**Project Εργαστηρίου ΒΔ (2020-21)**

Η ομάδα:  
     ΕΠΩΝΥΜΟ           ΟΝΟΜΑ        ΑΜ        ΕΤΟΣ  
1.  ΚΟΚΚΑΛΑ         ΒΑΣΙΛΙΚΗ     1067519        Γ'  
2.  ΠΑΠΑΔΟΠΟΥΛΟΥ   ΑΙΚΑΤΕΡΙΝΗ    1067535       Γ'

**\*\*Ο κώδικας της MySQL είναι κάτω κάτω.**

Χρησιμοποιησαμε για τη δημιουργια του UI: WPF(Windows Format Presentation) = C# και XAML. Ακομα για την δημιουργια δεδομενων χρησιμοποιησαμε fake randomly generated data.

Για την υλοποίηση αυτού του project κάναμε κάποιες υποθέσεις, κυρίως για την δημιουργία της βάσης. Αρχικά, θεωρήσαμε ότι κάθε εταιρία έχει συγκεκριμένα τμήματα, όπου το καθένα έχει έναν υπεύθυνο Διευθυντή-Manager και κάποιους Αξιολογητές-Evaluators που έχουν ανατεθεί για να ελέγχουν και αξιολογούν τους υπαλλήλους που ανήκουν στο συγκεκριμένο τμήμα. Παρόλα αυτά οι ίδιοι αξιολογητές έχουν τη δυνατότητα δημιουργίας θέσεων εργασίας σε ολόκληρη την εταιρία, δηλαδή σε οποιοδήποτε τμήμα . Επίσης, για να είναι διακριτό το ποιοι Υπάλληλοι-Employees ανήκουν σε ποιο Τμήμα-Department της εταιρίας, κάθε τμήμα έχει ένα αντιπροσωπευτικό χρώμα. Με λίγα λόγια στη βάση μας έχουμε στον πίνακα για τα τμήματα ένα string που είναι για τη δεκαεξαδική τιμή του χρώματος που έχει ένα τμήμα.

Στόχος μας ήταν για την διευκόλυνση μας, αλλά και για να αποφύγουμε τυχόν λάθη που θα οδηγούσαν σε πολύπλοκα μπερδέματα που θα οδηγούσαν σε λαθος αποθηκευμένα δεδομενα μεσα στη βαση. Για αυτό το λόγο ισχύει η παρακάτω διαδικασία για τις αξιολογήσεις:

1. Ένας **evaluator δημιουργεί μία ανοιχτή θέση** για πρόσληψη δηλώνοντας το όνομα της εργασίας (π.χ. Junior Developer) **χωρίς να χρειαζεται να δηλώσει department, γιατί αυτό συμπληρώνεται αυτόματα** καθώς κάθε Job αντιστοιχίζεται σε ένα department και το Job αυτό μπορεί να αντιπροσωπεύει πολλές θέσεις Job Positions, προφανώς και κάθε νεα θεση που δημιουργειται.
2. Με το που δημιουργείται, μπορεί και ο ίδιος ο **evaluator** να δει τη νεα θεση που δημιούργησε και στην καρτέλα που είναι μονο οι δικες του θεσεις εργασιας αλλα και στην καρτελα που είναι ολες οι θεσεις εργασιας που είναι ανοιχτες αυτή τη στιγμη σε ολοκληρη την εταιρια. Φυσικα μετα τη δημιουργια της μπορει και να την επεξεργαστει.

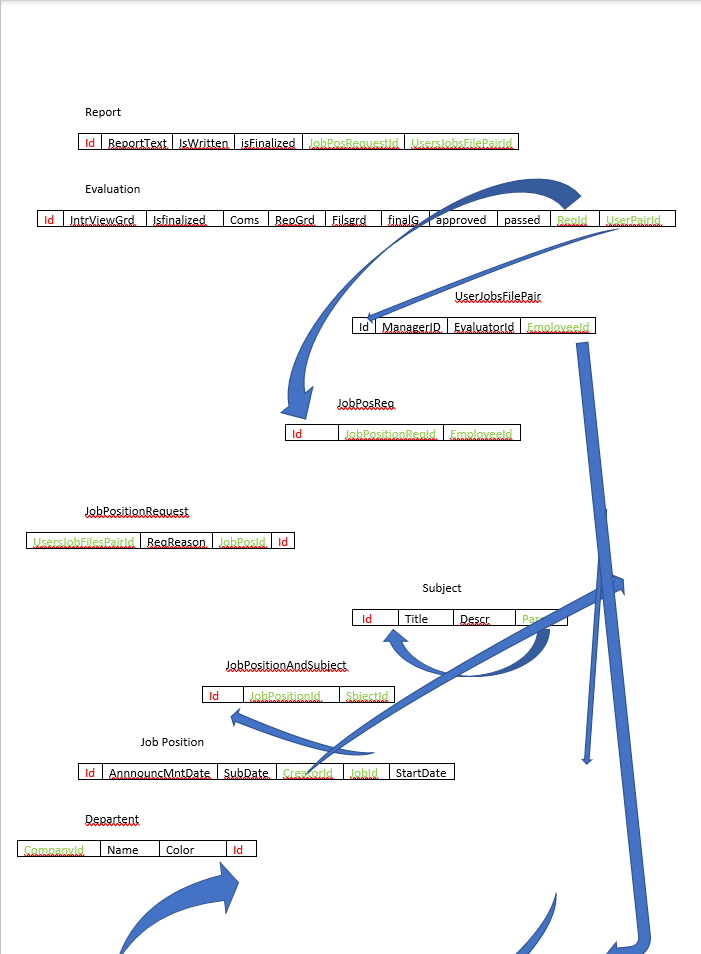
Έκτος απο αυτόν μπορουν και οι **Managers** να δουν τη νεα θεση εργασιας της οποιας τον μισθο μπορουν να επεξεργαστουν τον μισθο της. Ακόμη μπορουν και οι δυο να βλέπουν το συνολικο αριθμο αιτήσεων που έχουν δημιουργηθεί μέχρι στιγμης.

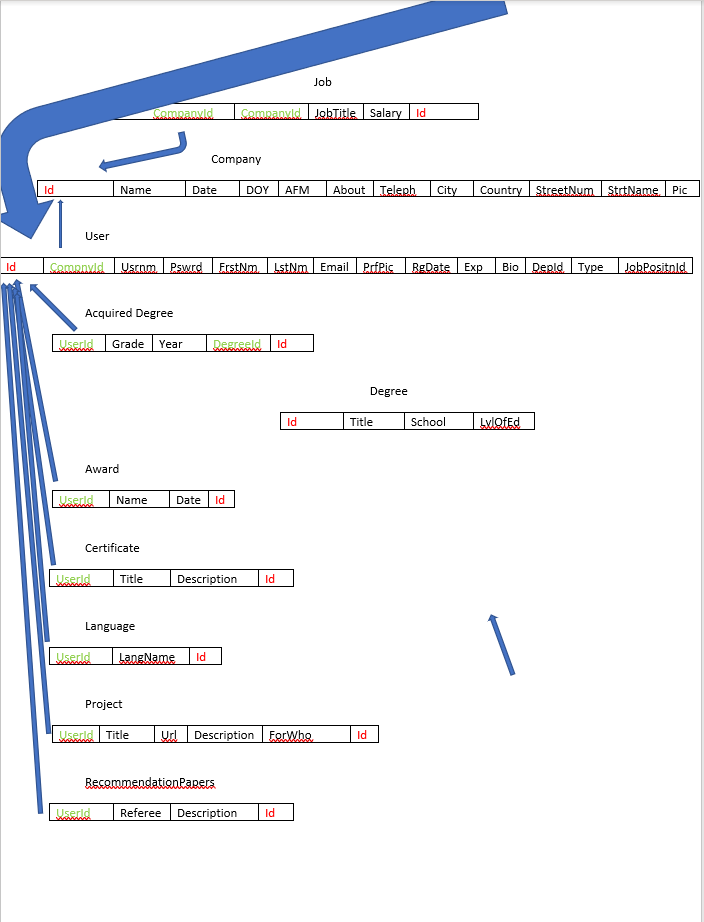
Τέλος, μπορούν και όλοι οι **υπάλληλοι** της συγκεκριμένης εταιρίας να δουν τη νεα θέση.

1. Τώρα, αν το επιθυμεί καποιος υπάλληλος μπορει να δημιουργήσει μια αίτηση για να αξιολογηθει και να αποκτησει αυτή τη θεση εργασιας. Όταν υποβαλλει την αιτηση καποιος ζητειται να συνταξει μια μικρη παραγραφο στην οποια αναφερει καποιους λογους για τους οποιους επιθυμει και θεωρει πως είναι ικανος να παρει την συγκεκριμενη θεση.
2. Με τη δημιουργια αυτης της αιτησης, **δημιουργείται αυτοματα μια άδεια αναφορα στον Manager του department που είναι ο συκγεκριμένος υπαλληλος που ζήτησε την αξιολογηση**. Ο manager λαμβάνει το κειμενακι που συνεταξε ο υπαλληλος και σε περιπτωση που επιθυμεί να δει τα στοιχεια του υπαλληλου αυτου, μπορει να παει σττην καρτελα Employees και να αναζητησει γράφοντας είτε το ονομα και επωνυμο του υπαλληλου είτε το username το και κανοντας κλικ στο κουμπι που αντιπροσωπευει τον υπαλληλο μπορει να μπει στο προφιλ του και να δει οποιαδηποτε εξτρα πληροφορια χρειαζεται για να δημιουργησει την αναφορα του. Φυσικα του δίνεται η δυνατοτητα επεξεργασιας της αναφορας και η προσωρινη της αποθηκευση. Αφου συνταξει την αναφορα του και είναι βεβαιως ότι είναι ολοκληρωμενη πατάει στο κουμπι “Finalize and Send” και η αναφορά του πλεον δηλώνεται ως ολοκληρωμενη.
3. Με την υποβολη της αναφορας **δημιουργείται αυτόματα στον Evaluator αδεια μια αναφορά αξιολόγησης** **που μέσα της έχει την αναφορα του manager** και ο ιδιος μπορει να προσθεσει τα σχολια από την συνεντευξη του με τον συγκεκριμενο υπαλληλο και τις 3 απαραιτητες βαθμολογιες. Φυσικα του δίνεται η δυνατοτητα επεξεργασιας των δικων του προσθεσεων και η προσωρινη τους αποθηκευση. Ο τελικος βαθμος υπολογίζεται αυτόματα. Όταν θεωρησει ότι είναι ολοκληρωμενη η αξιολογηση και δεν χρειαζεται καποια αλλαγη πατα στο κουμπι “Finalize and Send” και πλεον θεωρειται ολοκληρωμενη από τον Evaluator.
4. Έτσι με την υποβολή της **αξιολογησης** του Evaluator αυτή **στελενεται αυτόματα στον Manager του τμήματος**. Εκει ο manager μπορει να δει πόσες αξιολογησεις δεν είναι ολοκληρωμενες ακομα. Σε περιπτωση που εχουν ολοκληρωθει ολες οι αξιολογησεις για την συγκεκριμενη θεση εργασιας μπορει να επιλεξει να δει μονο τις αξιολογησεις που αφορουν αυτή τη θεση εργασιας και να επιλεξει τελικα ποιος εργαζομενος θα παρει την θεση. Τοτε παταει το πρασινο κουμπι για την επιβεββαιωση και σε περιπτωση που απορριπτει καποια αιτηση παταει το κοκκινο κουμπι επιβεβαιωσης.
5. Τελικα, «στελνεται» αυτοματα το αποτελεσμα της αξιολογησης στον υπαλληλο δηλώνοντας με πρασινα εντονα γραμματα ότι εχει περασει την αξιολογηση και του ζητειται να επιβεβαιωση την προαγωγη, ενώ με εντονα κοκκινα ότι δεν καταφερε να παρει την θεση εργασιας.

Φυσικα υπαρχουν και άλλες ενεργειες που μπορουν να πραγματοποιηθουν στην εφαρμογη αυτή. Αναλυτικότερα πιο κατω με τη βοηθεια screenshot εξηγουμε την εφαρμογη για κάθε χρηστη της.

**Το Σχεσιακο Διαργαμμα**





**Η ΕΦΑΡΜΟΓΗ**

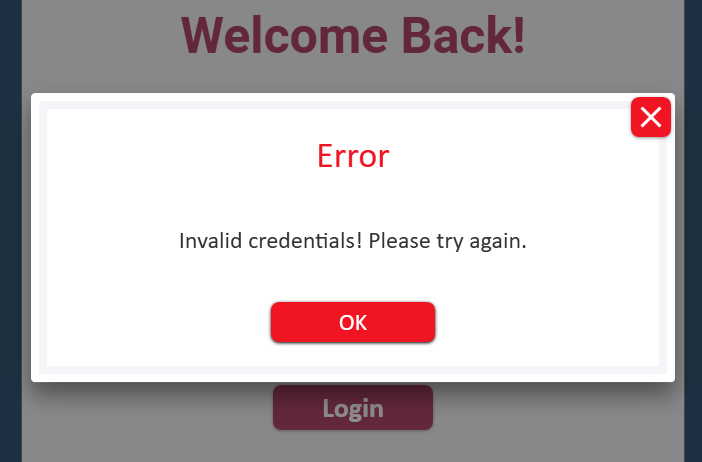
Αρχικα όλοι οι χρηστες όταν ανοιγουν την εφαρμογη βλεπουν το login page



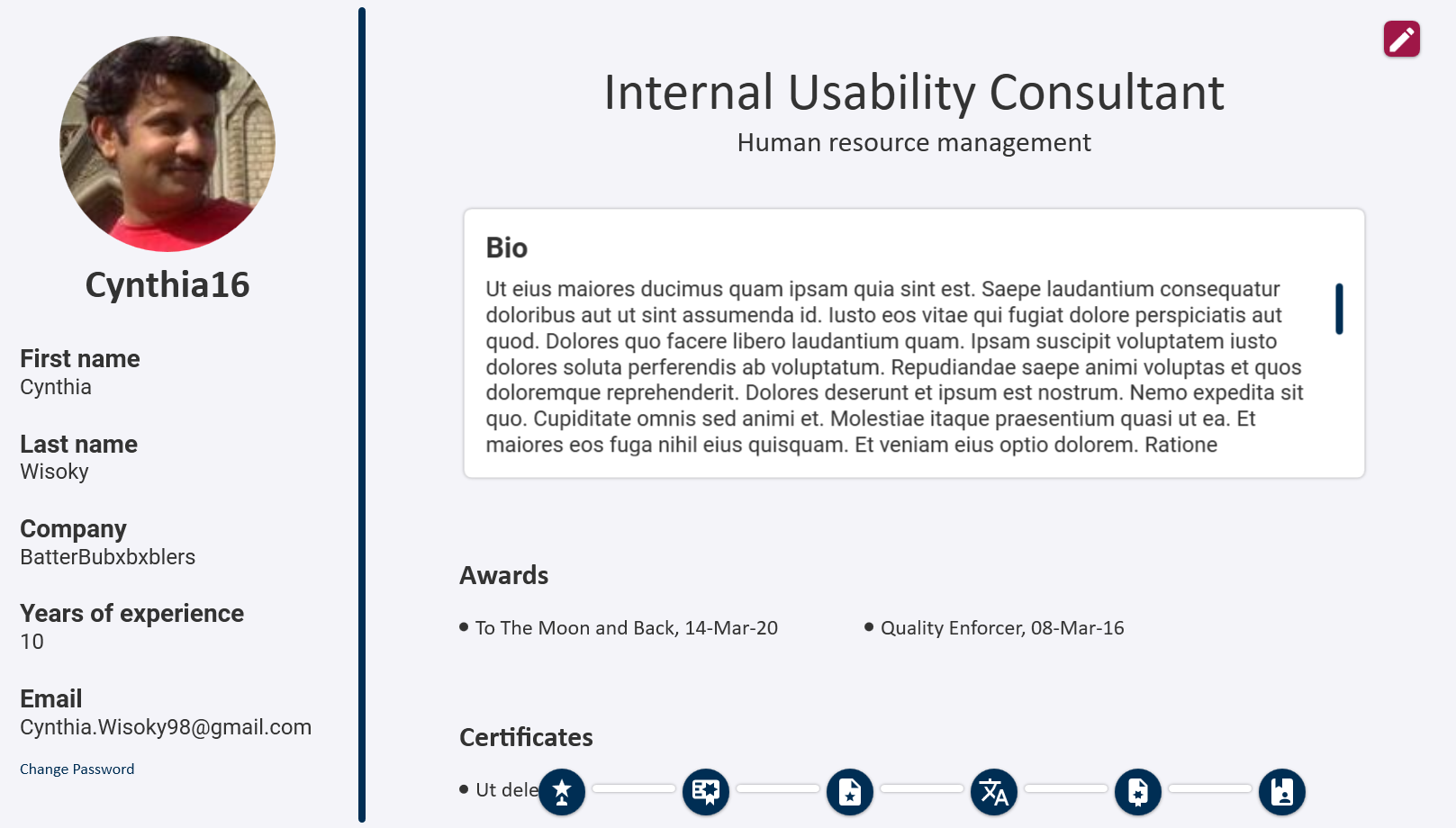
όπου τους ζητειται να βαλουν το username και το password τους.

Όταν πατήσουν το login γινεται αναζητηση στη βαση για καποιον χρηστη που να εχει τα δεδομενα που εχει βαλει ο χρηστης.

Σε περιπτωση που δεν βρεθει στη βαση καποιος χρηστης, δηλαδη εχουν καταχωρηθει λαθος στοιχεια εμφανιζεται αντιστοιχο μηνυμα.

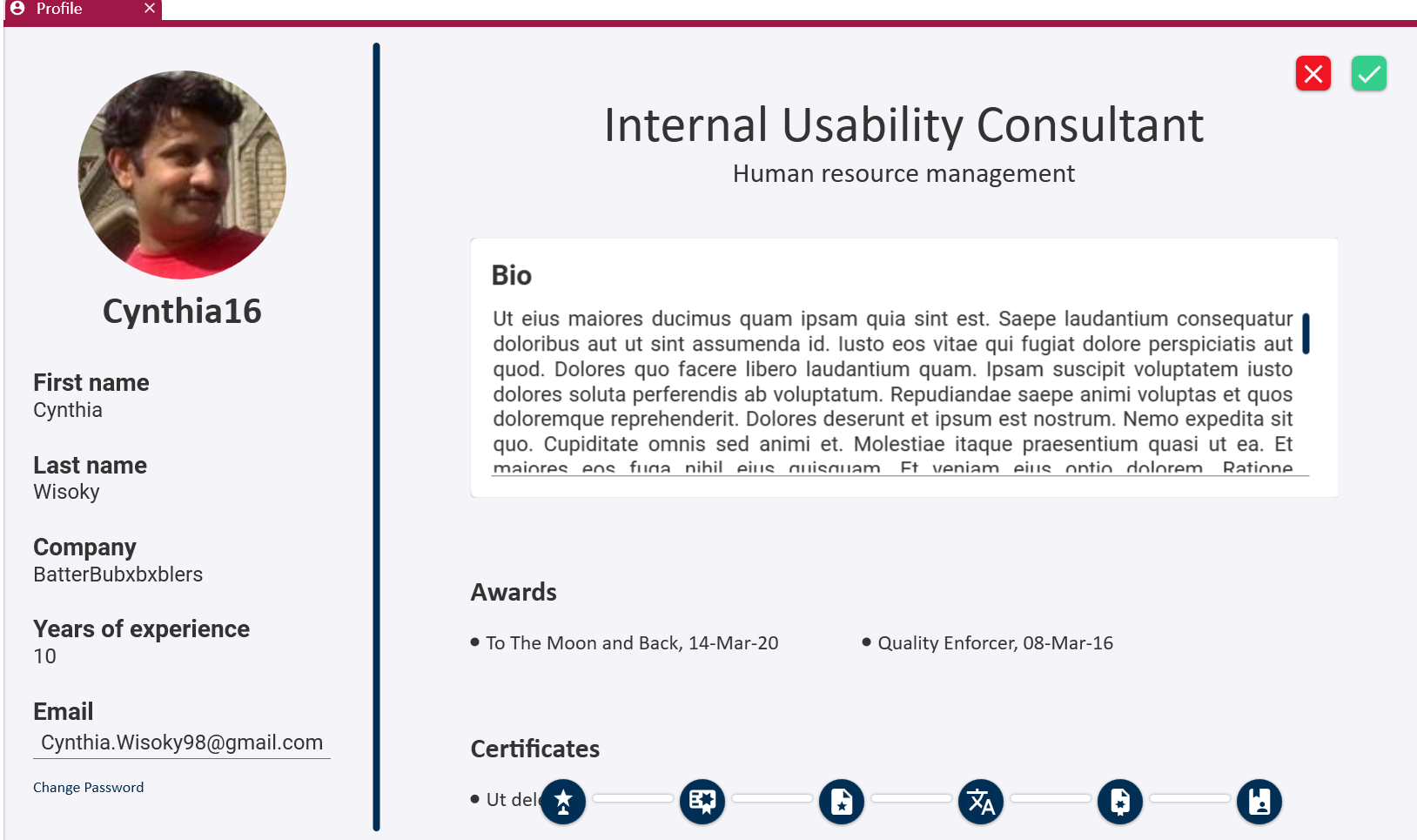


**Αν είναι Employee o χρήστης:**

****

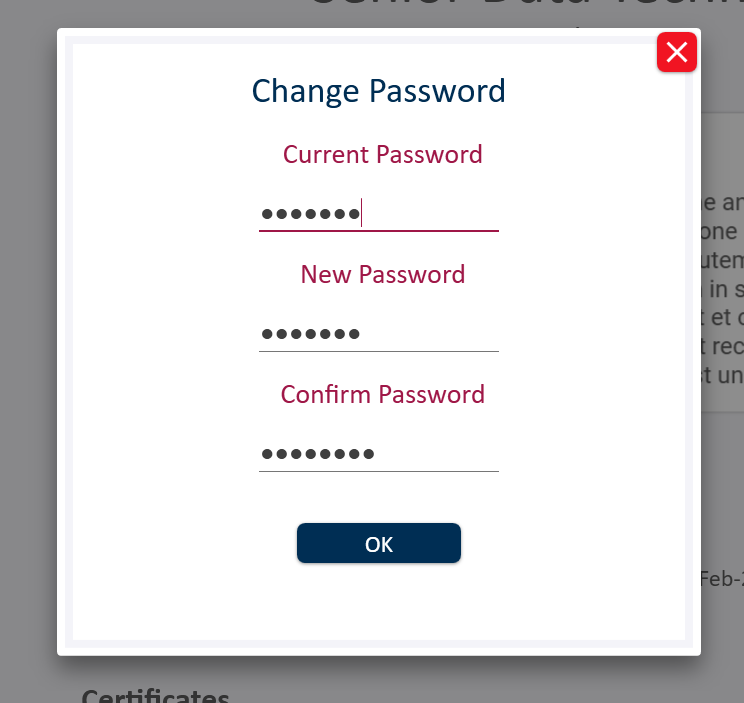
Μπορεί όπως και όλοι οι υπόλοιποι χρήστες, να βλέπει τα προσωπικά του στοιχεία και να τροποποιεί το βιογραφικό , το email και τον κωδικό πρόσβασής του.

