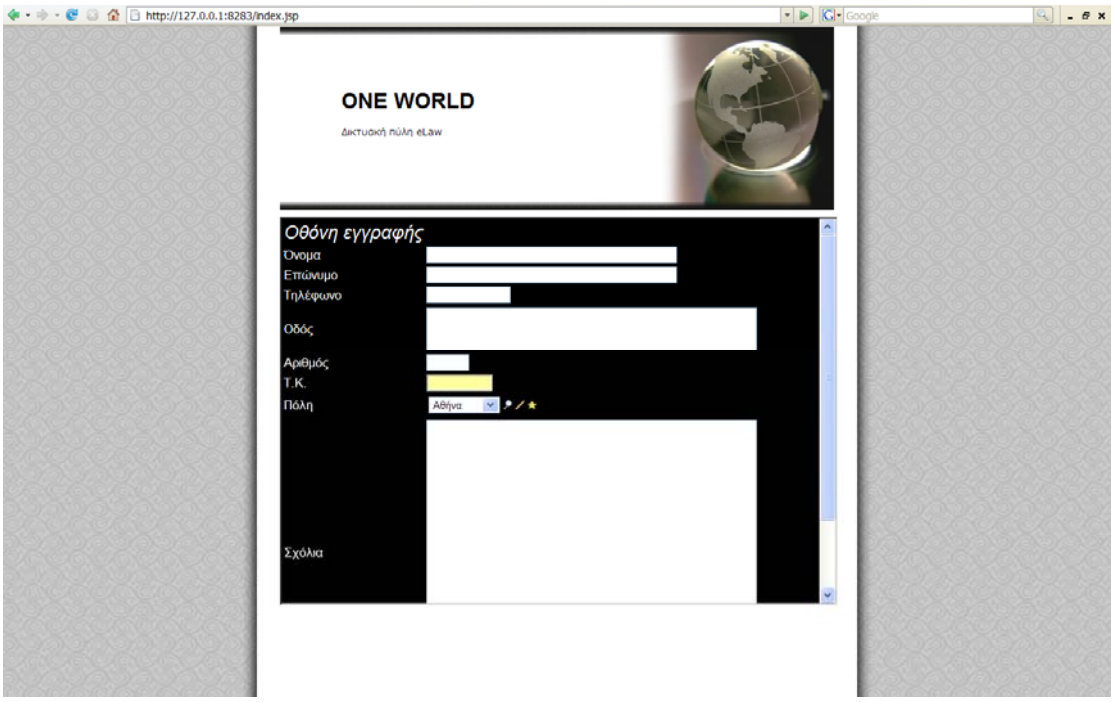
Σε περίπτωση πληκτρολόγησης λάθος κωδικού εμφανίζεται το επόμενο παράθυρο με την ένδειξη «Wrong password». Εδώ μας δίνεται η δυνατότητα επαναπληκτρολόγησης του κωδικού και η εισαγωγή μας στο σύστημα.



