

# ORIGIN

## Βάσεις Δεδομένων :

Αναφορά Project 2021-2022

### Στοιχεία Ομάδας :

Παπαδοπούλου Μαρία	ΑΜ: 1072494	3 <sup>ο</sup> έτος	up1072494@upnet.gr
Σφήκας Θεόδωρος	ΑΜ:1072550	3 <sup>ο</sup> έτος	up1072494@upnet.gr

# ΤΕΚΜΗΡΙΩΣΗ SQL

## ΕΙΣΑΓΩΓΗ

Στην παρούσα αναφορά περιέχεται η τεκμηρίωση του back-end μέρους της εφαρμογής, δηλαδή της βάσης δεδομένων. Πιο συγκεκριμένα, θα αιτιολογηθεί το σχήμα της βάσης, θα εξηγηθούν οι παραδοχές με βάση τις οποίες προέκυψαν τόσο η σχεδίαση, όσο και οι λειτουργίες της και θα παρουσιαστεί το σχεσιακό διάγραμμα που την περιγράφει, καθώς και ο κώδικας της.

## ΠΑΡΑΔΟΧΕΣ

Οι παραδοχές που αναφέρουμε χωρίζονται σε δύο κατηγορίες:

### A) Παραδοχές που αφορούν το σχήμα της βάσης

1. Η υπηρεσία παρέχει όλες τις ταινίες και όλα τα επεισόδια σε όλους τους πελάτες, οπότε οι πίνακες inventories/καταλόγων καταργούνται. Η παραδοχή αυτή βασίστηκε σε μελέτη των αναγκών της βάσης, από την οποία εξήχθη το συμπέρασμα ότι οι συγκεκριμένοι πίνακες δεν προσφέρουν κάποια διευκόλυνση, αντιθέτως μεγεθύνουν την πολυπλοκότητα των queries, απαιτώντας επιπλέον joins.
2. Για τις 3 κατηγορίες χρηστών ακολουθήθηκε μία πιο οντοκεντρική προσέγγιση, με τους πίνακες πελατών (customer), υπαλλήλων (employee) και διαχειριστών (admin) να αναφέρονται σε έναν κοινό πίνακα χρηστών (user) με ξένα κλειδιά τα δικά τους id πεδία, στο πεδίο id του πίνακα user. Ο πίνακας user περιέχει τα στοιχεία που όλοι οι χρήστες έχουν κοινά, ενώ οι πίνακες που αναφέρονται στις 3 κατηγορίες έχουν πεδίο id και ίσως επιπλέον δικά τους πεδία. Έτσι, αποφεύγεται η ύπαρξη χρηστών με ίδιο id και μειώνονται οι προσπελάσεις πινάκων σε περίπτωση που ψάχνουμε πληροφορίες για χρήστη με βάση το id. Ταυτόχρονα, η σύνδεση ενός χρήστη απαιτεί μία μόνο προσπέλαση, στον πίνακα χρηστών, όπως και η εύρεση του τρέχοντος χρήστη, που χρησιμοποιείται σε πολλά σημεία του κώδικα και κυρίως στα triggers που ανανεώνουν το log.
3. Τα στοιχεία που διατηρούμε για τους χρήστες στον πίνακα user είναι αυτά που υπήρχαν στον πίνακα customer στη βάση που δόθηκε, με τις εξής αλλαγές/προσθήκες: το πεδίο active δηλώνει ενεργό λογαριασμό, δηλαδή λογαριασμό που έχει δικαίωμα πρόσβασης στην υπηρεσία. Με βάση αυτό το πεδίο μπορούν να δημιουργηθούν ενεργοί/ανενεργοί λογαριασμοί πελατών από τον διαχειριστή και να

ανασταλούν λογαριασμοί πελατών από τους υπαλλήλους, χωρίς την οριστική διαγραφή τους. Όταν ένας λογαριασμός καθίσταται ανενεργός, ο πελάτης δεν μπορεί να συνδεθεί στη βάση. Ενεργοποίηση λογαριασμού μπορούν να κάνουν οι υπάλληλοι μέσω της δυνατότητας επεξεργασίας λογαριασμού πελάτη, που εξηγείται αναλυτικά στην αναφορά για το γραφικό περιβάλλον.

Έχει προστεθεί ένα ακόμα πεδίο pending στον πίνακα χρηστών με σκοπό την αποδοχή/απόρριψη λογαριασμών πελατών, που επιχειρούν να εγγραφούν για πρώτη φορά στην υπηρεσία, από τον διαχειριστή.

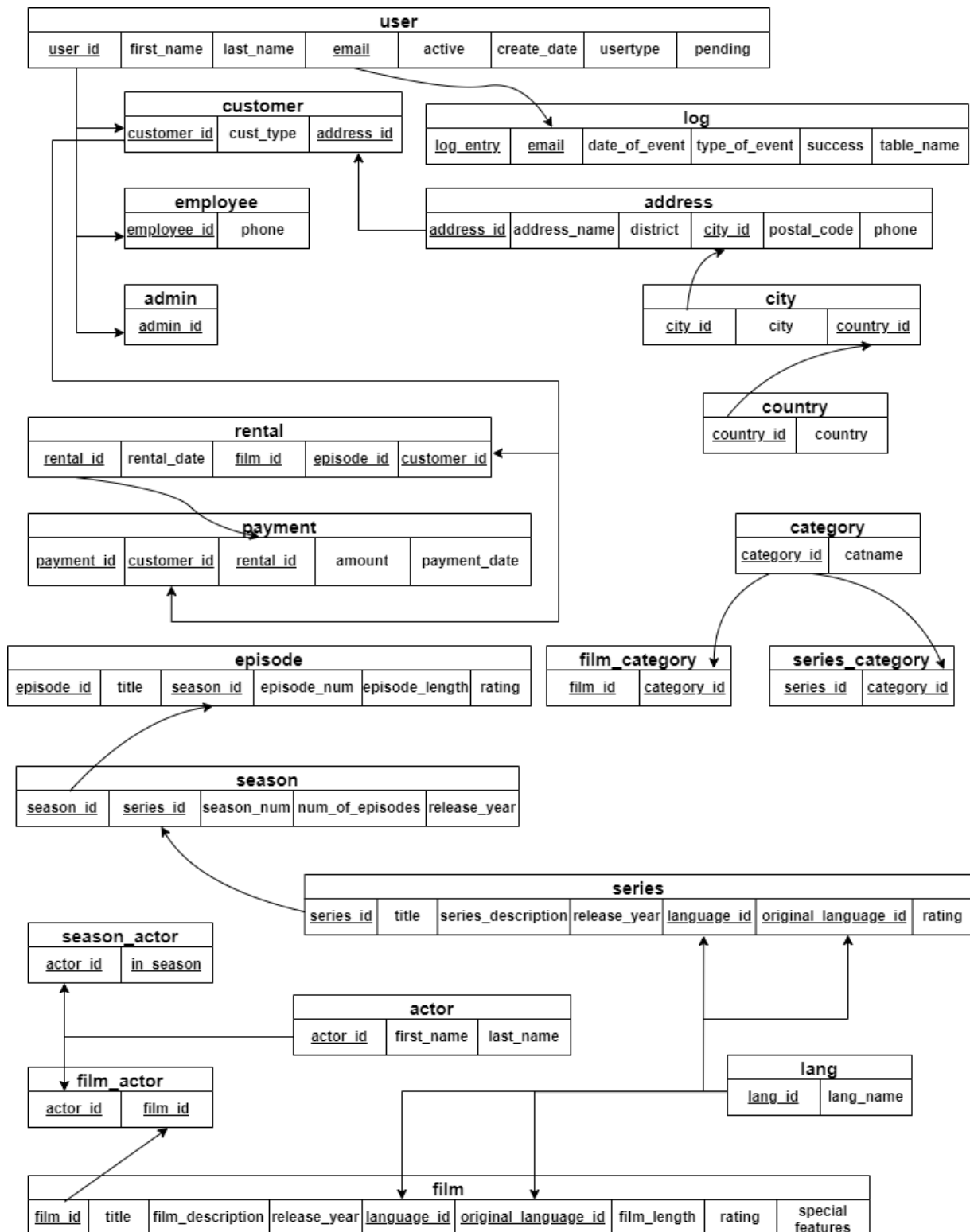
Το πεδίο address\_id έχει μεταφερθεί στον πίνακα πελατών (customer).