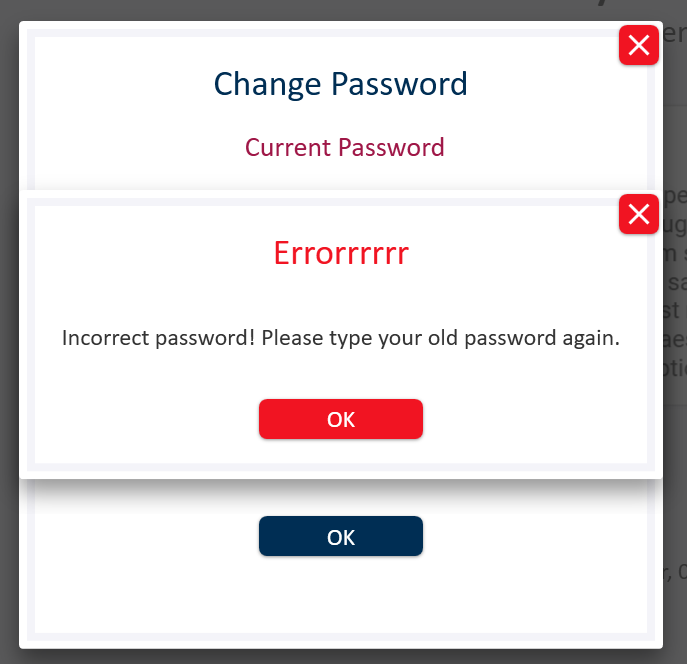
(Τροποποίηση)



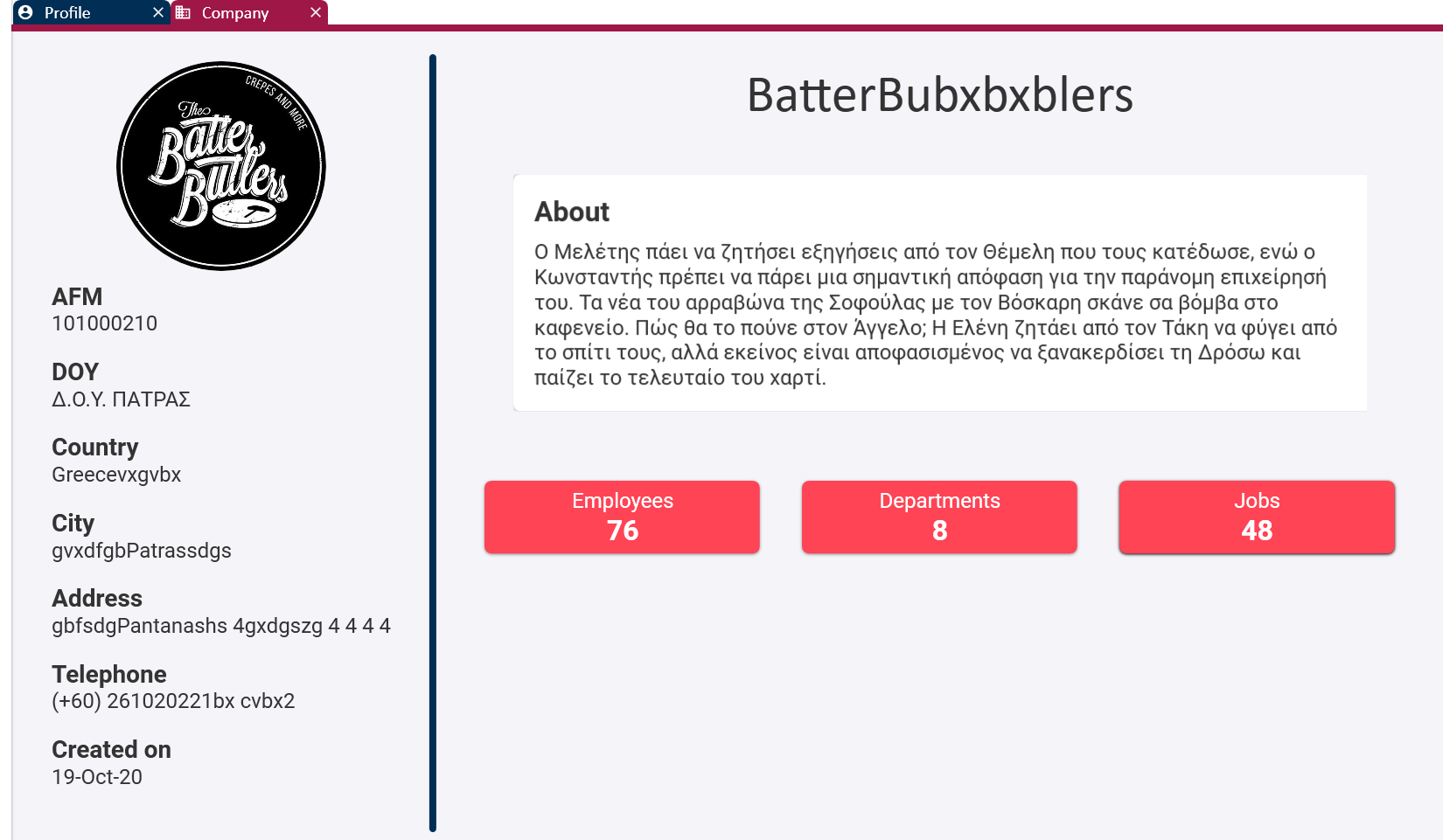
Κάτω από το βιογραφικό εμφανίζεται το αρχείο με όλα τα διπλώματα, τις συστατικές επιστολές, τις εργασίες ακόμα και τις ξένες γλώσσες τις οποίες γνωρίζει και κατέχει ο εκάστοτε εργαζόμενος. Οστόσω, δεν μπορεί να τα τροποποιήσει. Μόνο ο manager που είναι υπεύθυνος για το τμήμα το συγκεκριμένου μπορεί μονάχα να του προσθέσει κάποιο επίτευγμα.

Πατώντας το κουμπί changePassword μπορεί να αλλάξει τον κωδικό του

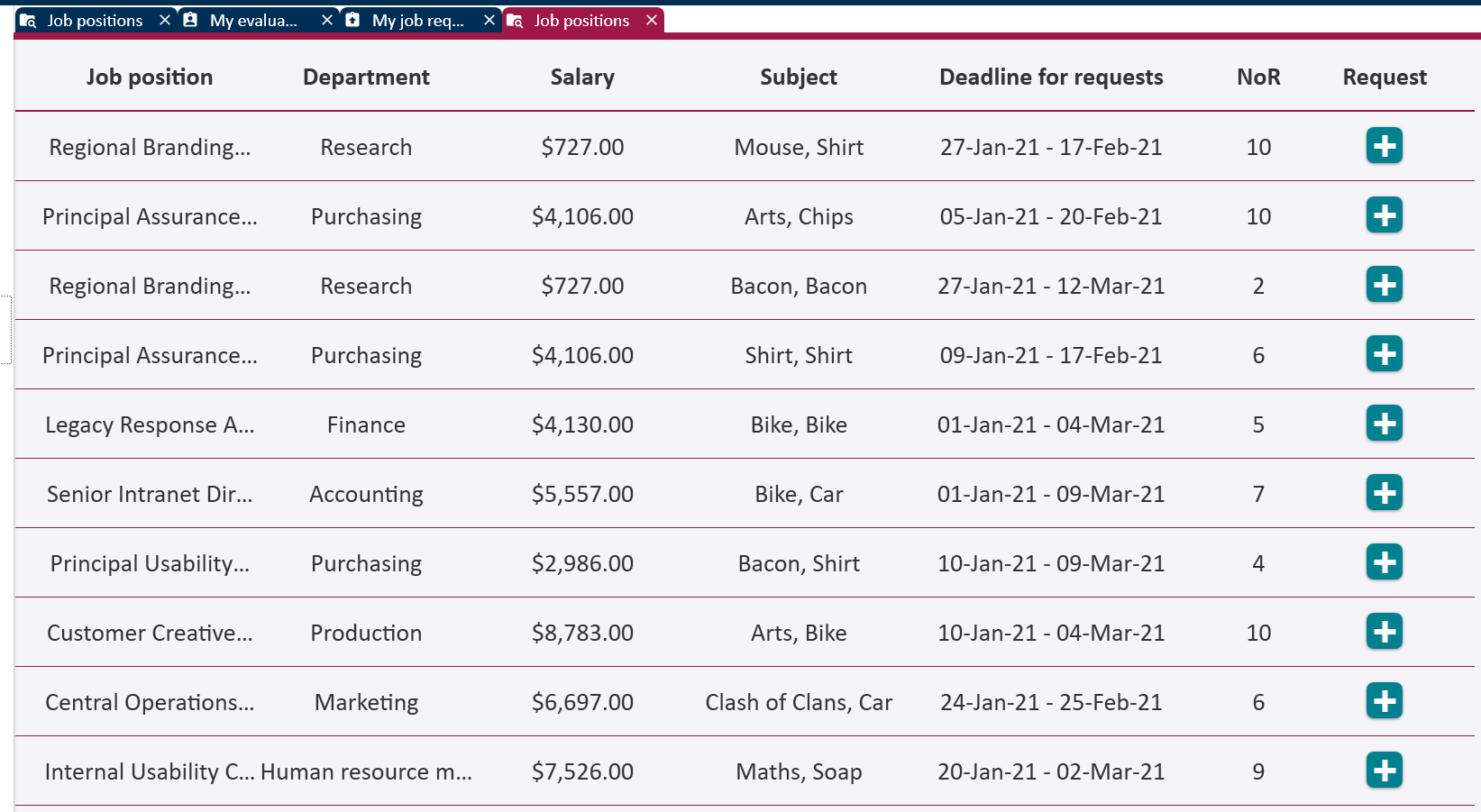


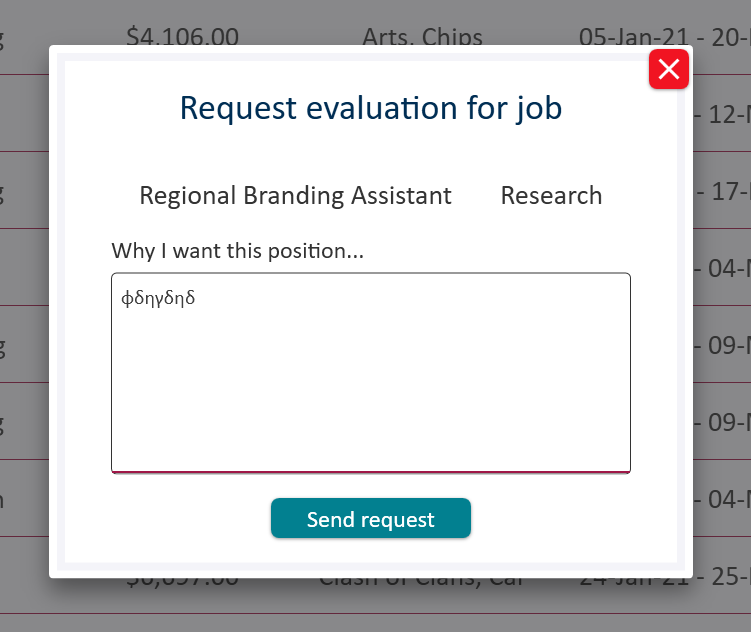


Μπορεί επίσης όπως και οι υπόλοιποι χρήστες να βλέπει τα στοιχεία της εταιρείας του.



Έπειτα, μπορεί να βλέπει τις θέσεις εργασίας που είναι διαθέσιμες για αίτηση ώστε να αλλάξει την τωρινή του . Υποβάλει αίτημα και έπειτα, το αίτημα αυτό έως ότου λάβει τα απαραίτητα (αξιολόγηση από evaluator και report από manager), παραμένει αρχειοθετημένο.

****



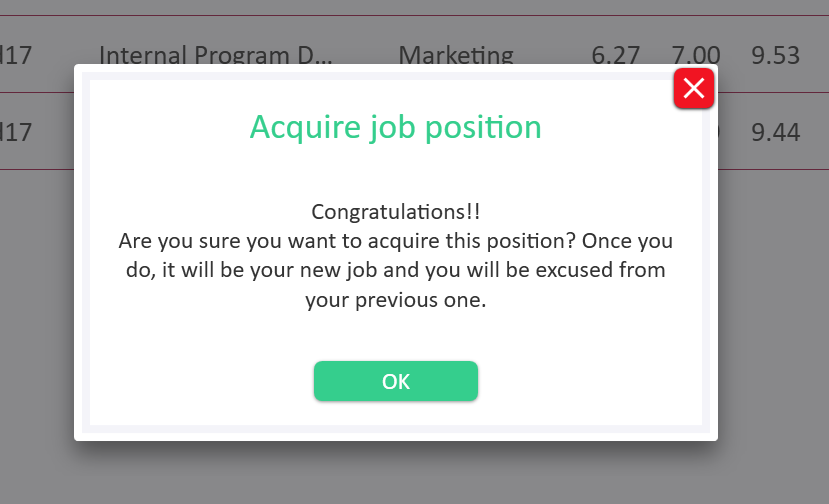
Εδώ ο εργαζόμενος μπορεί και βλέπει τις αιτήσεις που έχει κάνει. Αν θέλει, μπορεί να τις ακυρώσει.(μια κάθε φορά)

****

****

Για εκείνες στις οποίες υπέβαλε αίτηση , δεν την ακύρωσε και αξιολογήθηκε επ’αυτών, μπορεί και βλέπει τα αποτελέσματα:****

Αν σε κάποια από αυτές προήχθη ,μπορεί να πατήσει pass και να κατωχυρώσει τη θέση! (Η θέση δεν κατωχυρώνεται αυτόματα, αλλά πηγαίνει πρώτα στον manager ώστε να επιλέξει εκείνος εν τέλη, αν αυτέ η δουλειά πρέπει να πάει σε αυτόν τον εργαζόμενο(αφού μπορεί να έχει κατωχυρώσει περισσότερες απο μια ταυτοχρόνως)).



Τέλος, αφότου ελέγξει όσα ήθελε, μπορεί να αποσυνδεθεί από το σύστημα πατώντας το κουμπί log out. (Καθε χρήστης μπορεί να κάνει το ίδιο).

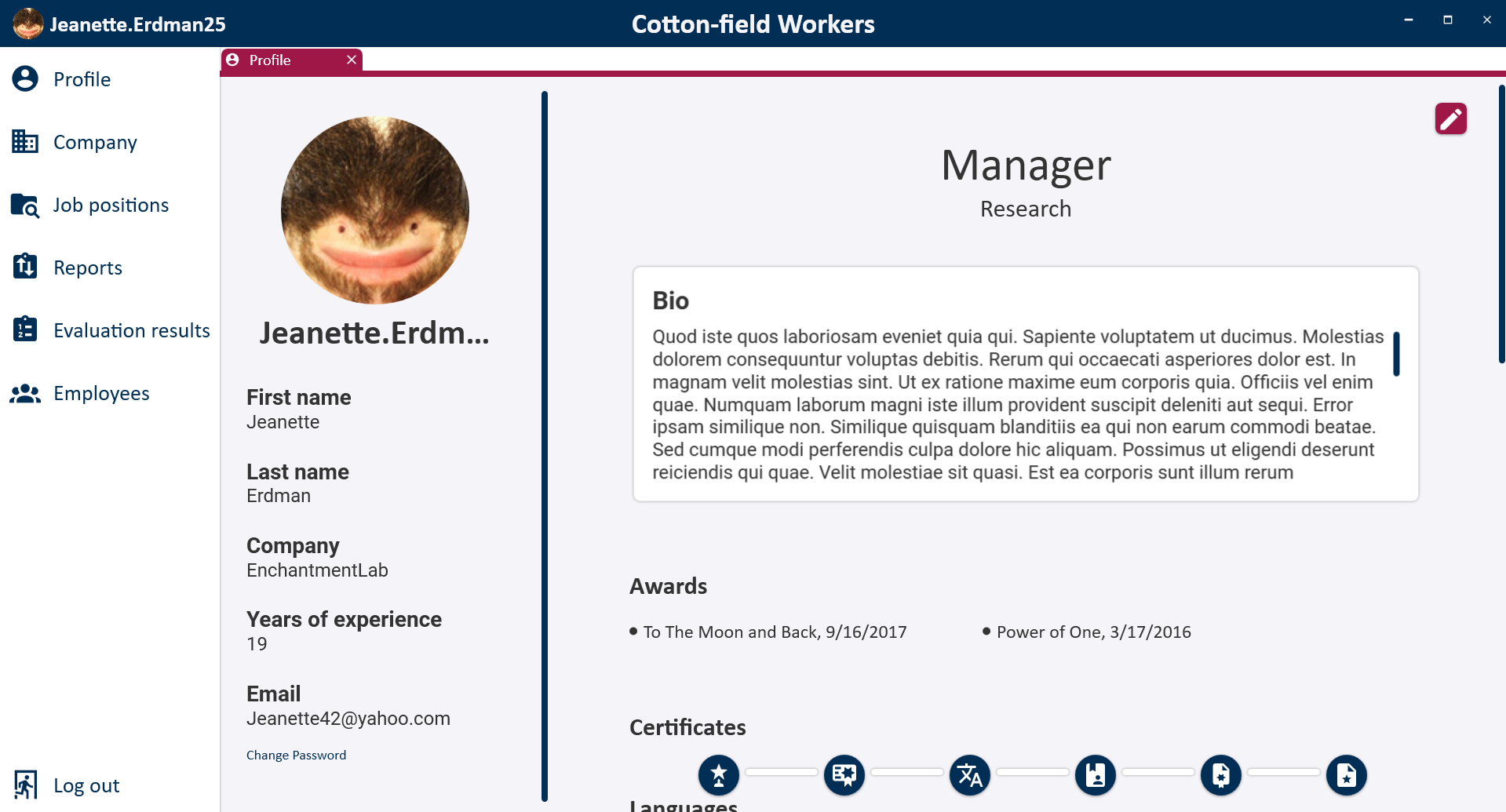


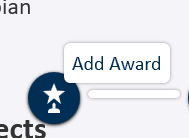
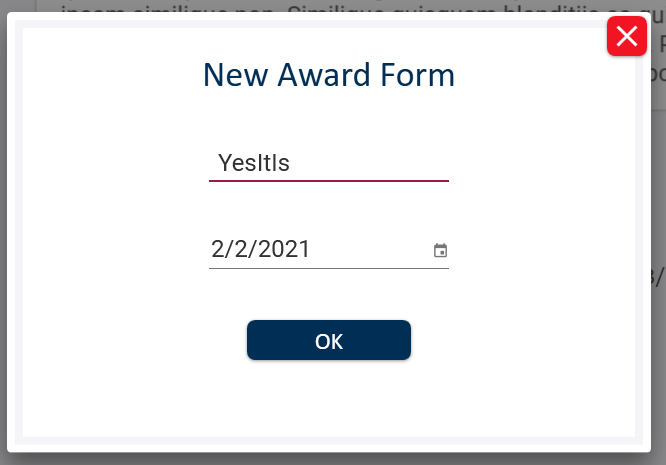
**Αν είναι Manager o χρήστης:**

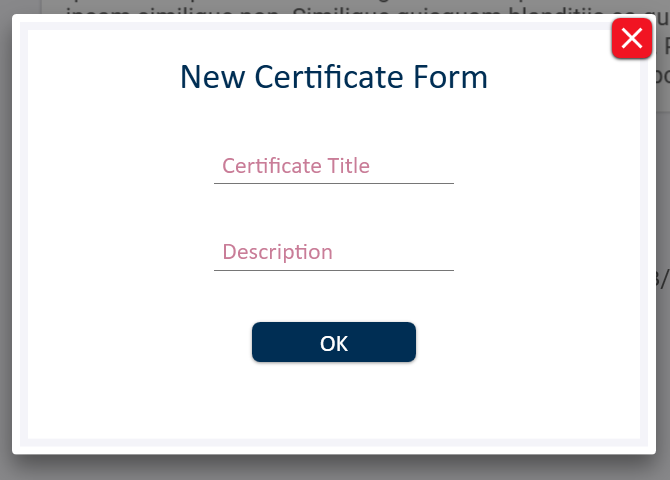
Όπως και ένας υπάλληλος μπορεί να ανοιξει μια καρτελα με το προφιλ του καθως και της δικης του εταιριας.

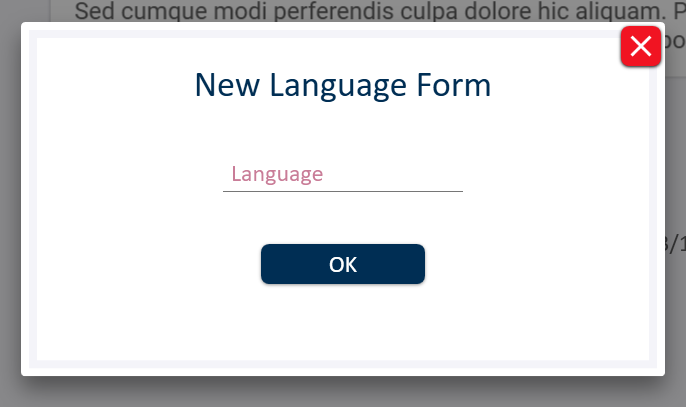
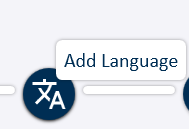
Σε αυτές ο ιδιος όμως εχει τη δικαιοδωσια να επεξεργαστει και καποια στοιχεια.

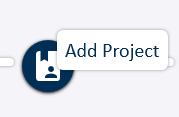
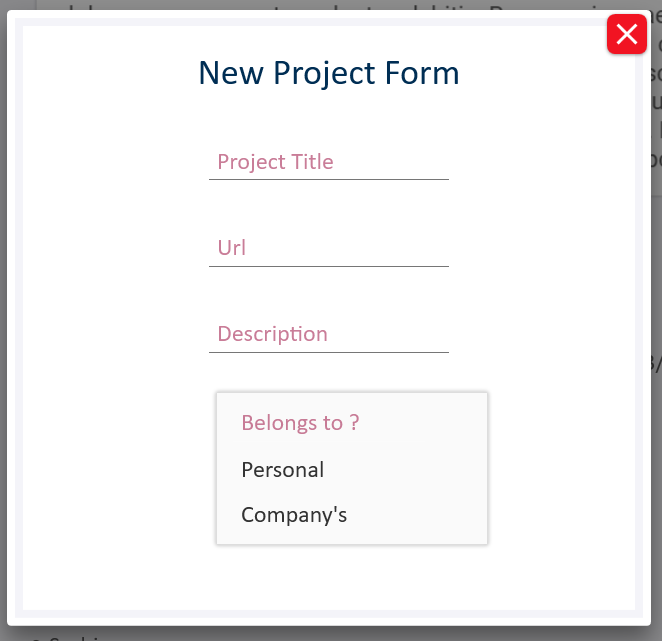
Παρακατω φαινεται το προφιλ του και με τα κατω κουμπια μπορει να προσθεσει βραβεια, certificates, recommendation papers, projects, degrees και γλωσσες. Το κατω μενου εμφανιζεται σε αυτόν αλλα και σε οποιον υπαλληλο επιλεξει να δει από την «Employees».