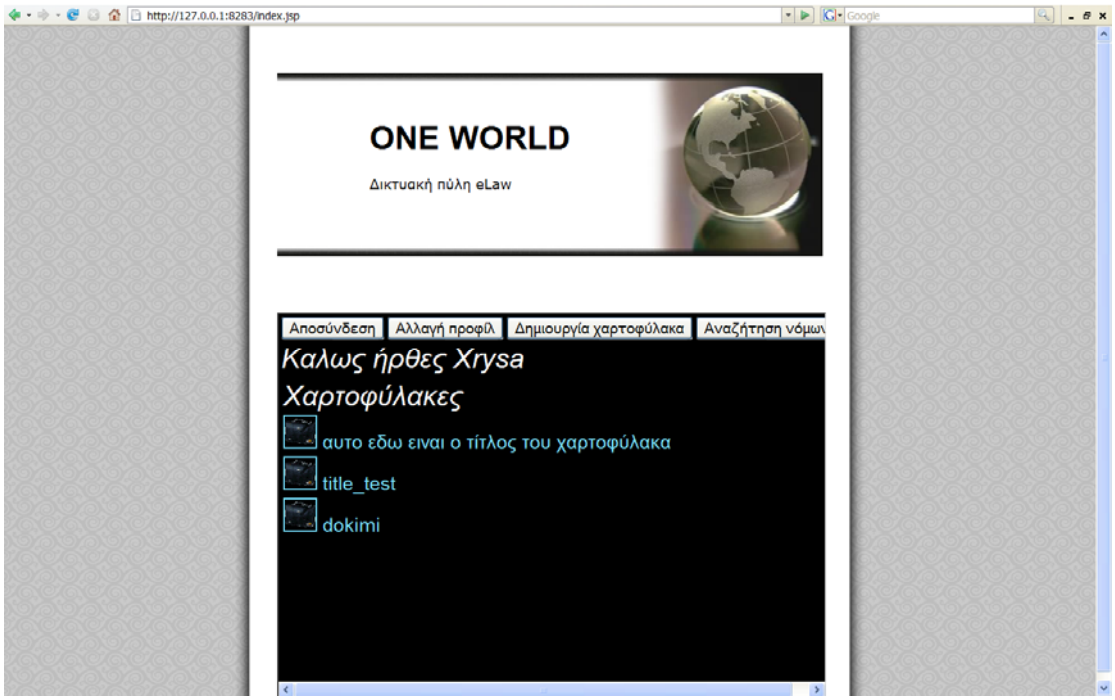
Επιπλέον υπηρεσίες που παρέχονται από τη σελίδα μας είναι η *Ανάκτηση κωδικού* και η *Εγγραφή*. Η ανάκτηση κωδικού είναι μια υπηρεσία στο σύστημά μας η οποία μας βοηθάει να ανακτήσουμε το κωδικό μας, απλά στέλνοντας ένα μήνυμα στο σύστημα ,και στη συνέχεια αυτό θα αναλάβει να μας αποστείλει στο e-mail μας το κωδικό. Αυτό φαίνεται στο επόμενο παράθυρο.



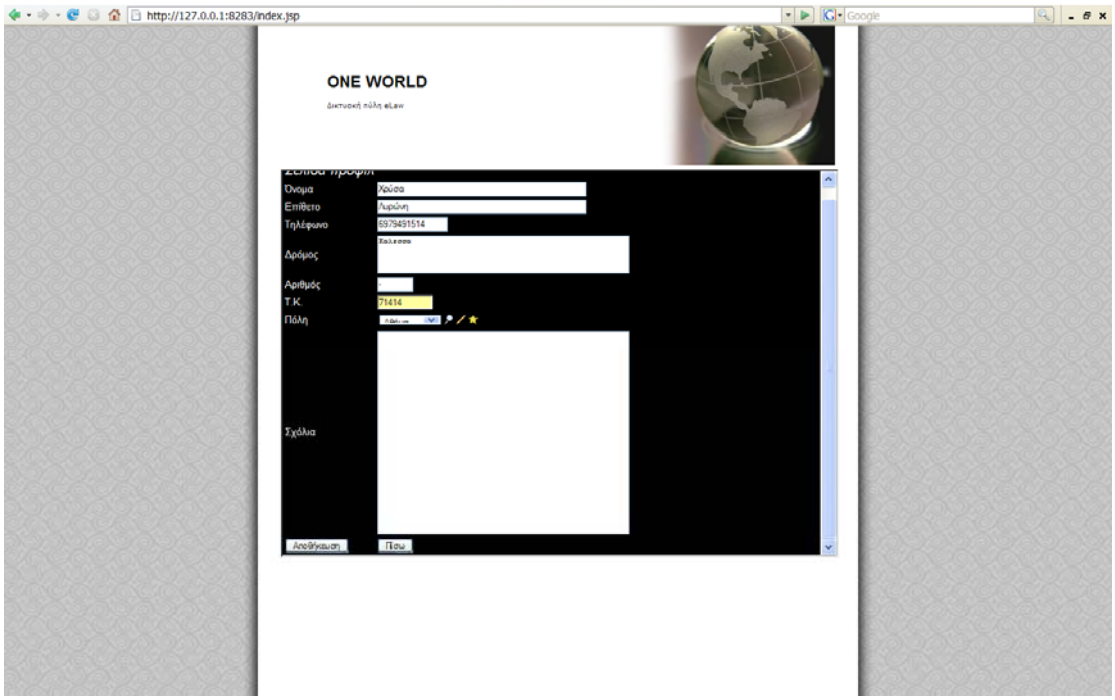
Εξίσου σημαντική υπηρεσία είναι η υπηρεσία της *εγγραφής χρήστη*. Με αυτή την υπηρεσία ο χρήστη που επισκέπτεται το σύστημά μας μπορεί να εισάγει τα απαραίτητα προσωπικά του στοιχεία και να γίνει μέλος του συστήματος με τη ιδιότητα του ως δικηγόρος. Στο επόμενο παράθυρο μπορούμε να δούμε αυτή την οθόνη εγγραφής



