

Βάσεις Δεδομένων Εξαμηνιαία Εργασία

Ιωάννης Δάρας (03115018 - daras.giannhs@gmail.com)

Μαρία Παρέλλη (03115155 - maryparelli@gmail.com)

"The great advantage of a hotel is that it is a refuge from home life."
- George Bernard Shaw

1 Οργάνωση κώδικα και δομή ιστοσελίδας

Στην παρούσα ενότητα περιγράφεται η οργάνωση του κώδικα και η δομή της ιστοσελίδας όπως αυτή προκύπτει από τις μεταβάσεις μεταξύ των διαφορετικών παρεχόμενων οθονών.

1.1 Οργάνωση κώδικα

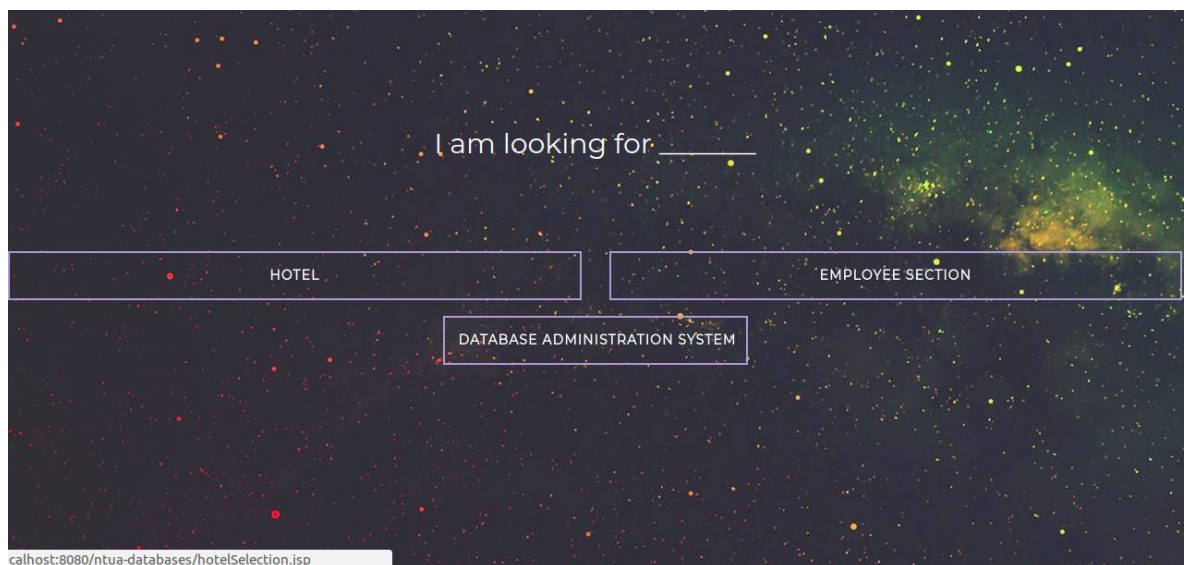
Κάνοντας unzip το αρχείο που έχει υποβληθεί στο mycourses βλέπουμε τους εξής φακέλους:

1. `./report/` : Περιέχει τα αρχεία για την ανάπτυξη της παρούσας αναφοράς σε \LaTeX καθώς και ένα export της σε pdf.
2. `WebContent`: Περιέχει το front-end της εφαρμογής. Ειδικότερα:
 - (α') `./WebContent/bootstrap` : Περιέχεται ο κώδικας του bootstrap framework που χρησιμοποιείται σε μεγάλο βαθμό (ιδιαίτερα το grid system που παρέχει) για το σχεδιασμό του css της εφαρμογής.
 - (β') `./WebContent/fancy_table`: Περιέχει μια υλοποίηση βασισμένη σε Javascript και css ενός διαδραστικού table που χρησιμοποιούμε στην εφαρμογή μας.
 - (γ') `./WebContent/favicons` : Περιέχει το fontawesome που χρησιμοποιείται για διάφορα favicons της εφαρμογής μας.
 - (δ') `./WebContent/images` : Περιέχει το background image αλλά και διάφορες άλλες εικόνες που χρησιμοποιούμε σε διάφορα σημεία στην εφαρμογή μας.
 - (ε') `./WebContent/js` : περιέχει το `calender.js` που πρακτικά είναι ένα custom ημερολόγιο που χρησιμοποιούμε για την επιλογή των start και finish dates στην εφαρμογή μας.
 - (στ') `./WebContent/META-INF` : περιέχει το manifest της εφαρμογής μας
 - (ζ') `./WebContent/WEB-INF` : περιέχει διαθέσιμες βιβλιοθήκες αλλά και το αρχείο `web.xml` που χρησιμεύει ως config file για την εφαρμογή μας.
 - (η') `./WebContent/*.jsp` : Τα αρχεία αυτά αποτελούν τις σελίδες της εφαρμογής μας.
 - (θ') `./WebContent/theme.css` : περιέχει το css template που γράψαμε για την εφαρμογή μας.
3. `./src/com/ntua/databases/` : Στον φάκελο αυτό συναντάμε τα εξής αρχεία:
 - (α') `./src/com/ntua/databases/Connector.java` : αποτελεί το αρχείο που μας εξασφαλίζει τη σύνδεση με τη βάση δεδομένων
 - (β') Διάφορα Servlets : Τα Servlets αυτά είναι Java αρχεία που επικοινωνούν με τη βάση (την ρωτάνε, τροποποιούν τα δεδομένα της, τα διαγράφουν, κ.α.) ανάλογα με τις εισόδους που βάζει στις αντίστοιχες φόρμες ο χρήστης. Στη συνέχεια, τα αποτελέσματα τους τα επιστρέφουν στα jsp files. Σημειώνεται ότι τα servlet χρησιμοποιούνται και για form validation.
4. `./src/com/ntuadatabases/` : Εδώ συναντάμε Servlets που αφορούν το κομμάτι του administration της βάσης δεδομένων. Συγκεκριμένα, έχουμε:

- (α') `./src/com/ntuadatabases/controller/` : Οι controllers για την ανταλλαγή πληροφοριών μεταξύ Servlet και jsp files.
 - (β') `./src/com/ntuadatabases/dao/` : Εδώ βρίσκονται Java files που ρωτάνε ή τροποποιούν το κομμάτι της βάσης που αφορά το database administration.
 - (γ') `./src/com/ntuadatabases/model/` : Εδώ ορίζονται μοντέλα (κλάσσεις) των οποίων τα instances περνάμε στα jsp files μέσω των controllers.
5. `./build` : Εδώ βρίσκονται τα build files της εφαρμογής μας.
 6. `./database-setup` : Εδώ βρίσκονται τα ακόλουθα αρχεία:
 - (α') `./database-setup/tables-creator` : sql script που δημιουργεί όλους τους πίνακες στη βάση δεδομένων μας.
 - (β') `./database-setup/database-initializer` : sql script που αρχικοποιεί τη βάση δεδομένων μας με διάφορα δεδομένα.
 - (γ') `./database-setup/database-triggers.sql` : sql script που περιέχει τους triggers της βάσης μας.
 - (δ') `./database-setup/database-indexes.sql` : sql script που ορίζει ευρετήρια για τη βάση μας.
 - (ε') `./database-setup/database-views.sql` : Εδώ ορίζονται όψεις της βάσης δεδομένων μας.
 - (στ') `./LICENCE` : περιέχει την άδεια της εφαρμογής
 - (ζ') `./README.md` : περιέχει το README file της εφαρμογής όπως αυτό πάρθηκε από το repository της ομάδας στο Github.

1.2 Δομή Ιστοσελίδας - Μεταβάσεις

Η αρχική σελίδα της εφαρμογής μας είναι το `welcome.jsp`, στο οποίο υπάρχουν 3 κουμπιά, το καθένα από τα οποία αντιστοιχεί σε μία από τις τρεις κατηγορίες χρηστών της εφαρμογής μας:

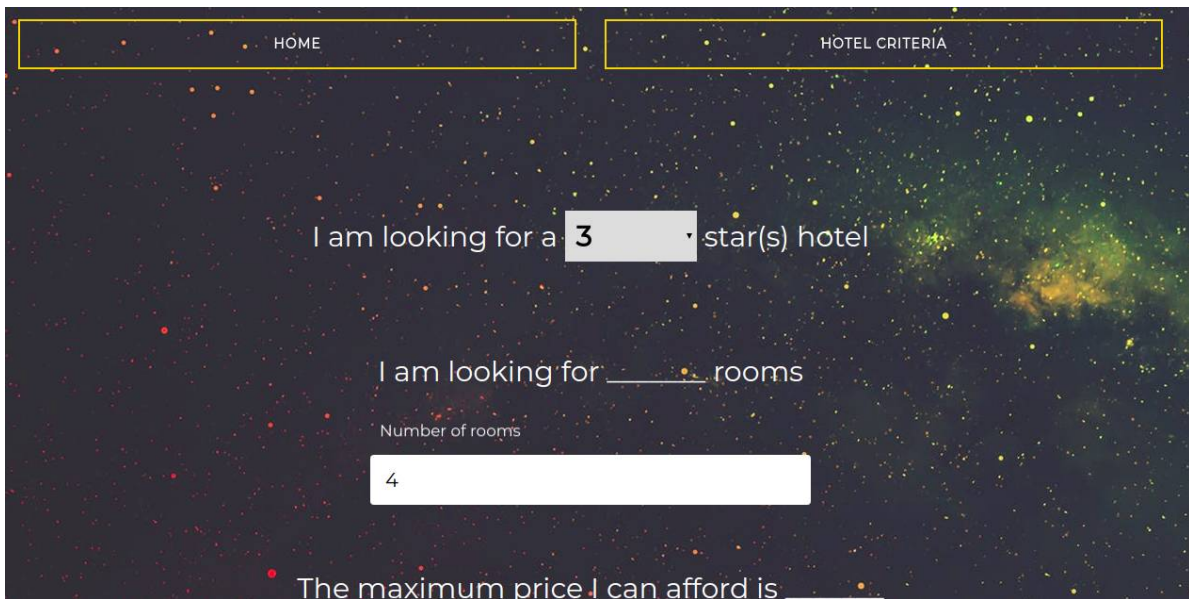


1. Hotel: το επιλέγουν οι πελάτες των ξενοδοχείων για να μεταβούν στη σελίδα `hotelSelection.jsp`, όπου θα θέσουν τα κριτήρια τους για την αναζήτηση ξενοδοχείου.
2. Employee Login: το επιλέγουν οι ενεργοί υπάλληλοι των ξενοδοχείων για να μεταβούν στην σελίδα `EmployeeLogin.jsp`, όπου γίνεται επιβεβαίωση της ταυτότητάς τους.
3. Database Administration System: το επιλέγει ο διαχειριστής της βάσης για να μεταβεί στη σελίδα `protectedLogin.jsp`, όπου θα εισάγει τον ειδικό κωδικό ώστε να αποκτήσει πρόσβαση στις λειτουργίες της βάσης δεδομένων.

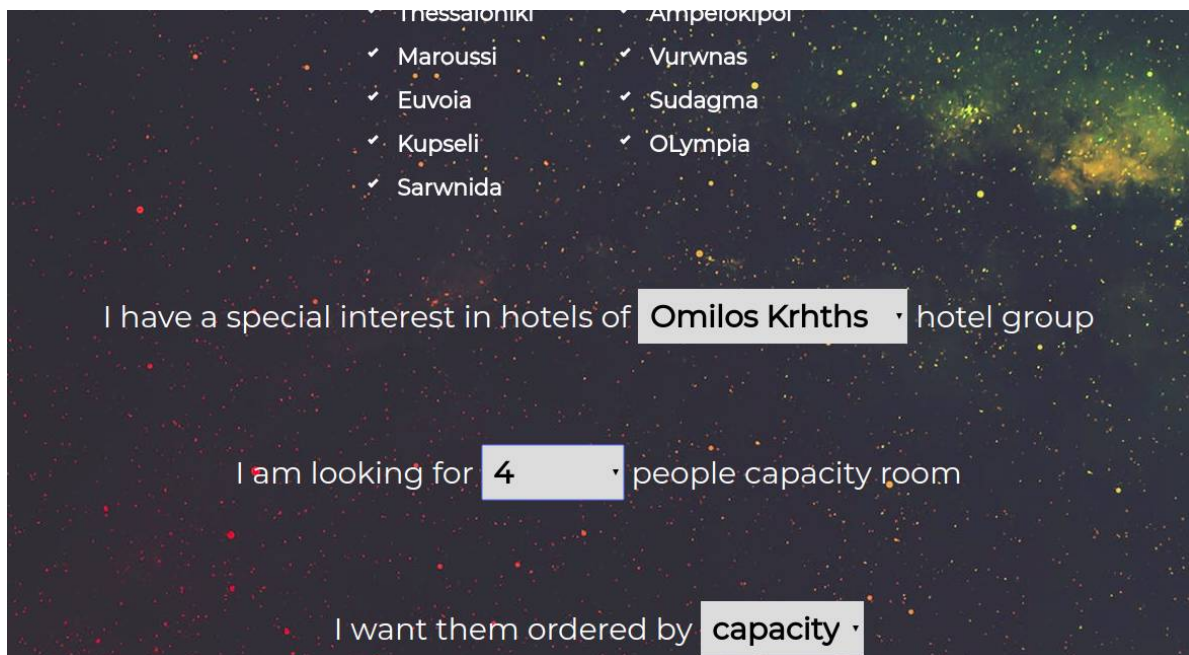
Συνεπώς εξάγουμε το συμπέρασμα ότι ανάλογα με το ποιος χρησιμοποιεί την ιστοσελίδα μας διαμορφώνονται 3 ξεχωριστές διαδρομές:

1. Διαδρομή πελάτη

Αφού ο πελάτης επιλέξει το κουμπί Hotel στην αρχική σελίδα, οδηγείται στη σελίδα hotelSelection.jsp, όπου θέτει τα κριτήρια για την επιλογή δωματίων. Συγκεκριμένα, μπορεί να επιλέξει κατηγορία ξενοδοχείου, αριθμό δωματίων, περιοχή, ημερομηνίες καθώς και όμιλο, στον οποίο θέλει να ανήκει.