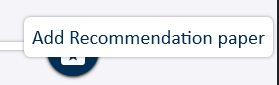
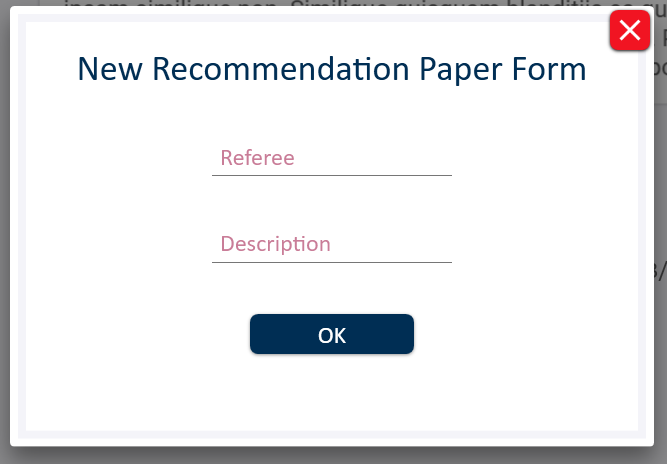








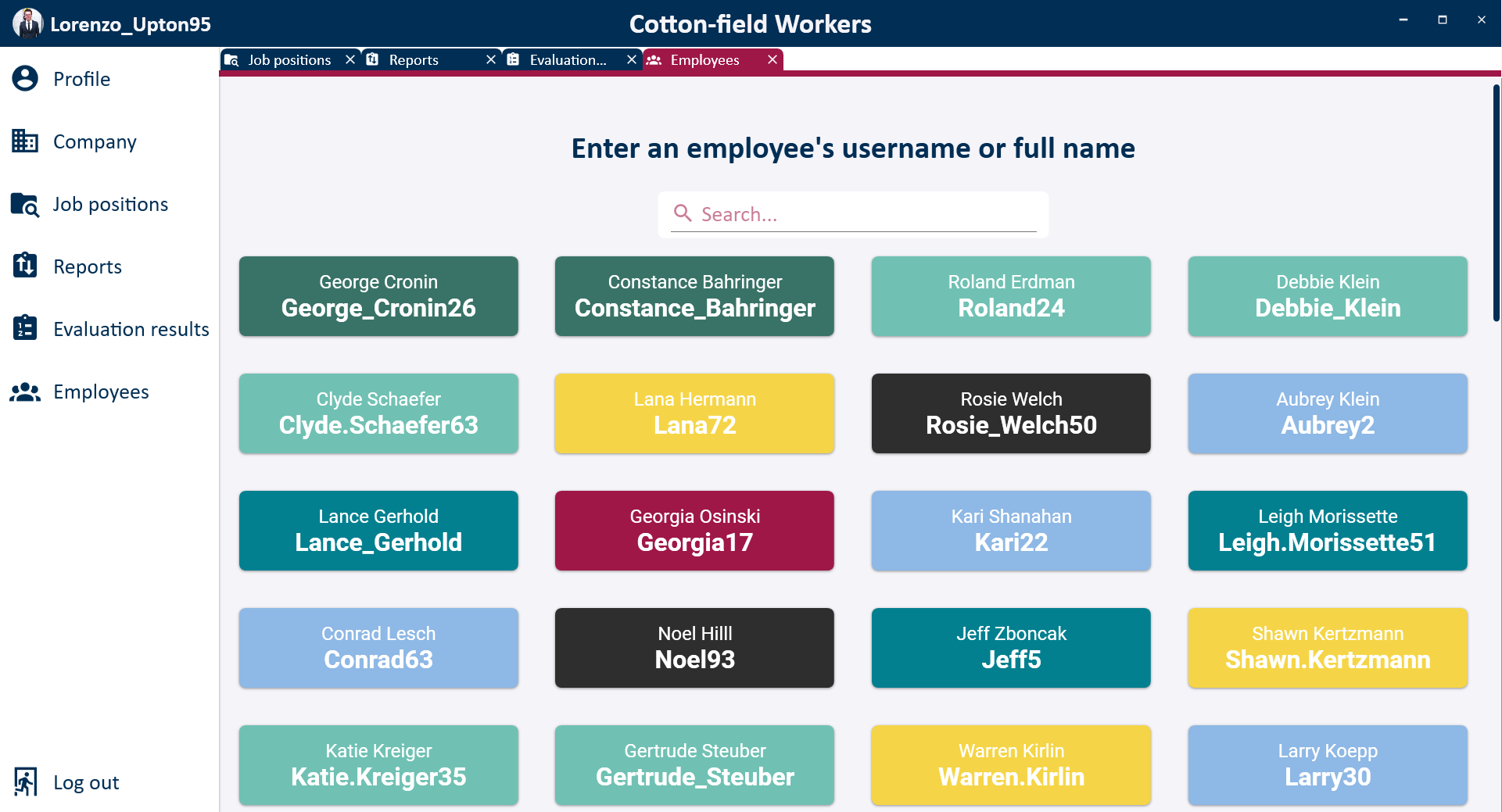


Με το που παταει ΟΚ δημιουργειται και προσθετεται στο προφιλ του το νέο τρεητ του.

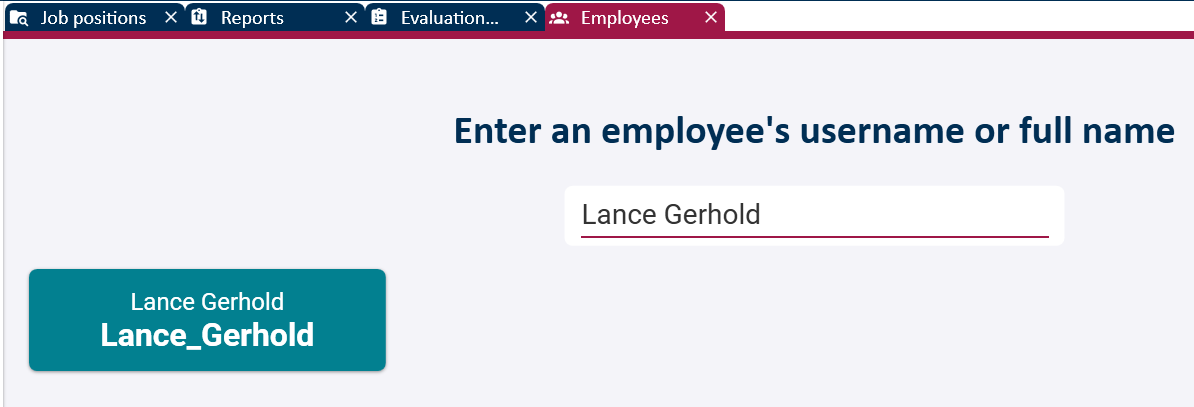
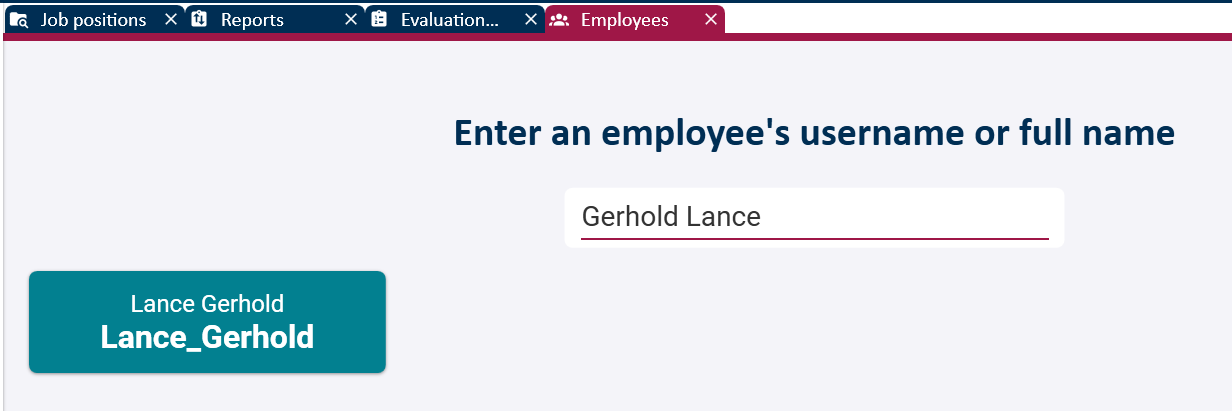
Μια άλλη καρτέλα που μπορει να ανοιξει είναι η “Job positions”. Σε αυτή μπορει να δει πληροφορίες για όλες τις ανοιχτες θέσεις εργασιας, όπως για παράδειγμα τον συνολικο αριθμο αιτησεων που εχουν γινει για μια συγκεκριμενη θεση και την τελευταία ημερα υποβολης αιτησεων, καθώς και να επεξεργαστεί τον μισθό τους.



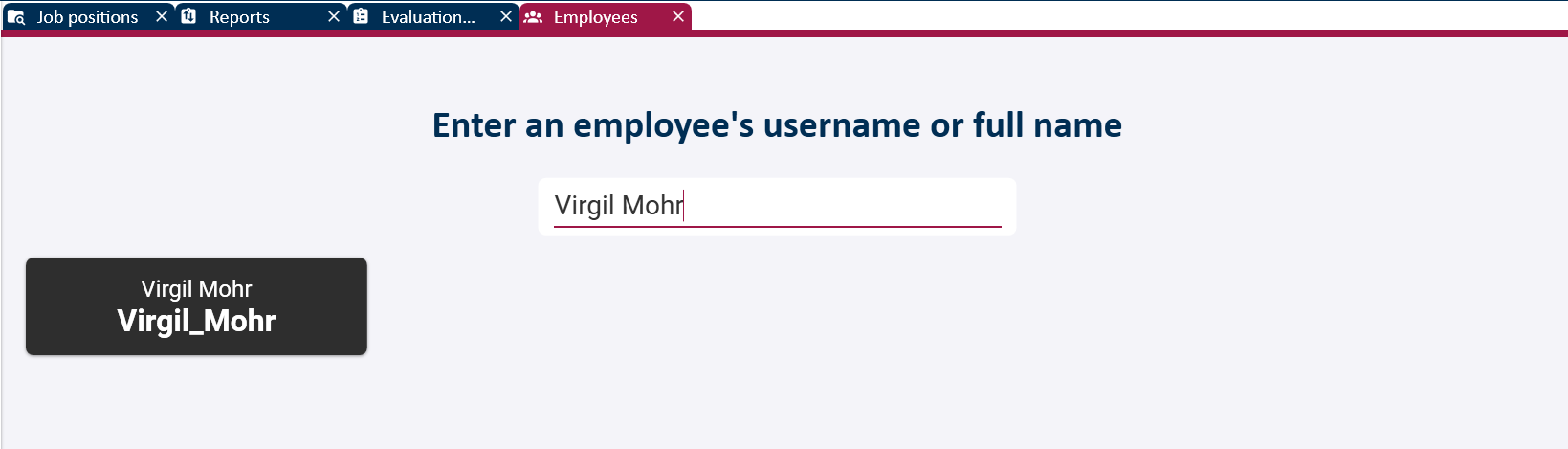
Επιπλέον μπορει να δει σε μια νεα καρτέλα όλους τους υπαλληλους της εταιριας στην οποια δουλευει. Όπως φαίνεται και στην εικονα παρακατω έχουμε δημιουργήσει για κάθε υπάλληλο ένα κουμπι που τον αντιπροσωπευει. Κάθε κουμπι έχει το χρώμα του τμήματος-deparmtent στο οποιο ανηκει ένας χρηστης. Ακομα έχει το Όνομα και το Επώνυμο του υπαλληλου και το username του. Μια δυνατότητα που δινεται στον manager είναι να αναζητησει κάποιον υπαλληλο πληκτρολογώντας είτε το όνομα και το επώνυμο του με οποιαδήποτε σειρά είτε το username του.



Αναζήτηση χρηστη με ονομα και επώνυμο:



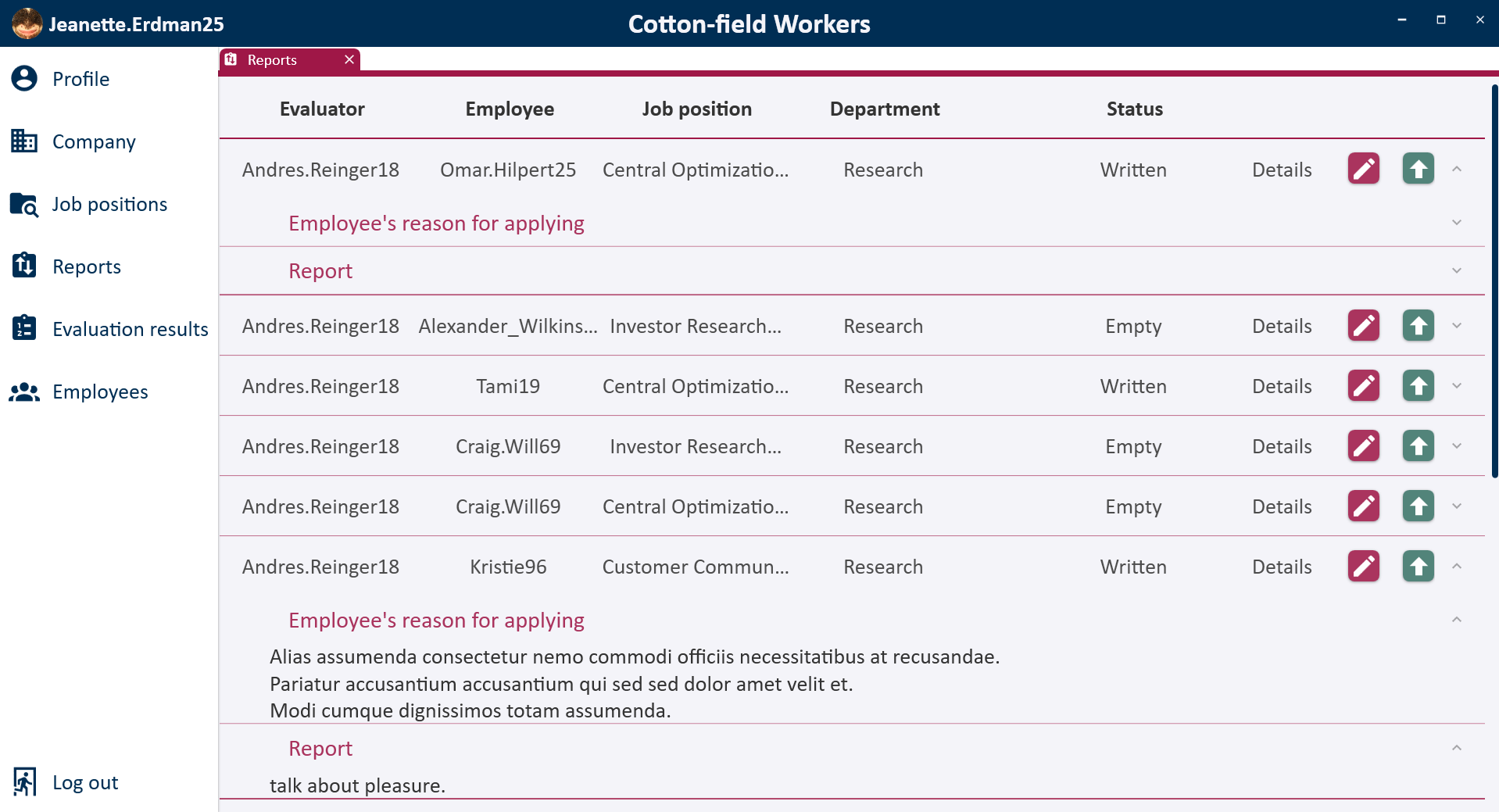
Αναζήτηση χρηστη με username:



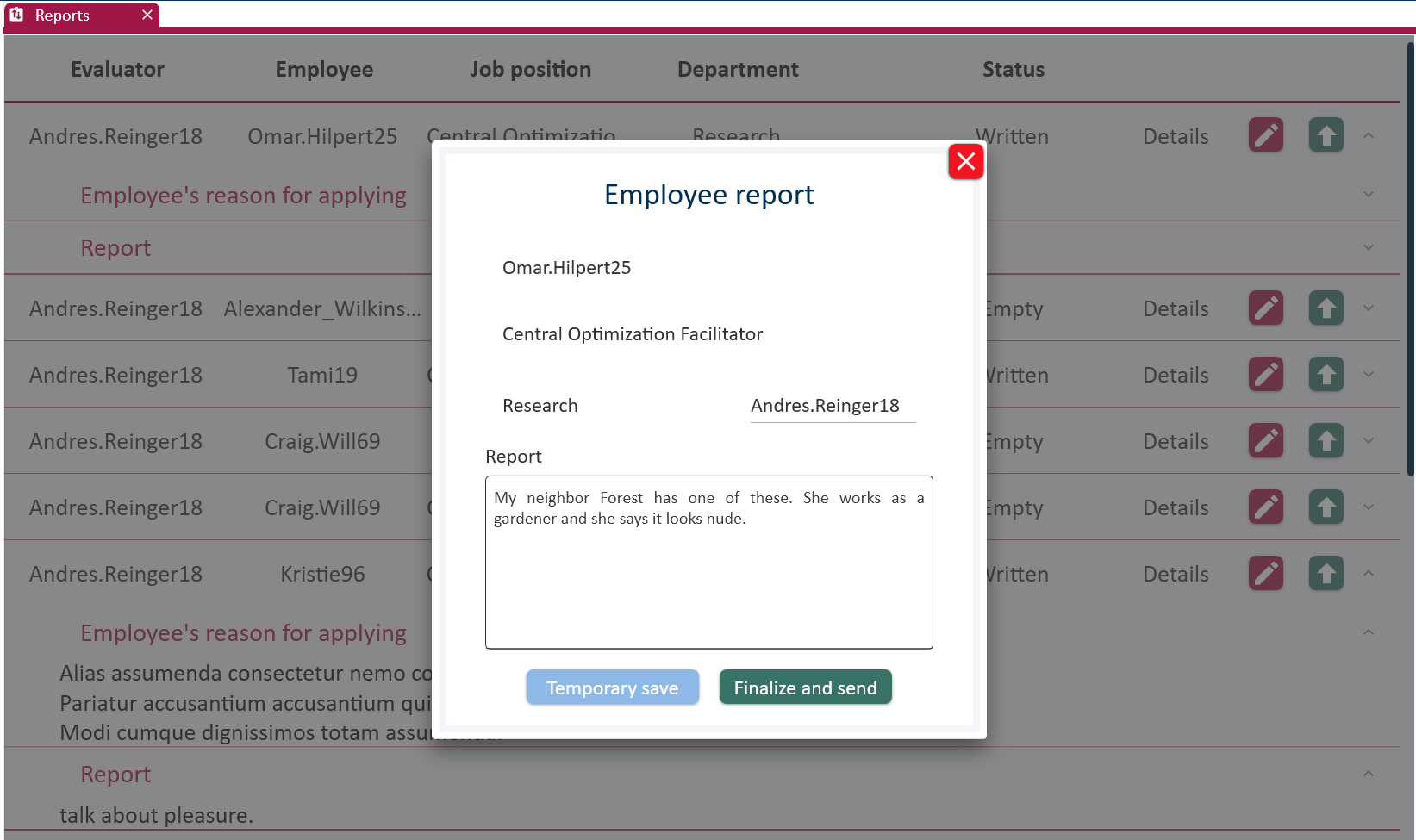
Σε περιπτωση που κάνει κλικ πάνω σε κάποιο κουμπι-υπάλληλο τον κατευθύνει στο προφίλ του συγκεκριμένου υπαλλήλου όπου του δίνεται η δυνατότητα εκτος από το να δει στοιχεια του να επεξεργαστει και καποια από αυτά.

………………………..

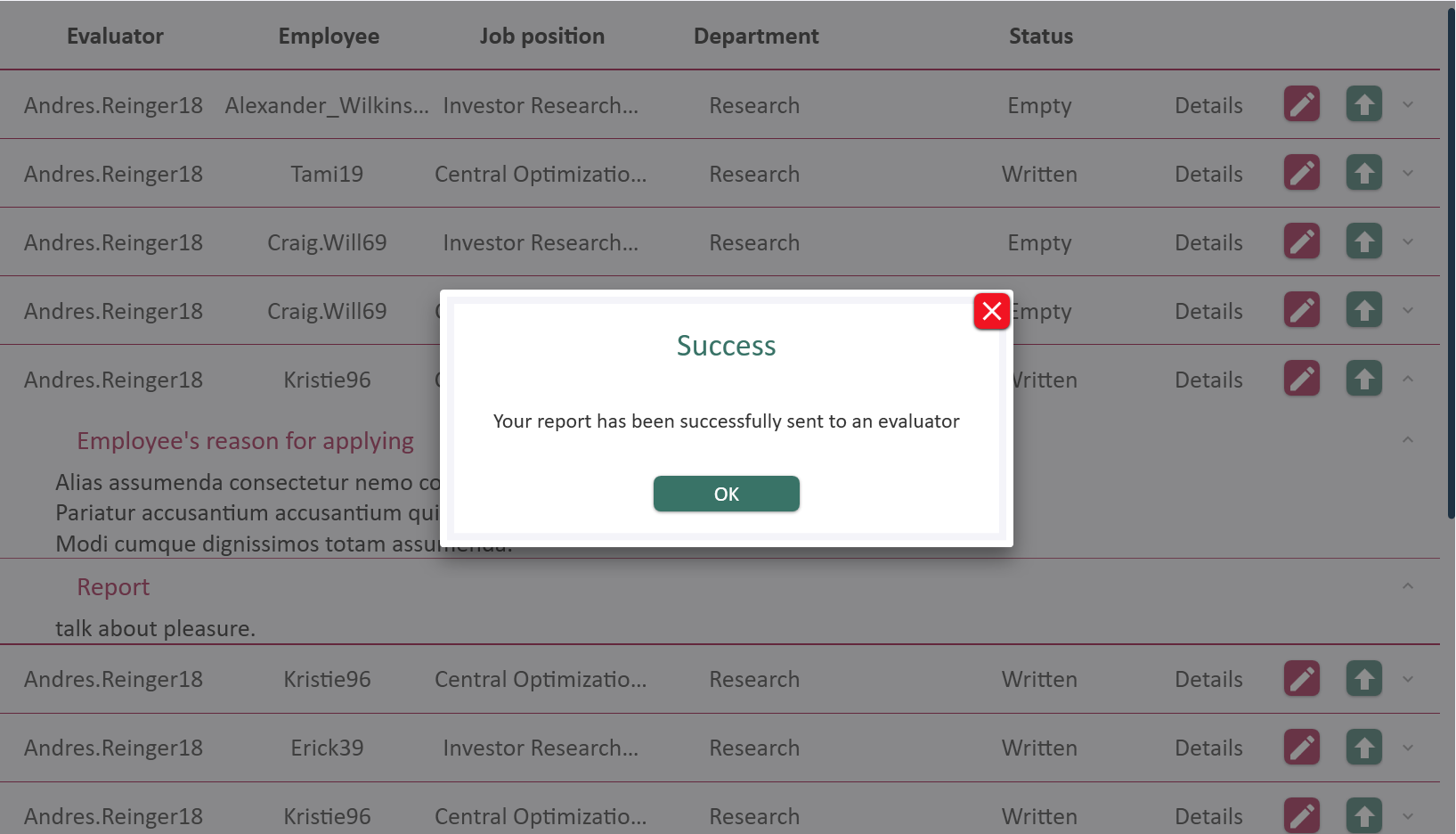
Παρακατω φαινεται η καρτελα με τις αναφορες του Manager. Σε περιπτωση που είναι κενή, είτε επιδη εχει διαγραψει μια προηγουμενη είτε επειδη μολις εχει δημιουργειθει μια αιτηση για αξιολογηση από καποιον υπαλληλο, στο status εχει το empty, αλλιως written. Η καρτελα αποτελειται από ένα data grid του οποιου κάθε γραμμη είναι ενας expander που μεσα εχει αλλους δυο empanders που εχουν το κειμενο που συνεταξε ο υπαλληλος αναφεροντας γιατι θελει τη δουλεια και την αναφορα του ιδιου του manager.