Ερχόμενοι ξανά στην υπηρεσία της εισόδου μας στο σύστημα, έχοντας πληκτρολογήσει σωστά το όνομα και το κωδικό χρήστη, μεταφερόμαστε στο επόμενο παράθυρο. Η υπηρεσία στην οποία βρισκόμαστε, είναι η πιο ενδιαφέρουσα στο σύστημά μας, μιας και αποτελεί καινοτομία στην ανάπτυξη διδικτυακών χώρων για δικηγόρους. Είναι η υπηρεσία δημιουργίας προσωπικού *χαρτοφύλακα* για τον εκάστοτε χρήστη-δικηγόρο καθώς και η διαχείριση αυτού από το δικηγόρο. Στο επόμενο παράθυρο μπορούμε να δούμε τις δυνατότητες αυτής της υπηρεσίας.



Στο κέντρο του παραθύρου εμφανίζονται κάποια εικονίδια με την αντίστοιχη ονομασία για τον κάθε χαρτοφύλακα, η οποία έχει δοθεί από τον ίδιο το χρήστη. Στο πάνω μέρος του παραθύρου, έχουμε υπό μορφή buttons κάποιες υπηρεσίες τις οποίες και θα αναλύσουμε στη συνέχεια. Με το κουμπί της **Αποσύνδεσης** αμέσως επιστρέφουμε στην αρχική σελίδα. Με το κουμπί της **Αλλαγής προφίλ** εμφανίζεται το επόμενο παράθυρο.