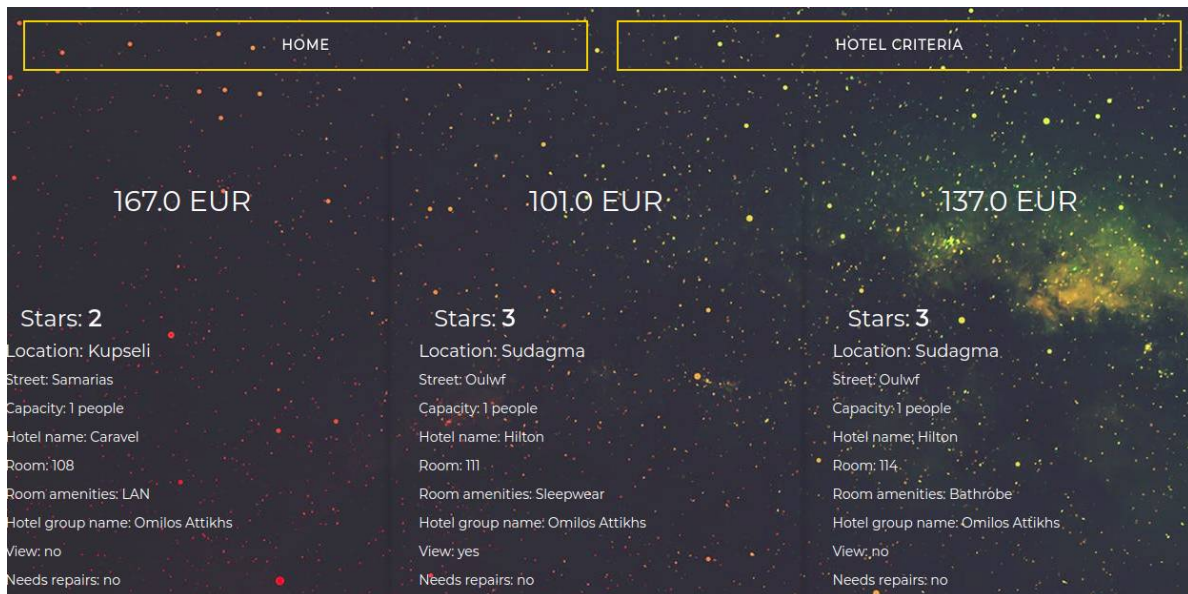


Φυσικά, του δίνεται η επιλογή να προσαρμόσει την αναζήτηση του συμπληρώνοντας όσα από τα παραπάνω κριτήρια επιθυμεί. Παράλληλα, του δίνεται η επιλογή να λάβει τα δωμάτια ταξινομημένα ανά κατηγορία είτε ανά περιοχή. Στη σελίδα αυτή δίνεται η δυνατότητα στον πελάτη να δει και τις όψεις της βάσης που αφορούν τον αριθμό των διαθέσιμων δωματίων ανά περιοχή και τον αριθμό των διαθέσιμων δωματίων ανά capacity.

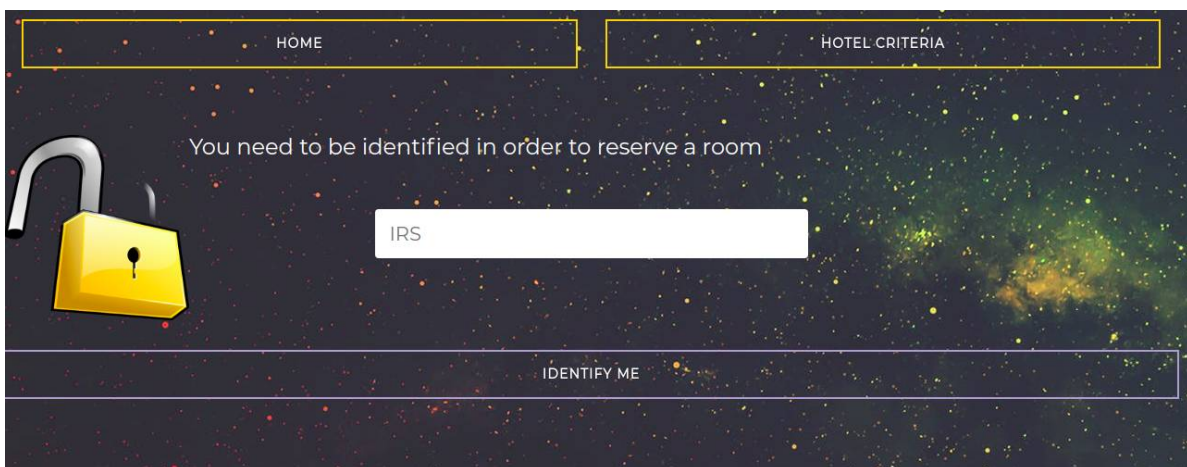


Πατώντας του κουμπί Query Me, ο χρήστης πραγματοποιεί την αναζήτηση. Αυτό επιτυγχάνεται ως εξής:

Τα πεδία, τα οποία συμπλήρωσε ο χρήστης στέλνονται ως παράμετροι στο servlet SearchServlet.java, όπου γίνονται τα κατάλληλα queries στη βάση και τα αποτελέσματα στέλνονται στη σελίδα roomselection.jsp. Στη σελίδα αυτή ο χρήστης μπορεί να δει τα χαρακτηριστικά και να επιλέξει ανάμεσα στα δωμάτια, που πληρούν τις προϋποθέσεις, που εισήγαγε.



Μόλις επιλέξει το δωμάτιο, που επιθυμεί μεταφέρεται στη σελίδα `identifyCustomer.jsp`, όπου του ζητείται να επιβεβαιώσει την ταυτότητα του δίνοντας το ΑΦΜ του. Εάν δεν υπάρχει στη βάση πελάτης καταχωρημένος με αυτό το ΑΦΜ, τότε ο πελάτης ανακατευθύνεται στη σελίδα `createCustomer.jsp`, στην οποία εισάγει τα στοιχεία του και δημιουργείται ο λογαριασμός του. Αντίθετα, εάν ο πελάτης είναι καταχωρημένος στη βάση τότε κατευθύνεται στη σελίδα `reservationDates.jsp`, στην οποία εισάγει της ημερομηνίες, στις οποίες επιθυμεί να γίνει η κράτηση. Εάν το δωμάτιο είναι ακόμη διαθέσιμο, τότε η κράτηση πραγματοποιείται με επιτυχία και ο πελάτης μεταφέρεται στη σελίδα `reservationReady.jsp`. Σε αντίθετη περίπτωση μεταφέρεται στη σελίδα `reservationProblem.jsp`.



Όλα τα παραπάνω σε επίπεδο εφαρμογής πραγματοποιούνται ως εξής: Όταν ο χρήστης επιλέξει δωμάτιο τα στοιχεία `room_id`, `hotel_id`, δηλαδή τα πρωτεύοντα κλειδιά της σχέσης `hotel_room` περνούν ως παράμετροι στο servlet `IdentifyCustomer.java`. Εκεί γίνεται query στη σχέση `customer`, για να διαπιστωθεί εάν ο πελάτης υπάρχει στη βάση. Ανάλογα με το αποτέλεσμα ο χρήστης κατευθύνεται και στην αντίστοιχη σελίδα, όπως αναλύθηκε παραπάνω. Εάν ο πελάτης υπάρχει, στη σελίδα `reservationDates.jsp` μεταφέρονται ως παράμετροι το `room_id`, το `hotel_id`, το ΑΦΜ του πελάτη καθώς και το ονοματεπώνυμό του. Εάν ο πελάτης δεν υπάρχει, κατευθύνεται στη σελίδα `createCustomer.jsp`, όπου μεταφέρονται ως παράμετροι, το `room_id`, το `hotel_id` καθώς και το ΑΦΜ του. Τα πεδία, που ο πελάτης συμπληρώνει στη σελίδα στέλνονται στο servlet `NewCustomer.java`, όπου γίνονται οι κατάλληλες εισαγωγές στη βάση και ο έλεγχος μεταφέρεται στη σελίδα `reservationDates.jsp`.

Welcome back Mpampis Apostolou

One last step

My start date is

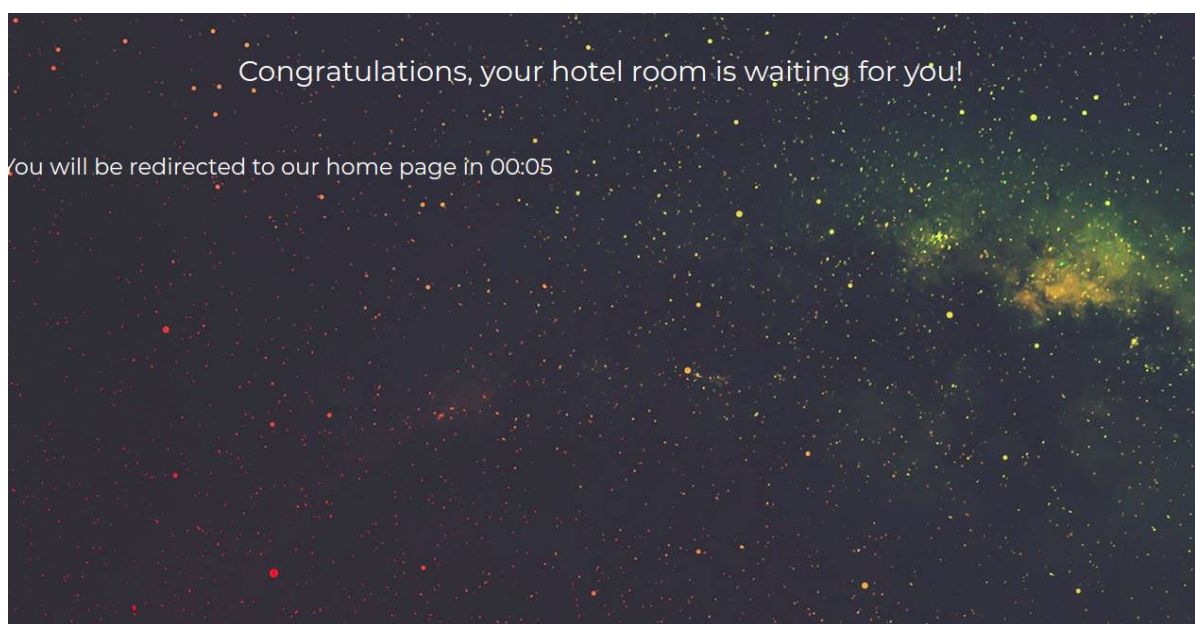
17 01 2019

My end date is

17 09 2022

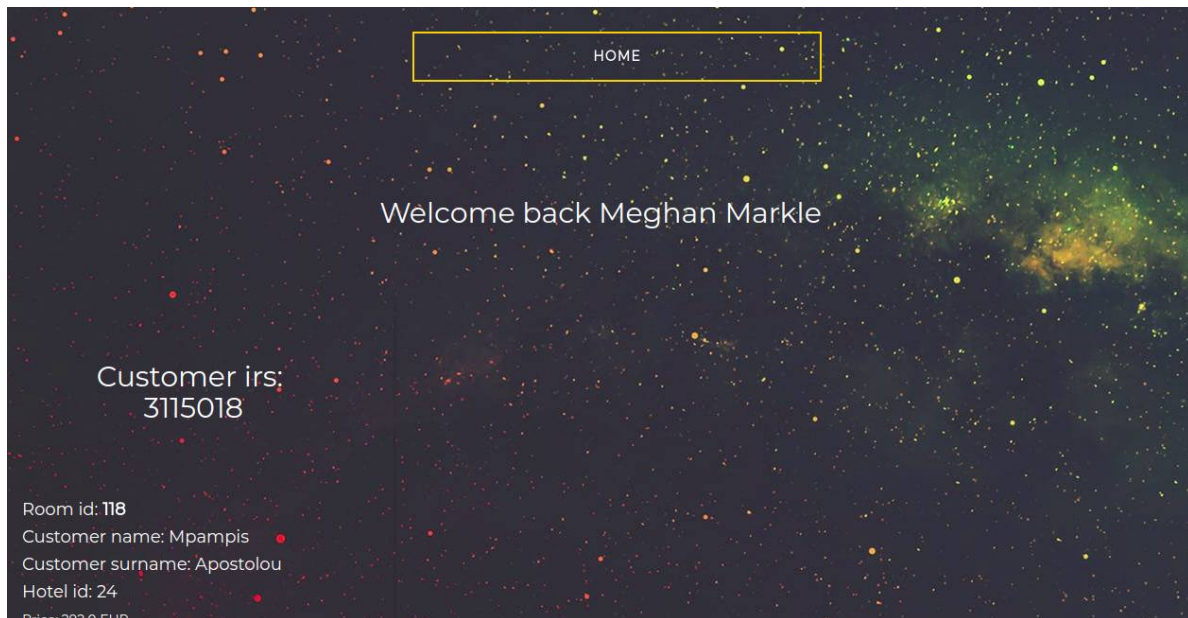
RESERVE

Αφού ο πελάτης εισάγει τις επιθυμητές ημερομηνίες για το δωμάτιο της επιλογής του, αυτές μεταφέρονται ως παράμετροι στο servlet `Reservation.java`, στο οποίο γίνεται έλεγχος εάν στο δωμάτιο αυτό έχει εντωμεταξύ γίνει κράτηση για τις ίδιες ημερομηνίες. Εάν όχι γίνεται η κατάλληλη εισαγωγή στη σχέση `reserves` και ο έλεγχος μεταφέρεται στη σελίδα `reservationReady.jsp`. Εάν ναι, τότε ο έλεγχος μεταφέρεται στη σελίδα `reservationProblem.jsp`.



2. Διαδρομή υπαλλήλου

Ο υπάλληλος ξενοδοχείου στη σελίδα `welcome.jsp` επιλέγει το κουμπί `Employee Section` και μεταφέρεται στη σελίδα `EmployeeLogin.jsp`, στην οποία εισάγει το ΑΦΜ του. Εάν αυτό αντιστοιχεί σε πεδίο στη βάση `Employee`, τότε η σύνδεση του είναι επιτυχής και μεταφέρεται στη σελίδα `employeeRent.jsp`. Αντίθετα εμφανίζεται μήνυμα λάθους. Στη σελίδα `employeeRent.jsp` ο υπάλληλος μπορεί να δει τις ανοιχτές κρατήσεις για τα ξενοδοχεία, στο οποίο εργάζεται.



Όταν απαιτηθεί να γίνει το check-in του πελάτη ο υπάλληλος επιλέγει την κράτηση, που του αντιστοιχεί και μεταφέρεται στη σελίδα `implementPayment.jsp`. Στη σελίδα αυτή ο υπάλληλος εισάγει μέθοδο πληρωμής και μόλις πραγματοποιηθεί με επιτυχία η ενοικίαση ο υπάλληλος μεταφέρεται πίσω στη σελίδα `employeeRent.jsp`, για να δει και τις υπόλοιπες διαθέσιμες κρατήσεις.