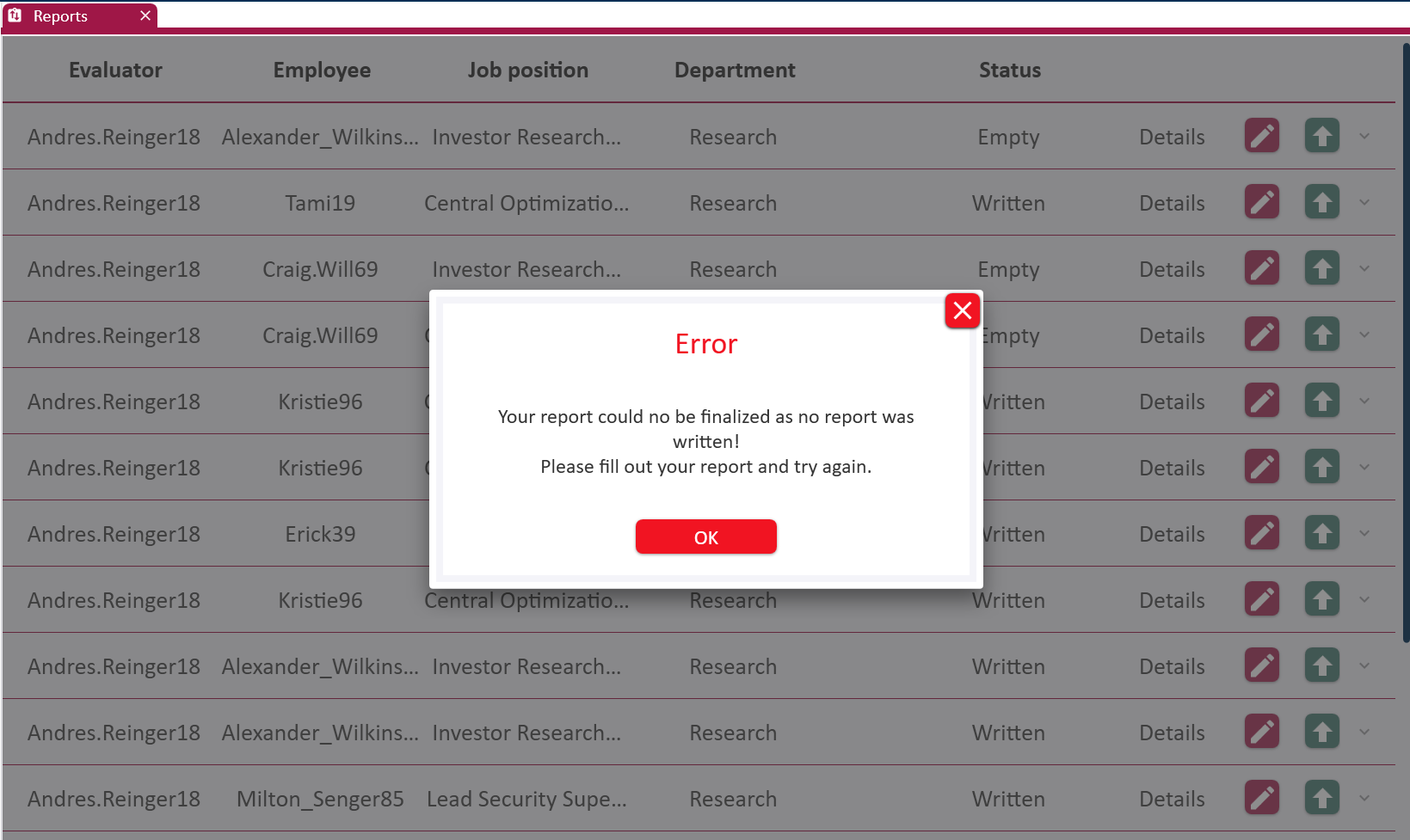
Σε περιπτωση που επιλεξει να επεξεργαστει ή να γραψει την αναφορα του παταει το ροζ edit button και του εμφανιζεται η παρακατω φορμα στην οποια δηλωνονται αυτοματα το username του υπαλληλου, η θεση εργασιας για την οποια ζητησε να αξιολογηθει καθως και το τμημα στο οποιο βρισκεται η θεση εργασιας αυτή. Ο manager επιλεγει σε ποιον evaluator επιθυμει να σταλει αυτή η αναφορα και στο πλαισιο συντάσσει την αναγορα του. Όταν τελειωσει με την επεξεργασια του του δινεται η δυνατοτητα να αποθηκευσει προσωρινα, ώστε να την ελεγξει ξανα καποια άλλη στιγμη ή να οριστικοποιησει την αναφορα του και να σταλει απευθειας και αυτοματα να δημιουργηθει ένα αιτημα αξιολογησης στον evaluator που επελεξε ο ιδιος.



Σε περιπτωση που πατήσει να οριστικοποιηθει μια αναφορα του που εχει συναξει εμφανιζεται το παρακατω μηνυμα και η στηλη IsFinalized στον πινακα Reports από 0 γινεται 1.



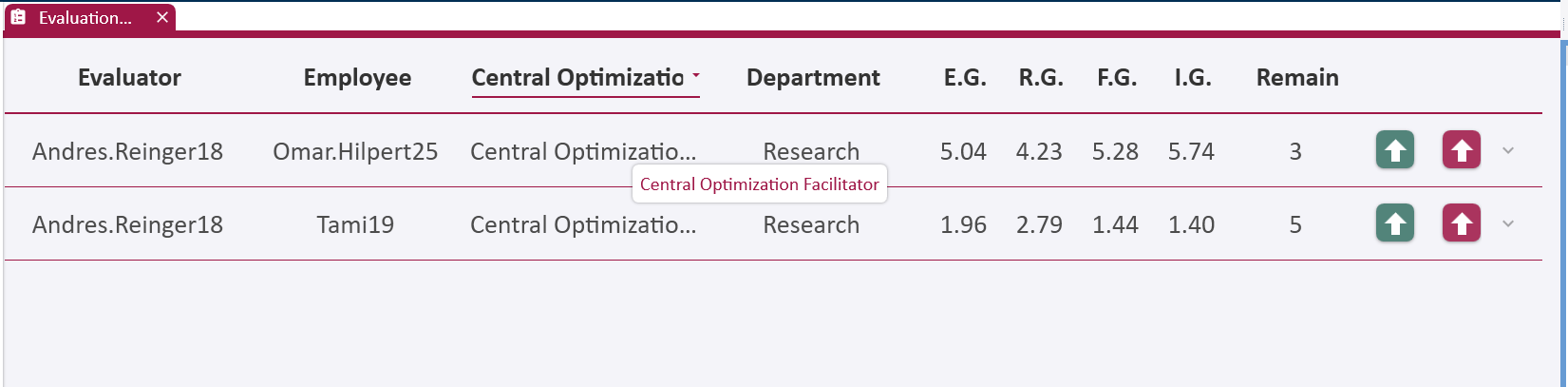
Σε περιπτωση όμως που πατησει να οριστικοποιηθει μια αδεια αναφορα του εμφανιζεται ένα error μηνυμα και φυσικα δεν οριστικοποιειται η αναφορα αυτή.



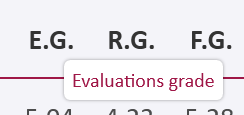
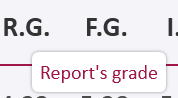
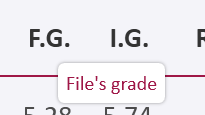
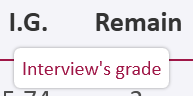
Μια ακομα καρτελα που βλεπει ο manager είναι η “Evaluation Results” στην οποια φαίνονται οι ολοκληρωμενες αξιολογησεις σορταρισμενες με τον τελικο βαθμο της αξιολοογησης σε φθινουσα σειρα.

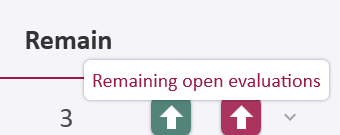
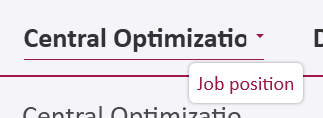


Το job position στη κεφαλιδα είναι ένα φιλτρο με το οποιο μπορει να διαλεξει να δει τις αξιολογησεις που αφορουν μια και μονο ανοιχτη θεση εργασιας. Για παραδειγμα αν επιλεξει την Central optimization Facilitator του εμφανιζονται ολες οι ανοιχτες θεσεις εγασιας που αφορουν τη συγκκριμενη δουλεια, για αυτό και στο παρακατω παραδειγμα στη στηλη Remain που αναφερει τον αριθμο τον αξιολογησεων που απομενουν να ολοκληρωθουν είναι διαφορετικοι:

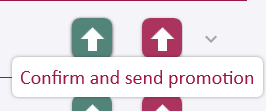
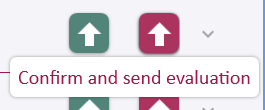


Κάθε στηλη στην κεφαλιδα, καθως και στα κουμπια, εχει ένα tooltip που εξηγει ακριβως τι είναι κάθε στηλη. Για παραδειγμα:

Για το πρασινο: Για το κοκκινο:

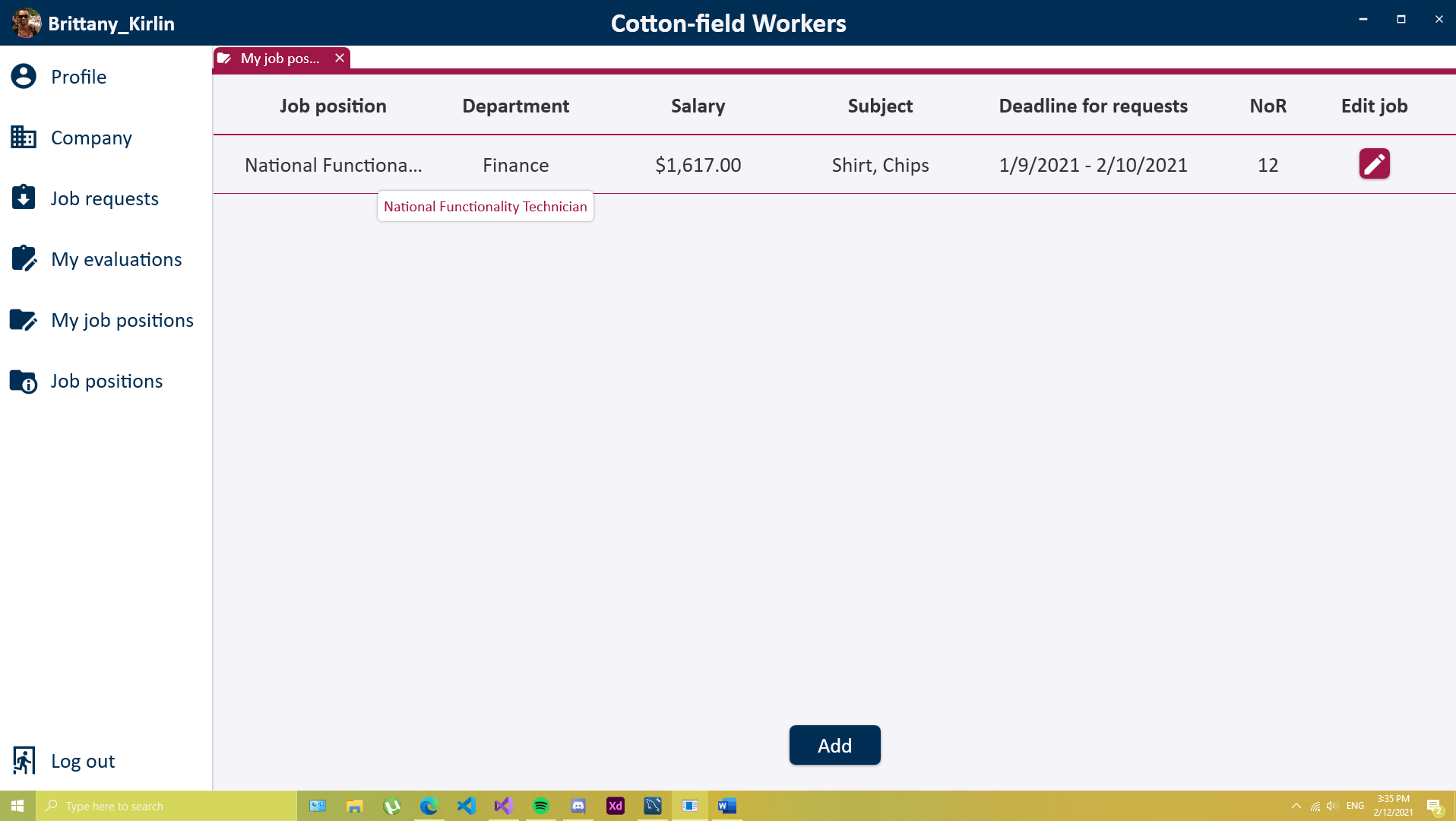
Σε περιπτωση που επιλεξει το πρασινο στέλνετε στον υπαλληλο και τον ενημερωνει ότι περασε την αξιολογηση ενώ στην περιπτωση του κοκκινου τον ενημερωνει ότι δεν καταφερε να παρει τη θεση.

**Αν είναι Evaluator o χρήστης:**

Εχει το δικαιωμα να δει στην καρτελα job positions ολες τις ανοιχτες θεσεις εργασιας που εχουν δημιουργηθει στην εταιρια του με τον αριθμο των αιτησεων που εχουν δημιουργηθει συνολικα για κάθε μια από αυτές, καθως και το τμημα στο οποιο βρισκονται τις προθεσμιες αλλα και τα αντικειμενα τα οποια αφορουν.

****

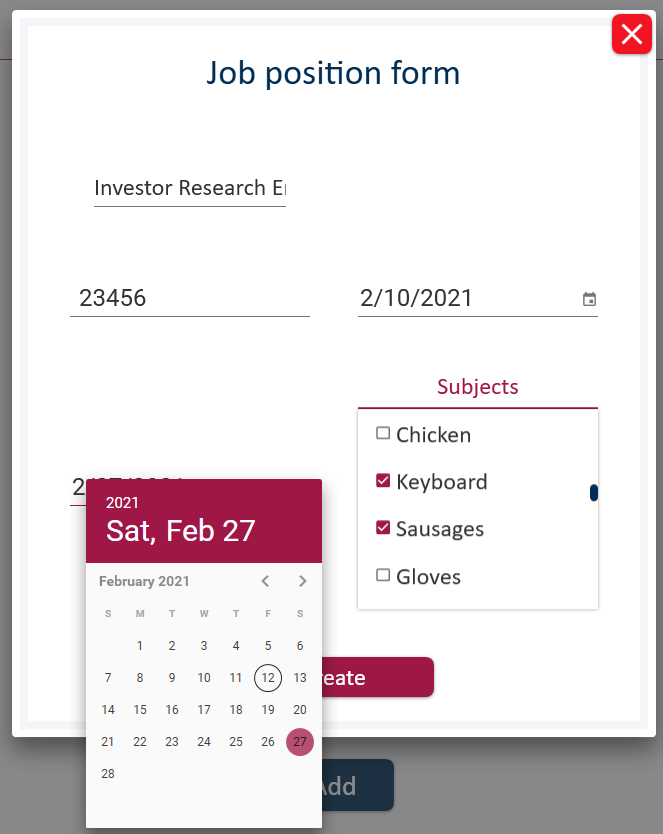
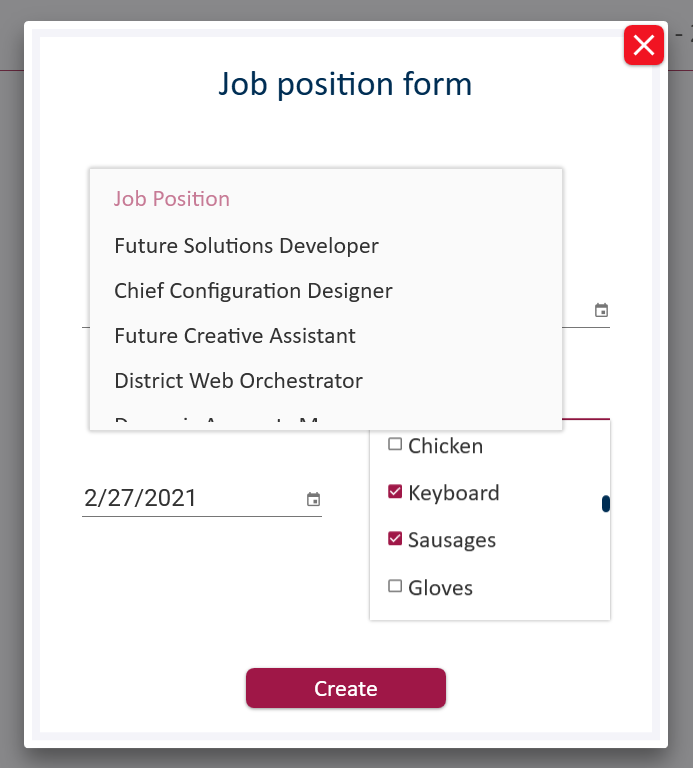
Ακομα αν το επιθυμει μπορει πηγαινοντας στο «My job positions» να δημιουργησει αλλα και να επεξεργαστει δικες του ανοιχτες θεσεις εργασιας.



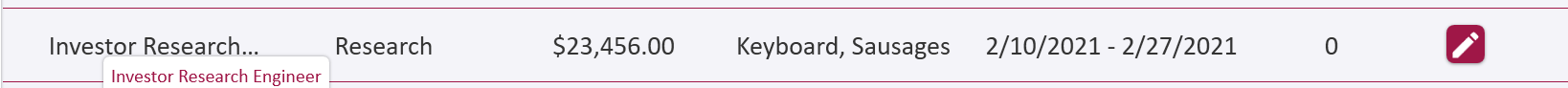
Πατώντας το κουμπι “Add” εμφανιζεται μια φορμα στην οποια μπορει να επιλεξει από τις υπαρχουσες δουλειες που εχει η εταιριας να δημιουργησει μια θεση. Ετσι γλιτωνουμε τυχον ερρος σε περιπτωση που προσπαθουσε να δημιουργησει καποια θεση που δεν υπαρχει αντιστοιχη δουλεια στην εταιρια.



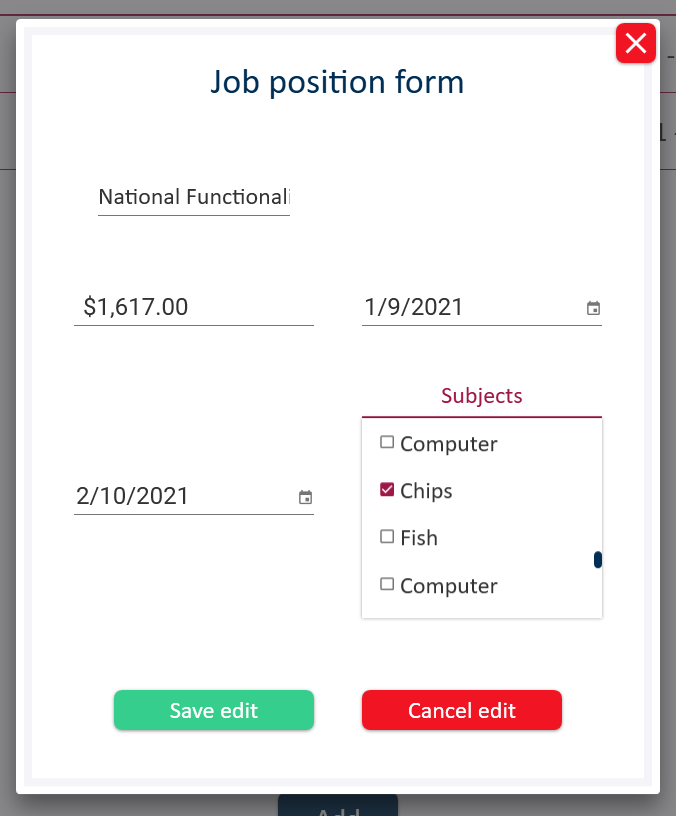
Τα ονοματα εργασιων που μπορει να επιλεξει: Τα θεματα και η ημερομινια :



Και ετσι στην παρακατω εικονα φαινεται η θεση που μολις δημιουργησε ο συγκεκριμενος evaluator.

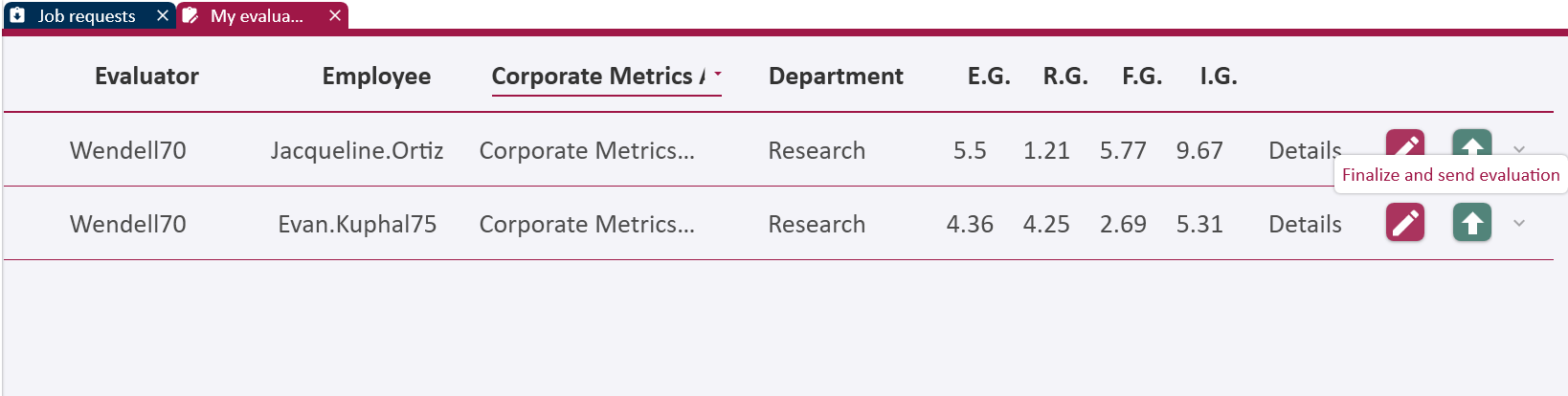


Σε περιπτωση που επιθυμουσε να την επεξεργαστει παταει στο ροζ κουμπι και εμφανιζεται η παρακατω φορμα.



Ακομα όπως φαινεται και πιο κατω μπορει να δει στην καρτελα “Job requests” τις συνολικες αιτησεις που εχουν γινει στην εταιρια. Αν κανει hover πανε από κάθε δεδομενο στο data grid εμφανιζεται ένα tooltip που εμφανιζει ολοκληρο το κειμενο, όπως πχ το username ενός employee. Ακομα μπορει να κανει το ιδιο και στο header, πχ NoR = Request’s number.

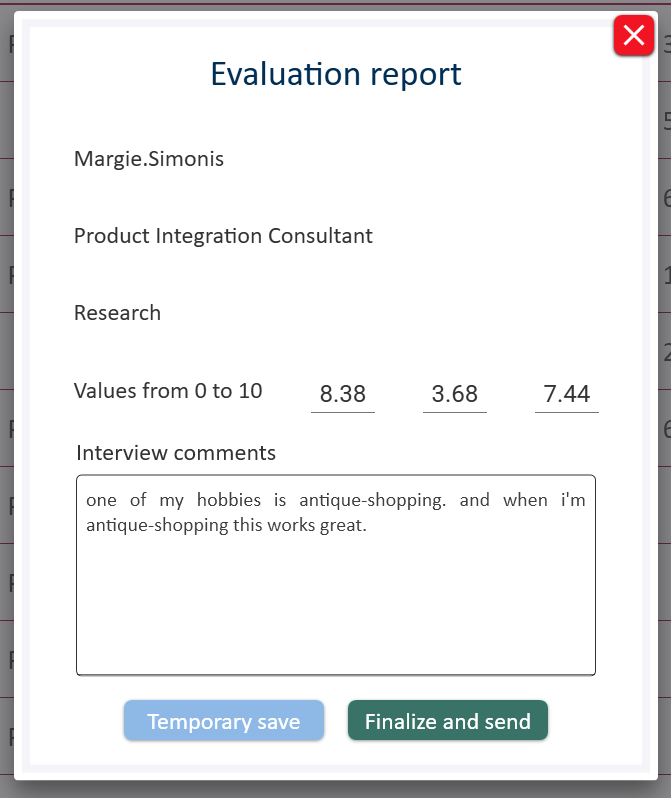


Tελος, μπορει ανοιγοντας την καρτελα “My evaluations” κάνοντας κλικ στο διπλανο μενου, να δει τις αξιολογησεις που έχει κανει ή που πρεπει να κανει και δεν εχουν οριστηκοποιηθει ακομα. Είναι παλι ένα data grid οπου κάθε γραμμη είναι ενας expander που εχει μεσα του αλλους δυο expanders έναν για τα σχολια του αξιολογητη από την συνεντευξη και έναν για την αναφορα του διευθυντη. Υπαρχει ενας picker στο job position οπου μπορει να επιλεξει να δει τις αξιολογησεις του που αφορουν μια συγκεκριμενη θεση εργασιας μονο όπως φαινεται στην εικονα πιο κατω. 

Ενώ παρακατω φαινονται ολες αξιολογησεις που εχει να κανει.



\* Όσες εχουν 0 δεν εχουν πραγματοποιηθει ακομα.

Μπορει να επεργαστει τις αξιολογησεις του πατωντας το ροζ κουμπι και τοτε του εμφανιζεται στη διπλανη φορμα.

Αρχικα φαινεται το username του υπαλληλου,

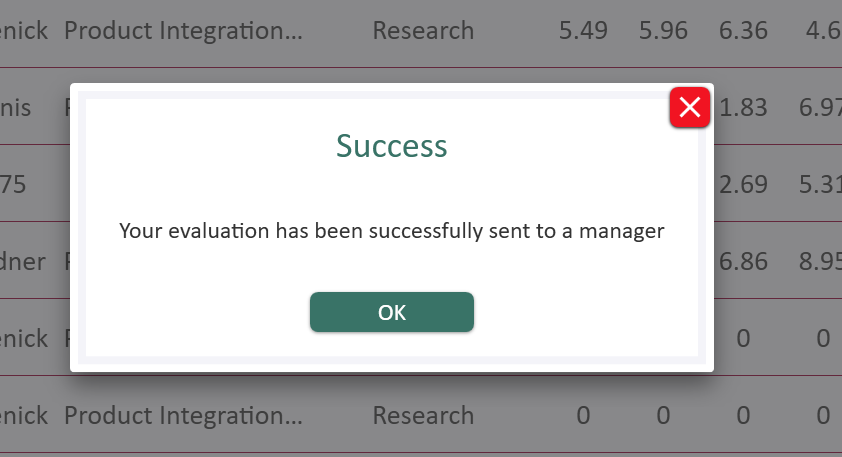
από κατω το ονομα της θεσης εργασιας και

μετα το τμημα στο οποιο βρισκεται αυτή.