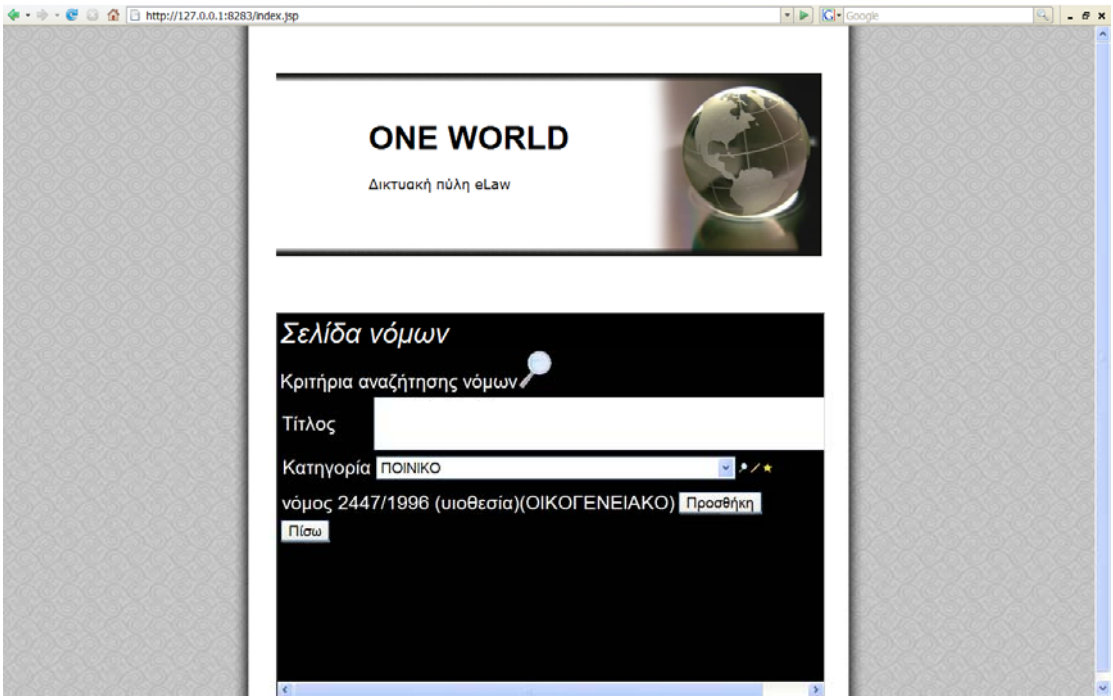


Σε αυτό το παράθυρο, ο χρήστης-δικηγόρος έχει τη δυνατότητα να παρέμβει στη ήδη συμπληρωμένη καρτέλα με τα προσωπικά του στοιχεία και να κάνει οποιαδήποτε αλλαγή. Στη συνέχεια πατώντας το κουμπί της *Αποθήκευσης*, ενημερώνεται αμέσως η Βάση με τα καινούργια πλέον στοιχεία.
Επιλέγοντας τώρα το κουμπί *Πίσω* της επιστροφής, μεταφερόμαστε ξανά στο προηγούμενο Παράθυρο. Η αμέσως επόμενη επιλογή μας, είναι η **Δημιουργία Χαρτοφύλακα.** Πατώντας το κουμπί της δημιουργίας, εμφανίζεται το επόμενο παράθυρο. Εδώ μπορούμε να δούμε ποιος είναι ο επισκέπτης του site στο πεδίο *κάτοχος*. Στη συνέχεια μπορούμε να δώσουμε ένα όνομα στο χαρτοφύλακα που έχουμε δημιουργήσει, καθώς και μια περιγραφή αυτού. Στη σελίδα αυτή του Χαρτοφύλακα έχουμε τη διάφορες δυνατότητες. Πατώντας το κουμπί της *Αποθήκευσης* αυτόματα αποθηκεύεται το όνομα και η περιγραφή που έχουμε δώσει στο Χαρτοφύλακά μας.

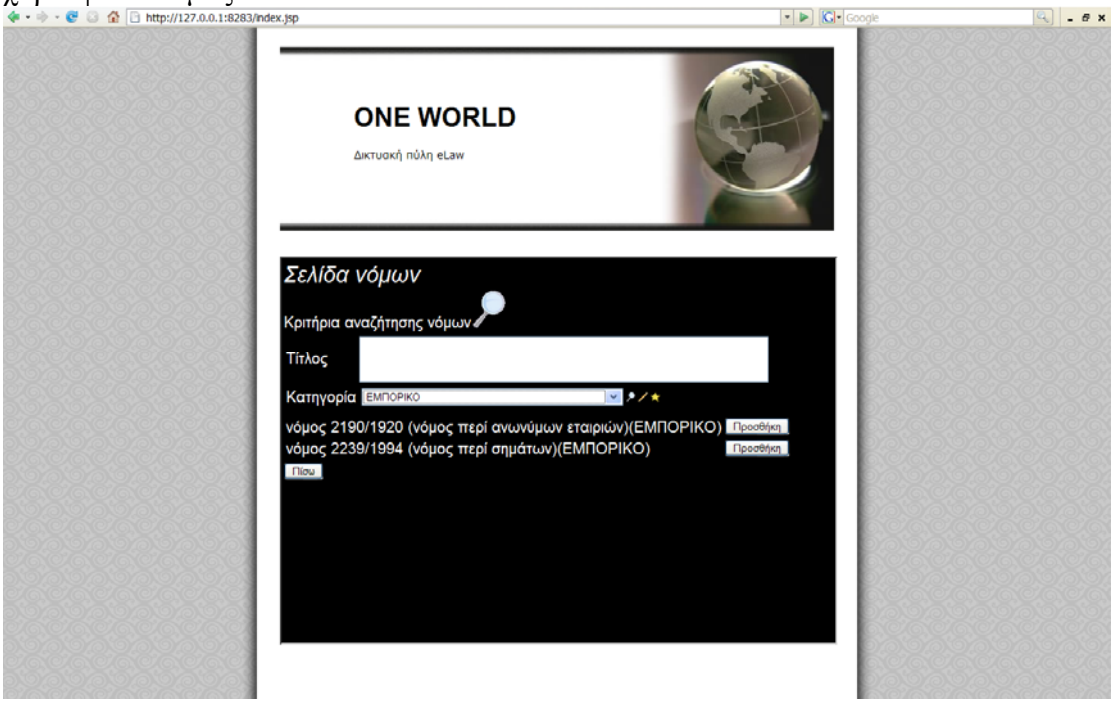
Μια επίσης σημαντική υπηρεσία, είναι η υπηρεσία των *Σημειώσεων*. Πατώντας το κουμπί της *Προσθήκης*, το οποίο βρίσκεται δίπλα στο πεδίο των σημειώσεων μεταφερόμαστε στο επόμενο παράθυρο.