Σε επίπεδο εφαρμογής τα παραπάνω πραγματοποιούνται ως εξής: Όταν ο υπάλληλος εισάγει το ΑΦΜ του αυτό περνάει ως παράμετρος στο servlet `IdentifyEmployee.java`, στο οποίο γίνεται query στη σχέση `employee`, για να διαπιστωθεί εάν ο υπάλληλος είναι καταχωρημένος στη βάση. Στη συνέχεια, γίνονται τα κατάλληλα queries στις σχέσεις `reserves`, `hotel_room` και `customer`, ώστε τα δεδομένα για τις διαθέσιμες κρατήσεις στο ξενοδοχείο εργασίας του υπαλλήλου να περαστούν στη σελίδα `EmployeeRent.jsp`. Αφού ο υπάλληλος επιλέξει τη κράτηση του πελάτη, ο οποίος πραγματοποιεί το check-in και εισάγει μέθοδο πληρωμής στη σελίδα `implementPayment.jsp`, τα δεδομένα της ενοικίασης (`room`, `hotel`, `price`, `irs_employee`, `irs_customer`, `start_date`, `finish_date`, `payment_amount`, `payment_method`) στέλνονται στο servlet `RentServlet.java`. Στο servlet αυτό γίνεται η κατάλληλη εισαγωγή στη σχέση `rents` και στη συνέχεια ο έλεγχος μεταφέρεται στη σελίδα `IdentifyEmployee.jsp`.

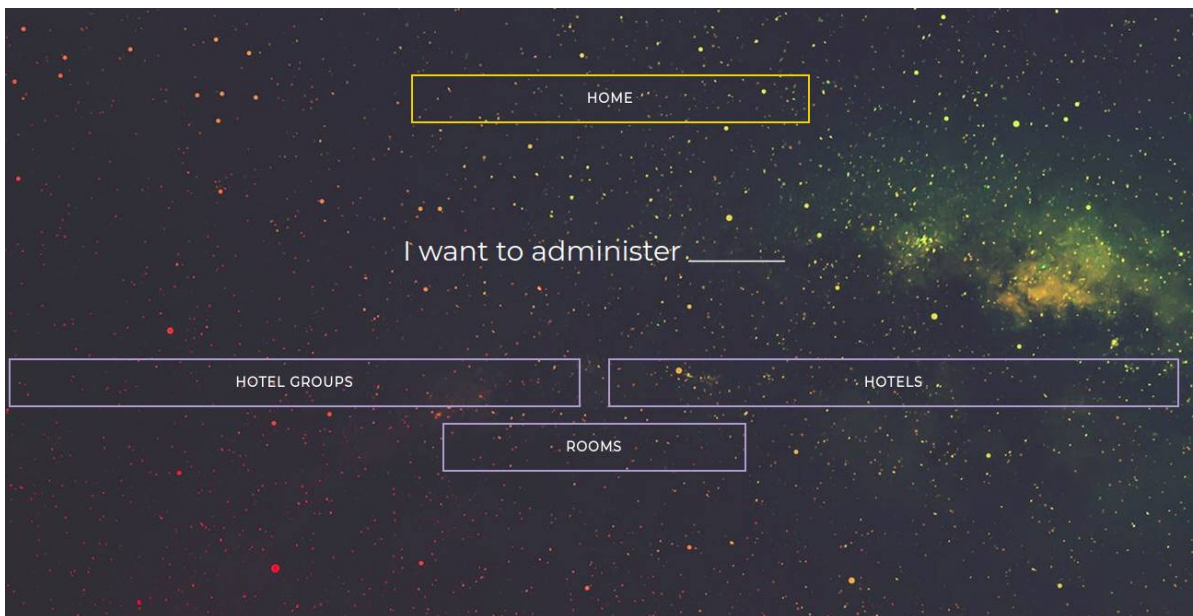
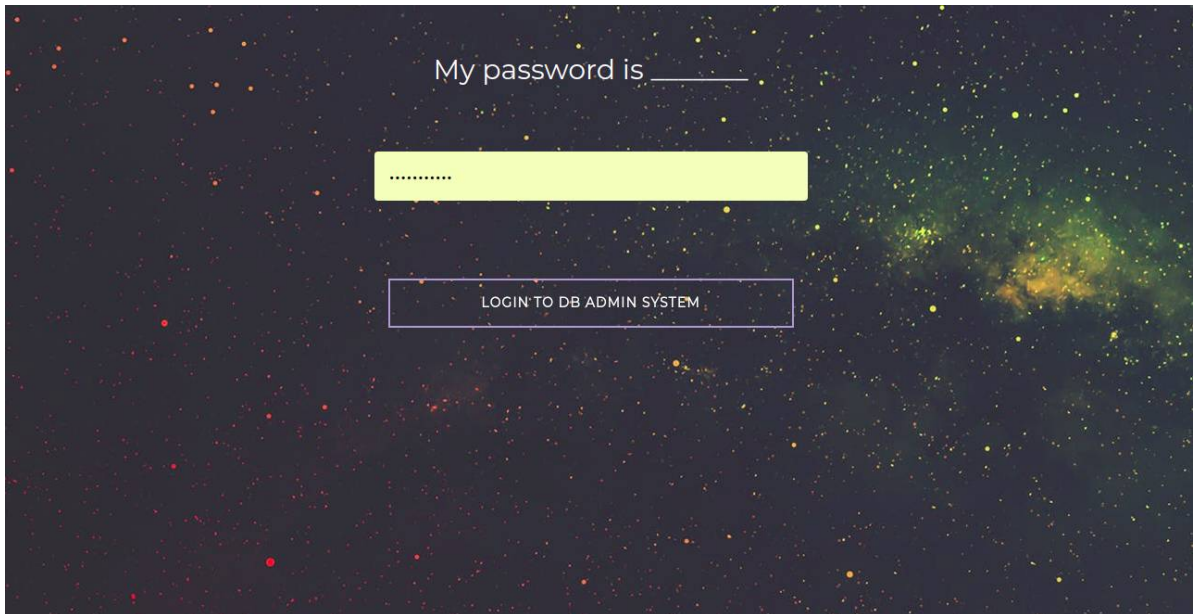
3. Διαδρομή διαχειριστή βάσης

Ο διαχειριστής της βάσης στη σελίδα `welcome.jsp` επιλέγει το κουμπί `Database Administration System` και μεταφέρεται στη σελίδα `protectedLogin.jsp`.

Εκεί εισάγει τον κωδικό του, ο οποίος στη συνέχεια μεταφέρεται ως παράμετρος στο servlet `Checker.java` και ελέγχεται η ορθότητα του. Αν είναι έγκυρος, ο διαχειριστής μεταφέρεται στη σελίδα `Administratorpage`, όπου επιλέγει ποιο πίνακα επιθυμεί να επεξεργαστεί. Οι επιλογές, που του παρέχονται είναι: `hotels`, `hotel_groups` και `rooms`.

- **Hotel groups.**

Εάν επιλέξει `hotel groups` μεταφέρεται στη σελίδα `listadminhotelgroups.jsp`, όπου του εμφανίζεται ένας πίνακας με όλες τις εγγραφές της σχέσης `hotel_group`. Αυτό γίνεται εφικτό μέσω του servlet `Hotelgroupscontroller.java` δίνοντας του ως παράμετρο `action=listhotelgroups`. Δίπλα σε κάθε εγγραφή, υπάρχει η επιλογή να τη διαγράψει ή να την επεξεργαστεί. Επιπλέον, υπάρχει στο πάνω μέρος της οθόνης κουμπί, το οποίο επιλέγει για να προσθέσει νέα εγγραφή. Εάν επιλεγεί να προστεθεί η να ανανεωθεί εγγραφή, ο διαχειριστής μεταφέρεται στη σελίδα `adminhotelgroups.jsp`, όπου συμπληρώνει τα κατάλληλα πεδία και στη συνέχεια οι τιμές τους περνάνε ως παράμετροι στο `hotelgroupscontroller` και γίνεται αντίστοιχα προσθήκη ή ανανέωση στη σχέση `hotel_groups`. Ομοίως, εάν επιλεγεί διαγραφή πεδίου, το `hotel_group_id` περνάει ως παράμετρος στο servlet μαζί με `action=delete`, όπου και πραγματοποιείται η διαγραφή του πεδίου.



Επιπλέον, δίπλα σε κάθε εγγραφή δίνεται επιλογή list_emails και list_phones, όπου μέσω του servlet Hotelgroupmailscontroller και Hotelgroupphonescontroller αντίστοιχα και δίνοντας τους τις κατάλληλες παραμέτρους και actions ο διαχειριστής μπορεί να δει σε λίστα τις εγγραφές στους πίνακες hotel_group_mails και hotel_group_phones, να τις διαγράψει ή να προσθέσει νέες.

- Hotels.

Εάν επιλέξει hotels ακολουθείται παρόμοια διαδικασία. Ο διαχειριστής μεταφέρεται στη σελίδα listadminhotels.jsp, όπου του εμφανίζεται ένας πίνακας με όλες τις εγγραφές της σχέσης hotel. Αυτό γίνεται εφικτό μέσω του servlet Hotelscontroller.java δίνοντας του ως παράμετρο action=listhotelgroup. Δίπλα σε κάθε εγγραφή, υπάρχει η επιλογή να τη διαγράψει ή να την επεξεργαστεί. Επιπλέον, υπάρχει στο πάνω μέρος της οθόνης κουμπί, το οποίο επιλέγει για να προσθέσει νέα εγγραφή. Εάν επιλεγεί να προστεθεί ή να ανανεωθεί εγγραφή, ο διαχειριστής μεταφέρεται στη σελίδα adminhotels.jsp, όπου συμπληρώνει τα κατάλληλα πεδία και στη συνέχεια οι τιμές τους περνάνε ως παράμετροι στο hotelscontroller και γίνεται αντίστοιχα προσθήκη ή ανανέωση στη σχέση hotels. Ομοίως, εάν επιλεγεί διαγραφή πεδίου, το hotel_id και το mail/phone περνάει ως παράμετρος στο servlet μαζί με action=delete, όπου και πραγματοποιείται η διαγραφή του πεδίου.

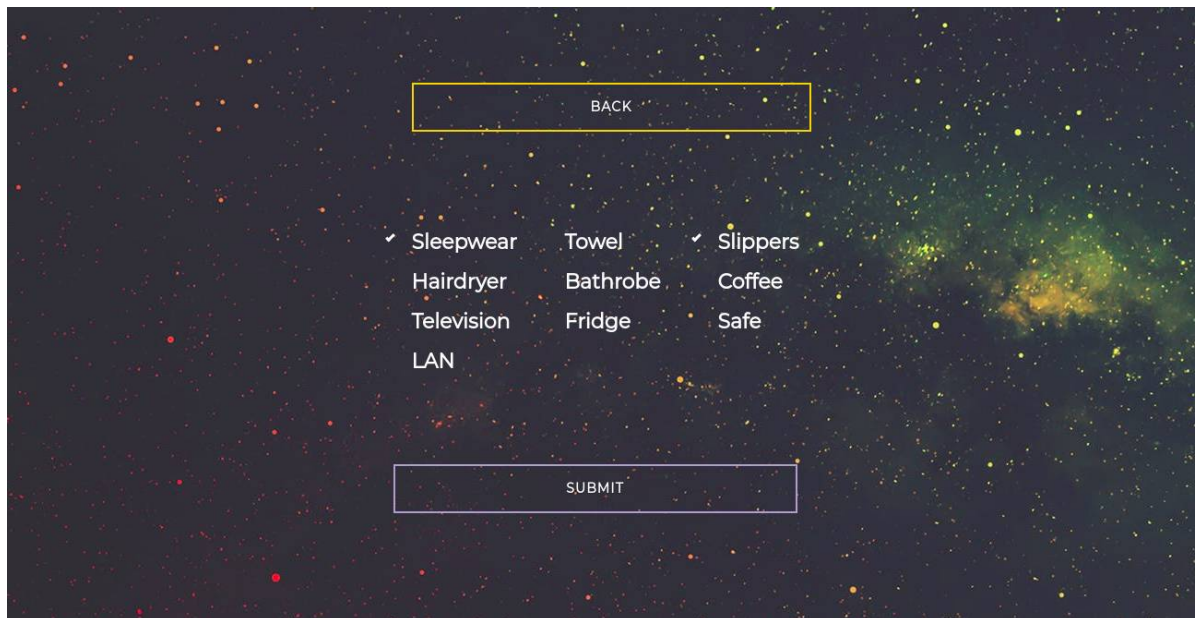
ADD NEW HOTEL												
HOTEL ID	HOTEL GROUP ID	NAME	STARS	NUMBER OF ROOMS	STREET	NUMBER	POSTAL	CITY	ACTION			
11	4	Thanos rooms	2	9	Tatoiou	13	14433	Thessaloniki	Update	Delete	Edit mails	Edit phones
12	3	Asterias	3	10	Mavilh	12	17418	Ampelokipoi	Update	Delete	Edit mails	Edit phones
13	3	Aphrodite	4	10	Mallias	13	16118	Maroussi	Update	Delete	Edit mails	Edit phones
14	3	Endless beach	4	10	Kritiou	24	15123	Vurwnas	Update	Delete	Edit mails	Edit phones
15	3	Fairytale	4	10	Patisiwn	28	14748	Euvoia	Update	Delete	Edit mails	Edit phones
16	4	Ta dwmatia ths	5	10	Kodriktonos	32	14998	Sudagma	Update	Delete	Edit mails	Edit phones

Επιπλέον, δίπλα σε κάθε εγγραφή δίνεται επιλογή edit_emails και edit_phones, όπου μέσω του servlet Hotelmailscontroller και Hotelphonescontroller αντίστοιχα και δίνοντας τους τις κατάλληλες παραμέτρους και actions ο διαχειριστής μπορεί να δει σε λίστα τις εγγραφές στους πίνακες hotel_mails και hotel_phones, να τις διαγράψει ή να προσθέσει νέες.

- Rooms.