Οι τρεις τιμες που συμπληρωνει είναι

1. συνεντευξης
2. αναφορας
3. βιογραφικου

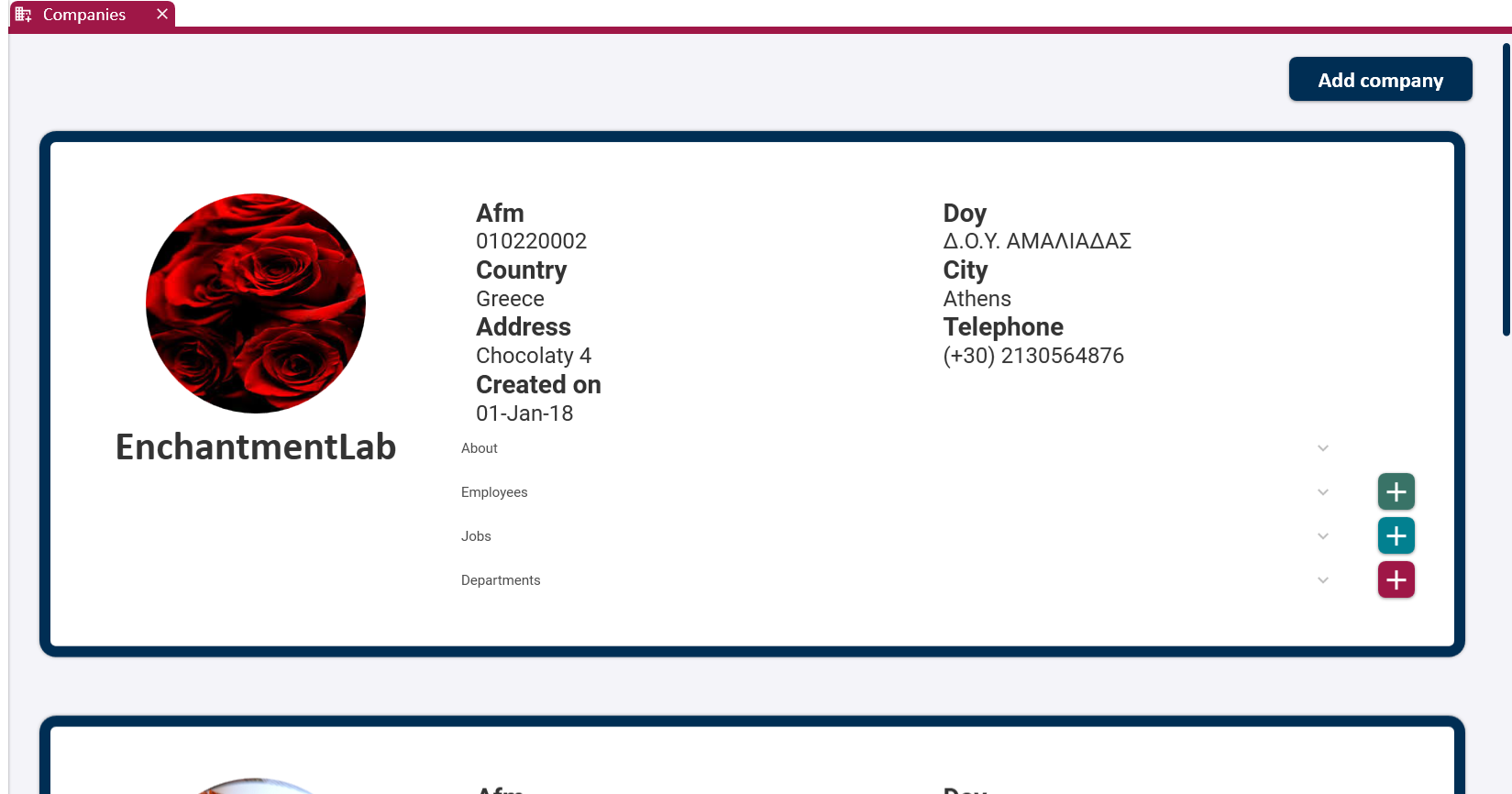
Μπορει να αποθηκευσει προσωρινα την αξιολογηση ή να την οριστικοποιησει και να σταλει στον διευθυντη του τμηματος και στον πινακα evaluations και στη στηλη IsFinalized το 0 να γινει 1.

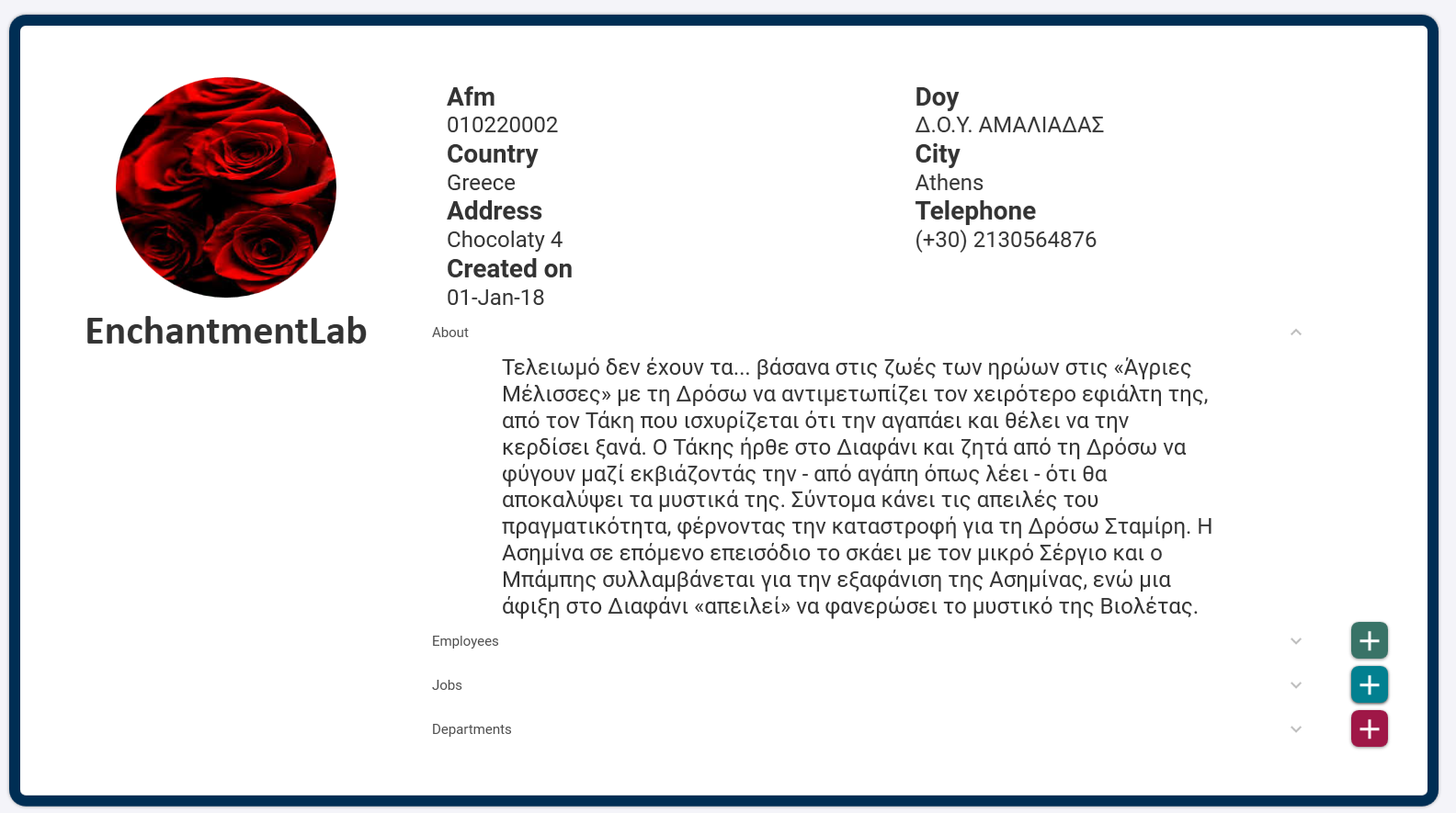


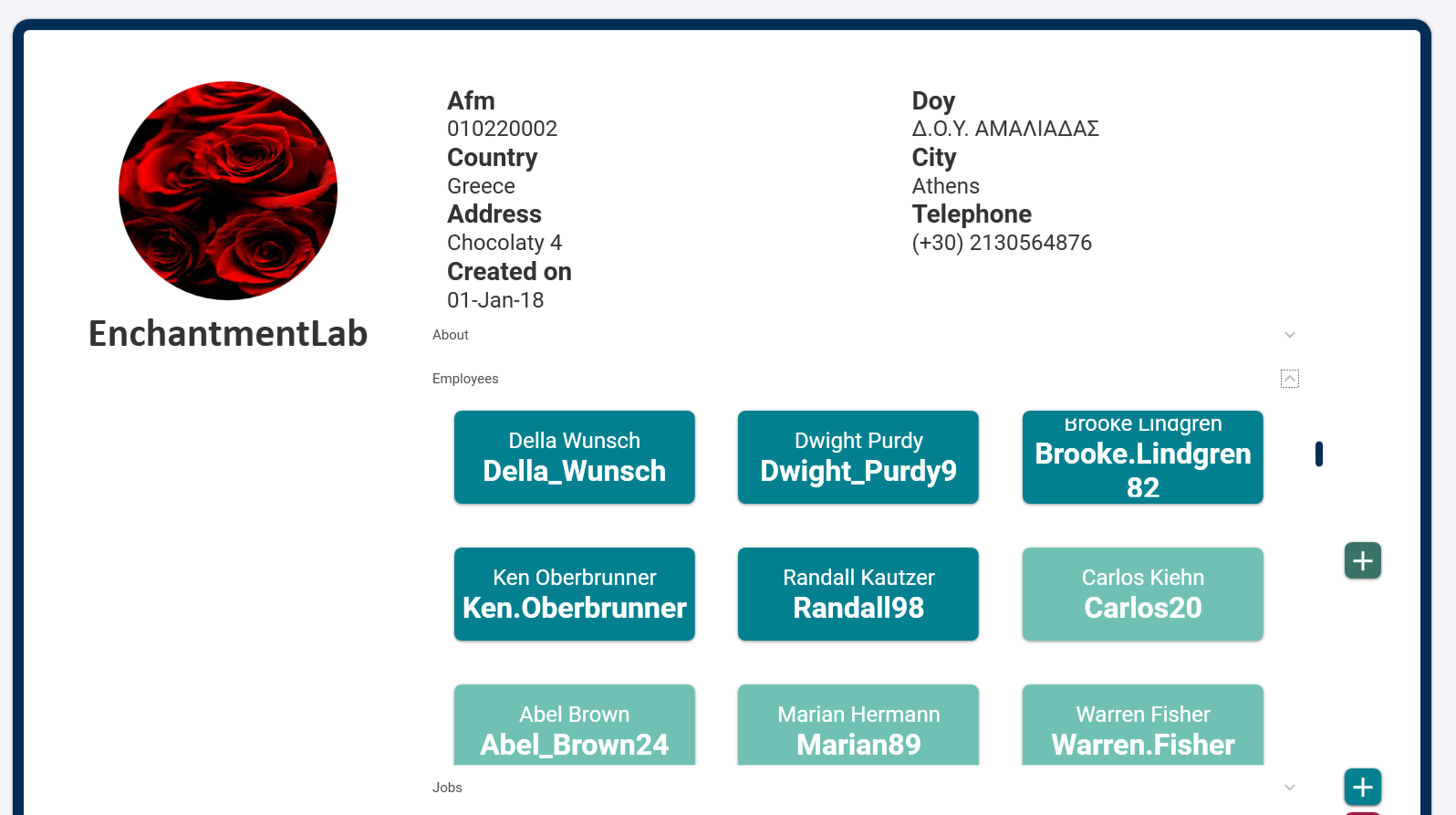
Πανω φαινεται το μηνυμη που εμφανιζεται στον αξιολογητη που τον ενημερωνει ότι οριστικοποιηθηκε με επιτυχια η αξιολογηση του και στλθηκε στον διευθυντη του τμηματος που είναι η θεση εργασιας.

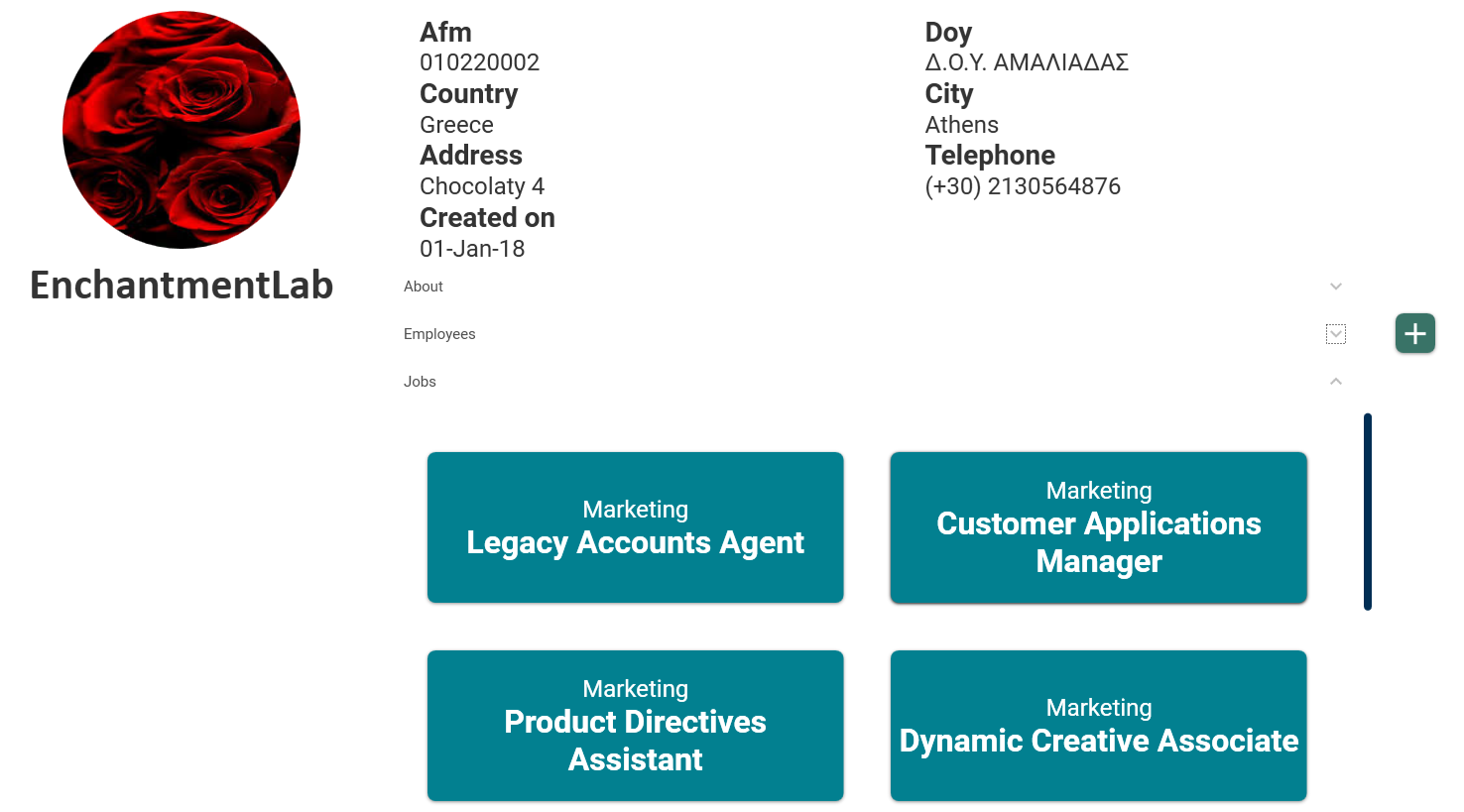
**Αν είναι Admin o χρήστης:**

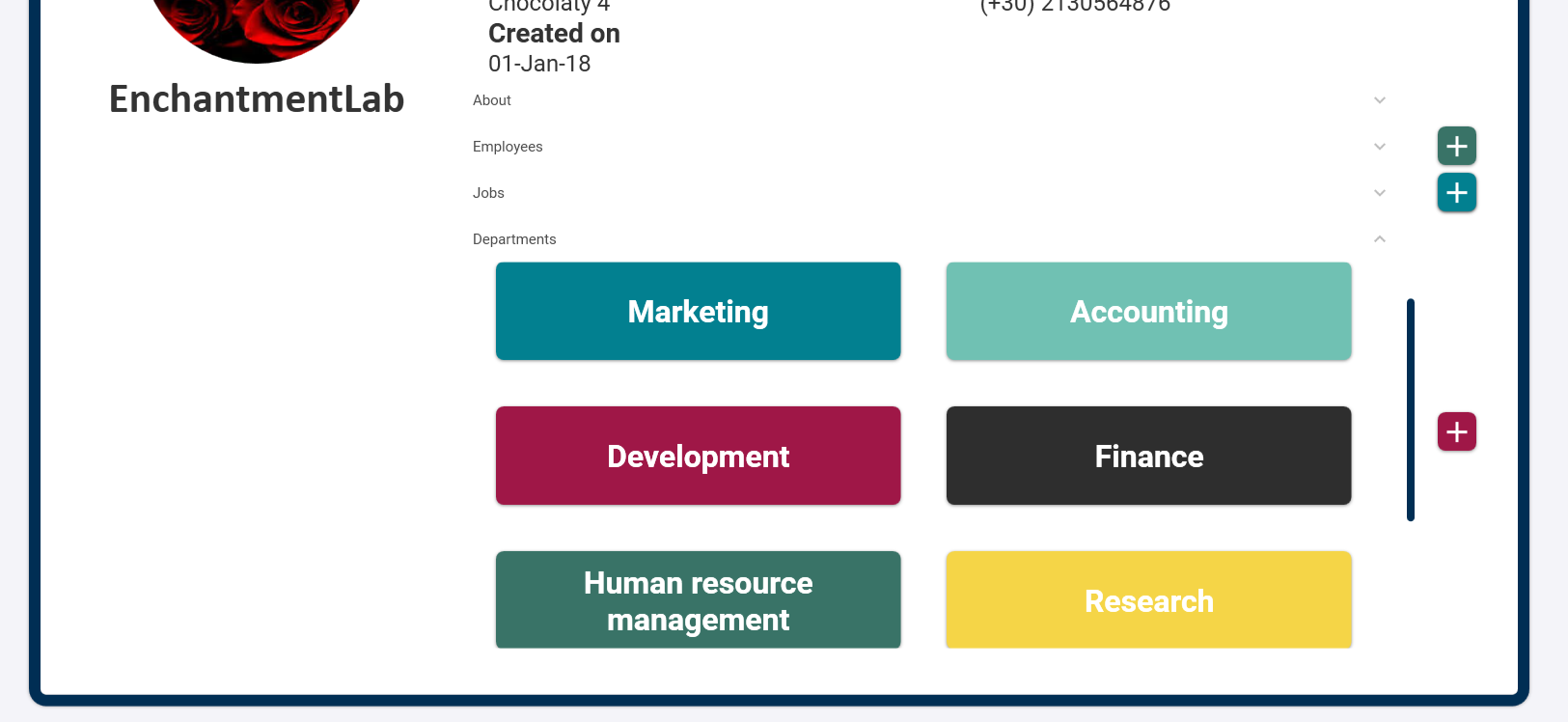
Τότε, πέρα από το προφίλ του, έχει πρόσβαση σε μία σελίδα στην οποία βλέπει και μπορεί να τροποποιήσει τα δεδομένα οποιασδήποτε εταιρίας του συστήματος .

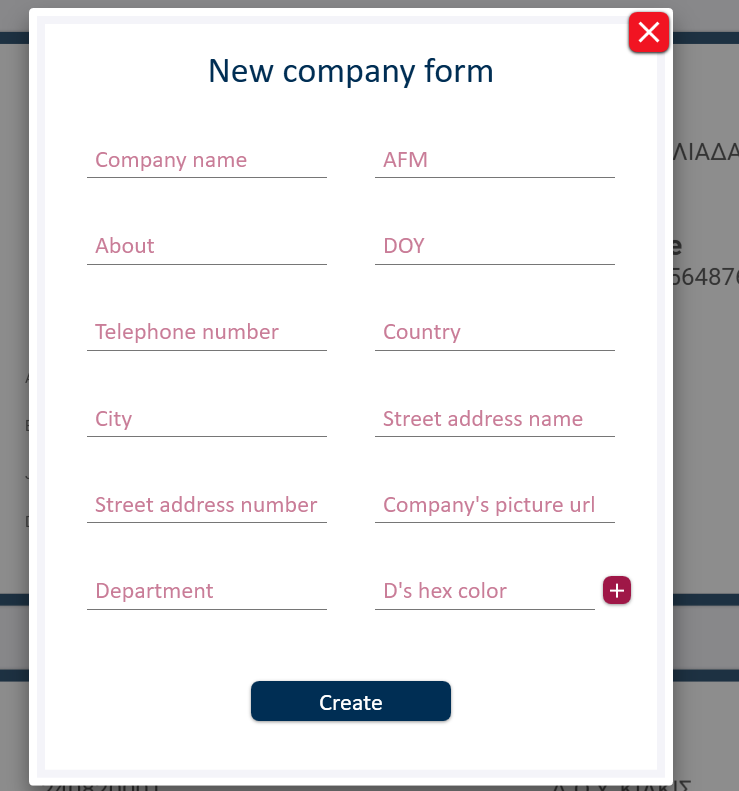




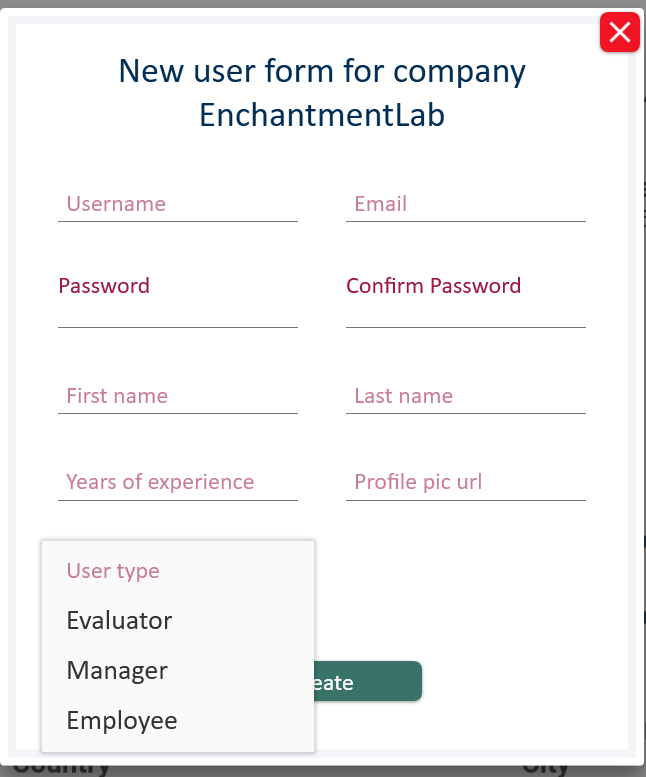
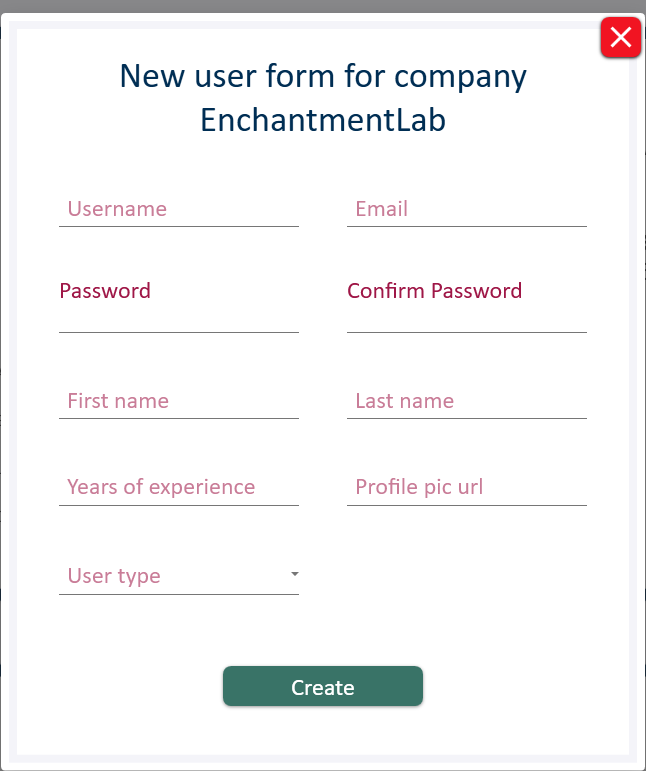