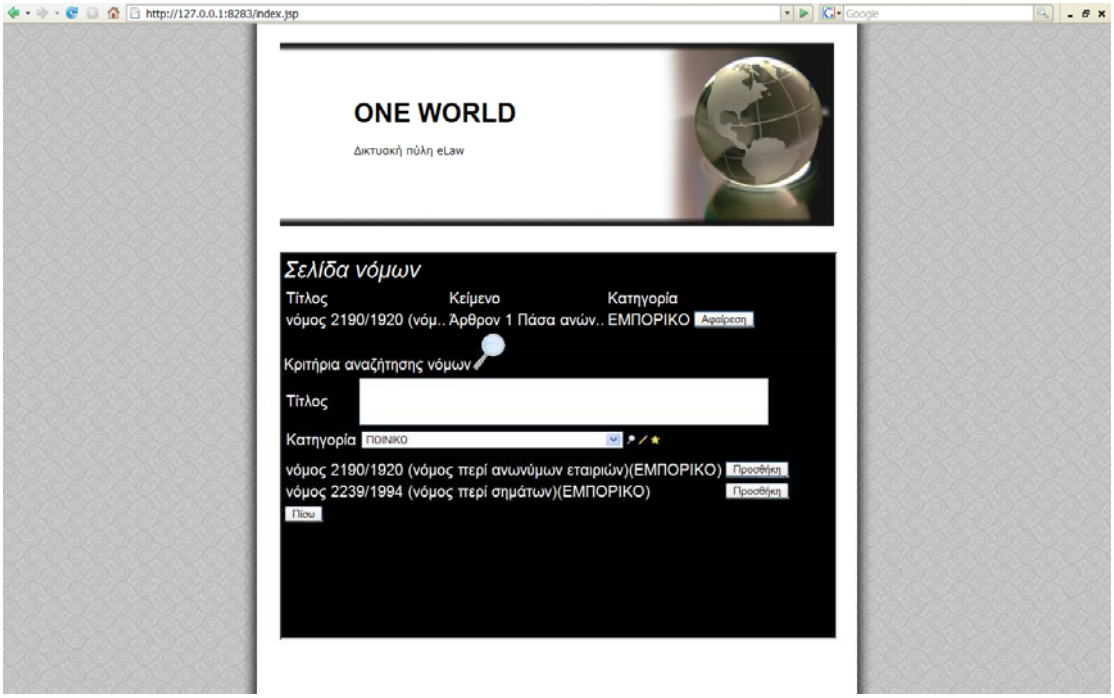
Εδώ μπορούμε να δώσουμε στο πεδίο *Σημείωση* το τίτλο της σημείωσης και στο πεδίο *Κείμενο* τη σημείωση που θέλουμε να καταγράψουμε. Πατώντας το κουμπί της *Αποθήκευσης*, αποθηκεύεται η σημείωση με τον αντίστοιχο τίτλο, ενώ πατώντας το κουμπί της διαγραφής διαγράφεται η συγκεκριμένη σημείωση.

Μια από τις πιο σημαντικές υπηρεσίες, είναι της διαχείρισης των νόμων. Με το κουμπί της *Διαχείρισης νόμων* εισερχόμαστε στο επόμενο παράθυρο.