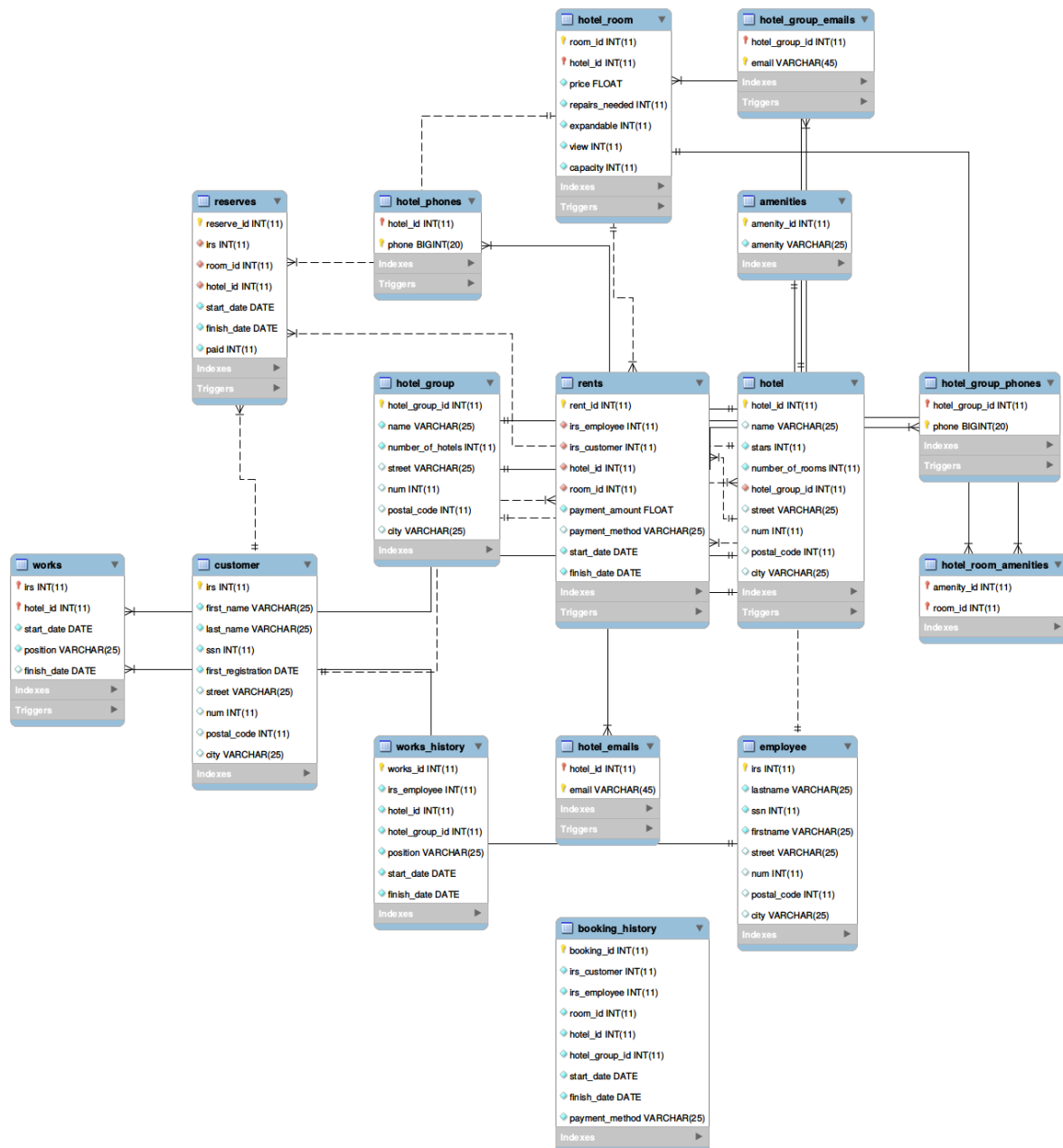
Εάν επιλέξει rooms, ο διαχειριστής μεταφέρεται στη σελίδα listadminrooms.jsp, όπου του εμφανίζεται ένας πίνακας με όλες τις εγγραφές της σχέσης hotel. Αυτό γίνεται εφικτό μέσω του servlet Hotelscontroller.java δίνοντας του ως παράμετρο action=listhotelgroups. Δίπλα σε κάθε εγγραφή, υπάρχει η επιλογή να τη διαγράψει ή να την επεξεργαστεί. Επιπλέον, υπάρχει στο πάνω μέρος της οθόνης κουμπί, το οποίο επιλέγει για να προσθέσει νέα εγγραφή. Εάν επιλεγεί να προστεθεί η να ανανεωθεί εγγραφή, ο διαχειριστής μεταφέρεται στη σελίδα adminhotels.jsp, όπου συμπληρώνει τα κατάλληλα πεδία και στη συνέχεια οι τιμές τους περνάνε ως παράμετροι στο hotelscontroller και γίνεται αντίστοιχα προσθήκη ή ανανέωση στη σχέση hotels. Ομοίως, εάν επιλεγεί διαγραφή πεδίου, το hotel_id και το mail/η phone περνάται ως παράμετρος στο servlet μαζί με action=delete, όπου και πραγματοποιείται η διαγραφή του πεδίου.

Επιπλέον, δίπλα σε κάθε εγγραφή υπάρχει η επιλογή edit_amenities, η οποία μεταφέρει τον διαχειριστή στη σελίδα listroomamenities, μέσω του servlet Roomamenitiescontroller.java και δίνοντας του ως παράμετρο το room_id και action=listamenities. Από τη σελίδα listroomamenities ο διαχειριστής επιλέγωντας add amenity, μεταβαίνει στη σελίδα adminroomamenities, όπου επιλέγει από μια λίστα με amenities ποιες θα προσθέσει.



2 Σχεσιακό Διάγραμμα

Το σχεσιακό διάγραμμα της εφαρμογής μας φαίνεται στο ακόλουθο σχήμα:



Σημείωση: Το relational model της βάσης μας κατασκευάστηκε από το tables-creator με Reverse Engineering μέσω του εργαλείου mysql-workbench το οποίο εγκαταστήσαμε σε περιβάλλον Ubuntu Linux. Συγκεκριμένα, συνδέουμε τη βάση που κατασκευάσαμε μέσω του tables-creator στο mysql-workbench και στη συνέχεια μέσω των επιλογών Database > Reverse Engineering παράγεται το ζητούμενο διάγραμμα από το mysql-workbench.

3 Αναλυτική επεξήγηση βάσης και επιλογών σχεδίασης

3.1 Customer

Ο πίνακας customer έχει τις εξής στήλες:

- irs: int **primary key**. Όπως και στους employee, το irs προσδιορίζει μοναδικά κάθε άνθρωπο άρα και κάθε customer. Συνεπώς, επιλέγεται ως primary key.
- first_name: varchar(25).
- last_name: varchar(25).

- ssn: int.
- first_registration: date.
- street: varchar(25).
- num: int.
- postal_code: int.
- city: varchar(25).

3.2 Hotel Groups

Ο πίνακας hotel_group έχει τις ακόλουθες στήλες:

- hotel_group_id: int **primary key**. Κάθε hotel group προσδιορίζεται μοναδικά από το id του. Προκειμένου να εξασφαλίσουμε τη μοναδικότητα του id ως primary key το βάζουμε **autoincrement** πεδίο στη βάση.
- name: varchar(25). Επιλέγουμε να **προσθέσουμε** τη στήλη name προκειμένου να μην απαιτούνται γνώσεις/πρόσβαση στη βάση προκειμένου ο χρήστης να επιλέξει την αλυσίδα ξενοδοχείων που τον ενδιαφέρει για τις διακοπές του.
- number_of_hotels: int. Η τιμή του πεδίου αυτού **μεταβάλλεται από triggers**. Παραπέμπουμε στην αντίστοιχη ενότητα για περαιτέρω πληροφορίες.
- street: varchar(25).
- num: int.
- postal_code: int.
- city: varchar(25).

3.3 Hotel Group Emails

Επειδή κάθε hotel group μπορεί να έχει πολλά emails (πλειότιμο) ορίζουμε πίνακα hotel_group_emails με **complex primary key** που αποτελείται από το email και το hotel_group_id. Ο πίνακας έχει τις ακόλουθες στήλες:

- hotel_id: int **foreign key** από τον πίνακα hotel_group.
- email: varchar(255). Η τιμή του πεδίου αυτού **ελέγχεται από triggers**.

3.4 Hotel Group Phones

Επειδή κάθε hotel group μπορεί να έχει πολλά phones (πλειότιμο) ορίζουμε πίνακα hotel_group_phones με **complex primary key** που αποτελείται από το phone και το hotel_group_id. Ο πίνακας έχει τις ακόλουθες στήλες:

- hotel_id: int **foreign key** από τον πίνακα hotel_group.
- phone: bigint. Η τιμή του πεδίου αυτού **ελέγχεται από triggers**.

3.5 Hotels

Ο πίνακας hotel έχει τις ακόλουθες στήλες:

- hotel_id: int primary key. Κάθε hotel προσδιορίζεται μοναδικά από το id του. Προκειμένου να εξασφαλίσουμε τη μοναδικότητα του id ως primary key το βάζουμε **autoincrement** πεδίο στη βάση.
- name: varchar(25).
- stars: int. Το εύρος τιμών ([1, 5]) αυτού του πεδίου **ελέγχεται από triggers**. Παραπέμπουμε στη σχετική ενότητα για περισσότερες πληροφορίες.
- number_of_rooms int: Η τιμή του πεδίου αυτού **μεταβάλλεται από triggers**. Παραπέμπουμε στην αντίστοιχη ενότητα για περαιτέρω πληροφορίες.
- hotel_group_id: int **foreign key**. Κάθε hotel που εισάγεται στη βάση πρέπει να ανήκει σε κάποιο hotel group. Συνεπώς, το πεδίο hotel_group_id ορίζεται ως foreign key από τον πίνακα hotel_group.
- street: varchar(25).
- num: int.
- postal_code: int.
- city: varchar(25).

3.6 Hotel Emails

Επειδή κάθε hotel μπορεί να έχει πολλά emails (πλειότιμο) ορίζουμε πίνακα hotel_emails με **complex primary key** που αποτελείται από το email και το hotel_id. Ο πίνακας έχει τις ακόλουθες στήλες:

- hotel_id: int **foreign key** από τον πίνακα hotel.
- email: varchar(255). Η τιμή του πεδίου αυτού **ελέγχεται από triggers**.

3.7 Hotel Phones

Επειδή κάθε hotel μπορεί να έχει πολλά phones (πλειότιμο) ορίζουμε πίνακα hotel_phones με **complex primary key** που αποτελείται από το phone και το hotel_id. Ο πίνακας έχει τις ακόλουθες στήλες:

- hotel_id: int **foreign key** από τον πίνακα hotel.
- phone bigint. Η τιμή του πεδίου αυτού **ελέγχεται από triggers**.

3.8 Πίνακας employee

Ο πίνακας employee έχει τις εξής στήλες:

- irs: int **primary key**. Όπως φαίνεται από το ER Model κάθε employee προσδιορίζεται μοναδικά από το irs του και συνεπώς επιλέγεται ως primary key. Το irs θεωρούμε ότι είναι ακέραιος αριθμός.
- lastname: varchar(25).
- firstname: varchar(25).
- ssn: int.
- street: varchar(25).
- num: int.
- postal_code: int.
- city: varchar(25).

3.9 Πίνακας works

Επιλέγουμε να κάνουμε τη σχέση works πίνακα επειδή ένας employee μπορεί να δουλεύει σε πολλά ξενοδοχεία και ένα ξενοδοχείο μπορεί να έχει πολλούς υπαλλήλους. Ο πίνακας works έχει **complex primary key** που αποτελείται από το **irs** που προσδιορίζει τον employee και το **hotel_id** που προσδιορίζει το ξενοδοχείο. Οι στήλες του πίνακα είναι:

- **irs**: int **foreign key** από τον πίνακα employee.
- **hotel_id**: int **foreign key** από τον πίνακα hotel.
- **start_date**: date
- **finish_date**: date
- **position**: varchar(25). Για τον πίνακα works η στήλη **position** ελέγχεται από triggers. Παραπέμπουμε στην αντίστοιχη ενότητα για περισσότερες πληροφορίες.

3.10 Hotel rooms

Ο πίνακας hotel_room έχει **complex primary key** που αποτελείται από τις στήλες **room_id** και **hotel_id**. Ο λόγος που επιλέγεται να μπει και το **hotel_id** στο primary key είναι νοηματικός: Το **hotel_room** είναι weak entity στο ER model και συνεπώς κάθε του entry πρέπει να προσδιορίζεται από τη σχέση που έχει με άλλο πίνακα στη βάση.

- **room_id**: int. Δεδομένου ότι το πεδίο αυτό είναι μέρος του primary key και το άλλο μέρος του primary key μπορεί να είναι κοινό για διαφορετικά entries, το πεδίο αυτό ορίζεται **autoincrement**.
- **hotel_id**: int **foreign key** από τον πίνακα hotel.
- **price**: double.
- **repairs_needed**: int. Το εύρος τιμών [0,1] του πεδίου αυτού ελέγχεται από triggers. Παραπέμπουμε στη σχετική ενότητα για περισσότερες πληροφορίες.
- **expandable**: int. Το εύρος τιμών [0,1] του πεδίου αυτού ελέγχεται από triggers. Παραπέμπουμε στη σχετική ενότητα για περισσότερες πληροφορίες.
- **view**: int. Το εύρος τιμών [0,1] του πεδίου αυτού ελέγχεται από triggers. Παραπέμπουμε στη σχετική ενότητα για περισσότερες πληροφορίες.
- **capacity**: int. Το εύρος τιμών [1,5] του πεδίου αυτού ελέγχεται από triggers. Παραπέμπουμε στη σχετική ενότητα για περισσότερες πληροφορίες.

3.11 Πίνακας reserves

Ο πίνακας reserves περιέχει τις κρατήσεις για τα ξενοδοχεία που υπάρχουν ακόμη στη βάση. Περιέχει τις ακόλουθες στήλες:

- **reserve_id**: int **primary key autoincrement**. Εξασφαλίζει τη μοναδικότητα κάθε κράτησης.
- **irs**: int **foreign key** από τον πίνακα customer.
- **room_id**: int **foreign key** από τον πίνακα hotel_room.
- **hotel_id**: int **foreign key** από τον πίνακα hotel.
- **hotel_group_id**: int **foreign key** από τον πίνακα hotel_group.
- **start_date**: date.
- **finish_date**: date.

3.12 Πίνακας rents

Ο πίνακας rents περιέχει τις κρατήσεις για τα ξενοδοχεία που υπάρχουν ακόμη στη βάση. Περιέχει τις ακόλουθες στήλες:

- rent_id: int **primary key autoincrement**. Εξασφαλίζει τη μοναδικότητα κάθε ενοικίασης.
- irs_employee: int **foreign key** από τον πίνακα employee.
- irs_customer: int **foreign key** από τον πίνακα customer.
- hotel_id: int **foreign key** από τον πίνακα hotel.
- room_id: int **foreign key** από τον πίνακα hotel_room.
- payment_amount: float.
- payment_method: varchar(25).
- start_date: date.
- finish_date: date.