4. Τα στοιχεία που διατηρούμε για τους πελάτες είναι το id τους (ξένο κλειδί στο id του πίνακα user), ο τύπος τους, για την υλοποίηση των τριών τύπων συνδρομής και ο κωδικός της διεύθυνσης κατοικίας τους (ξένος κλειδί στον πίνακα διευθύνσεων - address).
5. Δεν αποθηκεύουμε διεύθυνση για τους υπαλλήλους, αλλά μόνο τηλέφωνο (και προφανώς το id τους, ως ξένο κλειδί στο id των χρηστών – πίνακας user).
6. Για τους διαχειριστές δεν αποθηκεύουμε κανένα επιπλέον μέσο επικοινωνίας, παρά μόνο το email, που αποθηκεύεται για όλους τους χρήστες (και προφανώς το id τους, ως ξένο κλειδί στο id των χρηστών – πίνακας user).
7. Τα στοιχεία που διατηρούμε για τις σειρές είναι αντίστοιχα των στοιχείων για τις ταινίες. Για τις γλώσσες, θεωρούμε ότι όλη η σειρά έχει υποτίτλους στη γλώσσα που υπάρχει στο πεδίο language\_id.
8. Τα στοιχεία που διατηρούμε για τις σεζόν είναι: ένα μοναδικό id που λειτουργεί ως πρωτεύον κλειδί, το id της σειράς στην οποία η σεζόν ανήκει (ξένο κλειδί στο id των σειρών), ο αριθμός της σεζόν (1<sup>ος</sup>, 2<sup>ος</sup> κύκλο κλπ), το πλήθος των επεισοδίων που αυτή περιέχει και το έτος κυκλοφορίας.
9. Δημιουργήθηκαν οι συσχετίσεις σειράς-κατηγορίας και σεζόν-ηθοποιού, με αντίστοιχο τρόπο με τις ταινίες.
10. Τα στοιχεία που διατηρούμε για τα επεισόδια είναι: ένα μοναδικό id που λειτουργεί ως πρωτεύον κλειδί, τον τίτλο του επεισοδίου, το id της σεζόν στην οποία ανήκει το επεισόδιο (ξένο κλειδί στο πεδίο id των σεζόν), τον αριθμό του (1<sup>ο</sup>, 2<sup>ο</sup> επεισόδιο κλπ), η διάρκεια του και η σήμανση καταλληλότητας.

11. Για τις ενοικιάσεις διατηρούμε τον ίδιο πίνακα rental που δόθηκε και προσθέτουμε επιπλέον πεδίο episode\_id, ως ξένο κλειδί που αναφέρεται στο πεδίο id των επεισοδίων. Επειδή μία ενοικίαση αναφέρεται είτε σε ταινία, είτε σε επεισόδιο, σε κάθε γραμμή αυτού του πίνακα ένα από τα δύο ξένα κλειδιά που δηλώνουν το προϊόν για το οποίο έγινε η ενοικίαση (film\_id, episode\_id) είναι NULL και μόνο το άλλο έχει τιμή. Η πρακτική αυτή προτιμήθηκε έναντι της ύπαρξης ξεχωριστών πινάκων ενοικιάσεων για ταινίες και επεισόδια, επειδή αποκλείει την εύρεση ίδιων id μεταξύ των ενοικιάσεων και μειώνει τις προσπελάσεις αν πχ ψάχνουμε ενοικίαση ή προϊόν με βάση κάποια αγορά από τον πίνακα payment.
12. Τα στοιχεία που διατηρούμε για το log ενεργειών είναι: αύξοντας αριθμός καταχώρησης, email χρήστη που προκάλεσε την ενέργεια (ως ξένο κλειδί στο πεδίο email των χρηστών – πίνακας user), ημερομηνία και τύπος συμβάντος, τιμή που δηλώνει επιτυχία/αποτυχία ενέργειας και το όνομα του πίνακα που αυτή αφορούσε.
13. Η βάση χρησιμοποιεί 5 user-defined μεταβλητές:
- a) @user\_now : αποθηκεύει το id του τρέχοντος χρήστη (η αποθήκευση γίνεται κατά τη σύνδεση του στην υπηρεσία. Σε περίπτωση που ο χρήστης εγγράφεται για πρώτη φορά, τα triggers που ανανεώνουν το log χρησιμοποιούν έναν βοηθητικό admin)

Με βάση της παραπάνω παραδοχές προέκυψε το σχήμα της βάσης που φαίνεται στην επόμενη εικόνα:

## ΣΧΕΣΙΑΚΟ ΔΙΑΓΡΑΜΜΑ



Σημείωση: Από το σχεσιακό αποκρύπτονται 4 πίνακες οι οποίοι δημιουργήθηκαν καθαρά για βοηθητικούς σκοπούς, όπως εξηγείται παρακάτω. Οι πίνακες αυτοί δεν έχουν σχέση με τις απαιτήσεις δεδομένων της υπηρεσίας και δεν προσφέρουν καμία παραπάνω πληροφορία.