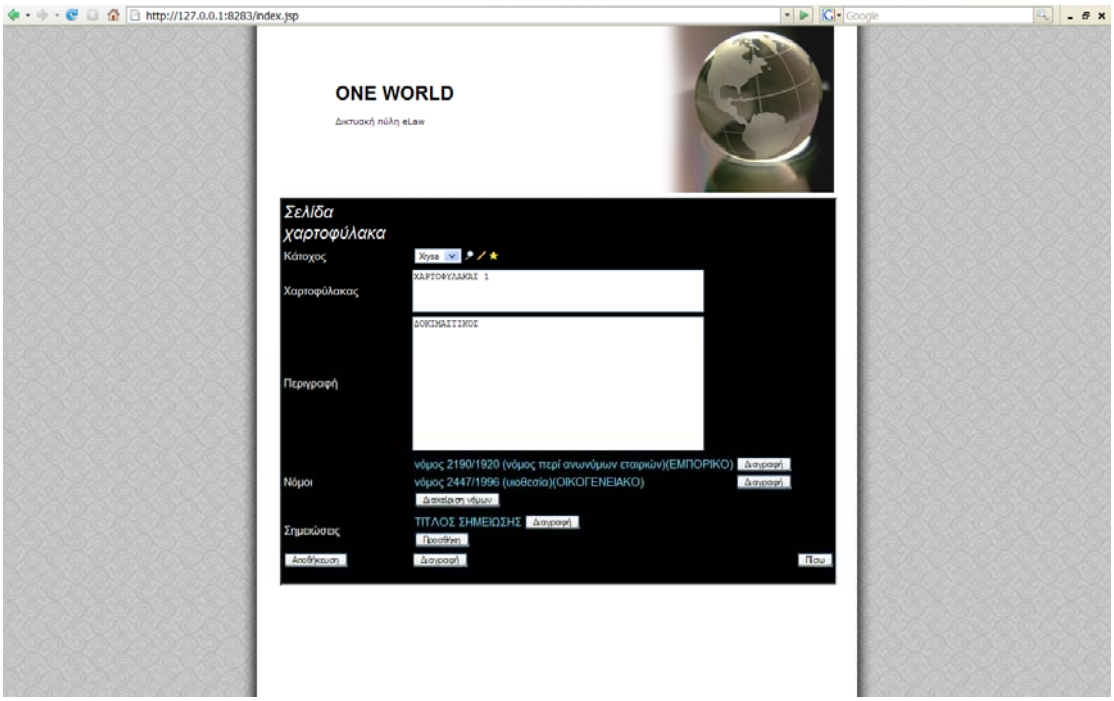


Σε αυτή τη σελίδα νόμων όπως την ονομάζουμε, έχουμε τη δυνατότητα Αναζήτησης των Νόμων. Επιλέγω κατηγορία και πατάω το εικονίδιο της αναζήτησης. Αμέσως εμφανίζονται οι νόμοι που υπάρχουν σύμφωνα με το κριτήριο του δικαίου. Επιλέγοντας το κουμπί της προσθήκης, ο συγκεκριμένος νόμος προστίθεται στο χαρτοφύλακά μας.

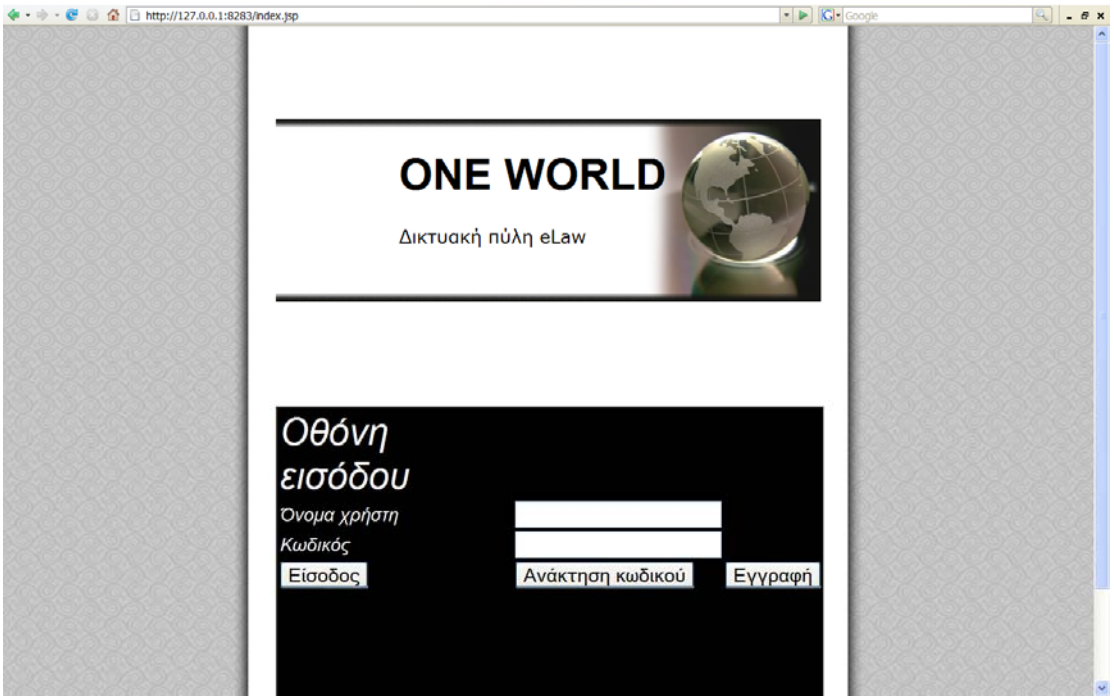




Το αποτέλεσμα φαίνεται στο επόμενο παράθυρο με τους νόμους που έχουμε ήδη προσθέσει.