3.13 Amenities

Θεωρήσαμε ότι η καλύτερη σχεδιαστική επιλογή είναι να υπάρχει ένα σύνολο από πιθανά amenities ώστε να υπάρχει ομοιομορφία στην παρεχόμενη πληροφορία των ξενοδοχείων. Έτσι, για τα amenities ορίζουμε δύο πίνακες: amenities, hotel_room_amenities.

3.13.1 Πίνακας amenities

Ο πίνακας αυτός περιέχει τις ακόλουθες στήλες:

- amenity_id: int **primary key autoincrement**. Είναι το αναγνωριστικό του συγκεκριμένου amenity.
- amenity: varchar(25).

3.13.2 Πίνακας hotel_room_amenities

Ο πίνακας αυτός περιέχει την πληροφορία των amenities που έχει κάθε δωμάτιο. Συγκεκριμένα, έχει ένα **complex primary key** που αποτελείται από το room_id και το amenity_id. Οι στήλες του πίνακα είναι τα μέρη του complex primary key του και είναι και οι δύο τύπου int.

3.14 Επιπλέον πίνακες

Επειδή πιθανώς χρειαζόμαστε να κρατάμε ένα ιστορικό των ανθρώπων που δούλεψαν στα ξενοδοχεία καθώς και όλες τις ενοικιάσεις-κρατήσεις που έγιναν ακόμα και αν δεν υπάρχει πλέον το δωμάτιο ορίζουμε τους ακόλουθους δύο πίνακες που οι εισαγωγές δεδομένων σε αυτούς γίνονται από triggers:

3.14.1 Πίνακας works_history

Οι στήλες του πίνακα είναι:

- works_id: int **primary key autoincrement**. Εξασφαλίζει τη μοναδικότητα κάθε γραμμής.
- irs_employee: int
- hotel_id: int
- hotel_group_id: int
- position: varchar(25)
- start_date: date.
- finish_date: date.

3.14.2 Πίνακας booking_history

Έχει τις ακόλουθες στήλες:

- booking_id: int **primary key autoincrement**. Εξασφαλίζει τη μοναδικότητα κάθε γραμμής.
- irs_customer: int.
- irs_employee: int.
- room_id: int.
- hotel_id: int.
- hotel_group_id: int.
- start_date: date.
- finish_date: date.
- payment_method: varchar(25).

4 Triggers

Στην ενότητα αυτοί αναλύουμε όλους τους triggers που έχουμε ορίσει στη βάση δεδομένων μας και βρίσκονται στο αρχείο ./database-setup/database-triggers.sql .

Οι triggers αυτοί χωρίζονται στις εξής κατηγορίες:

1. Triggers ελέγχου τιμών
2. Triggers update τιμών
3. Triggers ελέγχου νομιμότητας ενεργειών
4. Triggers διατήρησης ιστορικού

4.1 Triggers ελέγχου τιμών

Οι triggers αυτοί ουσιαστικά υλοποιούν τη συνάρτηση check που υπάρχει σε άλλα mysql συστήματα ώστε να διασφαλίσουν ότι μπαίνουν νοηματικά ορθές τιμές. Έχουμε τους ακόλουθους:

1. check_for_hotel_group_email_validity1: Ο trigger αυτός τεστάρει το email που πάει να εισαχθεί σε ένα hotel group απέναντι σε ένα regex expression που ελέγχει την ορθότητα του. Αν δεν περάσει το test, πετάει ένα exception.
2. check_for_hotel_group_email_validity2: Ο trigger αυτός τεστάρει το email που πάει να γίνει update σε ένα hotel group απέναντι σε ένα regex expression που ελέγχει την ορθότητα του. Αν δεν περάσει το test, πετάει ένα exception.
3. check_for_hotel_email_validity1: Ο trigger αυτός τεστάρει το email που πάει να εισαχθεί σε ένα hotel απέναντι σε ένα regex expression που ελέγχει την ορθότητα του. Αν δεν περάσει το test, πετάει ένα exception.
4. check_for_hotel_email_validity2: Ο trigger αυτός τεστάρει το email που πάει να γίνει update ένα hotel απέναντι σε ένα regex expression που ελέγχει την ορθότητα του. Αν δεν περάσει το test, πετάει ένα exception.
5. check_for_hotel_group_phone_validity1: Ο trigger αυτός ελέγχει αν το phone που πάει να εισαχθεί σε ένα hotel_group είναι έγκυρος αριθμός (δηλαδή, αν έχει έγκυρο αριθμό ψηφίων).

6. `check_for_hotel_group_phone_validity2`: Ο trigger αυτός ελέγχει αν το phone που πάει να γίνει update σε ένα hotel_group είναι έγκυρος αριθμός (δηλαδή, αν έχει έγκυρο αριθμό ψηφίων).
7. `check_for_hotel_phone_validity2`: Ο trigger αυτός ελέγχει αν το phone που πάει να γίνει εισαχθεί σε ένα hotel είναι έγκυρος αριθμός (δηλαδή, αν έχει έγκυρο αριθμό ψηφίων).
8. `check_for_hotel_phone_validity2`: Ο trigger αυτός ελέγχει αν το phone που πάει να γίνει update σε ένα hotel_group είναι έγκυρος αριθμός (δηλαδή, αν έχει έγκυρο αριθμό ψηφίων).
9. `check_for_stars_validity1`: Ο trigger αυτός αποτρέπει την εισαγωγή ενός ξενοδοχείου αν ο αριθμός των αστεριών του είναι εκτός από τα επιτρεπτά όρια του διαστήματος [1, 5].
10. `check_for_stars_validity2`: Ο trigger αυτός αποτρέπει την αλλαγή των αστεριών ενός ξενοδοχείου αν ο νέος αριθμός είναι εκτός από τα επιτρεπτά όρια του διαστήματος [1, 5].
11. `check_for_hotel_room_validity1`: Ο trigger αυτός ελέγχει ότι το δωμάτιο που εισάγεται έχει capacity εντός του διαστήματος [1, 5] και τις μεταβλητές view, expandable, repairs_needed εντός του διαστήματος [0, 1]. Σε διαφορετική περίπτωση, πετάει ένα sql exception.
12. `check_for_hotel_room_validity2`: Ο trigger αυτός ελέγχει ότι το δωμάτιο του οποίου ανανεώνονται οι πληροφορίες (update) θα έχει μετά την ανανέωση capacity εντός του διαστήματος [1, 5] και τις μεταβλητές view, expandable, repairs_needed εντός του διαστήματος [0, 1]. Σε διαφορετική περίπτωση, πετάει ένα sql exception.
13. `check_for_reserves_validity1`: Ο trigger αυτός ελέγχει ότι το νέο entry που εισάγεται στον πίνακα reserves έχει ορθές νοηματικά τιμές, δηλαδή boolean paid και ημέρα ολοκλήρωσης της κράτησης μεταγενέστερη χρονικά από την ημέρα έναρξης της κράτησης.
14. `check_for_reserves_validity2`: Ο trigger αυτός ελέγχει ότι το entry που γίνεται update στον πίνακα reserves έχει ορθές νοηματικά τιμές, δηλαδή boolean paid και ημέρα ολοκλήρωσης της κράτησης μεταγενέστερη χρονικά από την ημέρα έναρξης της κράτησης.
15. `check_for_rent_validity1`: Ο trigger αυτός ελέγχει ότι το νέο entry που εισάγεται στον πίνακα rents έχει ορθές νοηματικά τιμές, δηλαδή μη αρνητικό ποσό πληρωμής και ημέρα ολοκλήρωσης της ενοικίασης μεταγενέστερη χρονικά από την ημέρα έναρξης της ενοικίασης.
16. `check_for_rent_validity2`: Ο trigger αυτός ελέγχει ότι το entry που γίνεται update στον πίνακα rents έχει ορθές νοηματικά τιμές, δηλαδή μη αρνητικό ποσό πληρωμής και ημέρα ολοκλήρωσης της ενοικίασης μεταγενέστερη χρονικά από την ημέρα έναρξης της ενοικίασης.
17. `check_for_works_validity1`: Ο trigger αυτός ελέγχει ότι ένας νέος υπάλληλος σε ένα ξενοδοχείο θα έχει ημέρα ολοκλήρωσης συμβολαίου μεταγενέστερη χρονικά από την ημέρα έναρξης συμβολαίου.
18. `check_for_works_validity2`: Ο trigger αυτός ελέγχει ότι υπάλληλος που γίνεται update σε ένα ξενοδοχείο θα έχει ημέρα ολοκλήρωσης συμβολαίου μεταγενέστερη χρονικά από την ημέρα έναρξης συμβολαίου.

4.2 Triggers update τιμών

Σε αυτή την ενότητα βάζουμε όλους τους triggers που πυροδοτούνται για να ανανεώσουν κάποια πληροφορία σε έναν πίνακα όταν σε κάποιον άλλο πίνακα συμβαίνει μια αλλαγή (insert, delete or update).

Συγκεκριμένα, έχουμε τους ακόλουθους triggers:

1. `check_norooms1`: Ο trigger αυτός πυροδοτείται όταν εισάγεται ένα καινούργιο δωμάτιο ώστε ο αριθμός των δωματίων του αντίστοιχου ξενοδοχείου να αυξηθεί.
2. `check_norooms2`: Ο trigger αυτός πυροδοτείται όταν διαγράφεται ένα δωμάτιο ώστε ο αριθμός των δωματίων του αντίστοιχου ξενοδοχείου να μειωθεί.

3. `check_nohotels1`: Ο trigger αυτός πυροδοτείται όταν εισάγεται ένα καινούργιο ξενοδοχείο ώστε ο αριθμός των ξενοδοχείων του αντίστοιχου hotel group να αυξηθεί.
4. `check_nohotels2`: Ο trigger αυτός πυροδοτείται όταν διαγράφεται ένα ξενοδοχείο ώστε ο αριθμός των ξενοδοχείων του αντίστοιχου hotel group να μειωθεί.
5. `check_updateroom`: Ο trigger αυτός αναλαμβάνει να αλλάξει τους αριθμούς των δωματίων δύο ξενοδοχείων όταν ένα δωμάτιο μεταφέρεται από το ένα ξενοδοχείο στο άλλο.
6. `check_updatehotel`: Ο trigger αυτός αναλαμβάνει να αλλάξει τους αριθμούς ξενοδοχείων δύο ξενοδοχειακών αλυσίδων όταν ένα ξενοδοχείο μεταφέρεται από τη μία αλυσίδα στην άλλη.
7. `check_payment`: Ο trigger αυτός αναλαμβάνει να θέσει το πεδίο `paid` του πίνακα `reserves` στην τιμή 1 όταν γίνεται ενοικίαση για τη συγκεκριμένη κράτηση.

4.3 Triggers ελέγχου νομιμότητας ενεργειών

1. `check_reservation`: Ο trigger αυτός πριν γίνει `insert` στον πίνακα `reserves` ελέγχει ότι η κράτηση που πάει να γίνει δεν έχει επικάλυψη με κάποια άλλη κράτηση για το συγκεκριμένο δωμάτιο. Αν υπάρχει επικάλυψη, πετάει `exception` και αποτρέπει την εισαγωγή.
2. `check_for_manager_on_update`: Ο trigger αυτός κάθε φορά που πάει να γίνει `update` το `position` ενός εργαζομένου ελέγχει αν μετά το `update` θα υπάρχει τουλάχιστον ένας `manager` στο συγκεκριμένο ξενοδοχείο. Σε διαφορετική περίπτωση, πετάει `exception` και εμποδίζει το `update`.
3. `check_for_manager_on_delete`: Ο trigger αυτός κάθε φορά που πάει να γίνει `delete` ένας εργαζόμενος ελέγχει αν μετά το `delete` θα υπάρχει τουλάχιστον ένας `manager` στο συγκεκριμένο ξενοδοχείο. Σε διαφορετική περίπτωση, πετάει `exception` και εμποδίζει το `delete`.

4.4 Triggers διατήρησης ιστορικού

1. `create_workhistory_entry1`: Ο trigger αυτός για κάθε εισαγωγή στον πίνακα `works` βάζει τις κατάλληλες πληροφορίες στον πίνακα `works_history` προκειμένου να υπάρχει ένα ιστορικό υπαλλήλων στα διάφορα `positions` για όλα τα ξενοδοχεία και τις ξενοδοχειακές αλυσίδες.
2. `create_workhistory_entry2`: Ο trigger αυτός για κάθε `update` στον πίνακα `works` κάνει `insert` τις κατάλληλες πληροφορίες στον πίνακα `works_history` προκειμένου να υπάρχει ένα ιστορικό υπαλλήλων στα διάφορα `positions` για όλα τα ξενοδοχεία και τις ξενοδοχειακές αλυσίδες.
3. `create_bookinghistory_entry1`: Ο trigger αυτός για κάθε εισαγωγή στον πίνακα `rents` βάζει τις κατάλληλες πληροφορίες στον πίνακα `booking_history` προκειμένου να υπάρχει ένα ιστορικό όλων των κρατήσεων/ενοικιάσεων που έγιναν μέσω της εφαρμογής ακόμα και αν το δωμάτιο, το ξενοδοχείο ή η ξενοδοχειακή αλυσίδα δεν είναι πλέον διαθέσιμα στη βάση.
4. `create_bookinghistory_entry2`: Ο trigger αυτός για κάθε `update` στον πίνακα `rents` κάνει το αντίστοιχο `update` στον πίνακα `booking_history` προκειμένου να υπάρχει ένα ιστορικό όλων των κρατήσεων/ενοικιάσεων που έγιναν μέσω της εφαρμογής ακόμα και αν το δωμάτιο, το ξενοδοχείο ή η ξενοδοχειακή αλυσίδα δεν είναι πλέον διαθέσιμα στη βάση.