## Β) Παραδοχές που αφορούν τη λειτουργικότητα της βάσης

1. Η ερμηνεία/λειτουργία των πεδίων active και pending του πίνακα χρηστών (user) εξηγήθηκε στο Α μέρος.
2. Τα procedures των ζητούμενων 3.1 και 3.3 περνάνε τα αποτελέσματα τους σε βοηθητικούς πίνακες που αποκρύπτονται από το σχεσιακό που παραθέσαμε, επειδή δεν έχουν σχέση με τις απαιτήσεις δεδομένων της υπηρεσίας. Οι πίνακες αυτοί δημιουργήθηκαν για να είναι ευκολότερη η ανάκτηση των αποτελεσμάτων από την java.
3. Δημιουργήθηκε ένα ακόμα procedure, η pay4product, η οποία με βάση το πρώτο όρισμα, που δηλώνει τον τύπο του προϊόντος που νοικιάζεται (ταινία/επεισόδιο) και τον τύπο του πελάτη με id το δεύτερο όρισμα, αποφασίζει την τιμή χρέωσης.
4. Από triggers, δημιουργήθηκαν αυτά που ζητούσαν τα ερωτήματα 4.1, 4.2 και 4.3 καθώς και μία σειρά από after insert/delete/update triggers για όλους τους πίνακες της βάσης. Τα τελευταία, αναλαμβάνουν την ανανέωση του πίνακα log εφόσον μία συναλλαγή είναι επιτυχής. Σε περίπτωση αποτυχίας της συναλλαγής, την ανανέωση του log αναλαμβάνει η java.
5. Λόγω της χρήση του InnoDB Engine, επιτρέπεται αποκλειστικά η χρήση ευρετηρίων τύπου B-Trees, ενώ δεν υποστηρίζονται Hash ευρετήρια. Παρατηρήσαμε ότι υπήρχαν έτοιμα στη βάση πολλά indexes για τα πεδία που χρειαζόμασταν για την προσπέλαση των πινάκων (συνήθως αυτά αποτελούσαν ξένα κλειδιά). Έτσι, δημιουργήθηκαν μόνο δύο επιπλέον ευρετήρια, ένα για το πεδίο rental\_date του πίνακα ενοικιάσεων (rental) και ένα για το επίθετο (πεδίο last\_name) των ηθοποιών (πίνακας actor). Και τα δύο χρησιμοποιούνται για το procedure του ερωτήματος 3.1, ενώ το δεύτερο χρησιμοποιείται και στα procedures των ερωτημάτων 3.4 α και β. Όπως φαίνεται και από τον κώδικα χρησιμοποιήθηκαν και όσα από τα ήδη υπάρχοντα ευρετήρια κρίθηκαν απαραίτητα.

## ΚΩΔΙΚΑΣ SQL

-- Table Creation Schema

-- Table structure for table "actor"

```
CREATE TABLE IF NOT EXISTS actor (  
    actor_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,  
    first_name varchar(45) NOT NULL,  
    last_name varchar(45) NOT NULL,  
    PRIMARY KEY (actor_id)  
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
```

-- Table structure for table "language"

```
CREATE TABLE IF NOT EXISTS lang (  
    lang_id tinyint(3) unsigned NOT NULL AUTO_INCREMENT,  
    lang_name char(20) NOT NULL,  
    PRIMARY KEY (lang_id)  
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
```

-- Table structure for table "film"

```
CREATE TABLE IF NOT EXISTS film (  
    film_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,  
    title varchar(128) NOT NULL,  
    film_description text,  
    release_year year(4) DEFAULT NULL,  
    language_id tinyint(3) unsigned NOT NULL,  
    original_language_id tinyint(3) unsigned DEFAULT NULL,  
    film_length smallint(5) unsigned DEFAULT NULL,  
    rating enum('G', 'PG', 'PG-13', 'R', 'NC-17') DEFAULT 'G',  
    special_features set("Trailers", "Commentaries", "Deleted Scenes",  
"Behind the Scenes") DEFAULT NULL,  
    PRIMARY KEY (film_id),  
    CONSTRAINT fk_film_language FOREIGN KEY (language_id) REFERENCES  
lang (lang_id) ON DELETE RESTRICT ON UPDATE CASCADE,  
    CONSTRAINT fk_film_language_original FOREIGN KEY  
(original_language_id) REFERENCES lang (lang_id) ON DELETE RESTRICT ON  
UPDATE CASCADE  
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
```

-- Table structure for table "film\_actor"

```
CREATE TABLE IF NOT EXISTS film_actor (  
    actor_id smallint(5) unsigned NOT NULL,  
    film_id smallint(5) unsigned NOT NULL,  
    PRIMARY KEY (actor_id, film_id),  
    CONSTRAINT fk_film_actor_actor FOREIGN KEY (actor_id) REFERENCES  
actor (actor_id) ON DELETE CASCADE ON UPDATE CASCADE,  
    CONSTRAINT fk_film_actor_film FOREIGN KEY (film_id) REFERENCES film  
(film_id) ON DELETE CASCADE ON UPDATE CASCADE  
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
```

```

-- Table structure for table "category"
CREATE TABLE IF NOT EXISTS category (
    category_id tinyint(3) unsigned NOT NULL AUTO_INCREMENT,
    catname varchar(25) NOT NULL,
    PRIMARY KEY (category_id)
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "film_category"
CREATE TABLE IF NOT EXISTS film_category (
    film_id smallint(5) unsigned NOT NULL,
    category_id tinyint(3) unsigned NOT NULL,
    PRIMARY KEY (film_id, category_id),
    CONSTRAINT fk_film_category_category FOREIGN KEY (category_id)
REFERENCES category (category_id) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_film_category_film FOREIGN KEY (film_id) REFERENCES
film (film_id) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "series"
CREATE TABLE IF NOT EXISTS series (
    series_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    title varchar(128) NOT NULL,
    series_description text,
    release_year year(4) DEFAULT NULL,
    language_id tinyint(3) unsigned NOT NULL,
    original_language_id tinyint(3) unsigned DEFAULT NULL,
    rating enum('G', 'PG', 'PG-13', 'R', 'NC-17') DEFAULT 'G',
    PRIMARY KEY (series_id),
    CONSTRAINT fk_series_language FOREIGN KEY (language_id) REFERENCES
lang (lang_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_series_language_original FOREIGN KEY
(original_language_id) REFERENCES lang (lang_id) ON DELETE RESTRICT ON
UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "season"
CREATE TABLE IF NOT EXISTS season (
    season_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    series_id smallint(5) unsigned NOT NULL,
    season_num tinyint(2) unsigned NOT NULL,
    num_of_episodes tinyint(2),
    release_year year(4) DEFAULT NULL,
    PRIMARY KEY(season_id),
    CONSTRAINT season_to_series FOREIGN KEY (series_id) REFERENCES
series(series_id) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