Εκτός όμως από τη δυνατότητα προσθήκης ενός νόμου, έχουμε τη δυνατότητα και να διαγράψουμε ένα νόμο που δεν μας ενδιαφέρει πια, πατώντας το κουμπί της *Διαγραφής*. Σε αυτό το σημείο, μας δίδεται και η δυνατότητα να αποθηκεύσουμε ότι ενέργειες έχουμε κάνει σε αυτό το χαρτοφύλακα και με το κουμπί της επιστροφής *Πίσω*, να επιστρέψουμε στην αρχική σελίδα.



Επίλογος

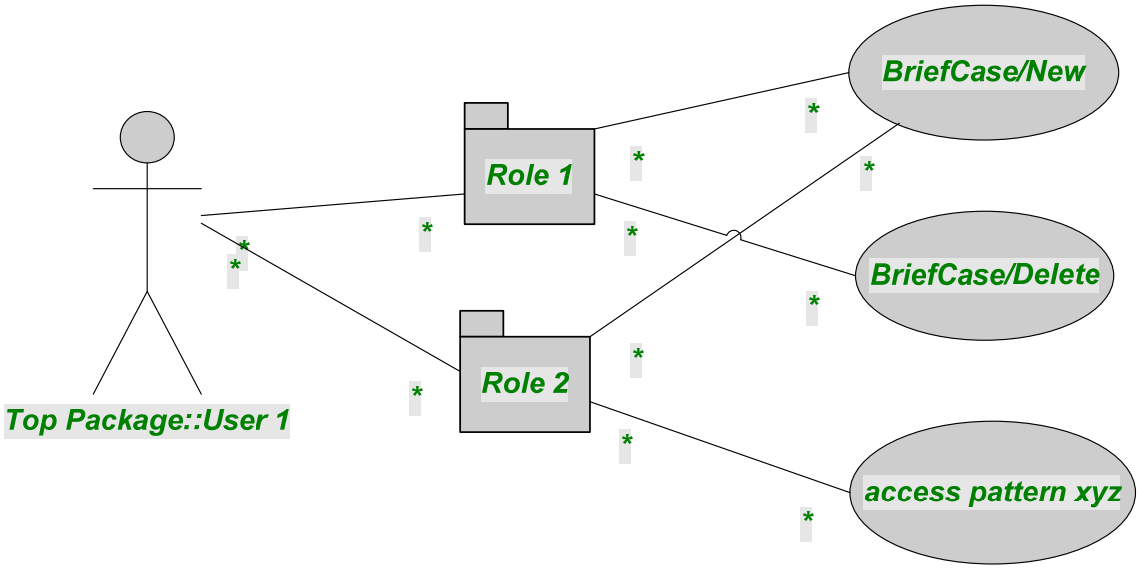
Τι μελλοντική εξέλιξη μπορεί να υπάρξει

Σε αυτό το σημείο, θα πρέπει να αναφέρουμε το εξής σημαντικό στοιχείο. Το σύστημά μας έχει αναπτυχθεί με βάση κάποιο πυρήνα πάνω στον οποίο και βασίζεται η ανάπτυξη του ιστοχώρου του eLAW.

Η δομή του συστήματος στο θέμα απόδοσης των δικαιωμάτων είναι η ακόλουθη.

- Κάθε ενέργεια στο σύστημα, έχει ένα access pattern [κωδικός ενέργειας]
  Πχ. BriefCase/New σημαίνει δημιουργία νέου χαρτοφύλακα
- Υπάρχουν στο σύστημα πολλοί ρόλοι [Roles] χρηστών, ένας ρόλος συνδέεται με ένα ή περισσότερα access patterns. Δηλαδή ένας ρόλος μπορεί να κάνει τις λειτουργίες που αντιστοιχίζονται σε αυτά τα access patterns.
- Κάθε χρήστης μπορεί να έχει πολλούς ρόλους.

Αν θα θέλαμε να το αναπαραστήσουμε και σχεδιαγραμματικά, θα μπορούσε να ήταν όπως φαίνεται στο επόμενο διάγραμμα.



Εικόνα 4 Διάγραμμα χρήσης Ρόλων

Το πλεονέκτημα της απόδοσης ρόλων στο κάθε χρήστη, είναι ότι με αυτό τον τρόπο, δίδεται η δυνατότητα στο διαχειριστή του συστήματος, να προσαρμόσει τα δικαιώματα του κάθε χρήστη.