4.5 Κώδικας για τους triggers

Ο κώδικας για τους triggers που αναλύσαμε βρίσκεται στο αρχείο `./database-setup/database-triggers.sql` και φαίνεται παρακάτω:

```

1 use ntua_db;
2
3 delimiter |
4 create trigger check_norooms1 after insert on hotel_room
5   for each row
6   begin
7     update hotel set number_of_rooms = number_of_rooms+1 where hotel_id=new.hotel_id;

```



```

8   end;
9   |
10
11  create trigger check_norooms2 after delete on hotel_room
12    for each row
13    begin
14      update hotel set number_of_rooms = number_of_rooms-1 where hotel_id=old.hotel_id;
15    end;
16    |
17
18  create trigger check_nohotels1 after insert on hotel
19    for each row
20    begin
21      update hotel_group set number_of_hotels = number_of_hotels+1 where hotel_group_id=new.
22        hotel_group_id;
23    end;
24    |
25
26  create trigger check_nohotels2 after delete on hotel
27    for each row
28    begin
29      update hotel_group set number_of_hotels = number_of_hotels-1 where hotel_group_id=old.
30        hotel_group_id;
31    end;
32    |
33
34  create trigger check_updateroom after update on hotel_room
35    for each row
36    begin
37      if (new.hotel_id <> old.hotel_id) then
38        update hotel set number_of_rooms=number_of_rooms+1 where hotel_id=new.hotel_id;
39        update hotel set number_of_rooms=number_of_rooms-1 where hotel_id=old.hotel_id;
40      end if;
41    end;
42    |
43
44  create trigger check_updatehotel after update on hotel
45    for each row
46    begin
47      if (new.hotel_group_id <> old.hotel_group_id) then
48        update hotel_group set number_of_hotels=number_of_hotels+1 where hotel_group_id=new.
49          hotel_group_id;
50        update hotel_group set number_of_hotels=number_of_hotels-1 where hotel_group_id=old.
51          hotel_group_id;
52      end if;
53    end;
54    |
55
56  create trigger check_payment after insert on rents
57    for each row
58    begin
59      update reserves set paid=1 where (irs=new.irs_customer and room_id=new.room_id and
60        start_date=new.start_date and finish_date=new.finish_date);
61    end;
62    |
63
64  create trigger check_reservation before insert on reserves
65    for each row
66    begin
67      if (exists(select * from reserves where new.room_id=room_id and not(new.finish_date<=
68        start_date or new.start_date>=finish_date))) then
69        signal sqlstate '45000' set message_text = 'Sorry,another user booked the room';
70      end if;
71    end;

```

```

69 |
70 |
71 |
72 create trigger check_for_hotel_group_email_validity1 before insert on hotel_group_emails
73 for each row
74 begin
75     if (new.email not like '%@%.%' or new.email like '@%' or new.email like '%@@%') then
76         signal sqlstate '45000' set message_text = 'Wrong email';
77     end if;
78 end;
79 |
80 |
81 create trigger check_for_hotel_group_email_validity2 before update on hotel_group_emails
82 for each row
83 begin
84     if (new.email not like '%@%.%' or new.email like '@%' or new.email like '%@@%') then
85         signal sqlstate '45000' set message_text = 'Wrong email';
86     end if;
87 end;
88 |
89 |
90 |
91 create trigger check_for_hotel_email_validity1 before insert on hotel_emails
92 for each row
93 begin
94     if (new.email not like '%@%.%' or new.email like '@%' or new.email like '%@@%') then
95         signal sqlstate '45000' set message_text = 'Wrong email';
96     end if;
97 end;
98 |
99 |
100 create trigger check_for_hotel_email_validity2 before update on hotel_emails
101 for each row
102 begin
103     if (new.email not like '%@%.%' or new.email like '@%' or new.email like '%@@%') then
104         signal sqlstate '45000' set message_text = 'Wrong email';
105     end if;
106 end;
107 |
108 |
109 create trigger check_for_hotel_group_phone_validity1 before insert on hotel_group_phones
110 for each row
111 begin
112     if (new.phone < 10000000 or new.phone > 99999999999) then
113         signal sqlstate '45000' set message_text = 'Wrong phone';
114     end if;
115 end;
116 |
117 |
118 create trigger check_for_hotel_group_phone_validity2 before update on hotel_group_phones
119 for each row
120 begin
121     if (new.phone < 10000000 or new.phone > 99999999999) then
122         signal sqlstate '45000' set message_text = 'Wrong phone';
123     end if;
124 end;
125 |
126 |
127 create trigger check_for_hotel_phone_validity1 before insert on hotel_phones
128 for each row
129 begin
130     if (new.phone < 10000000 or new.phone > 99999999999) then
131         signal sqlstate '45000' set message_text = 'Wrong phone';
132     end if;
133 end;
134 |
135 |

```

```

136 create trigger check_for_hotel_phone_validity2 before update on hotel_phones
137 for each row
138 begin
139     if (new.phone<10000000 or new.phone>99999999999) then
140         signal sqlstate '45000' set message_text = 'Wrong phone';
141     end if;
142 end;
143 |
144
145
146 create trigger check_for_stars_validity1 before insert on hotel
147 for each row
148 begin
149     if (new.stars<=0 or new.stars>5) then
150         signal sqlstate '45000' set message_text = 'Wrong star argument';
151     end if;
152
153 end;
154 |
155
156 create trigger check_for_stars_validity2 before update on hotel
157 for each row
158 begin
159     if (new.stars<=0 or new.stars>5) then
160         signal sqlstate '45000' set message_text = 'Wrong star argument';
161     end if;
162 end;
163 |
164
165
166 create trigger check_for_hotel_room_validity1 before insert on hotel_room
167 for each row
168 begin
169     if (new.view<0 or new.view>1 or new.capacity>4 or new.capacity<0 or new.expandable<0 or
170         new.expandable>1 or new.price<0) then
171         signal sqlstate '45000' set message_text = 'Wrong hotel room arguments';
172     end if;
173 |
174
175 create trigger check_for_hotel_room_validity2 before update on hotel_room
176 for each row
177 begin
178     if (new.view<0 or new.view>1 or new.capacity>4 or new.capacity<0 or new.expandable<0 or
179         new.expandable>1 or new.price<0) then
180         signal sqlstate '45000' set message_text = 'Wrong hotel room arguments';
181     end if;
182 end;
183 |
184
185 create trigger check_for_reserves_validity1 before insert on reserves
186 for each row
187 begin
188     if (new.paid<0 or new.paid>1 or new.finish_date<=new.start_date) then
189         signal sqlstate '45000' set message_text = 'Wrong reservation';
190     end if;
191 end;
192 |
193
194 create trigger check_for_reserves_validity2 before update on reserves
195 for each row
196 begin
197     if (new.paid<0 or new.paid>1 or new.finish_date<=new.start_date) then
198         signal sqlstate '45000' set message_text = 'Wrong reservation';
199     end if;
200 end;

```

```

201 |
202 |
203 |
204 create trigger check_for_rent_validity1 before insert on rents
205     for each row
206     begin
207         if (new.payment_amount<0 or new.finish_date <= new.start_date ) then
208             signal sqlstate '45000' set message_text = 'Error in rents arguments';
209         end if;
210     end;
211 |
212 |
213 |
214 create trigger check_for_rent_validity2 before update on rents
215     for each row
216     begin
217         if (new.payment_amount<0 or new.finish_date <= new.start_date ) then
218             signal sqlstate '45000' set message_text = 'Error in rents arguments';
219         end if;
220     end;
221 |
222 |
223 |
224 create trigger check_for_works_validity1 before insert on works
225     for each row
226     begin
227         if (new.finish_date<new.start_date) then
228             signal sqlstate '45000' set message_text = 'Work dates error';
229         end if;
230     end;
231 |
232 |
233 |
234 create trigger check_for_works_validity2 before update on works
235     for each row
236     begin
237         if (new.finish_date<new.start_date) then
238             signal sqlstate '45000' set message_text = 'Work dates error';
239         end if;
240     end;
241 |
242 |
243 |
244 |
245 create trigger check_for_manager_on_update before update on works
246     for each row
247     begin
248         if (new.position <> old.position and old.position='manager') then
249             if ( not exists(select * from works where new.hotel_id=hotel_id and new.position='
manager'and new.finish_date>curdate())) then
250                 signal sqlstate '45000' set message_text = 'Every hotel must have a manager';
251             end if;
252         end if;
253     end;
254 |
255 |
256 |
257 create trigger check_for_manager_on_delete before delete on works
258     for each row
259     begin
260         if (old.position='manager') then
261             if ( (select count(position) from works where old.position='manager' and old.
finish_date>curdate())>1) then
262                 signal sqlstate '45000' set message_text = 'Every hotel must have a manager';
263             end if;
264         end if;
265     end;

```

```

266 |
267 |
268 create trigger create_workhistory_entry1 after insert on works
269 for each row
270 begin
271     insert into works_history(irs_employee,hotel_id,hotel_group_id,position,start_date,
272     finish_date) values (new.irs,new.hotel_id,(select hotel_group_id from hotel where
273     hotel_id=new.hotel_id),new.position,new.start_date,new.finish_date);
274 |
275 create trigger create_workhistory_entry2 after update on works
276 for each row
277 begin
278     insert into works_history(irs_employee,hotel_id,hotel_group_id,position,start_date,
279     finish_date) values (new.irs,new.hotel_id,(select hotel_group_id from hotel where
280     hotel_id=new.hotel_id),new.position,new.start_date,new.finish_date);
281 |
282 create trigger create_bookinghistory_entry1 after insert on rents
283 for each row
284 begin
285     insert into booking_history(irs_employee,irs_customer,room_id,hotel_id,hotel_group_id,
286     start_date,finish_date,payment_method) values (new.irs_employee,new.irs_customer,new.
287     room_id,new.hotel_id,(select hotel_group_id from hotel where hotel_id=new.hotel_id),new.
288     start_date,new.finish_date,new.payment_method);
289 |
290 create trigger create_bookinghistory_entry2 after update on rents
291 for each row
292 begin
293     update booking_history set irs_employee=new.irs_employee,irs_customer=new.irs_customer,
294     room_id=new.room_id,start_date=new.start_date,finish_date=new.finish_date,payment_method
295     =new.payment_method,hotel_id=new.hotel_id,hotel_group_id=(select hotel_group_id from
296     hotel where hotel_id=new.hotel_id);
297 |
298 delimiter ;

```

5 Περαιτέρω περιορισμοί

Στην ενότητα αυτή αναλύουμε περιορισμούς που δεν καλύπτονται από τους triggers και από τις σχεδιαστικές επιλογές που κάναμε. Η ανάλυση γίνεται τόσο σε επίπεδο εφαρμογής όσο και σε επίπεδο βάσης δεδομένων.

5.1 Επίπεδο βάσης

Εκτός από τους περιορισμούς που εισάγουν τα triggers έχουμε και τους ακόλουθους περιορισμούς:

- Όλα τα foreign key constraints έχουν οριστεί με on delete cascade property προκειμένου να επιτρέπονται τα deletes στους πίνακες "γονείς". Αυτό δεν σημαίνει ότι χάνεται χρήσιμη πληροφορία με την διαγραφή ενός ξενοδοχείου ή μιας ξενοδοχειακής αλυσίδας. Διατηρούμε μέσω triggers ενημερωμένους για όλο το ιστορικό τους πίνακες booking_history και works_history οι οποίοι δεν έχουν foreign key constraint ώστε να έχουμε όλο το ιστορικό των αλλαγών που αφορά employees και ενοικιάσεις σε δωμάτια.
- Όλα τα πεδία των πινάκων που αντιστοιχούν σε entities και σχέσεις του ER Model έχουν οριστεί NOT NULL καθώς αφενός θεωρούμε ότι είναι χρήσιμες πληροφορίες που διασφαλίζουν την πληρότητα και την ομοιομορφία της βάσης και αφετέρου έτσι είναι ευκολότερα διαχειρίσιμος και επεκτάσιμος ο κώδικας εφαρμογής.

- Η σχέση works έχει ως complex primary key το hotel_id μαζί το irs του υπαλλήλου. Αυτό σημαίνει ότι δεν μπορεί ένας υπάλληλος να δουλεύει στο ίδιο ξενοδοχείο σε δύο positions. Ο περιορισμός αυτός τέθηκε καθώς δεν προκύπτει σενάριο διπλοθεσίας από την εκφώνηση. Σε μια τέτοια εκδοχή, θα έπρεπε να οριστεί ένα autoincrement primary key στον πίνακα works. Στην παρούσα μορφή, ο περιορισμός αυτός όμως διασφαλίζει και ότι υπάρχει μοναδικό record στον πίνακα works για κάθε υπάλληλο ενώ στο σενάριο του autoincrement primary key δεν θα υπήρχε τέτοιος περιορισμός. Σημειώνουμε, ότι το να δουλεύει ένας υπάλληλος σε δύο διακριτά χρονικά διαστήματα στο ίδιο ξενοδοχείο προβλέπεται κανονικά από τη βάση με τη διαγραφή του υπαλλήλου και την επανατοποθέτηση του μετά την επαναπρόσληψη. Η πρώτη πρόσληψη δεν χάνεται καθώς κρατάμε το πλήρες ιστορικό με triggers στον πίνακα works_history.
- Ορίζουμε έναν πίνακα amenities στον οποίο μπαίνουν τα πιθανά amenities, αντί να αφήνουμε στον πίνακα hotel_room_amenities να υπάρχει μια στήλη varchar(25) για τα amenities. Αυτή η επιλογή γίνεται καθώς θεωρούμε πως είναι αυθαίρετο κάθε ξενοδοχείο να βάζει με δικό του τρόπο τα amenities (πχ: tv ή television ;) και αν επιτρέπαμε κάτι τέτοιο θα είχαμε νοηματικά όμοια αντικείμενα να είναι ασυσχέτιστα στη βάση δεδομένων μας.

5.2 Επίπεδο εφαρμογής

Σε επίπεδο εφαρμογής, έχουμε ορίσει τους ακόλουθους περιορισμούς:

1. Δεν μπορείς να φάξεις ξενοδοχείο ή να κάνεις κράτηση για ημερομηνίες που ανήκουν στο παρελθόν (μήνυμα λάθους: *'You can't book the past'*).
2. Δεν μπορείς να φάξεις ξενοδοχείο ή να κάνεις κράτηση για start date που είναι μεταγενέστερο χρονικά του finish date (μήνυμα λάθους: *'Dates collision'*).
3. Μπορείς να αναζητήσεις ξενοδοχείο αφήνοντας κενές τις ημερομηνίες έναρξης και λήξης ή συμπληρώνοντας και τις δύο. Αν συμπληρώσεις μόνο τη μία από τις δύο παίρνεις σχετικό μήνυμα λάθους για να αλλάξεις την επιλογή σου.
4. Δεν μπορείς να κάνεις κράτηση αν δεν έχεις ορίσει και το start και το finish date.
5. Οι επιλογές ως προς το payment method στο rent είναι MasterCard, PayPal, Cash καθώς θεωρούμε ότι βάζοντας selectbox στις επιλογές διασφαλίζεται μια ομοιομορφία στη βάση (αντί να αφήνεις για παράδειγμα ελεύθερο κείμενο για εισαγωγή στη φόρμα).

6 Εφαρμογή και SQL

Στην ανάπτυξη της εφαρμογής χρησιμοποιήθηκαν διάφορα sql statements για την επικοινωνία, την τροποποίηση και την ανάκτηση δεδομένων από την βάση. Στην ενότητα αυτή, εξηγούμε τον τρόπο λειτουργίας ορισμένων λειτουργιών της εφαρμογής από την σκοπιά της sql.