```



```

-- Table structure for table "season-actor"
CREATE TABLE IF NOT EXISTS season_actor(
    actor_id smallint(5) unsigned NOT NULL,
    in_season smallint(5) unsigned NOT NULL,
    PRIMARY KEY (actor_id, in_season),
    CONSTRAINT fk_series_actor_actor FOREIGN KEY (actor_id) REFERENCES
actor (actor_id) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_series_actor_season FOREIGN KEY (in_season)
REFERENCES season(season_id) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "series-category"
CREATE TABLE IF NOT EXISTS series_category (
    series_id smallint(5) unsigned NOT NULL,
    category_id tinyint(3) unsigned NOT NULL,
    PRIMARY KEY (series_id, category_id),
    CONSTRAINT fk_series_category_category FOREIGN KEY (category_id)
REFERENCES category (category_id) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_series_category_film FOREIGN KEY (series_id)
REFERENCES series (series_id) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "episode"
CREATE TABLE IF NOT EXISTS episode (
    episode_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    title varchar(128) NOT NULL,
    season_id smallint(5) unsigned NOT NULL,
    episode_num tinyint(2) unsigned NOT NULL,
    episode_length smallint(5) unsigned DEFAULT NULL,
    rating enum('G', 'PG', 'PG-13', 'R', 'NC-17') DEFAULT 'G',
    PRIMARY KEY(episode_id),
    CONSTRAINT fk_episode_season FOREIGN KEY (season_id) REFERENCES
season(season_id) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "country"
CREATE TABLE IF NOT EXISTS country (
    country_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    country varchar(50) NOT NULL,
    PRIMARY KEY (country_id)
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "city"
CREATE TABLE IF NOT EXISTS city (
    city_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    city varchar(50) NOT NULL,
    country_id smallint(5) unsigned NOT NULL,
    PRIMARY KEY (city_id),

```

```

        CONSTRAINT fk_city_country FOREIGN KEY (country_id) REFERENCES
country (country_id) ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "address"
CREATE TABLE IF NOT EXISTS address (
    address_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    address_name varchar(50) NOT NULL,
    district varchar(20) DEFAULT NULL,
    city_id smallint(5) unsigned NOT NULL,
    postal_code varchar(10) DEFAULT NULL,
    phone varchar(20) NOT NULL,
    PRIMARY KEY (address_id),
    CONSTRAINT fk_address_city FOREIGN KEY (city_id) REFERENCES city
(city_id) ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "user"
CREATE TABLE IF NOT EXISTS user(
    user_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    first_name varchar(45) NOT NULL,
    last_name varchar(45) NOT NULL,
    email varchar(50) DEFAULT NULL,
    active tinyint(1) NOT NULL DEFAULT 1,
    create_date datetime NOT NULL,
    usertype enum("CUSTOMER", "ADMIN", "EMPLOYEE"),
    pending boolean,
    PRIMARY KEY (user_id),
    UNIQUE KEY (email)
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "customer"
CREATE TABLE IF NOT EXISTS customer (
    customer_id smallint(5) unsigned NOT NULL,
    cust_type enum("FILMS ONLY", "SERIES ONLY", "FILMS AND SERIES")
DEFAULT NULL,
    address_id smallint(5) unsigned DEFAULT NULL,
    PRIMARY KEY (customer_id),
    CONSTRAINT fk_customer_address FOREIGN KEY (address_id) REFERENCES
address (address_id) ON DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT fk_user_customer FOREIGN KEY (customer_id) REFERENCES
user(user_id) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "employee"
CREATE TABLE IF NOT EXISTS employee (
    employee_id smallint(5) unsigned NOT NULL,
    phone int(10) unsigned DEFAULT NULL,

```

```

        PRIMARY KEY(employee_id),
        CONSTRAINT fk_user_employee FOREIGN KEY (employee_id) REFERENCES
user(user_id) ON DELETE CASCADE ON UPDATE CASCADE
    ) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "admin"
CREATE TABLE IF NOT EXISTS admin (
    admin_id smallint(5) unsigned NOT NULL,
    PRIMARY KEY(admin_id),
    CONSTRAINT fk_user_admin FOREIGN KEY (admin_id) REFERENCES
user(user_id) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "rental"
CREATE TABLE IF NOT EXISTS rental (
    rental_id int(11) NOT NULL AUTO_INCREMENT,
    rental_date datetime NOT NULL,
    film_id smallint(5) unsigned DEFAULT NULL,
    episode_id smallint(5) unsigned DEFAULT NULL,
    customer_id smallint(5) unsigned NOT NULL,
    PRIMARY KEY (rental_id),
    CONSTRAINT fk_rental_customer FOREIGN KEY (customer_id) REFERENCES
customer (customer_id) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_rental_film_film FOREIGN KEY (film_id) REFERENCES
film (film_id) ON DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT fk_rental_episode_ep FOREIGN KEY (episode_id) REFERENCES
episode (episode_id) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "payment"
CREATE TABLE IF NOT EXISTS payment (
    payment_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    customer_id smallint(5) unsigned,
    rental_id int(11),
    amount decimal(5,2) DEFAULT 0.4,
    payment_date datetime NOT NULL,
    PRIMARY KEY (payment_id),
    CONSTRAINT fk_payment_customer FOREIGN KEY (customer_id) REFERENCES
customer (customer_id) ON DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT fk_payment_rental FOREIGN KEY (rental_id) REFERENCES
rental (rental_id) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table structure for table "log"
CREATE TABLE IF NOT EXISTS log (
    log_entry int(11) NOT NULL AUTO_INCREMENT,
    email varchar(70),
    date_of_event DATETIME NOT NULL,

```

```

    type_of_event enum("INSERT", "UPDATE", "DELETE"),
    success boolean NOT NULL,
    table_name varchar(50) NOT NULL,
    PRIMARY KEY(log_entry),
    CONSTRAINT fk_log_user FOREIGN KEY (email) REFERENCES user (email)
ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

-- Table indexes

CREATE INDEX IF NOT EXISTS rentdate ON rental(rental_date);

CREATE INDEX IF NOT EXISTS lastname ON actor(last_name);

-- DB procedures (+ user-defined variables, auxiliary tables)
--@block set global variables (= prices for products according to
subscription type)
SET @films_only = 0.4;
SET @series_only = 0.2;
SET @films_series = 0.3;
SET @series_films = 0.1;
SET @user_now = -1;

-- @block query3.1
CREATE TABLE IF NOT EXISTS z3_1films(
    film_id smallint(5) NOT NULL,
    title varchar(128) NOT NULL
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

CREATE TABLE IF NOT EXISTS z3_1series(
    series_id smallint(25) NOT NULL,
    title varchar(128) NOT NULL
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

CREATE PROCEDURE IF NOT EXISTS zhtoumeno3_1 (IN c enum('m', 's'), IN
number INT, IN startdate DATE, IN enddate DATE)
BEGIN
    DECLARE done INT;
    DECLARE msid SMALLINT(5);
    DECLARE mst VARCHAR(128);

    IF (c = 'm') THEN
    BEGIN

        DECLARE result_m CURSOR FOR SELECT rental.film_id
        FROM rental USE INDEX (rentdate)
        WHERE rental.rental_date BETWEEN startdate AND enddate AND
rental.film_id IS NOT NULL