Παράθεση ιδεών για ηλεκτρονικό υπολογιστή τσέπης

Η επιτυχής άσκηση του δικηγορικού επαγγέλματος προϋποθέτει τη σωστή διευθέτηση και τη συνεχή παρακολούθηση των ημερομηνιών και του χρόνου εν γένει. Το δίκαιο είναι γεμάτο διαδικαστικούς κανόνες που στηρίζονται στην παρέλευση του χρόνου. Η ορθή απονομή Δικαιοσύνης συνδέεται άμεσα με την τήρηση των προβλεπομένων προθεσμιών και εν γένει διαδικαστικών προϋποθέσεων. Οι προθεσμίες παρέρχονται, ο χρόνος παραγραφής συμπληρώνεται, τα δικαιώματα αποσβεννύονται, εάν δεν ασκηθούν, ο αριθμός πινακίου ή εκθέματος εκφωνείται, ο διάδικος δικάζεται ερήμην, ο κατηγορούμενος δικάζεται ως ει παρών, συχνά δε ο πρώτος βαθμός δικαιοδοσίας απόλλυται με δυσμενείς ως επί το πλείστον συνέπειες.

Από την άλλη πλευρά τόσο η νομοθεσία όσο και οι αποφάσεις των Δικαστηρίων διαμορφώνουν το νομικό πλαίσιο μες στο οποίο εν γένει ασκούνται τα ένδικα δικαιώματα και την κρατούσα δικαιική κατάσταση. Η παρακολούθηση και ενημέρωση επί της διαρκώς εξελισσόμενης νομοθεσίας και νομολογίας γίνεται ακόμη πιο ευχερής με τη χρήση της τεχνολογίας των ηλεκτρονικών υπολογιστών.

Μέχρι σήμερα έχει καταστεί πολύ σημαντική η προσφορά της τράπεζας νομικών πληροφοριών του Δικηγορικού Συλλόγου Αθηνών, που είναι η μόνη η οποία διαθέτει υποδομή για την ενημέρωση επί της δημοσιεύσεως δικαστικών αποφάσεων και επί των εκθεμάτων των Δικαστηρίων, πέρα από την ενημέρωση επί της νομοθεσίας και της νομολογίας την οποία επαρκώς παρέχει. Συνήθως κάποιος έχει πρόσβαση σε αυτές μέσω του ηλεκτρονικού υπολογιστή στο σπίτι ή το γραφείο.

Θα ήταν ομοίως ιδιαίτερα χρήσιμη η προσφορά της χρήσης ενός υπολογιστή τσέπης με προγράμματα που θα παρείχαν πέρα από τις προαναφερόμενες υπηρεσίες, και ειδική ενημέρωση για την καθ' ημέραν πορεία της εκδίκασης των αστικών υποθέσεων στο Πρωτοδικείο και το Ειρηνοδικείο Ηρακλείου ανά αριθμό πινακίου και ανά αίθουσα και των ποινικών υποθέσεων στα Πλημμελειοδικεία του Ηρακλείου και στο Εφετείο Κρήτης ανά αριθμό εκθέματος. Η εν λόγω υπηρεσία θα εξοικονομούσε στο μαχόμενο δικηγόρο πολύ χρόνο, διότι θα αποφευγόταν η άσκοπη σπατάλη χρόνου και θα περιοριζόταν ο συνεχής φόβος της απώλειας της σειράς, και ομοίως θα αποτρεπόταν το άγχος που δημιουργεί η διαρκής μετάβαση από τη μια αίθουσα στην άλλη και από το ένα κτίριο στο άλλο για τον έλεγχο της πορείας της κάθε υπόθεσης.

Βέβαια η λειτουργία ενός τέτοιου συστήματος προϋποθέτει την ύπαρξη συστήματος ηλεκτρονικών υπολογιστών στην έδρα του δικάζοντος Δικαστηρίου, τη χρήση αντίστοιχου προγράμματος και την παροχή των εν λόγω πληροφοριών από τη γραμματεία της έδρας του συνεδριάζοντος Δικαστηρίου, που είναι ήδη επιφορτισμένη με την ευθύνη της πιστής τήρησης των πρακτικών της δίκης.

Βιβλιογραφία

- http://www.w3schools.com/sql/sql_intro.asp
- http://www.gigfoot.net/software/index.html
- http://www.dsa.gr
- http://www.nasw.org/users/nbauman/lawdb.htm
- http://www.lawnet.gr/
- http://lawdb.intrasoftnet.gr
- http://www.parliament.gr
- http://www.et.gr/
- http://elawyer.blogspot.com/2005_09_01_archive.html