6.1 Selections

Στην υποενότητα αυτή παραθέτουμε μερικά από τα selection queries που γίνονται μέσω της εφαρμογής μας.

1. Ιδιότητα city από τις εγγραφές της σχέσης hotel, καταργώντας τις διπλές τιμές

```
1 select distinct cities from hotel
```

2. Όλες οι εγγραφές στη σχέση hotel_mails, στις οποίες η ιδιότητα hotel_id έχει την τιμή 1

```
1 select * from hotel_mails where hotel_id=1 ;
```

3. hotel_id από τις εγγραφές στη σχέση works που πληρούν τις προϋποθέσεις το ΑΦΜ του υπαλλήλου να είναι 1245 και να είναι ακόμα ενεργός στο συγκεκριμένο ξενοδοχείο.

```
1 select hotel_id from works where irs=1245 and finish_date>curdate();
```

6.2 Ενώσεις - Joins

Στην εφαρμογή μας χρησιμοποιούμε συχνά selection queries που γίνονται σε joined πίνακες. Ένα παράδειγμα για την περιγραφή των amenities, που παρέχονται από το δωμάτιο με τον δοσμένο κωδικό δωματίου:

```
1 select hotel_room_amenities.room_id, amenities.amenity from hotel_room_amenities inner join
   amenities on hotel_room_amenities.amenity_id=amenities.amenity_id where room_id=2
```

6.3 Insertions

Τροποποιήσεις στη βάση γίνονται και με την εισαγωγή νέων στοιχείων. Για παράδειγμα, με το ακόλουθο query Θέτουμε νέες τιμές στις ιδιότητες της εγγραφής του πίνακα hotel, για την οποία το hotel_id είναι αυτό που ορίστηκε:

```
1 update hotel set hotel_group_id=?,name=? ,stars=?, street=?, num=?, postal_code=?, city=?
   where hotel_id=?;
```

6.4 Deletions

Ο admin της βάσης από την εφαρμογή μας μπορεί αφού συνδεθεί να διαγράψει στοιχεία από τη βάση. Για παράδειγμα, η διαγραφή ενός ξενοδοχείου μέσω της εφαρμογής μας μπορεί να πυροδοτήσει το ακόλουθο query:

```
1 delete from hotel where hotel_id=2;
```

6.5 Δυναμικά queries - Παράδειγμα: αναζήτηση δωματίων με βάση πολλά κριτήρια

Τα περισσότερα queries στην εφαρμογή μας αναπτύσσονται δυναμικά, με την έννοια ότι ξεκινάμε από ένα μικρό query και το μεγαλώνουμε ανάλογα με το αν ο χρήστης έχει συμπληρώσει ή όχι πεδία της φόρμας. Στην υποενότητα αυτή αναλύουμε το σύνθετο query της αναζήτησης δωματίου με βάση όλα τα κριτήρια που παρέχονται στην εφαρμογή. Μελετάμε τη σύνθετη περίπτωση που ο χρήστης έχει επιλέξει και συγκεκριμένο αριθμό δωματίων που αναζητά.

Αρχικά, παραθέτουμε το ακόλουθο κομμάτι κώδικα:

```
1 String columns="hotel_group.name as name2,hotel.name,room_id, hotel_room.hotel_id, price ,
   repairs_needed, expandable, view, capacity, stars, hotel.hotel_group_id, hotel.street,
   hotel.num, hotel.postal_code, hotel.city";
2
3 String columns2="tmp.name2,tmp.name,tmp.room_id, tmp.hotel_id, price, repairs_needed,
   expandable, view, capacity, stars, tmp.hotel_group_id, tmp.street, tmp.num, tmp.
   postal_code, tmp.city";
4
5 String firstQuery="create temporary table tmp select "+columns+" from hotel_room inner join
   hotel on hotel_room.hotel_id=hotel.hotel_id inner join hotel_group on hotel.
   hotel_group_id=hotel_group.hotel_group_id where (";
6
7 if (! stars.equals("")) {
8     firstQuery=firstQuery+"stars="+stars+" and ";
9 }
10 if (! capacity.equals("")) {
11     firstQuery=firstQuery+"capacity >="+capacity+" and ";
12 }
13 if (! group.equals("")) {
14     firstQuery=firstQuery+"hotel_group.hotel_group_id="+group+" and ";
15 }
16
17 if (! price.equals("")) {
18     firstQuery=firstQuery+"hotel_room.price<="+price+" and ";
19 }
20
21 int len=locations.length;
```

```

22     int index=1;
23     if (len>0) {
24         firstQuery=firstQuery+"hotel.city in (" ;
25     }
26     for (String x : locations) {
27         if (index==len) {
28             firstQuery=firstQuery+"\ '"+x+"\ '"+" ) and ";
29         }
30         else {
31             firstQuery=firstQuery+"\ '"+x+"\ '"+" , ";
32         }
33         index=index+1;
34     }

```

Με αυτόν τον τρόπο, σύμφωνα με το ποια κριτήρια ο πελάτης έχει ορίσει επιστρέφονται σε προσωρινό πίνακα οι ιδιότητες hotel_group.name, hotel.name, room_id, hotel_room.hotel_id, price, repairs_needed, expandable, view, capacity, stars, hotel.hotel_group_id, hotel.street, hotel.num, hotel.postal_code, hotel.city από την ένωση των σχέσεων hotel_room, hotel και hotel_group.

Στη συνέχεια με τον εξής κώδικα:

```

1 String case1="create temporary table tmp2 select distinct "+columns2+" from tmp order by "+
  option+";";
2 String secondQuery="create temporary table tmp2 select distinct "+columns2+" from tmp left
  join reserves on tmp.room_id=reserves.room_id where ( (reserves.room_id is null) or (" ;
3
4 secondQuery=secondQuery+"(finish_date<\ '"+start+"\ ' ) or (start_date>\ '"+finish+"\ ' )";

```

Γίνεται ένωση της προηγούμενης προσωρινής σχέσης με τη σχέση reserve και επιστρέφονται σε δεύτερο προσωρινό πίνακα οι ιδιότητες των δωματίων, τα οποία είναι διαθέσιμα τις ημερομηνίες που εισήγαγε(αν εισήγαγε) ο πελάτης.

Τέλος, έχουμε:

```

1 String helpQuery="create temporary table tmp3 like tmp2";
2 helpQuery="insert tmp3 select * from tmp2";
3
4 if (!num.equals("")) {
5     finalQuery="select * from tmp2 where hotel_id in (select hotel_id from tmp3 group
  by hotel_id having count(hotel_id)>="+num+" )";
6 }
7 else {
8     finalQuery="select * from tmp2";
9 }

```

Εδώ δημιουργείται ένα αντίγραφο του δεύτερου προσωρινού πίνακα και εμφανίζονται στο χρήστη οι ιδιότητες των δωματίων, τα οποία πληρούν επιπροσθέτως το κριτήριο το ξενοδοχείο, στο οποίο ανήκουν να έχει διαθέσιμο αριθμό δωματίων ίσο ή μεγαλύτερο με αυτό που έχει επιλέξει ο χρήστης.

6.6 Δημιουργία views βάσης

Στην εκφώνηση ζητείται να δημιουργήσουμε views της βάσης στα οποία ο χρήστης θα μπορεί να βλέπει τον αριθμό των διαθέσιμων δωματίων ανά capacity και τον αριθμό των διαθέσιμων δωματίων ανά location. Για την δημιουργία των συγκεκριμένων views τρέχουμε τα ακόλουθα queries που βρίσκονται στο φάκελο database-setup στο αρχείο database-views.sql :

```

1 create view hotel_room_capacity_view as select capacity ,count(room_id) as
  rooms_per_capacity from hotel_room where room_id not in (select room_id from reserves
  where curdate() between start_date and finish_date) group by capacity;
2 create view hotel_room_location_view as select htl.city ,count(htl_room.room_id) as
  rooms_per_city from hotel as htl inner join hotel_room as htl_room on htl.hotel_id=
  htl_room.hotel_id where htl_room.room_id not in (select room_id from reserves where
  curdate() between start_date and finish_date) group by city;

```

Τα views αυτά ο χρήστης μπορεί να τα προσπελάσει από τη σελίδα hotelSelection.jsp πατώντας τα αντίστοιχα buttons.

7 Ευρετήρια

Τα ευρετήρια επιβαρύνουν τη διαδικασία των insertions, μπορούν όμως σε ορισμένες περιπτώσεις να επιταχύνουν σημαντικά το selection. Ιδιαίτερα στην περίπτωση που το selection γίνεται συνήθως με κάποιο where clause που αφορά στήλη που δεν είναι key, τα indexes μπορούν να επιταχύνουν σημαντικά την διαδικασία.

7.1 Ευρετήρια που ορίζει η mysql

Η mysql ορίζει από μόνη της ως ευρετήρια όλα τα primary και τα foreign keys. Συνεπώς, όλα τα πεδία που έχουμε ορίσει ως primary ή foreign keys υφίστανται indexing.

7.2 Δικά μας ευρετήρια

Η χρήση ευρετηρίων έχει νόημα στις περιπτώσεις που τα selections είναι πολύ περισσότερα από τα insertions και ιδιαίτερα όταν γίνονται με where clause που αφορά στήλες που δεν είναι keys. Αυτή ακριβώς είναι η περίπτωση της αναζήτησης δωματίου: Ψάχνουμε πολύ συχνότερα δωμάτια για ένα προορισμό από ότι πραγματικά κλείνουμε, και όταν ψάχνουμε, ψάχνουμε με βάση τα start και finish dates. Είναι λοιπόν χρήσιμο, στους πίνακες reserves και rents που γίνονται οι αναζητήσεις να χρησιμοποιήσουμε indexes. Έτσι, ορίζουμε τα ακόλουθα δύο indexes:

```
1 create index reserves_date_index on reserves (start_date, finish_date);
2 create index rents_date_index on rents (start_date, finish_date);
```

Ακόμη, όταν ψάχνουμε στο ιστορικό των εργαζομένων, μας αφορούν συνήθως πληροφορίες όπως πόσοι εργαζόμενοι υπήρχαν σε μια συγκεκριμένη χρονική περίοδο. Η αναζήτηση γίνεται με βάση τα dates και όχι με βάση κάποιο κλειδί. Άρα, έχει και εδώ πιθανώς νόημα να ορίσουμε το index:

```
1 create index works_history_date_index on works_history (start_date, finish_date);
```

Τέλος, για τα bookings που έχουν γίνει μας ενδιαφέρει συχνά ο τρόπος πληρωμής και οι ημερομηνίες (παρά το booking id). Άρα, ορίζουμε και εδώ το index:

```
1 create index booking_history_index on booking_history (start_date, finish_date,
payment_method);
```

8 Σύστημα και γλώσσες προγραμματισμού

1. Το παρόν project υλοποιήθηκε σε Linux Ubuntu σύστημα.
2. Για την υλοποίηση της Βάσης Δεδομένων χρησιμοποιείται MySQL Server (έκδοση 5.7.22).
3. Για το server-side της εφαρμογής χρησιμοποιείται Java.
4. Για την επικοινωνία μεταξύ client και server χρησιμοποιήθηκε Apache Tomcat.
5. Ως προς το front-end της εφαρμογής χρησιμοποιήθηκε html για την εισαγωγή των στοιχείων της σελίδας, css για τον καλλωπισμό των στοιχείων αυτών και Javascript για form validation, δημιουργία διαδραστικών στοιχείων (όπως κουμπιών), δημιουργία ενός custom calendar, κ.α. . Για το css χρησιμοποιήσαμε το bootstrap framework και ιδιαίτερα ότι αφορά τους containers και το grid σύστημα που προσφέρει για responsive στοίχιση στοιχείων.

Πιο συγκεκριμένα, χρησιμοποιούνται jsp αρχεία τα οποία περιέχουν Java κώδικα και μέσω του Apache Tomcat γίνονται render σε html ώστε να μπορεί να τα βλέπει ο client. Τα αρχεία αυτά επικοινωνούν με τη βάση μέσω κατάλληλων Servlets (Java files) τα οποία συνομιλούν με τη ΒΔ και στέλνουν πληροφορίες πίσω στα jsp files για εμφάνιση στο χρήστη της εφαρμογής.

9 Αναλυτικά βήματα εγκατάστασης της εφαρμογής

Τα βήματα που χρειάζεται να υλοποιήσει κάποιος προκειμένου να εγκαταστήσει την εφαρμογή μας σε σύστημα Linux είναι τα ακόλουθα:

1. Εγκατάσταση mysql. Σε ubuntu, η εγκατάσταση μπορεί να γίνει από το terminal με τη χρήση της εντολής:

```
1 sudo apt-get install mysql-server
```

2. Δημιουργία βάσης δεδομένων.

Αφού ολοκληρώσουμε με το installation και το setup του mysql-server (ορίζοντας κωδικό για το χρήστη root) πληκτρολογούμε στο terminal:

```
1 mysql -u root -p
```

Αφού εισάγουμε τον κωδικό και συνδεθούμε γράφουμε:

```
1 create database ntua_db;
```

3. Δημιουργία πινάκων στη βάση δεδομένων.

Κλείνουμε το session με τη mysql, κάνουμε cd στο φάκελο database-setup και τρέχουμε την ακόλουθη εντολή:

```
1 mysql -u root -p ntua_db < tables-creator.sql
```

4. Εισαγωγή των triggers.

Για να εισάγουμε στη βάση δεδομένων μας τους triggers που έχουμε ορίσει γράφουμε:

```
1 mysql -u root -p ntua_db < database-triggers.sql
```

5. Αρχικοποίηση με δεδομένα.

Για να αρχικοποιήσουμε τη βάση δεδομένων μας τρέχουμε στο terminal:

```
1 mysql -u root -p ntua_db < database-initializer.sql
```

6. Δημιουργία views βάσης.

```
1 mysql -u root -p ntua_db < database-views.sql
```

7. Εγκατάσταση JAVA.

Στο τέλος της εγκατάστασης, θα πρέπει να υπάρχει στο `.bashrc` ορισμένη η μεταβλητή `JAVA_HOME` που να δείχνει στο φάκελο εγκατάστασης της JAVA στον υπολογιστή.

8. Εγκατάσταση Eclipse.

9. Εγκατάσταση Apache Tomcat.

Στο τέλος της εγκατάστασης, θα πρέπει να υπάρχει στο `.bashrc` ορισμένη η μεταβλητή `CATALINA_HOME` που να δείχνει στο φάκελο εγκατάστασης του Apache Tomcat στον υπολογιστή.