```

```

GROUP BY rental.film_id
ORDER BY count(rental.film_id) DESC
LIMIT 0, number;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;

SET done = 0;
OPEN result_m;

REPEAT
FETCH result_m INTO msid;
IF (done = 0) THEN
    SELECT title INTO mst
    FROM film
    WHERE film_id = msid;
    INSERT INTO z3_1films VALUES(msid,mst);
END IF;
UNTIL (done =1)
END REPEAT;
CLOSE result_m;

END;
ELSEIF (c = 's') THEN
BEGIN

    DECLARE result_s CURSOR FOR SELECT season.series_id
    FROM season INNER JOIN episode ON season.season_id =
episode.season_id
    INNER JOIN rental ON episode.episode_id = rental.episode_id
    WHERE rental.episode_id IN
    (SELECT rental.episode_id
    FROM rental USE INDEX (rentdate) WHERE rental.rental_date
BETWEEN startdate AND enddate AND rental.episode_id IS NOT NULL
    )
    GROUP BY season.series_id
    ORDER BY count(rental.episode_id) DESC
    LIMIT 0, number;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;

    SET done = 0;
    OPEN result_s;
    REPEAT
    FETCH result_s INTO msid;
    IF (done = 0) THEN
        SELECT title INTO mst
        FROM series
        WHERE series_id= msid;

```

```

            INSERT INTO z3_1series VALUES(msid,mst);
        END IF;
    UNTIL (done =1)
    END REPEAT;
    CLOSE result_s;

    END;
    END IF;
END$

-- @block call zhtoumeno3_1
CALL zhtoumeno3_1('m', 2, '2005-06-15 01:25:08', '2005-9-19 03:42:27');

-- @block zhtoumeno3_2
CREATE PROCEDURE IF NOT EXISTS zhtoumeno3_2(IN em VARCHAR(50), IN d
DATE, OUT rentals SMALLINT(3))
BEGIN
    DECLARE done INT;
    DECLARE cust_id SMALLINT(5) UNSIGNED;

    SELECT customer_id
    INTO cust_id
    FROM customer
    WHERE customer_id IN
    (SELECT user_id FROM user WHERE email=em);

    SELECT count(rental_id) INTO rentals
    FROM rental USE INDEX (fk_rental_customer)
    WHERE rental.customer_id = cust_id AND DATE(rental.rental_date) = d
    GROUP BY rental.customer_id;

END$

-- @block test zhtoumeno3_2
SET @temp=0;
CALL zhtoumeno3_2('MAE.FLETCHER@sakilacustomer.org', '2005-9-19',
@temp);

-- @block zhtoumeno3_3
DROP TABLE z3_3films;
DROP TABLE z3_3series;
CREATE TABLE IF NOT EXISTS z3_3films(
    dates VARCHAR(25) NOT NULL,
    film_payments DECIMAL(7,2) NOT NULL
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

CREATE TABLE IF NOT EXISTS z3_3series(
    dates VARCHAR(25) NOT NULL,

```

```

        episode_payments DECIMAL(7,2) NOT NULL
    ) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

DROP PROCEDURE IF EXISTS zhtoumeno3_3;
CREATE PROCEDURE IF NOT EXISTS zhtoumeno3_3()
BEGIN
    DECLARE done INT;
    DECLARE f_sum DECIMAL(7,2);
    DECLARE f_date VARCHAR(25);
    DECLARE s_sum DECIMAL(7,2);
    DECLARE s_date VARCHAR(25);

    DECLARE films_sum CURSOR FOR
        SELECT sum(payment.amount), EXTRACT(YEAR_MONTH FROM
payment.payment_date)
        FROM payment INNER JOIN rental ON payment.rental_id =
rental.rental_id
        WHERE rental.film_id IS NOT NULL
        GROUP BY EXTRACT(YEAR_MONTH FROM payment.payment_date)
        ORDER BY EXTRACT(YEAR_MONTH FROM payment.payment_date) ASC;

    DECLARE series_sum CURSOR FOR
        SELECT sum(payment.amount), EXTRACT(YEAR_MONTH FROM
payment.payment_date)
        FROM payment INNER JOIN rental ON payment.rental_id =
rental.rental_id
        WHERE rental.film_id IS NULL
        GROUP BY EXTRACT(YEAR_MONTH FROM payment.payment_date)
        ORDER BY EXTRACT(YEAR_MONTH FROM payment.payment_date) ASC;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;
    open films_sum;
    open series_sum;

    delete from z3_3films;
    SET done = 0;
    REPEAT
    FETCH films_sum INTO f_sum, f_date;
    IF (done = 0) THEN
        INSERT INTO z3_3films VALUES (f_date, f_sum);
    END IF;
    UNTIL(done = 1)
    END REPEAT;

    SET done = 0;
    delete from z3_3series;
    REPEAT
    FETCH series_sum INTO s_sum, s_date;

```

```

    IF (done = 0) THEN
        INSERT INTO z3_3series VALUES (s_date, s_sum);
    END IF;
    UNTIL(done = 1)
    END REPEAT;

    SELECT dates AS "DATE", film_payments AS "INCOME FROM FILMS" FROM
z3_3films;
    SELECT dates AS "DATE", episode_payments AS "INCOME FROM SERIES"
FROM z3_3series;
END$

-- @block test zhtoumeno3_3
call zhtoumeno3_3();

-- @block zhtoumeno3_4a
DROP PROCEDURE IF EXISTS zhtoumeno3_4a$
CREATE PROCEDURE IF NOT EXISTS zhtoumeno3_4a(IN lastname1 VARCHAR(45),
IN lastname2 VARCHAR(45))
BEGIN

    SELECT count(actor_id) AS 'Actor Count With Same Lastname'
    FROM actor USE INDEX(lastname)
    WHERE last_name BETWEEN lastname1 AND lastname2;

    SELECT first_name AS "First name", last_name AS "Last name"
    FROM actor USE INDEX(lastname)
    WHERE last_name BETWEEN lastname1 AND lastname2;
END$

-- @block test zhtoumeno3_4a
call zhtoumeno3_4a("Cunt", "Dick");

-- @block zhtoumeno3_4b
CREATE TABLE IF NOT EXISTS z3_4b(
    first_name varchar(45) NOT NULL,
    last_name varchar(45) NOT NULL
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

DROP PROCEDURE IF EXISTS zhtoumeno3_4b;
CREATE PROCEDURE zhtoumeno3_4b(IN lastname VARCHAR(45) )
BEGIN
    DECLARE done INT;
    DECLARE fn VARCHAR(45);
    DECLARE ln VARCHAR(45);

    DECLARE fullname CURSOR FOR
    SELECT first_name, last_name