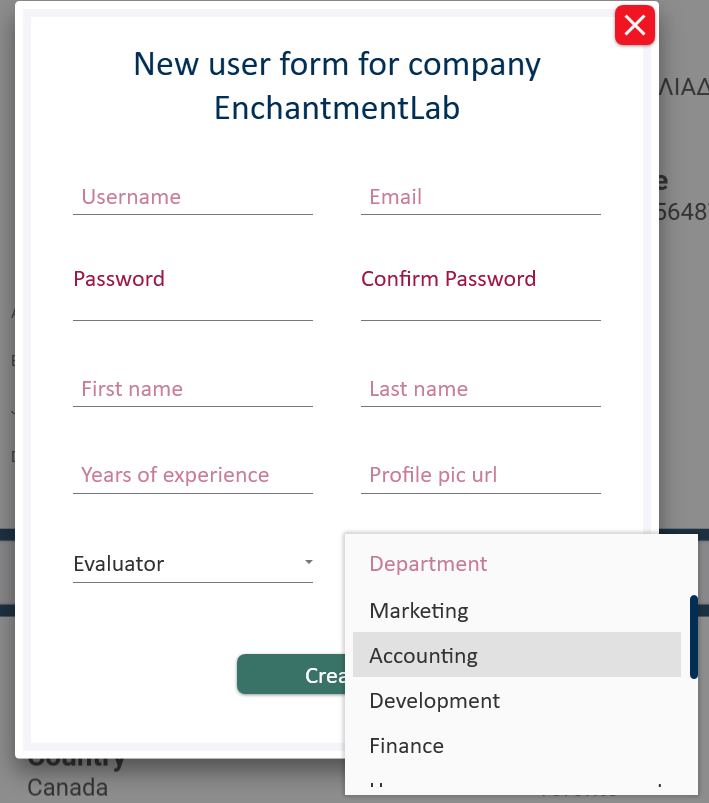




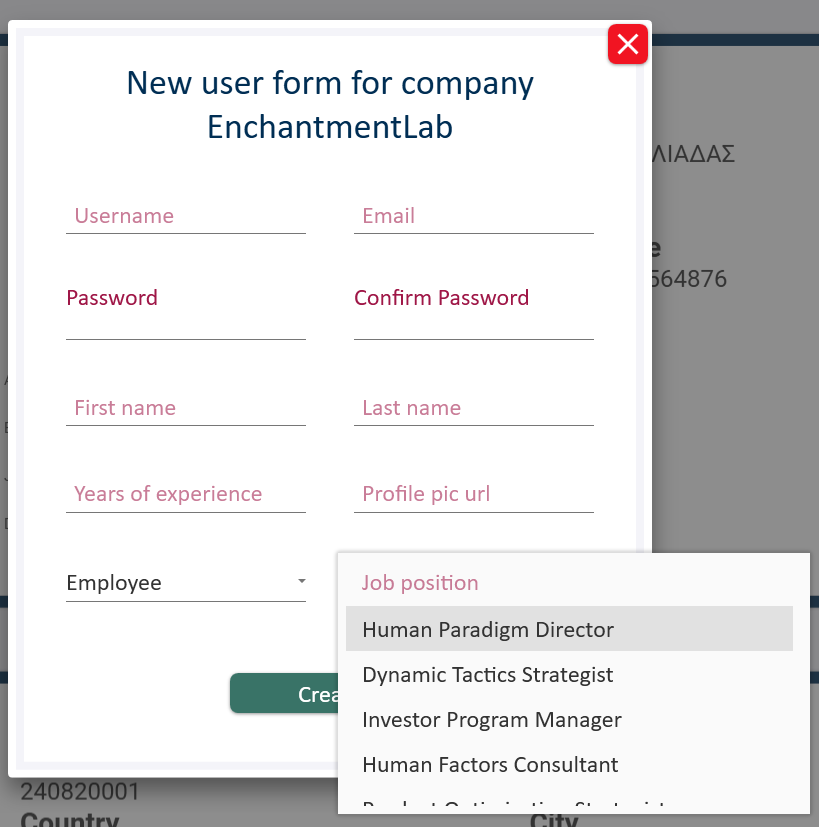
Μπορεί να προσθέσει νέα εταιρία στο σύστημα 

Ή, να προσθέσει χρήστες,θέσεις εργασίας και τμήματα σε μία εταιρία



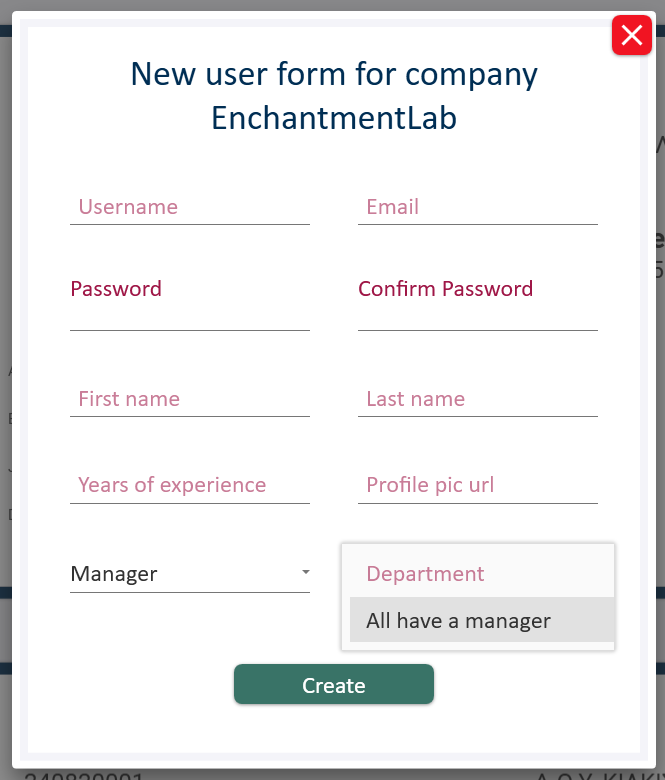
Σε περιπτωση που επιλεξει να δημιουργησει καποιον αξιολογητη

του εμφανιζει διπλα τα τμηματα που υπαρχουν ηδη σστην εταιρια **αυτόματα για να αποφυγουμε τυχον λαθη** για να τον θεσει υπευθυνο.

Σε περιπτωση που επιλεξει να δημιουργησει έναν υπαλληλο,

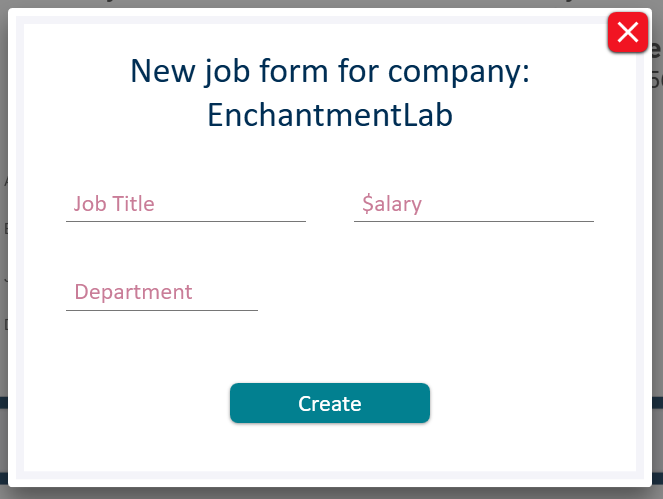
του εμφανιζει μετα **αυτοματα** τις ανοιχτες θεσεις εργασιας που υπαρχουν στην εταιρια για να του αναθεσει καποια.

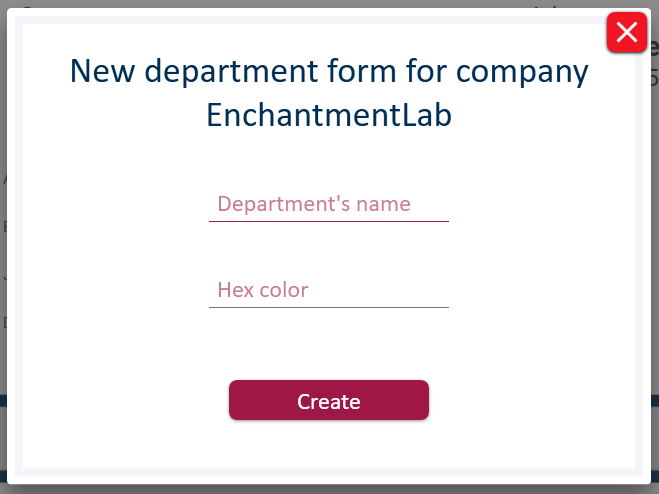
Ετσι αποφευγουμε τυχον λαθη.



Ενώ σε περιπτωση που επιλεξει να δημιουργησει καποιον μαναγερ του εμφανιζει όλα τα τμηματα που ΔΕΝ εχουν καποιον υπευθυνο διευθυντη.

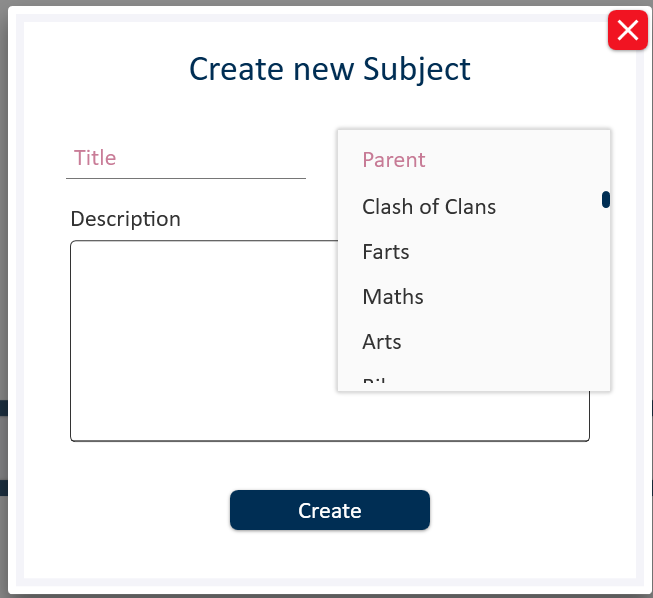
Στη συγκεκριμενη περιπτωση εχουν όλα και δεν μπορει να δημιουργηθει καποιος διευθυντης.





Έπειτα, μπορεί να δημιουργήσει νέους τομείς ενασχόλησης (Subjects). Κάθε subject έχει ένα μοναδικό αντικείμενο, ευρύτερης έννοιας στο οποίο και ανήκει ενώ μπορεί να έχει πολλά τα οποία ανήκουν σε αυτό. Ο Administrator μπορεί να δει όλους τους υπάρχοντες τομείς και να δημιουργήσει καινούριους .





Τέλος, μπορεί να βλέπει την σελίδα Log στην οποία καταγράφεται κάθε “action” που πραγματοποιείται στο σύστημα, από οποιοδήποτε χρήστη ακόμη και απ τον ίδιο τον admin ή άλλους.(Εγώ και η Κατερίνα είμαστε μόνο)



Παρακάτω είναι ολόκληρος ο κώδικας της MySQL που χρησιμοποιήσαμε για την δημιουργία της εφαρμογής μας.

Συγκεκριμένα ο κώδικας για το Α.3:

*Stored Procedures*

3.1:

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetEmployeeJobRequestsData;

DELIMITER $

/\* 3.1 STORED PROCEDURE

Gets the first and last name of an employee and shows the:

job position requests, finalized and approved by manager evaluations, evaluator's first and last name

IF evaluations not finalized ====== pending

\*/

create procedure GetEmployeeJobRequestsData(in employeeFirstName VARCHAR(120) , in employeeLastName VARCHAR(120))

begin

SELECT `e`.`IsAprovedByManager`, `e`.`IsFinalized`, `e`.`FilesGrade`, `e`.`FinalGrade`, `e`.`InterviewGrade`,

`e`.`ReportGrade`, `u0`.`Username`, `u0`.`FirstName`, `u0`.`LastName`,

`u1`.`Username`, `u1`.`FirstName`, `u1`.`LastName`, `j0`.`AnnouncementDate`, `j0`.`SubmissionDate`, `j1`.`JobTitle`,

`j1`.`Salary`,`d`.`DepartmentName`, `r`.`IsFinalized`

FROM `Evaluations` AS `e`

INNER JOIN `UsersJobFilesPairs` AS `u` ON `e`.`UsersJobFilesPairId` = `u`.`Id`

INNER JOIN `Users` AS `u0` ON `u`.`EmployeeId` = `u0`.`Id`

INNER JOIN `Users` AS `u1` ON `u`.`EvaluatorId` = `u1`.`Id`

INNER JOIN `Users` AS `u2` ON `u`.`ManagerId` = `u2`.`Id`

INNER JOIN `JobPositionRequests` AS `j` ON `e`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

LEFT JOIN `Reports` AS `r` ON `u`.`Id` = `r`.`UsersJobFilesPairId`

/\* Finds the manager with the specified id AND the evaluations that have the value 0 in the is aproved by manager column \*/

WHERE ((`u0`.`FirstName` = employeeFirstName) AND (`u0`.`LastName` = employeeLastName))

/\* They are ordered by their final grade first then the evaluation's id and so on...\*/

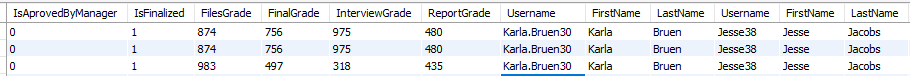
ORDER BY `u0`.`Id`, `u1`.`Id`, `u2`.`Id`, `u`.`Id`, `j`.`Id`, `j0`.`Id`, `j1`.`Id`, `d`.`Id`, `r`.`Id`,`e`.`FinalGrade`, `e`.`Id`;

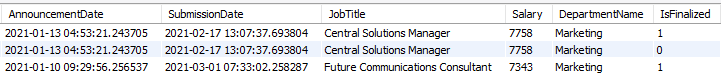
end$

DELIMITER ;

CALL GetEmployeeJobRequestsData('Cecilia', 'Nienow');

Μεσα στο workbench:





3.2:

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS ManageEvaluatorEvaluationsForOneJobPosition;

DELIMITER $

/\* 3.2 STORED PROCEDURE

Gets the jobPositionId and evaluatorId and shows the:

if final grade exists finalises the evaluation

IF evaluations not finalized ====== pending

\*/

create procedure ManageEvaluatorEvaluationsForOneJobPosition(in jobPositionId INT , in evaluatorId INT, INOUT dataList varchar(4000))

begin

DECLARE finished INTEGER DEFAULT 0;

/\*Declares a temp variable for the final grade of an evaluation\*/

DECLARE finalGrade INT;

DECLARE filesGrade INT;

DECLARE interviewGrade INT;

DECLARE reportGrade INT;

DECLARE evaluationId INT;

DECLARE isFinal tinyint(1);

DEClARE cursorEvaluation

CURSOR FOR

SELECT `e`.`IsFinalized`, `e`.`FilesGrade`, `e`.`FinalGrade`, `e`.`InterviewGrade`,

`e`.`ReportGrade`, `e`.`Id`

FROM `Evaluations` AS `e`

INNER JOIN `UsersJobFilesPairs` AS `u` ON `e`.`UsersJobFilesPairId` = `u`.`Id`

INNER JOIN `Users` AS `u0` ON `u`.`EmployeeId` = `u0`.`Id`

INNER JOIN `Users` AS `u1` ON `u`.`EvaluatorId` = `u1`.`Id`

INNER JOIN `Users` AS `u2` ON `u`.`ManagerId` = `u2`.`Id`

INNER JOIN `JobPositionRequests` AS `j` ON `e`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

LEFT JOIN `Reports` AS `r` ON `u`.`Id` = `r`.`UsersJobFilesPairId`

/\* Finds the evaluation by the evaluatiors id the job position's id and must is finalized column be 0 \*/

WHERE ((`u1`.`Id` = evaluatorId) AND (`j0`.`Id` = jobPositionId) AND (`e`.`IsFinalized` = 0))

/\* They are ordered by their final grade first then the evaluation's id and so on...\*/

ORDER BY `u0`.`Id`, `u1`.`Id`, `u2`.`Id`, `u`.`Id`, `j`.`Id`, `j0`.`Id`, `j1`.`Id`, `d`.`Id`, `r`.`Id`,`e`.`FinalGrade`, `e`.`Id`;

-- declare NOT FOUND handler

DECLARE CONTINUE HANDLER

FOR NOT FOUND SET finished = 1;

OPEN cursorEvaluation;

getEvaluation: LOOP

FETCH cursorEvaluation INTO isFinal, filesGrade, finalGrade, interviewGrade, reportGrade, evaluationId;

/\*If the required 3 grades are NOT null \*/

if(filesGrade != 0 AND interviewGrade != 0 AND reportGrade != 0 AND finalGrade != 0)

then

/\* Sets them as finalized \*/

set isFinal = 1;

end if;

/\* Updates the final grade column of the evaluations table's row with id the specified id\*/

update vaseis.evaluations set IsFinalized = isFinal Where Id = evaluationId;

IF finished = 1 THEN

LEAVE getEvaluation;

END IF;

-- build the data list

SET dataList = CONCAT(evaluationId,"-",isFinal,"-", filesGrade,"-", finalGrade,"-", interviewGrade,"-", reportGrade,";;;;;",dataList);

END LOOP getEvaluation;

CLOSE cursorEvaluation;

end$

DELIMITER ;

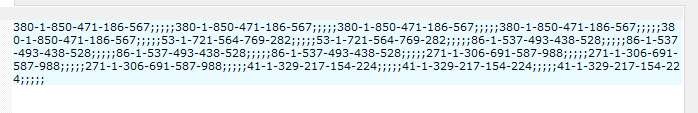
SET @list = '';

CALL ManageEvaluatorEvaluationsForOneJobPosition(155, 273, @list);

SELECT @list;

Μεσα στο workbench:

Ο 2ος αριθμος είναι το IsFinalized που πλεον είναι 1 παντου



3.3:

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS ShowEvaluationsForJobPosition;

DELIMITER $

/\* 3.3 STORED PROCEDURE

Shows all the evaluations for a job position by the job position's id

with their total number of evaluations and remaining- not finalized evaluations

\*/

create procedure ShowEvaluationsForJobPosition(in jobPositionId INT, INOUT dataList varchar(4000))

begin

DECLARE finished INTEGER DEFAULT 0;

/\*Declares a temp variable for the final grade of an evaluation\*/

DECLARE id INT;

DECLARE finalGrade INT;

DECLARE filesGrade INT;

DECLARE interviewGrade INT;

DECLARE reportGrade INT;

DECLARE evaluationId INT;

DECLARE isFinal tinyint(1);

DECLARE message VARCHAR(200);

DECLARE jobTitle VARCHAR(200);

DECLARE numOfREmainingEvaluations INT;

DECLARE numOfTotalEvaluations INT;

DEClARE cursorEvaluation

CURSOR FOR

SELECT `e`.`Id`, `j1`.`JobTitle`,`e`.`IsFinalized`, `e`.`FilesGrade`, `e`.`FinalGrade`, `e`.`InterviewGrade`,

`e`.`ReportGrade`

FROM `Evaluations` AS `e`

INNER JOIN `UsersJobFilesPairs` AS `u` ON `e`.`UsersJobFilesPairId` = `u`.`Id`

INNER JOIN `Users` AS `u0` ON `u`.`EmployeeId` = `u0`.`Id`

INNER JOIN `Users` AS `u1` ON `u`.`EvaluatorId` = `u1`.`Id`

INNER JOIN `Users` AS `u2` ON `u`.`ManagerId` = `u2`.`Id`

INNER JOIN `JobPositionRequests` AS `j` ON `e`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

LEFT JOIN `Reports` AS `r` ON `u`.`Id` = `r`.`UsersJobFilesPairId`

/\* Finds the evaluation by the evaluatiors id the job position's id and must is finalized column be 0 \*/

WHERE (`j0`.`Id` = jobPositionId)

/\* They are ordered by finalized first, then their final grade first then the evaluation's id and so on...\*/

ORDER BY `e`.`IsFinalized`, `e`.`FinalGrade`, `e`.`Id`;

-- declare NOT FOUND handler

DECLARE CONTINUE HANDLER

FOR NOT FOUND SET finished = 1;

OPEN cursorEvaluation;

SET numOfREmainingEvaluations = 0;

SET numOfTotalEvaluations = 0;

getEvaluation: LOOP

/\* For aech row gets the required data \*/

FETCH cursorEvaluation INTO id, jobTitle, isFinal, filesGrade, finalGrade, interviewGrade, reportGrade;

/\*If the evaluations is NOT finalized \*/

if isFinal = 0

then

/\* Sets the message accordingly \*/

set message = "NOT finalized evaluation";

/\* Adds 1 to the number of remaining evaluations \*/

set numOfREmainingEvaluations = numOfREmainingEvaluations + 1;

/\* Adds 1 to the number of total evaluations \*/

set numOfTotalEvaluations = numOfTotalEvaluations + 1;

else

/\* Sets the message accordingly \*/

set message = "Finalized evaluation";

/\* Adds 1 to the number of total evaluations \*/

set numOfTotalEvaluations = numOfTotalEvaluations + 1;

end if;

IF finished = 1 THEN

LEAVE getEvaluation;

END IF;

-- build the data list

SET dataList = CONCAT\_WS("\n",id,"-",jobTitle,"-//", message, "//","Total:",numOfTotalEvaluations,"Remain:",numOfREmainingEvaluations,"//",isFinal,"-", filesGrade,"-", finalGrade,"-", interviewGrade,"-", reportGrade,";;;;;",dataList);

END LOOP getEvaluation;

CLOSE cursorEvaluation;

end$

DELIMITER ;

SET @list = '';

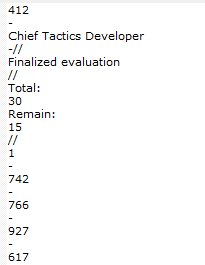
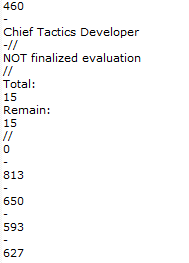
CALL ShowEvaluationsForJobPosition(102, @list);

SELECT @list;

Μεσα στο workbench:

Τελικα συνολικα 30 αξιολογησεις (1)

Τελευταια αξιολογηση που δεν είναι ολοκληρωμενη και συνολυκα μη ολοκληρωμενες 15(2)



*Triggers*

Companies:

/\* Drops the trigger if it exists \*/

DROP TRIGGER IF EXISTS DenyCompanyDataChange;

DELIMITER $

/\* Creates a trigger that fires when an update on the companies table of the database occurs.

If someone tries to alter the AFM DOY and Name of a company it ignores the change and keeps the old values

\*/

CREATE TRIGGER DenyCompanyDataChange

BEFORE UPDATE ON companies

FOR EACH ROW

BEGIN

/\* Do NOT apply the changes keep the old values \*/

SET NEW.Name = OLD.Name;

SET NEW.AFM = OLD.AFM;

SET NEW.DOY = OLD.DOY;

END$

Users:

/\* Drops the trigger if it exists \*/

DROP TRIGGER IF EXISTS DenyUserDataChange;

DELIMITER $

/\* Createsa trigger that fires when an updata on the users table of the database occurs.