10. Download του binary file του MySQL JDBC Driver.

11. Αποσυμπίεση του τελευταίου αρχείου

12. Μεταφορά του jar file που προέκυψε από την αποσυμπίεση στο φάκελο `$CATALINA_HOME/lib`.

Σημείωση: Η μεταβλητή `CATALINA_HOME` ορίστηκε στο βήμα 9.

13. Import στο Eclipse του project. Αυτό γίνεται ακολουθώντας τη διαδρομή: File > Import

14. Εισαγωγή ενός instance του Apache Tomcat Server στο Eclipse.

Για να το πετύχουμε αυτό ακολουθούμε τη διαδρομή στο Eclipse: Window > Preferences > Server > Runtime Environments > Add και διαλέγουμε από τη λίστα τον Apache Tomcat στην έκδοση που τον κατεβάσαμε.

15. Configuration του Apache Tomcat Instance.

Προκειμένου να μπορέσει το Instance με τη βάση δεδομένων μας πρέπει να μπορέσει να τη δει. Για να συμβεί αυτό απαιτείται μια διαδικασία configuration. Συγκεκριμένα, αλλάζουμε το αρχείο `server.xml` του server που δημιουργήθηκε στο ακόλουθο:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 Licensed to the Apache Software Foundation (ASF) under one or more
4 contributor license agreements. See the NOTICE file distributed with
5 this work for additional information regarding copyright ownership.
6 The ASF licenses this file to You under the Apache License, Version 2.0
7 (the "License"); you may not use this file except in compliance with
8 the License. You may obtain a copy of the License at
9
10 http://www.apache.org/licenses/LICENSE-2.0
11
12 Unless required by applicable law or agreed to in writing, software
13 distributed under the License is distributed on an "AS IS" BASIS,
14 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15 See the License for the specific language governing permissions and
16 limitations under the License.
17 -->
18 <!-- Note: A "Server" is not itself a "Container", so you may not
19 define subcomponents such as "Valves" at this level.
20 Documentation at /docs/config/server.html
21 -->
22 <Server port="8005" shutdown="SHUTDOWN">
23   <Listener className="org.apache.catalina.startup.VersionLoggerListener"/>
24   <!-- Security listener. Documentation at /docs/config/listeners.html
25   <Listener className="org.apache.catalina.security.SecurityListener" />
26   <!-->
27   <!--APR library loader. Documentation at /docs/apr.html -->
28   <Listener SSLEngine="on" className="org.apache.catalina.core.AprLifecycleListener"/>
29   <!-- Prevent memory leaks due to use of particular java/javax APIs-->
30   <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener"/>
31   <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener"/>
32   <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener"/>
33
34   <!-- Global JNDI resources
35   Documentation at /docs/jndi-resources-howto.html
36   -->
37   <GlobalNamingResources>
38     <!-- Editable user database that can also be used by
39     UserDatabaseRealm to authenticate users
40     -->
41     <Resource auth="Container" description="User database that can be updated and
42     saved" factory="org.apache.catalina.users.MemoryUserDatabaseFactory" name="
43     UserDatabase" pathname="conf/tomcat-users.xml" type="org.apache.catalina.
44     UserDatabase"/>
45   </GlobalNamingResources>
46
47   <!-- A "Service" is a collection of one or more "Connectors" that share
48   a single "Container" Note: A "Service" is not itself a "Container",
49   so you may not define subcomponents such as "Valves" at this level.
50   Documentation at /docs/config/service.html
51   -->
```

```

47 <Service name="Catalina">
48
49 <!--The connectors can use a shared executor, you can define one or more named
thread pools-->
50 <!--
51 <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
52     maxThreads="150" minSpareThreads="4"/>
53 -->
54
55
56 <!-- A "Connector" represents an endpoint by which requests are received
and responses are returned. Documentation at :
57     Java HTTP Connector: /docs/config/http.html
58     Java AJP Connector: /docs/config/ajp.html
59     APR (HTTP/AJP) Connector: /docs/apr.html
60     Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
61 -->
62
63 <Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort=
"8443"/>
64 <!-- A "Connector" using the shared thread pool-->
65 <!--
66 <Connector executor="tomcatThreadPool"
67     port="8080" protocol="HTTP/1.1"
68     connectionTimeout="20000"
69     redirectPort="8443" />
70 -->
71 <!-- Define a SSL/TLS HTTP/1.1 Connector on port 8443
72     This connector uses the NIO implementation. The default
73     SSLImplementation will depend on the presence of the APR/native
74     library and the useOpenSSL attribute of the
75     AprLifecycleListener.
76     Either JSSE or OpenSSL style configuration may be used regardless of
77     the SSLImplementation selected. JSSE style configuration is used below.
78 -->
79 <!--
80 <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
81     maxThreads="150" SSLEnabled="true">
82     <SSLHostConfig>
83         <Certificate certificateKeystoreFile="conf/localhost-rsa.jks"
84             type="RSA" />
85     </SSLHostConfig>
86 </Connector>
87 -->
88 <!-- Define a SSL/TLS HTTP/1.1 Connector on port 8443 with HTTP/2
89     This connector uses the APR/native implementation which always uses
90     OpenSSL for TLS.
91     Either JSSE or OpenSSL style configuration may be used. OpenSSL style
92     configuration is used below.
93 -->
94 <!--
95 <Connector port="8443" protocol="org.apache.coyote.http11.Http11AprProtocol"
96     maxThreads="150" SSLEnabled="true" >
97     <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />
98     <SSLHostConfig>
99         <Certificate certificateKeyFile="conf/localhost-rsa-key.pem"
100             certificateFile="conf/localhost-rsa-cert.pem"
101             certificateChainFile="conf/localhost-rsa-chain.pem"
102             type="RSA" />
103     </SSLHostConfig>
104 </Connector>
105 -->
106
107 <!-- Define an AJP 1.3 Connector on port 8009 -->
108 <Connector port="8009" protocol="AJP/1.3" redirectPort="8443"/>
109
110
111 <!-- An Engine represents the entry point (within Catalina) that processes

```

```

112     every request. The Engine implementation for Tomcat stand alone
113     analyzes the HTTP headers included with the request, and passes them
114     on to the appropriate Host (virtual host).
115     Documentation at /docs/config/engine.html —>
116
117     <!-- You should set jvmRoute to support load-balancing via AJP ie :
118     <Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">
119     —>
120     <Engine defaultHost="localhost" name="Catalina">
121
122         <!--For clustering, please take a look at documentation at:
123         /docs/cluster-howto.html (simple how to)
124         /docs/config/cluster.html (reference documentation) —>
125         <!--
126         <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
127         —>
128
129         <!-- Use the LockOutRealm to prevent attempts to guess user passwords
130         via a brute-force attack —>
131         <Realm className="org.apache.catalina.realm.LockOutRealm">
132             <!-- This Realm uses the UserDatabase configured in the global JNDI
133             resources under the key "UserDatabase". Any edits
134             that are performed against this UserDatabase are immediately
135             available for use by the Realm. —>
136             <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="
137             UserDatabase"/>
138         </Realm>
139
140         <Host appBase="webapps" autoDeploy="true" name="localhost" unpackWARs="true">
141
142             <!-- SingleSignOn valve, share authentication between web applications
143             Documentation at: /docs/config/valve.html —>
144             <!--
145             <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
146             —>
147
148             <!-- Access log processes all example.
149             Documentation at: /docs/config/valve.html
150             Note: The pattern used is equivalent to using pattern="common" —>
151             <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
152             pattern="%h %l %u %t &quot;%r&quot; %s %b" prefix="localhost_access_log" suffix=".
153             txt"/>
154
155             <Context docBase="ntua-databases" path="/ntua-databases" reloadable="true"
156             source="org.eclipse.jst.jee.server:ntua-databases"/></Host>
157         </Context>
158
159         <ResourceLink name="jdbc/ntua_db"
160             global="jdbc/ntua_db"
161             type="javax.sql.DataSource"/>
162     </Context>
163
164 </Engine>
165 </Service>
166 </Server>

```

προσαρμόζοντας το docBase στο όνομα του project στο Eclipse.

Στη συνέχεια, αλλάζουμε το context.xml του Server που δημιουργήθηκε στο ακόλουθο αρχείο:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 Licensed to the Apache Software Foundation (ASF) under one or more
4 contributor license agreements. See the NOTICE file distributed with
5 this work for additional information regarding copyright ownership.
6 The ASF licenses this file to You under the Apache License, Version 2.0
7 (the "License"); you may not use this file except in compliance with

```

```

8 the License. You may obtain a copy of the License at
9
10 http://www.apache.org/licenses/LICENSE-2.0
11
12 Unless required by applicable law or agreed to in writing, software
13 distributed under the License is distributed on an "AS IS" BASIS,
14 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15 See the License for the specific language governing permissions and
16 limitations under the License.
17 <!-- The contents of this file will be loaded for each web application --><Context>
18
19 <!-- Default set of monitored resources. If one of these changes, the -->
20 <!-- web application will be reloaded. -->
21 <WatchedResource>WEB-INF/web.xml</WatchedResource>
22 <WatchedResource>WEB-INF/tomcat-web.xml</WatchedResource>
23 <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>
24
25 <!-- Uncomment this to disable session persistence across Tomcat restarts -->
26 <!--
27 <Manager pathname="" />
28 -->
29 <Resource name="jdbc/ntua_db" auth="Container" type="javax.sql.DataSource"
30 maxActive="100" maxIdle="30" maxWait="10000"
31 username="root" password="[insert_your_pass_here]" driverClassName="com
32 .mysql.jdbc.Driver"
33 url="jdbc:mysql://localhost:3306/ntua_db"/>
34 </Context>

```

προσαρμόζοντας το πεδίο password στον κωδικό που έχουμε ορίσει στη βάση μας για το χρήστη root.

16. Τροποποίηση του κωδικού για τη βάση δεδομένων που βρίσκεται στο αρχείο:

./src/com/ntua/databases/Connector.java

17. Εκτέλεση εφαρμογής.

Για την εκτέλεση της εφαρμογής, δεξί κλικ στο root folder του project στο Eclipse και Run As > Run on Server.

10 Κατασκευή βάσης δεδομένων

Η βάση δεδομένων κατασκευάζεται με τα ακόλουθα DDL:

```

1 use ntua_db;
2
3 create table employee (irs integer not null primary key, lastname varchar(25) not null,
4 ssn integer not null, firstname varchar(25) not null, street varchar(25), num integer,
5 postal_code integer, city
6 varchar(25));
7
8 create table hotel_group (hotel_group_id integer not null auto_increment primary key, name
9 varchar(25) not null,
10 number_of_hotels integer not null, street varchar(25), num integer, postal_code integer,
11 city
12 varchar(25));
13
14 create table hotel_group_emails (hotel_group_id integer not null, email varchar(45) not null
15 ,
16 primary key (hotel_group_id, email), foreign key (hotel_group_id) references hotel_group(
17 hotel_group_id) on delete cascade);
18
19 create table hotel_group_phones (hotel_group_id integer, phone bigint not null, primary key
20 (hotel_group_id, phone), foreign key (hotel_group_id) references hotel_group(
21 hotel_group_id) on delete cascade);
22

```

```

16
17 create table hotel (hotel_id integer not null auto_increment primary key,name varchar(25),
18 stars integer not null, number_of_rooms integer not null,
19 hotel_group_id integer not null,street varchar(25), num integer, postal_code integer,
city
20 varchar(25), foreign key (hotel_group_id) references hotel_group(hotel_group_id) on
delete cascade);
21
22 create table hotel_phones (hotel_id integer not null, phone bigint not null,
23 primary key (hotel_id,phone), foreign key (hotel_id) references hotel(hotel_id) on delete
cascade);
24
25 create table hotel_emails (hotel_id integer not null, email
26 varchar(45) not null, primary key (hotel_id,email), foreign key (hotel_id) references
hotel(hotel_id) on delete cascade);
27
28 create table works (irs integer not null,hotel_id
29 integer not null, start_date date not null, position varchar(25) not null,
30 finish_date date, primary key (irs,hotel_id),foreign key (irs) references employee(irs)
on delete cascade, foreign key (hotel_id) references hotel(hotel_id) on delete cascade);
31
32 create table customer(irs integer not null primary key,first_name varchar(25) not null,
33 last_name varchar(25) not null, ssn integer not null, first_registration date not null,
34 street varchar(25), num integer, postal_code integer, city
35 varchar(25));
36
37 create table amenities(amenity_id integer not null auto_increment primary key,
38 amenity varchar(25) not null);
39
40
41 create table hotel_room (room_id integer not null auto_increment, hotel_id integer not null
,price float not null, repairs_needed integer not null, expandable
42 integer not null, view integer not null, capacity integer not null,primary key (room_id,
hotel_id),
43 foreign key (hotel_id) references hotel(hotel_id) on delete cascade);
44
45
46 create table hotel_room_amenities(amenity_id integer not null,
47 room_id integer not null,primary key(amenity_id,room_id),foreign key (amenity_id)
references amenities(amenity_id) on delete cascade, foreign key (room_id) references
hotel_room(room_id) on delete cascade);
48
49 create table reserves (reserve_id integer not null auto_increment primary key,
50 irs integer not null, room_id integer not null, hotel_id integer not null,
51 start_date date not null, finish_date date not null, paid integer not null
52 ,foreign key (irs) references customer(irs) on delete cascade, foreign key (room_id)
references hotel_room(room_id) on delete cascade
53 ,foreign key (hotel_id) references hotel(hotel_id) on delete cascade);
54
55 create table rents (rent_id integer not null auto_increment primary key,
56 irs_employee integer not null,
57 irs_customer integer not null,
58 hotel_id integer not null, room_id integer not null,payment_amount float not null,
payment_method varchar(25),start_date date not null,finish_date date not null,
59 foreign key (irs_employee) references employee(irs) on delete cascade, foreign key (
irs_customer) references customer(irs) on delete cascade,
60 foreign key (hotel_id) references hotel(hotel_id) on delete cascade, foreign key (room_id
) references hotel_room(room_id) on delete cascade);
61
62 create table booking_history (booking_id integer auto_increment primary key,
63 irs_customer integer not null,
64 irs_employee integer not null,room_id integer not null,hotel_id integer not null,
hotel_group_id integer not null,start_date date not null,finish_date date not null,
65 payment_method varchar(25) not null);
66
67 create table works_history(works_id integer auto_increment primary key,irs_employee integer
not null,hotel_id integer not null,hotel_group_id integer not null,

```


68 `position varchar(25) not null , start_date date not null , finish_date date not null);`

11 Αναφορές

[1] Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. Database system concepts. Vol. 4. New York: McGraw-Hill, 1997.

[2] MySQL Reference