```



```

FROM actor USE INDEX(lastname)
WHERE actor.last_name = lastname;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;

SET done = 0;
OPEN fullname;
IF (FOUND_ROWS() > 1) THEN
    SELECT FOUND_ROWS() AS 'Count';
END IF;

delete from z3_4b;

REPEAT
FETCH fullname INTO fn, ln;
IF(done = 0) THEN
    INSERT INTO z3_4b VALUES (fn, ln);
END IF;
UNTIL(done =1)
END REPEAT;

CLOSE fullname;
SELECT first_name AS "First Name", last_name AS "Last Name" FROM
z3_4b;
END;

-- @block test zhtoumeno3_4b
call zhtoumeno3_4b("DAVIS");

--@block payment_procedure
CREATE PROCEDURE pay4product(IN productType enum("film", "episode"), IN
custID SMALLINT(5))
BEGIN
    DECLARE customer_type enum ("FILMS ONLY", "SERIES ONLY", "FILMS AND
SERIES");
    DECLARE to_pay DECIMAL(5,2);

    SELECT cust_type INTO customer_type
    FROM customer
    WHERE customer_id = custID;

    IF (customer_type = "FILMS ONLY") THEN
        SET to_pay = @films_only;
    ELSEIF (customer_type = "SERIES ONLY") THEN
        SET to_pay = @series_only;
    ELSE
        IF (productType = "episode") THEN
            SET to_pay = @series_films;
        ELSE

```

```

        SET to_pay = @films_series;
    END IF;

END IF;

SELECT to_pay;
END$

-- DB triggers
-- trigger 4_1 notes
-- deletes on rental/payment made only by the admin
-- -- (we might need the triggers if the admin wants to delete some
records to free up space in the database)
-- before update on rental/payment abort (no one can change a
rental/payment that has already been made)

-- @block rental_made (after insert) IN DB
CREATE TRIGGER afterINSrental AFTER INSERT ON rental
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);

    SELECT email INTO email_now
    FROM user
    WHERE user.user_id = @user_now;

    INSERT INTO LOG VALUES(NULL, email_now, NOW(), "INSERT", 1,
"rental");
END$

-- @block payment_made (after insert) IN DB
CREATE TRIGGER afterINSpayment AFTER INSERT ON payment
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);

    SELECT email INTO email_now
    FROM user
    WHERE user.user_id = @user_now;

    INSERT INTO LOG VALUES(NULL, email_now, NOW(), "INSERT", 1,
"payment");
END$

-- @block delete rental
CREATE TRIGGER beforeDELrental BEFORE DELETE ON rental
FOR EACH ROW

```

```

BEGIN
    DECLARE type_now enum("CUSTOMER", "ADMIN", "EMPLOYEE");
    DECLARE email_now VARCHAR(70);

    SELECT usertype, email INTO type_now, email_now
    FROM user
    WHERE user_id = @user_now;

    IF (type_now != "ADMIN") THEN
        INSERT INTO log VALUES(NULL, email_now, CURDATE(), "DELETE", 0,
"rental");
        SIGNAL SQLSTATE VALUE "45000"
        SET MESSAGE_TEXT ="You are not authorized to change this
value.";
    END IF;

END$

CREATE TRIGGER beforeDELpayment BEFORE DELETE ON payment
FOR EACH ROW
BEGIN
    DECLARE type_now enum("CUSTOMER", "ADMIN", "EMPLOYEE");
    DECLARE email_now VARCHAR(70);

    SELECT usertype, email INTO type_now, email_now
    FROM user
    WHERE user_id = @user_now;
    IF (type_now != "ADMIN") THEN
        INSERT INTO log VALUES(NULL, email_now, CURDATE(), "DELETE", 0,
"payment");
        SIGNAL SQLSTATE VALUE "45000"
        SET MESSAGE_TEXT ="You are not authorized to change this
value.";
    END IF;

END$

CREATE TRIGGER beforeUPrental BEFORE UPDATE ON rental
FOR EACH ROW
BEGIN
    DECLARE type_now enum("CUSTOMER", "ADMIN", "EMPLOYEE");
    DECLARE email_now VARCHAR(70);

    SELECT usertype, email INTO type_now, email_now
    FROM user
    WHERE user_id = @user_now;

    IF (type_now != "ADMIN") THEN

```

```

        INSERT INTO log VALUES(NULL, email_now, CURDATE(), "UPDATE", 0,
"rental");
        SIGNAL SQLSTATE VALUE "45000"
        SET MESSAGE_TEXT ="You are not authorized to change this
value.";
        END IF;

END$

CREATE TRIGGER beforeUPpayment BEFORE UPDATE ON payment
FOR EACH ROW
BEGIN
    DECLARE type_now enum("CUSTOMER", "ADMIN", "EMPLOYEE");
    DECLARE email_now VARCHAR(70);

    SELECT usertype, email INTO type_now, email_now
    FROM user
    WHERE user_id = @user_now;

    IF (type_now != "ADMIN") THEN
        INSERT INTO log VALUES(NULL, email_now, CURDATE(), "UPDATE", 0,
"payment");
        SIGNAL SQLSTATE VALUE "45000"
        SET MESSAGE_TEXT ="You are not authorized to change this
value.";
        END IF;

END$

-- changed mod result (mod =1 means that the rental that is about to
happen is multiple of 3 and deserves discount)
-- @block trigger4_2 IN DB
CREATE TRIGGER discount BEFORE INSERT ON payment
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    DECLARE num_of_rents SMALLINT(3);

    SELECT email INTO email_now
    FROM user
    WHERE user.user_id = NEW.customer_id;

    call zhtoumeno3_2(email_now, DATE(NEW.payment_date), num_of_rents);

    IF num_of_rents % 3 = 0 THEN
        SET NEW.amount = NEW.amount/2;
    END IF;

END$

```

```

-- @block trigger4_3 (before update)
-- a customer can only change his own profile settings through the gui.
An employee can change any customers settings or his own through the
gui
CREATE TRIGGER declineUpdate BEFORE UPDATE ON user
FOR EACH ROW
BEGIN
    DECLARE em VARCHAR(50);
    DECLARE curuser_type enum("ADMIN","EMPLOYEE","CUSTOMER");

    SELECT email, usertype INTO em, curuser_type
    FROM user
    WHERE user.user_id = @user_now;

    IF (curuser_type = "CUSTOMER") THEN
        IF (OLD.usertype != NEW.usertype OR OLD.email != NEW.email OR
OLD.user_id != NEW.user_id) THEN
            SIGNAL SQLSTATE VALUE "45000"
            SET MESSAGE_TEXT = "You are not authorized to change this
value.";
        END IF;
    ELSEIF (curuser_type = "EMPLOYEE") THEN
        IF (OLD.email != NEW.email) THEN
            SIGNAL SQLSTATE VALUE "45000"
            SET MESSAGE_TEXT = "You are not authorized to change this
value.";
        END IF;
    END IF;
END$

-- @block AFTER DELETE TRIGGERS FOR ALL TABLES IN DB
-- actor
CREATE TRIGGER delactor AFTER DELETE ON actor
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"actor");

END$

-- language
CREATE TRIGGER dellang AFTER DELETE ON lang
FOR EACH ROW
BEGIN

```

```

    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"lang");
END$

-- film
CREATE TRIGGER delfilm AFTER DELETE ON film
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"film");

END$

-- film_actor
CREATE TRIGGER delfilm_actor AFTER DELETE ON film_actor
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"film_actor");
END$

-- category
CREATE TRIGGER delcategory AFTER DELETE ON category
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"category");
END$

-- film_category
CREATE TRIGGER delfilm_category AFTER DELETE ON film_category
FOR EACH ROW