If the connected user is NOT of type Administrator (0) then set the new values as the old ones

Keeps old correct values

\*/

CREATE TRIGGER DenyUserDataChange

BEFORE UPDATE ON users

FOR EACH ROW

BEGIN

/\* Gets the connected user \*/

SET @connectedUser = USER();

/\* If the users type is NOT 0 == Admin... \*/

IF( connectedUser.Type != 0)

THEN

/\* Do NOT apply the changes keep the od values \*/

SET NEW.Username = OLD.Username;

SET NEW.Type = OLD.Type;

SET NEW.Id = OLD.Id;

SET NEW.RegistrationDate = OLD.RegistrationDate;

SET NEW.YearsOfExperience = OLD.YearsOfExperience;

END IF;

END$

Log:

/\* Drops the trigger if it exists \*/

DROP TRIGGER IF EXISTS AddCompanyDataToLog;

DELIMITER $

/\* Creates a trigger that fires when an insert on the companies table of the database occurs.\*/

CREATE TRIGGER AddCompanyDataToLog

BEFORE INSERT ON companies

FOR EACH ROW

BEGIN

/\* Gets the connected user \*/

SET @connectedUser = USER();

INSERT INTO `vaseis`.`logs`

(`Username`, `Action`, `Details`, `When`)

VALUES

(connectedUser.Username, "New comapny created", "by admin", NOW());

END$

/\* Drops the trigger if it exists \*/

DROP TRIGGER IF EXISTS UpdateUserDataToLog;

DELIMITER $

/\* Creates a trigger that fires when an insert on the companies table of the database occurs.\*/

CREATE TRIGGER UpdateCompanyDataToLog

BEFORE INSERT ON companies

FOR EACH ROW

BEGIN

/\* Gets the connected user \*/

SET @connectedUser = USER();

INSERT INTO `vaseis`.`logs`

(`Username`, `Action`, `Details`, `When`)

VALUES

(connectedUser.Username, "A company's data was altered by", connectedUser.Username, NOW());

END$

/\* Drops the trigger if it exists \*/

DROP TRIGGER IF EXISTS AddUserDataToLog;

DELIMITER $

/\* Creates a trigger that fires when an insert on the users table of the database occurs.\*/

CREATE TRIGGER AddUserDataToLog

BEFORE INSERT ON users

FOR EACH ROW

BEGIN

/\* Gets the connected user \*/

SET @connectedUser = USER();

INSERT INTO `vaseis`.`logs`

(`Username`, `Action`, `Details`, `When`)

VALUES

(connectedUser.Username, "New user created", "by admin", NOW());

END$

/\* Drops the trigger if it exists \*/

DROP TRIGGER IF EXISTS UpdateUserDataToLog;

DELIMITER $

/\* Creates a trigger that fires when an insert on the users table of the database occurs.\*/

CREATE TRIGGER UpdateUserDataToLog

BEFORE INSERT ON users

FOR EACH ROW

BEGIN

/\* Gets the connected user \*/

SET @connectedUser = USER();

INSERT INTO `vaseis`.`logs`

(`Username`, `Action`, `Details`, `When`)

VALUES

(connectedUser.Username, "A user's data was altered by", connectedUser.Username, NOW());

END$

/\* Drops the trigger if it exists \*/

DROP TRIGGER IF EXISTS AddJobDataToLog;

DELIMITER $

/\* Creates a trigger that fires when an insert on the jobs table of the database occurs.\*/

CREATE TRIGGER AddJobDataToLog

BEFORE INSERT ON jobs

FOR EACH ROW

BEGIN

/\* Gets the connected user \*/

SET @connectedUser = USER();

INSERT INTO `vaseis`.`logs`

(`Username`, `Action`, `Details`, `When`)

VALUES

(connectedUser.Username, "New job created", "by admin", NOW());

END$

/\* Drops the trigger if it exists \*/

DROP TRIGGER IF EXISTS AddDepartmentDataToLog;

DELIMITER $

/\* Creates a trigger that fires when an insert on the departments table of the database occurs.\*/

CREATE TRIGGER AddDepartmentDataToLog

BEFORE INSERT ON departments

FOR EACH ROW

BEGIN

/\* Gets the connected user \*/

SET @connectedUser = USER();

INSERT INTO `vaseis`.`logs`

(`Username`, `Action`, `Details`, `When`)

VALUES

(connectedUser.Username, "New department created", "by admin", NOW());

END$

Υπολοιπος κωδικας mysql:

use vaseis;

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetCompanyEmployeesss;

DELIMITER $

/\* Creates a stored procedure that gets all that employees that are in a company\*/

create procedure GetCompanyEmployeesss(in companyId INT)

begin

SELECT `u`.`Bio`,`u`.`Email`, `u`.`FirstName`, `u`.`FullName`,

`u`.`LastName`, `u`.`Password`, `u`.`ProfilePicture`, `u`.`RegistrationDate`, `u`.`Type`,

`u`.`Username`, `u`.`YearsOfExperience`, `d`.`DepartmentName`,

`j0`.`JobTitle`,

`a`.`AcquiredDate`, `a`.`Name`,`c0`.`Description`, `c0`.`Title`,

`p`.`Description`, `p`.`MadeForWho`, `p`.`Title`, `p`.`Url`, `r`.`Description`, `r`.`Referee`,

`a0`.`Grade`, `a0`.`YearEarned`

FROM `Users` AS `u`

LEFT JOIN `Departments` AS `d` ON `u`.`DepartmentId` = `d`.`Id`

LEFT JOIN `Companies` AS `c` ON `d`.`CompanyId` = `c`.`Id`

LEFT JOIN `JobPositions` AS `j` ON `u`.`JobPositionId` = `j`.`Id`

LEFT JOIN `Jobs` AS `j0` ON `j`.`JobId` = `j0`.`Id`

LEFT JOIN `Awards` AS `a` ON `u`.`Id` = `a`.`UserId`

LEFT JOIN `Certificates` AS `c0` ON `u`.`Id` = `c0`.`UserId`

LEFT JOIN `Projects` AS `p` ON `u`.`Id` = `p`.`UserId`

LEFT JOIN `RecomendationPapers` AS `r` ON `u`.`Id` = `r`.`UserId`

LEFT JOIN `AcquiredDegrees` AS `a0` ON `u`.`Id` = `a0`.`UserId`

/\* Finds the company with the specified id and the users that have the value 1 (UserType = Employee) in the Type column \*/

WHERE (`d`.`CompanyId` = companyId) AND (`u`.`Type` = 1)

ORDER BY `u`.`Id`, `a`.`Id`, `c0`.`Id`, `p`.`Id`, `r`.`Id`, `a0`.`Id`;

end$

DELIMITER ;

/\* Calls the Procedure and requests all the information about every uesr that is of type employee (TYPE = 1)\*/

CALL GetCompanyEmployeesss(1);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetManagerUnfinishedReports;

DELIMITER $

/\* Creates a stored procedure that gets all that reports that are not finilised yet\*/

create procedure GetManagerUnfinishedReports(in managerId INT)

begin

SELECT `r`.`Id`, `r`.`IsFinalized`, `r`.`IsWritten`, `r`.`JobPositionRequestId`, `r`.`ReportText`, `r`.`UserDataModelId`, `r`.`UsersJobFilesPairId`, `u`.`Id`, `u`.`EmployeeId`, `u`.`EvaluatorId`, `u`.`ManagerId`, `u0`.`Id`, `u0`.`Bio`, `u0`.`CompanyDataModelId`, `u0`.`DepartmentId`, `u0`.`Email`, `u0`.`FirstName`, `u0`.`FullName`, `u0`.`JobPositionId`, `u0`.`LastName`, `u0`.`Password`, `u0`.`ProfilePicture`, `u0`.`RegistrationDate`, `u0`.`Type`, `u0`.`Username`, `u0`.`YearsOfExperience`, `u1`.`Id`, `u1`.`Bio`, `u1`.`CompanyDataModelId`, `u1`.`DepartmentId`, `u1`.`Email`, `u1`.`FirstName`, `u1`.`FullName`, `u1`.`JobPositionId`, `u1`.`LastName`, `u1`.`Password`, `u1`.`ProfilePicture`, `u1`.`RegistrationDate`, `u1`.`Type`, `u1`.`Username`, `u1`.`YearsOfExperience`, `u2`.`Id`, `u2`.`Bio`, `u2`.`CompanyDataModelId`, `u2`.`DepartmentId`, `u2`.`Email`, `u2`.`FirstName`, `u2`.`FullName`, `u2`.`JobPositionId`, `u2`.`LastName`, `u2`.`Password`, `u2`.`ProfilePicture`, `u2`.`RegistrationDate`, `u2`.`Type`, `u2`.`Username`, `u2`.`YearsOfExperience`, `j`.`Id`, `j`.`JobPositionId`, `j`.`RequestsReason`, `j`.`UserDataModelId`, `j`.`UsersJobFilesPairId`, `j0`.`Id`, `j0`.`AnnouncementDate`, `j0`.`CreatorId`, `j0`.`JobId`, `j0`.`StartDate`, `j0`.`SubmissionDate`, `j1`.`Id`, `j1`.`CompanyId`, `j1`.`DepartmentId`, `j1`.`JobTitle`, `j1`.`Salary`, `d`.`Id`, `d`.`Color`, `d`.`CompanyId`, `d`.`DepartmentName`, `j2`.`Id`, `j2`.`JobPositionId`, `j2`.`RequestsReason`, `j2`.`UserDataModelId`, `j2`.`UsersJobFilesPairId`

FROM `Reports` AS `r`

INNER JOIN `UsersJobFilesPairs` AS `u` ON `r`.`UsersJobFilesPairId` = `u`.`Id`

INNER JOIN `Users` AS `u0` ON `u`.`ManagerId` = `u0`.`Id`

INNER JOIN `Users` AS `u1` ON `u`.`EvaluatorId` = `u1`.`Id`

INNER JOIN `Users` AS `u2` ON `u`.`EmployeeId` = `u2`.`Id`

INNER JOIN `JobPositionRequests` AS `j` ON `r`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

LEFT JOIN `JobPositionRequests` AS `j2` ON `u`.`Id` = `j2`.`UsersJobFilesPairId`

/\* Finds the manager with the specified id AND the reports that have 0 in the is finalised column \*/

WHERE (`u`.`ManagerId` = managerId) AND (`r`.`IsFinalized` = FALSE)

ORDER BY `r`.`Id`, `u`.`Id`, `u0`.`Id`, `u1`.`Id`, `u2`.`Id`, `j`.`Id`, `j0`.`Id`, `j1`.`Id`, `d`.`Id`, `j2`.`Id`;

end$

DELIMITER ;

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetEvluationsNotApprovedByManager;

DELIMITER $

/\* Creates a stored procedure that gets all that evaluations that are finalised but NOT approved by a manager yet\*/

create procedure GetEvluationsNotApprovedByManager(in managerId INT)

begin

SELECT `e`.`Comments`, `e`.`FilesGrade`, `e`.`FinalGrade`, `e`.`InterviewGrade`,

`e`.`ReportGrade`, `u0`.`Username`,

`u1`.`Username`, `j0`.`AnnouncementDate`, `j0`.`SubmissionDate`, `j1`.`JobTitle`,

`j1`.`Salary`,`d`.`DepartmentName`,

`r`.`ReportText`

FROM `Evaluations` AS `e`

INNER JOIN `UsersJobFilesPairs` AS `u` ON `e`.`UsersJobFilesPairId` = `u`.`Id`

INNER JOIN `Users` AS `u0` ON `u`.`EmployeeId` = `u0`.`Id`

INNER JOIN `Users` AS `u1` ON `u`.`EvaluatorId` = `u1`.`Id`

INNER JOIN `Users` AS `u2` ON `u`.`ManagerId` = `u2`.`Id`

INNER JOIN `JobPositionRequests` AS `j` ON `e`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

LEFT JOIN `Reports` AS `r` ON `u`.`Id` = `r`.`UsersJobFilesPairId`

/\* Finds the manager with the specified id AND the evaluations that have the value 0 in the is aproved by manager column \*/

WHERE ((`u`.`ManagerId` = managerId) AND (`e`.`IsAprovedByManager` = FALSE)) AND (`e`.`IsFinalized` = TRUE)

/\* They are ordered by their final grade first then the evaluation's id and so on...\*/

ORDER BY `e`.`FinalGrade`, `e`.`Id`, `u`.`Id`, `u0`.`Id`, `u1`.`Id`, `u2`.`Id`, `j`.`Id`, `j0`.`Id`, `j1`.`Id`, `d`.`Id`, `r`.`Id`;

end$

DELIMITER ;

CALL GetEvluationsNotApprovedByManager(5);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetEvluationsApprovedByManager;

DELIMITER $

/\* Creates a stored procedure that gets all that evaluations that are finalised but NOT approved by a manager yet\*/

create procedure GetEvluationsApprovedByManager(in managerId INT)

begin

SELECT `e`.`Comments`, `e`.`FilesGrade`, `e`.`FinalGrade`, `e`.`InterviewGrade`,

`e`.`ReportGrade`, `u0`.`Username`,

`u1`.`Username`, `j0`.`AnnouncementDate`, `j0`.`SubmissionDate`, `j1`.`JobTitle`,

`j1`.`Salary`,`d`.`DepartmentName`,

`r`.`ReportText`

FROM `Evaluations` AS `e`

INNER JOIN `UsersJobFilesPairs` AS `u` ON `e`.`UsersJobFilesPairId` = `u`.`Id`

INNER JOIN `Users` AS `u0` ON `u`.`EmployeeId` = `u0`.`Id`

INNER JOIN `Users` AS `u1` ON `u`.`EvaluatorId` = `u1`.`Id`

INNER JOIN `Users` AS `u2` ON `u`.`ManagerId` = `u2`.`Id`

INNER JOIN `JobPositionRequests` AS `j` ON `e`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

LEFT JOIN `Reports` AS `r` ON `u`.`Id` = `r`.`UsersJobFilesPairId`

/\* Finds the manager with the specified id AND the evaluations that have the value 0 in the is aproved by manager column \*/

WHERE ((`u`.`ManagerId` = managerId) AND (`e`.`IsAprovedByManager` = TRUE)) AND (`e`.`IsFinalized` = TRUE)

/\* They are ordered by their final grade first then the evaluation's id and so on...\*/

ORDER BY `e`.`FinalGrade`, `e`.`Id`, `u`.`Id`, `u0`.`Id`, `u1`.`Id`, `u2`.`Id`, `j`.`Id`, `j0`.`Id`, `j1`.`Id`, `d`.`Id`, `r`.`Id`;

end$

DELIMITER ;

CALL GetEvluationsApprovedByManager(5);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetJobPositionData;

DELIMITER $

/\* Creates a stored procedure that gets all the evaluations that are finalised but NOT approved by a manager yet\*/

/\*And gets the number of evaluations that are not finalized yet consernig the specified job position\*/

create procedure GetJobPositionData(in jobPositionId INT)

begin

SELECT `e`.`Id`, `e`.`Comments`, `e`.`FilesGrade`, `e`.`FinalGrade`, `e`.`InterviewGrade`, `e`.`IsAprovedByManager`,

`e`.`IsFinalized`, `e`.`JobPositionRequestId`, `e`.`ReportGrade`, `u0`.`Username`,

`u1`.`Username`, `j0`.`AnnouncementDate`, `j0`.`SubmissionDate`,`j1`.`JobTitle`,

`j1`.`Salary`,`d`.`DepartmentName`

FROM `Evaluations` AS `e`

INNER JOIN `UsersJobFilesPairs` AS `u` ON `e`.`UsersJobFilesPairId` = `u`.`Id`

INNER JOIN `Users` AS `u0` ON `u`.`EmployeeId` = `u0`.`Id`

INNER JOIN `Users` AS `u1` ON `u`.`EvaluatorId` = `u1`.`Id`

INNER JOIN `Users` AS `u2` ON `u`.`ManagerId` = `u2`.`Id`

INNER JOIN `JobPositionRequests` AS `j` ON `e`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

LEFT JOIN `Reports` AS `r` ON `u`.`Id` = `r`.`UsersJobFilesPairId`

/\* Findsthe job position with the specified id and its evaluations that are not completed \*/

WHERE ((`j0`.`Id` = jobPositionId) AND (`e`.`IsAprovedByManager` = FALSE)) AND (`e`.`IsFinalized` = TRUE)

ORDER BY `j0`.`Id`, `e`.`Id`, `u`.`Id`, `u0`.`Id`, `u1`.`Id`, `u2`.`Id`, `j`.`Id`, `j1`.`Id`, `d`.`Id`, `r`.`Id`;

SELECT count(\*) AS `NumOfRemainingEvaluation`

FROM (select `Comments` FROM `Evaluations` AS `e`

INNER JOIN `JobPositionRequests` AS `j` ON `e`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

WHERE ((`e`.`IsAprovedByManager` = FALSE) AND (`e`.`IsFinalized` = FALSE) AND (`j0`.`Id` = jobPositionId))) AS `Remaining evaluations`;

end$

DELIMITER ;

CALL GetJobPositionData(112);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetConnectedUserProfileData;

DELIMITER $

/\* Creates a stored procedure that gets all the required data for a user's profile page

by the user's usernae and password \*/

create procedure GetConnectedUserProfileData(in username CHAR, in password CHAR)

begin

SELECT `t`.`Bio`,`t`.`Email`, `t`.`FirstName`, `t`.`FullName`, `t`.`JobPositionId`, `t`.`LastName`, `t`.`ProfilePicture`,

`t`.`RegistrationDate`, `t`.`Type`, `t`.`Username`, `t`.`YearsOfExperience`, `t`.`JobTitle`, `t`.`DepartmentName`, `a`.`Grade`, `a`.`YearEarned`,

`a0`.`AcquiredDate`, `a0`.`Name`, `c0`.`Description`, `c0`.`Title`,`l`.`Name`, `r`.`Description`, `r`.`Referee`

FROM (

SELECT `u`.`Id`, `u`.`Bio`, `u`.`CompanyId`, `u`.`DepartmentId`, `u`.`Email`, `u`.`FirstName`, `u`.`FullName`, `u`.`JobPositionId`, `u`.`LastName`, `u`.`Password`, `u`.`ProfilePicture`, `u`.`RegistrationDate`, `u`.`Type`, `u`.`Username`, `u`.`YearsOfExperience`, `j`.`Id` AS `Id0`, `j`.`AnnouncementDate`, `j`.`CreatorId`, `j`.`JobId`, `j`.`StartDate`, `j`.`SubmissionDate`, `j0`.`Id` AS `Id1`, `j0`.`CompanyId` AS `CompanyId0`, `j0`.`DepartmentId` AS `DepartmentId0`, `j0`.`JobTitle`, `j0`.`Salary`, `d`.`Id` AS `Id2`, `d`.`Color`, `d`.`CompanyId` AS `CompanyId1`, `d`.`DepartmentName`, `c`.`Id` AS `Id3`, `c`.`AFM`, `c`.`About`, `c`.`City`, `c`.`CompanyPicture`, `c`.`Country`, `c`.`DOY`, `c`.`DateCreated`, `c`.`Name`, `c`.`StreetName`, `c`.`StreetNumber`, `c`.`TelephoneNumber`

FROM `Users` AS `u`

LEFT JOIN `JobPositions` AS `j` ON `u`.`JobPositionId` = `j`.`Id`

LEFT JOIN `Jobs` AS `j0` ON `j`.`JobId` = `j0`.`Id`

LEFT JOIN `Departments` AS `d` ON `u`.`DepartmentId` = `d`.`Id`

LEFT JOIN `Companies` AS `c` ON `d`.`CompanyId` = `c`.`Id`

WHERE (`u`.`Username` = username) AND (`u`.`Password` = password)

LIMIT 1

) AS `t`

LEFT JOIN `AcquiredDegrees` AS `a` ON `t`.`Id` = `a`.`UserId`

LEFT JOIN `Awards` AS `a0` ON `t`.`Id` = `a0`.`UserId`

LEFT JOIN `Certificates` AS `c0` ON `t`.`Id` = `c0`.`UserId`

LEFT JOIN `Languages` AS `l` ON `t`.`Id` = `l`.`UserId`

LEFT JOIN `RecomendationPapers` AS `r` ON `t`.`Id` = `r`.`UserId`

ORDER BY `t`.`Id`, `a`.`Id`, `a0`.`Id`, `c0`.`Id`, `l`.`Id`, `r`.`Id`;

end$

DELIMITER ;

CALL GetConnectedUserProfileData("0xCAFFEEBABA", "neraki23");

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetConnectedUserJobRequestsData;

DELIMITER $

/\* Creates a stored procedure that gets all the required data for a user's job requests' page

by the user's id \*/

create procedure GetConnectedUserJobRequestsData(in userId INT)

begin

SELECT `j`.`RequestsReason`, `u`.`Id`, `u`.`EmployeeId`, `u`.`EvaluatorId`, `u0`.`Username`,

`j0`.`AnnouncementDate`, `j0`.`SubmissionDate`,`j1`.`JobTitle`, `j1`.`Salary`,

`d`.`DepartmentName`, `t`.`Title`

FROM `JobPositionRequests` AS `j`

LEFT JOIN `UsersJobFilesPairs` AS `u` ON `j`.`UsersJobFilesPairId` = `u`.`Id`

LEFT JOIN `Users` AS `u0` ON `u`.`EmployeeId` = `u0`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

LEFT JOIN `JobPositionRequests` AS `j2` ON `j0`.`Id` = `j2`.`JobPositionId`

LEFT JOIN

(

SELECT `j3`.`JobPositionId`, `j3`.`SubjectId`, `s`.`Id`, `s`.`Description`, `s`.`SubjectId` AS `SubjectId0`, `s`.`Title`, `j4`.`Id` AS `Id0`, `j4`.`AnnouncementDate`, `j4`.`CreatorId`, `j4`.`JobId`, `j4`.`StartDate`, `j4`.`SubmissionDate`

FROM `JobPositionSubjects` AS `j3`

INNER JOIN `Subjects` AS `s` ON `j3`.`SubjectId` = `s`.`Id`

INNER JOIN `JobPositions` AS `j4` ON `j3`.`JobPositionId` = `j4`.`Id`

) AS `t` ON `j0`.`Id` = `t`.`JobPositionId`

WHERE `u`.`EmployeeId` = userId

ORDER BY `j`.`Id`, `j0`.`Id`, `j1`.`Id`, `d`.`Id`, `j2`.`Id`, `t`.`JobPositionId`, `t`.`SubjectId`, `t`.`Id`, `t`.`Id0`;

end$

DELIMITER ;

CALL GetConnectedUserJobRequestsData(1);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetConnectedEmployeeEvaluationsData;

DELIMITER $

/\* Creates a stored procedure that gets all the required data for a user's job requests' page

by the user's id \*/

create procedure GetConnectedEmployeeEvaluationsData(in userId INT)

begin

SELECT `e`.`FilesGrade`, `e`.`FinalGrade`, `e`.`InterviewGrade`, `e`.`Passed`, `e`.`ReportGrade`,

/\* The employee's username and next is the evaluator's username, then the job's title, the salary and the department's name\*/

`u0`.`Username`, `u1`.`Username`,`j1`.`JobTitle`, `j1`.`Salary`, `d`.`DepartmentName`

FROM `Evaluations` AS `e`

INNER JOIN `UsersJobFilesPairs` AS `u` ON `e`.`UsersJobFilesPairId` = `u`.`Id`

INNER JOIN `Users` AS `u0` ON `u`.`EmployeeId` = `u0`.`Id`

INNER JOIN `Users` AS `u1` ON `u`.`EvaluatorId` = `u1`.`Id`

INNER JOIN `JobPositionRequests` AS `j` ON `e`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

/\* The employee id must be equal to the specified id and the bool in the evaluations table for is aproved by manager must be true \*/

WHERE (`u`.`EmployeeId` = userId) AND (`e`.`IsAprovedByManager` = TRUE);

end$

DELIMITER ;

CALL GetConnectedEmployeeEvaluationsData(1);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetCompanyOpenJobPositions;

DELIMITER $

/\* Creates a stored procedure that gets all the required data for a user's job requests' page

by the user's id \*/

create procedure GetCompanyOpenJobPositions(in companyId INT)

begin

SELECT `j`.`AnnouncementDate`,`j`.`SubmissionDate`,`j0`.`JobTitle`, `j0`.`Salary`,`d`.`DepartmentName`,

/\*The subject's title \*/

`t`.`Title`

FROM `JobPositions` AS `j`

INNER JOIN `Jobs` AS `j0` ON `j`.`JobId` = `j0`.`Id`

INNER JOIN `Departments` AS `d` ON `j0`.`DepartmentId` = `d`.`Id`

INNER JOIN `Companies` AS `c` ON `d`.`CompanyId` = `c`.`Id`

LEFT JOIN `JobPositionRequests` AS `j1` ON `j`.`Id` = `j1`.`JobPositionId`

LEFT JOIN (

SELECT `j2`.`JobPositionId`, `j2`.`SubjectId`, `s`.`Id`, `s`.`Description`, `s`.`SubjectId` AS `SubjectId0`, `s`.`Title`, `j3`.`Id` AS `Id0`, `j3`.`AnnouncementDate`, `j3`.`CreatorId`, `j3`.`JobId`, `j3`.`StartDate`, `j3`.`SubmissionDate`

FROM `JobPositionSubjects` AS `j2`

INNER JOIN `Subjects` AS `s` ON `j2`.`SubjectId` = `s`.`Id`

INNER JOIN `JobPositions` AS `j3` ON `j2`.`JobPositionId` = `j3`.`Id`

) AS `t` ON `j`.`Id` = `t`.`JobPositionId`

WHERE (`c`.`Id` = companyId) AND (`j`.`SubmissionDate` > CURRENT\_TIMESTAMP())

ORDER BY `j`.`Id`, `j0`.`Id`, `d`.`Id`, `c`.`Id`, `j1`.`Id`, `t`.`JobPositionId`, `t`.`SubjectId`, `t`.`Id`, `t`.`Id0`;

end$

DELIMITER ;

CALL GetCompanyOpenJobPositions(1);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetCompaniesWithTheirData;

DELIMITER $

/\* Creates a stored procedure that gets all the required data for the admin's companies' page

Basically for every comapny its departments, jobs and users \*/

create procedure GetCompaniesWithTheirData()

begin

SELECT `c`.`AFM`, `c`.`About`, `c`.`City`, `c`.`CompanyPicture`, `c`.`Country`, `c`.`DOY`, `c`.`DateCreated`, `c`.`Name`,

`c`.`StreetName`, `c`.`StreetNumber`, `c`.`TelephoneNumber`,

/\* User data \*/

`t`.`FullName`,`t`.`Username`,

/\*Job data\*/

`j`.`JobTitle`

FROM `Companies` AS `c`

LEFT JOIN (

/\* The department data \*/

SELECT`d`.`Color`,`d`.`DepartmentName`

FROM `Departments` AS `d`

LEFT JOIN `Users` AS `u` ON `d`.`Id` = `u`.`DepartmentId`

) AS `t` ON `c`.`Id` = `t`.`CompanyId`

LEFT JOIN `Jobs` AS `j` ON `c`.`Id` = `j`.`CompanyId`

ORDER BY `c`.`Id`, `t`.`Id`, `t`.`Id0`, `j`.`Id`;

end$

DELIMITER ;

CALL GetCompaniesWithTheirData();

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetSubjectsWithData;

DELIMITER $

/\* Creates a stored procedure that gets all the required data for the admin's subjects' page

Basically for every subject its parent, description and name \*/

create procedure GetSubjectsWithData()

begin

SELECT `s`.`Description`, `s`.`Title`,

/\*Parent data (just the title)\*/

`s0`.`Title`

FROM `Subjects` AS `s`

LEFT JOIN `Subjects` AS `s0` ON `s`.`Id` = `s0`.`SubjectId`

ORDER BY `s`.`Id`, `s0`.`Id`;

end$

DELIMITER ;

CALL GetSubjectsWithData();

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetCompanyProfileData;

DELIMITER $

/\* Creates a stored procedure that gets all the required data for the admin's subjects' page

Basically for every subject its parent, description and name \*/

create procedure GetCompanyProfileData(in companyId INT)

begin

SELECT `t`.`AFM`, `t`.`About`, `t`.`City`, `t`.`CompanyPicture`, `t`.`Country`, `t`.`DOY`, `t`.`DateCreated`, `t`.`Name`, `t`.`StreetName`, `t`.`StreetNumber`,

`t`.`TelephoneNumber`,

/\* The number of users in total in this company \*/

COUNT(`u`.`Id`),

/\* The number of departments in total \*/

COUNT(`d`.`Id`),

/\* The number of jobs in total in this company \*/

COUNT(`j`.`Id`)

FROM (

SELECT `c`.`Id`, `c`.`AFM`, `c`.`About`, `c`.`City`, `c`.`CompanyPicture`, `c`.`Country`, `c`.`DOY`, `c`.`DateCreated`, `c`.`Name`, `c`.`StreetName`, `c`.`StreetNumber`, `c`.`TelephoneNumber`

FROM `Companies` AS `c`

WHERE `c`.`Id` = companyId

LIMIT 1

) AS `t`

LEFT JOIN `Users` AS `u` ON `t`.`Id` = `u`.`CompanyId`

LEFT JOIN `Departments` AS `d` ON `t`.`Id` = `d`.`CompanyId`

LEFT JOIN `Jobs` AS `j` ON `t`.`Id` = `j`.`CompanyId`

ORDER BY `t`.`Id`, `u`.`Id`, `d`.`Id`, `j`.`Id`;

end$

DELIMITER ;

CALL GetCompanyProfileData(1);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetDepartmentEmployees;

DELIMITER $

/\* Creates a stored procedure that gets all the required data for the manager's employees' page

Basically for every employee that is in the same company as the manager gets their username and full name \*/

create procedure GetDepartmentEmployees(in companyId INT)

begin

SELECT `u`.`FullName`, `u`.`Username`

FROM `Users` AS `u`

LEFT JOIN `Departments` AS `d` ON `u`.`DepartmentId` = `d`.`Id`

LEFT JOIN `Companies` AS `c` ON `d`.`CompanyId` = `c`.`Id`

WHERE (`d`.`CompanyId` = companyId) AND (`u`.`Type` = 1)

ORDER BY `u`.`Id`, `a`.`Id`, `c0`.`Id`, `p`.`Id`, `r`.`Id`, `a0`.`Id`;

end$

DELIMITER ;

CALL GetCompanyProfileData(1);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS GetManagerNotFinalizedReports;

DELIMITER $

/\* Creates a stored procedure that gets all the required data for the manager's employees' page

Basically for every employee that is in the same company as the manager gets their username and full name \*/

create procedure GetManagerNotFinalizedReports(in managerId INT)

begin

SELECT `r`.`ReportText`, `u0`.`Username`,`u1`.`Username`, `j`.`RequestsReason`, `j0`.`AnnouncementDate`, `j0`.`SubmissionDate`,

`j1`.`JobTitle`, `j1`.`Salary`, `d`.`DepartmentName`

FROM `Reports` AS `r`

INNER JOIN `UsersJobFilesPairs` AS `u` ON `r`.`UsersJobFilesPairId` = `u`.`Id`

INNER JOIN `Users` AS `u0` ON `u`.`ManagerId` = `u0`.`Id`

INNER JOIN `Users` AS `u1` ON `u`.`EvaluatorId` = `u1`.`Id`

INNER JOIN `Users` AS `u2` ON `u`.`EmployeeId` = `u2`.`Id`

INNER JOIN `JobPositionRequests` AS `j` ON `r`.`JobPositionRequestId` = `j`.`Id`

INNER JOIN `JobPositions` AS `j0` ON `j`.`JobPositionId` = `j0`.`Id`

INNER JOIN `Jobs` AS `j1` ON `j0`.`JobId` = `j1`.`Id`

INNER JOIN `Departments` AS `d` ON `j1`.`DepartmentId` = `d`.`Id`

LEFT JOIN `JobPositionRequests` AS `j2` ON `u`.`Id` = `j2`.`UsersJobFilesPairId`

WHERE (`u`.`ManagerId` = managerId) AND (`r`.`IsFinalized` = FALSE)

ORDER BY `r`.`Id`, `u`.`Id`, `u0`.`Id`, `u1`.`Id`, `u2`.`Id`, `j`.`Id`, `j0`.`Id`, `j1`.`Id`, `d`.`Id`, `j2`.`Id`;

end$

DELIMITER ;

CALL GetManagerNotFinalizedReports(1);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS CreateEvaluationByFinalizedReport;

DELIMITER $

/\* Creates a stored procedure that creates and inserts in the evaluations table

a new evaluation with default values\*/

create procedure CreateEvaluationByFinalizedReport(in jobPositionRequestId INT, in usersJobFilesPairId INT)

begin

INSERT INTO `Evaluations` (`Comments`, `FilesGrade`, `FinalGrade`, `InterviewGrade`, `IsAprovedByManager`, `IsFinalized`, `JobPositionRequestId`, `Passed`,

`ReportGrade`, `UsersJobFilesPairId`)

VALUES ('', 0, 0, 0, false, false, jobPositionRequestId, false, 0, usersJobFilesPairId);

SELECT `Id`

FROM `Evaluations`

WHERE ROW\_COUNT() = 1 AND `Id` = LAST\_INSERT\_ID();

end$

DELIMITER ;

CALL CreateEvaluationByFinalizedReport(1, 1);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS UpdateJobSalaryByManager;

DELIMITER $

/\* Creates a stored procedure that creates and inserts in the evaluations table

a new evaluation with default values\*/

create procedure UpdateJobSalaryByManager(in salary INT, in jobId INT)

begin

UPDATE `Jobs` SET `Salary` = salary

WHERE `Id` = jobId;

end$

DELIMITER ;

CALL UpdateJobSalaryByManager(1, 1);

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS AddToJobSubjectPair;

DELIMITER $

/\* Creates a stored procedure that creates and inserts in the evaluations table

a new evaluation with default values\*/

create procedure AddToJobSubjectPair(in jobPositionId INT , in subjectId INT)

begin

INSERT INTO `JobPositionSubjects` (`JobPositionId`, `SubjectId`)

VALUES (jobPositionId, subjectId);

end$

DELIMITER ;

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS CreateJobPositionByEvaluator;

DELIMITER $

/\* Creates a stored procedure that creates and inserts in the evaluations table

a new evaluation with default values\*/

create procedure CreateJobPositionByEvaluator(in announcementDate DATETIME, in salary INT, in submissionDate DATETIME, in jobId INT, in creatorId INT,in jobPositionId INT , in subjectId INT )

begin

INSERT INTO `JobPositions` (`AnnouncementDate`, `CreatorId`, `JobId`, `StartDate`, `SubmissionDate`)

VALUES (announcementDate, submissionDate, jobId, null, creatorId);

SELECT `Id`

FROM `JobPositions`;

/\* Adds to the pairs a new pair with the selected subject \*/

call AddToJobSubjectPair(jobPositionId, subjectId);

/\* Updates the job's salary\*/

UPDATE `Jobs` SET `Salary` = salary

WHERE `Id` = jobId;

end$

DELIMITER ;

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS CreateJobPositionSubjectPair;

DELIMITER $

/\* Creates a stored procedure that creates and inserts in the evaluations table

a new evaluation with default values\*/

create procedure CreateJobPositionSubjectPair(in jobPositionId INT , in subjectId INT )

begin

INSERT INTO `JobPositionSubjects` (`JobPositionId`, `SubjectId`)

VALUES (jobPositionId, subjectId);

end$

DELIMITER ;

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS DeleteJobPositionSubjectPair;

DELIMITER $

/\* Creates a stored procedure that creates and inserts in the evaluations table

a new evaluation with default values\*/

create procedure DeleteJobPositionSubjectPair(in jobPositionId INT , in subjectId INT )

begin

DELETE FROM `JobPositionSubjects`

WHERE `JobPositionId` = jobPositionId AND `SubjectId` = subjectId;

end$

DELIMITER ;

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS DeleteJobPositionSubjectPair;

DELIMITER $

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS PrintNewJobPosition;

DELIMITER $

/\* Creates a stored procedure that gets the newly create job position from the job positions table\*/

create procedure PrintNewJobPosition(in jobPositionId INT )

begin

SELECT `t`.`JobTitle`, `t`.`Salary`,

`t`.`DepartmentName`

FROM (

SELECT `j`.`AnnouncementDate`, `j`.`SubmissionDate`, `d`.`DepartmentName`

FROM `JobPositions` AS `j`

INNER JOIN `Jobs` AS `j0` ON `j`.`JobId` = `j0`.`Id`

INNER JOIN `Departments` AS `d` ON `j0`.`DepartmentId` = `d`.`Id`

WHERE `j`.`Id` = jobPositionId

LIMIT 1

) AS `t`

LEFT JOIN (

SELECT `s`.`Title`

FROM `JobPositionSubjects` AS `j1`

INNER JOIN `Subjects` AS `s` ON `j1`.`SubjectId` = `s`.`Id`

INNER JOIN `JobPositions` AS `j2` ON `j1`.`JobPositionId` = `j2`.`Id`

) AS `t0` ON `t`.`Id` = `t0`.`JobPositionId`

ORDER BY `t`.`Id`, `t`.`Id0`, `t`.`Id1`, `t0`.`JobPositionId`, `t0`.`SubjectId`, `t0`.`Id`, `t0`.`Id0`;

end$

DELIMITER ;

/\* Deletes the procedure if it exists \*/

DROP PROCEDURE IF EXISTS PrintNewJobPosition;

DELIMITER $

/\* Deletes the procedure if it exists \*/

CREATE:

CREATE TABLE `acquireddegrees` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Grade` int(11) NOT NULL,

`YearEarned` datetime(6) NOT NULL,

`UserId` int(11) NOT NULL,

`DegreeId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_AcquiredDegrees\_DegreeId` (`DegreeId`),

KEY `IX\_AcquiredDegrees\_UserId` (`UserId`),

CONSTRAINT `FK\_AcquiredDegrees\_Degrees\_DegreeId` FOREIGN KEY (`DegreeId`) REFERENCES `degrees` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_AcquiredDegrees\_Users\_UserId` FOREIGN KEY (`UserId`) REFERENCES `users` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=201 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `awards` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Name` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`AcquiredDate` datetime(6) NOT NULL,

`UserId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Awards\_UserId` (`UserId`),

CONSTRAINT `FK\_Awards\_Users\_UserId` FOREIGN KEY (`UserId`) REFERENCES `users` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=505 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `certificates` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Title` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Description` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`UserId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Certificates\_UserId` (`UserId`),

CONSTRAINT `FK\_Certificates\_Users\_UserId` FOREIGN KEY (`UserId`) REFERENCES `users` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=305 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `companies` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Name` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`DateCreated` datetime(6) NOT NULL,

`DOY` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`AFM` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`About` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`TelephoneNumber` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`City` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Country` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`StreetNumber` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`StreetName` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`CompanyPicture` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

PRIMARY KEY (`Id`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `degrees` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Title` int(11) NOT NULL,

`School` int(11) NOT NULL,

`LevelOfEducation` int(11) NOT NULL,

PRIMARY KEY (`Id`)

) ENGINE=InnoDB AUTO\_INCREMENT=51 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `departments` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`DepartmentName` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Color` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`CompanyId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Departments\_CompanyId` (`CompanyId`),

CONSTRAINT `FK\_Departments\_Companies\_CompanyId` FOREIGN KEY (`CompanyId`) REFERENCES `companies` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=33 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `evaluations` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`InterviewGrade` int(11) DEFAULT NULL,

`IsFinalized` tinyint(1) NOT NULL,

`Comments` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`ReportGrade` int(11) DEFAULT NULL,

`FilesGrade` int(11) DEFAULT NULL,

`FinalGrade` int(11) DEFAULT NULL,

`IsAprovedByManager` tinyint(1) NOT NULL,

`Passed` tinyint(1) DEFAULT NULL,

`JobPositionRequestId` int(11) NOT NULL,

`UsersJobFilesPairId` int(11) NOT NULL,

`UserDataModelId` int(11) DEFAULT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Evaluations\_JobPositionRequestId` (`JobPositionRequestId`),

KEY `IX\_Evaluations\_UserDataModelId` (`UserDataModelId`),

KEY `IX\_Evaluations\_UsersJobFilesPairId` (`UsersJobFilesPairId`),

CONSTRAINT `FK\_Evaluations\_JobPositionRequests\_JobPositionRequestId` FOREIGN KEY (`JobPositionRequestId`) REFERENCES `jobpositionrequests` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_Evaluations\_UsersJobFilesPairs\_UsersJobFilesPairId` FOREIGN KEY (`UsersJobFilesPairId`) REFERENCES `usersjobfilespairs` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_Evaluations\_Users\_UserDataModelId` FOREIGN KEY (`UserDataModelId`) REFERENCES `users` (`id`) ON DELETE RESTRICT

) ENGINE=InnoDB AUTO\_INCREMENT=512 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `jobpositionrequests` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`RequestsReason` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`JobPositionId` int(11) NOT NULL,

`UsersJobFilesPairId` int(11) DEFAULT NULL,

`UserDataModelId` int(11) DEFAULT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_JobPositionRequests\_JobPositionId` (`JobPositionId`),

KEY `IX\_JobPositionRequests\_UserDataModelId` (`UserDataModelId`),

KEY `IX\_JobPositionRequests\_UsersJobFilesPairId` (`UsersJobFilesPairId`),

CONSTRAINT `FK\_JobPositionRequests\_JobPositions\_JobPositionId` FOREIGN KEY (`JobPositionId`) REFERENCES `jobpositions` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_JobPositionRequests\_UsersJobFilesPairs\_UsersJobFilesPairId` FOREIGN KEY (`UsersJobFilesPairId`) REFERENCES `usersjobfilespairs` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_JobPositionRequests\_Users\_UserDataModelId` FOREIGN KEY (`UserDataModelId`) REFERENCES `users` (`id`) ON DELETE RESTRICT

) ENGINE=InnoDB AUTO\_INCREMENT=501 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `jobpositions` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`AnnouncementDate` datetime(6) DEFAULT NULL,

`SubmissionDate` datetime(6) DEFAULT NULL,

`StartDate` datetime(6) DEFAULT NULL,

`CreatorId` int(11) DEFAULT NULL,

`JobId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_JobPositions\_JobId` (`JobId`),

CONSTRAINT `FK\_JobPositions\_Jobs\_JobId` FOREIGN KEY (`JobId`) REFERENCES `jobs` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=163 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `jobpositionsubjects` (

`SubjectId` int(11) NOT NULL,

`JobPositionId` int(11) NOT NULL,

PRIMARY KEY (`JobPositionId`,`SubjectId`),

KEY `IX\_JobPositionSubjects\_SubjectId` (`SubjectId`),

CONSTRAINT `FK\_JobPositionSubjects\_JobPositions\_JobPositionId` FOREIGN KEY (`JobPositionId`) REFERENCES `jobpositions` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_JobPositionSubjects\_Subjects\_SubjectId` FOREIGN KEY (`SubjectId`) REFERENCES `subjects` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `jobs` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`JobTitle` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Salary` int(11) NOT NULL,

`CompanyId` int(11) DEFAULT NULL,

`DepartmentId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Jobs\_CompanyId` (`CompanyId`),

KEY `IX\_Jobs\_DepartmentId` (`DepartmentId`),

CONSTRAINT `FK\_Jobs\_Companies\_CompanyId` FOREIGN KEY (`CompanyId`) REFERENCES `companies` (`id`) ON DELETE RESTRICT,

CONSTRAINT `FK\_Jobs\_Departments\_DepartmentId` FOREIGN KEY (`DepartmentId`) REFERENCES `departments` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=201 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `languages` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Name` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`UserId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Languages\_UserId` (`UserId`),

CONSTRAINT `FK\_Languages\_Users\_UserId` FOREIGN KEY (`UserId`) REFERENCES `users` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=502 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `logs` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Username` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Action` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Details` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`When` datetime(6) NOT NULL,

PRIMARY KEY (`Id`)

) ENGINE=InnoDB AUTO\_INCREMENT=58 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `projects` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Title` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Url` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Description` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`MadeForWho` int(11) NOT NULL,

`UserId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Projects\_UserId` (`UserId`),

CONSTRAINT `FK\_Projects\_Users\_UserId` FOREIGN KEY (`UserId`) REFERENCES `users` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=604 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `recomendationpapers` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Referee` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Description` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`UserId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_RecomendationPapers\_UserId` (`UserId`),

CONSTRAINT `FK\_RecomendationPapers\_Users\_UserId` FOREIGN KEY (`UserId`) REFERENCES `users` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=402 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `reports` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`ReportText` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`IsWritten` tinyint(1) NOT NULL,

`IsFinalized` tinyint(1) NOT NULL,

`JobPositionRequestId` int(11) NOT NULL,

`UsersJobFilesPairId` int(11) NOT NULL,

`UserDataModelId` int(11) DEFAULT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Reports\_JobPositionRequestId` (`JobPositionRequestId`),

KEY `IX\_Reports\_UserDataModelId` (`UserDataModelId`),

KEY `IX\_Reports\_UsersJobFilesPairId` (`UsersJobFilesPairId`),

CONSTRAINT `FK\_Reports\_JobPositionRequests\_JobPositionRequestId` FOREIGN KEY (`JobPositionRequestId`) REFERENCES `jobpositionrequests` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_Reports\_UsersJobFilesPairs\_UsersJobFilesPairId` FOREIGN KEY (`UsersJobFilesPairId`) REFERENCES `usersjobfilespairs` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_Reports\_Users\_UserDataModelId` FOREIGN KEY (`UserDataModelId`) REFERENCES `users` (`id`) ON DELETE RESTRICT

) ENGINE=InnoDB AUTO\_INCREMENT=501 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `subjects` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Title` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Description` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`SubjectId` int(11) DEFAULT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Subjects\_SubjectId` (`SubjectId`),

CONSTRAINT `FK\_Subjects\_Subjects\_SubjectId` FOREIGN KEY (`SubjectId`) REFERENCES `subjects` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=25 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `users` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`Type` int(11) NOT NULL,

`Username` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Password` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`FirstName` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`LastName` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`FullName` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`Email` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`ProfilePicture` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`RegistrationDate` datetime(6) NOT NULL,

`YearsOfExperience` int(11) NOT NULL,

`Bio` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci,

`DepartmentId` int(11) DEFAULT NULL,

`CompanyId` int(11) DEFAULT NULL,

`JobPositionId` int(11) DEFAULT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_Users\_CompanyId` (`CompanyId`),

KEY `IX\_Users\_DepartmentId` (`DepartmentId`),

KEY `IX\_Users\_JobPositionId` (`JobPositionId`),

CONSTRAINT `FK\_Users\_Companies\_CompanyId` FOREIGN KEY (`CompanyId`) REFERENCES `companies` (`id`) ON DELETE RESTRICT,

CONSTRAINT `FK\_Users\_Departments\_DepartmentId` FOREIGN KEY (`DepartmentId`) REFERENCES `departments` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_Users\_JobPositions\_JobPositionId` FOREIGN KEY (`JobPositionId`) REFERENCES `jobpositions` (`id`) ON DELETE RESTRICT

) ENGINE=InnoDB AUTO\_INCREMENT=299 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `usersjobfilespairs` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`ManagerId` int(11) NOT NULL,

`EvaluatorId` int(11) NOT NULL,

`EmployeeId` int(11) NOT NULL,

PRIMARY KEY (`Id`),

KEY `IX\_UsersJobFilesPairs\_EmployeeId` (`EmployeeId`),

KEY `IX\_UsersJobFilesPairs\_EvaluatorId` (`EvaluatorId`),

KEY `IX\_UsersJobFilesPairs\_ManagerId` (`ManagerId`),

CONSTRAINT `FK\_UsersJobFilesPairs\_Users\_EmployeeId` FOREIGN KEY (`EmployeeId`) REFERENCES `users` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_UsersJobFilesPairs\_Users\_EvaluatorId` FOREIGN KEY (`EvaluatorId`) REFERENCES `users` (`id`) ON DELETE CASCADE,

CONSTRAINT `FK\_UsersJobFilesPairs\_Users\_ManagerId` FOREIGN KEY (`ManagerId`) REFERENCES `users` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=201 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;