```

```

BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"film_category");
END$

-- series
CREATE TRIGGER delseries AFTER DELETE ON series
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"series");
END$

-- season
CREATE TRIGGER delseason AFTER DELETE ON season
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"season");
END$

-- season_actor
CREATE TRIGGER delseason_actor AFTER DELETE ON season_actor
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"season_actor");
END$

-- series_category
CREATE TRIGGER delseries_category AFTER DELETE ON series_category
FOR EACH ROW

```

```

BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"series_category");
END$

```

```

CREATE TRIGGER delepisode AFTER DELETE ON episode
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"episode");
END$

```

```

-- country
CREATE TRIGGER delcountry AFTER DELETE ON country
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"country");
END$

```

```

-- city
CREATE TRIGGER delcity AFTER DELETE ON city
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"city");
END$

```

```

-- address
CREATE TRIGGER deladdress AFTER DELETE ON address
FOR EACH ROW
BEGIN

```



```

DECLARE email_now VARCHAR(50);
SELECT email INTO email_now
FROM user
WHERE user_id=@user_now;
INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"address");
END$

```

```

-- user
CREATE TRIGGER deluser AFTER DELETE ON user
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"user");
END$

```

```

-- customer
CREATE TRIGGER delcustomer AFTER DELETE ON customer
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"customer");
END$

```

```

-- employee
CREATE TRIGGER delemployee AFTER DELETE ON employee
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"employee");
END$

```

```

-- admin
CREATE TRIGGER deladmin AFTER DELETE ON admin
FOR EACH ROW
BEGIN

```

```

    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"admin");
END$

-- rental
CREATE TRIGGER delrental AFTER DELETE ON rental
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"rental");
END$

-- payment
CREATE TRIGGER delpayment AFTER DELETE ON payment
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "DELETE", 1,
"payment");
END$

-- @block AFTER INSERT TRIGGERS FOR ALL TABLES
-- actor
CREATE TRIGGER insactor AFTER INSERT ON actor
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"actor");

END$

-- language
CREATE TRIGGER inslang AFTER INSERT ON lang

```

```

FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"lang");
END$

-- film
CREATE TRIGGER insfilm AFTER INSERT ON film
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"film");

END$

-- film_actor
CREATE TRIGGER insfilm_actor AFTER INSERT ON film_actor
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"film_actor");
END$

-- category
CREATE TRIGGER inscategory AFTER INSERT ON category
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"category");
END$

-- film_category

```

```

CREATE TRIGGER insfilm_category AFTER INSERT ON film_category
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"film_category");
END$

-- series
CREATE TRIGGER insseries AFTER INSERT ON series
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"series");
END$

-- season
CREATE TRIGGER insseason AFTER INSERT ON season
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"season");
END$

-- season_actor
CREATE TRIGGER insseason_actor AFTER INSERT ON season_actor
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"season_actor");
END$

-- series_category

```

```

CREATE TRIGGER insseries_category AFTER INSERT ON series_category
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"series_category");
END$

```

```

CREATE TRIGGER insepisode AFTER INSERT ON episode
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"episode");
END$

```

```

-- country
CREATE TRIGGER inscountry AFTER INSERT ON country
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"country");
END$

```

```

-- city
CREATE TRIGGER inscity AFTER INSERT ON city
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"city");
END$

```

```

-- address
CREATE TRIGGER insaddress AFTER INSERT ON address

```

```

FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"address");
END$

-- user
CREATE TRIGGER insuser AFTER INSERT ON user
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"user");
END$

-- customer
CREATE TRIGGER inscustomer AFTER INSERT ON customer
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"customer");
END$

-- employee
CREATE TRIGGER insemployee AFTER INSERT ON employee
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"employee");
END$

-- admin
CREATE TRIGGER insadmin AFTER INSERT ON admin

```

```

FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "INSERT", 1,
"admin");
END$

-- (rental + payment exists above, according to query 4.1)

-- @block AFTER UPDATE TRIGGERS FOR ALL TABLES
CREATE TRIGGER upactor AFTER UPDATE ON actor
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"actor");

END$

-- language
CREATE TRIGGER uplang AFTER UPDATE ON lang
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"lang");
END$

-- film
CREATE TRIGGER upfilm AFTER UPDATE ON film
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"film");

```

```

END$

-- film_actor
CREATE TRIGGER upfilm_actor AFTER UPDATE ON film_actor
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"film_actor");
END$

-- category
CREATE TRIGGER upcategory AFTER UPDATE ON category
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"category");
END$

-- film_category
CREATE TRIGGER upfilm_category AFTER UPDATE ON film_category
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"film_category");
END$

-- series
CREATE TRIGGER upseries AFTER UPDATE ON series
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"series");

```



```

END$

-- season
CREATE TRIGGER upseason AFTER UPDATE ON season
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"season");
END$

-- season_actor
CREATE TRIGGER upseason_actor AFTER UPDATE ON season_actor
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"season_actor");
END$

-- series_category
CREATE TRIGGER upseries_category AFTER UPDATE ON series_category
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"series_category");
END$

CREATE TRIGGER upepisode AFTER UPDATE ON episode
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"episode");
END$

```

```

-- country
CREATE TRIGGER upcountry AFTER UPDATE ON country
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"country");
END$

```

```

-- city
CREATE TRIGGER upcity AFTER UPDATE ON city
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"city");
END$

```

```

-- address
CREATE TRIGGER upaddress AFTER UPDATE ON address
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"address");
END$

```

```

-- user
CREATE TRIGGER upuser AFTER UPDATE ON user
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"user");
END$

```

```

-- customer
CREATE TRIGGER upcustomer AFTER UPDATE ON customer
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"customer");
END$

-- employee
CREATE TRIGGER upemployee AFTER UPDATE ON employee
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"employee");
END$

-- admin
CREATE TRIGGER upadmin AFTER UPDATE ON admin
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"admin");
END$

-- rental
CREATE TRIGGER uprental AFTER UPDATE ON rental
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"rental");
END$

```

```
-- payment
CREATE TRIGGER uppayment AFTER UPDATE ON payment
FOR EACH ROW
BEGIN
    DECLARE email_now VARCHAR(50);
    SELECT email INTO email_now
    FROM user
    WHERE user_id=@user_now;
    INSERT INTO log VALUES (NULL, email_now, NOW(), "UPDATE", 1,
"payment");
END$
```

# ΤΕΚΜΗΡΙΩΣΗ GUI

## Εισαγωγή :

Το κομμάτι αυτό της αναφοράς μας θα επικεντρωθεί στο γραφικό μέρος του Project και την σύνδεση αυτού με την βάση δεδομένων. Θα αναλύσουμε βηματικά τις σχεδιαστικές επιλογές καθώς και ανάδειξη του τρόπου χρήσης και των δυνατοτήτων του γραφικού περιβάλλοντος. Για την υλοποίηση του project δεν χρησιμοποιήθηκε κάποιο εξειδικευμένο java IDE ( Integrated Development Environment, όπως το Netbeans, Eclipse κ.α. ) αλλά το vscode με τα απαραίτητα java extensions. Θεωρούμε σημαντικό να προσθέσουμε εδώ πως απαραίτητη προϋπόθεση για την λειτουργία του Project μας αποτελούν: Η ορθή ύπαρξη της βάσης, java runtime environment, το service της mariadb server και τέλος η ύπαρξη της javafx βιβλιοθήκης.

Ειδικότερα για την υλοποίηση επιλέξαμε το maven java archetype καθώς είναι ένα από τα πιο διαδεδομένα και σύγχρονα archetypes και dependency managers. Η δομή του archetype ακολουθήθηκε αυστηρά με στόχο το modularity και το reparability του κώδικα. Για την δημιουργία των γραφικών χρησιμοποιήθηκε η βιβλιοθήκη javafx. Επιλέξαμε την javafx υπερ της swing καθώς έχει πιο εκσυγχρονισμένα στοιχεία, έχει εδραιωθεί τόσο σε desktop εφαρμογές όσο και στο web σε εφαρμογές δομημένες στην java, είναι crossplatform και τέλος υποστηρίζει styling με την χρήση css. Για την δημιουργία των FXML resources χρησιμοποιήθηκε το εργαλείο Scenebuilder.

Επιπροσθέτως θα θέλαμε να επισημάνουμε πως ο κώδικας της java δεν είναι ούτε optimized ούτε scalable για εφαρμογές σε big data. Θεωρήσαμε πιο σημαντική την δημιουργία ενός όσο το δυνατότερο εύχρηστου και λειτουργικού UI ενώ κρίνουμε πως το optimization είναι εκτός των απαιτήσεων του project. Αναγνωρίζουμε περιοχές που ο κώδικας είναι επιρρεπής σε sql injection attacks, ωστόσο λόγω της πίεσης του χρόνου και του μεγάλου όγκου του Project δεν εκτελέσαμε εκτενή αμυντικό προγραμματισμό. Τέλος τα παράθυρα που δημιουργούνται κατά το gui δεν είναι scalable αλλά undecorated. Δεν ήμασταν ικανοποιημένοι από το Scaling των στοιχείων καθώς αλλάζει το μέγεθος του παραθύρου, δουλειά που αποδείχθηκε υπερβολικά χρονοβόρα και δύσκολη. Για αυτό επιλέξαμε τα μεγέθη των παραθύρων να είναι fixed καθώς και να χωράνε ακόμα και σε μικρές αναλύσεις οθόνης. Ως εκ τούτου περιοριστήκαμε σε διαστάσεις λίγο μικρότερες του 1366x768<sub>px</sub>.

Εν κατακλείδι παρόλο τα όσα αναφέραμε θεωρούμε πως επικεντρωθήκαμε και προσθέσαμε στοιχεία λειτουργικότητας καθώς πήραμε αποφάσεις και κάνουμε παραδοχές για τον εμπλουτισμό τους. Προσωπικά κρίνουμε πως η εφαρμογή είναι πλήρως λειτουργική και ελπίζουμε να καλύπτει ουσιαστικά τις ζητούμενες προδιαγραφές.

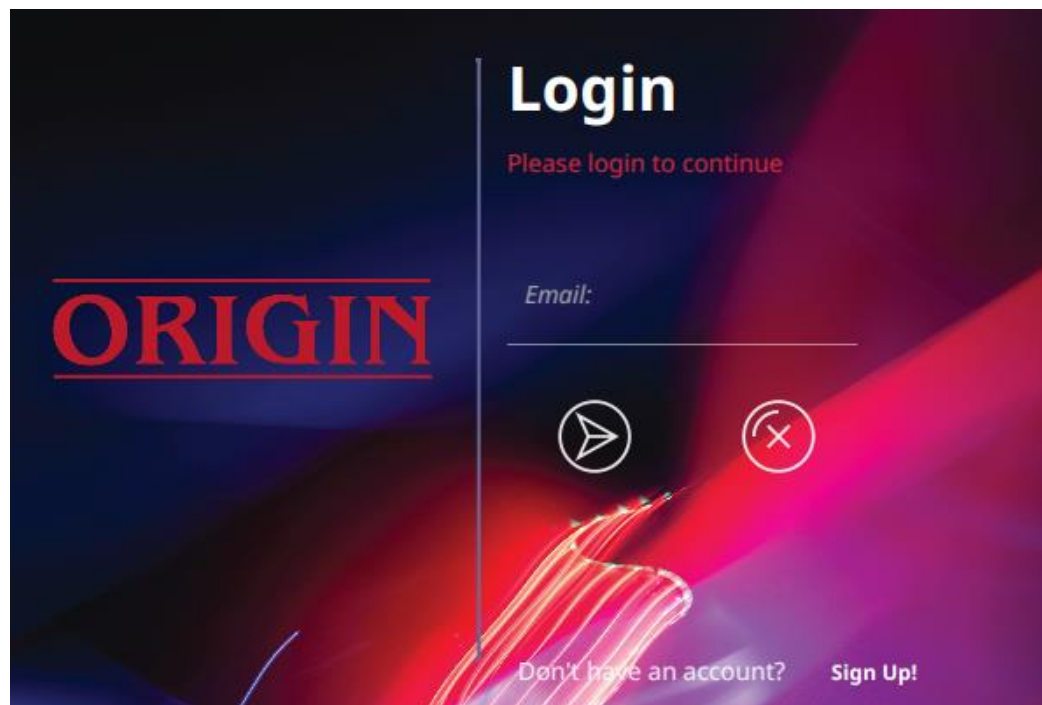
#### Παραδοχές :

- 1) Επισημαίνεται στα ζητούμενα πως μόνο ο admin έχει την δυνατότητα να δημιουργήσει πελάτες. Για λόγους λειτουργικότητας και εμπλουτισμού κατά την διαδικασία του log-in, υπάρχει και η δυνατότητα για sign-up όπου ο λογαριασμός δεν θεωρείται ενεργός αλλά περιμένει έγκριση από κάποιο διαχειριστή
- 2) Η χρήση του πεδίου active στον user μας παρουσίασε ένα πρόβλημα ερμηνείας. Είχαμε δύο επιλογές, είτε με την έννοια του ενεργού θα παρουσιάζουμε ποιοί χρήστες είναι κάθε στιγμή συνδεδεμένοι, είτε το αν ο λογαριασμός του κάθε χρήστη είναι active/inactive. Με το δεύτερο υλοποιούμε μία μορφή ban/suspend χωρίς την απαραίτητη οριστική διαγραφή από την βάση που μπορεί να κάνει μόνο ο Admin. Κρίναμε πως το δεύτερο πρόσθετε περισσότερο functionality στην εφαρμογή μας.

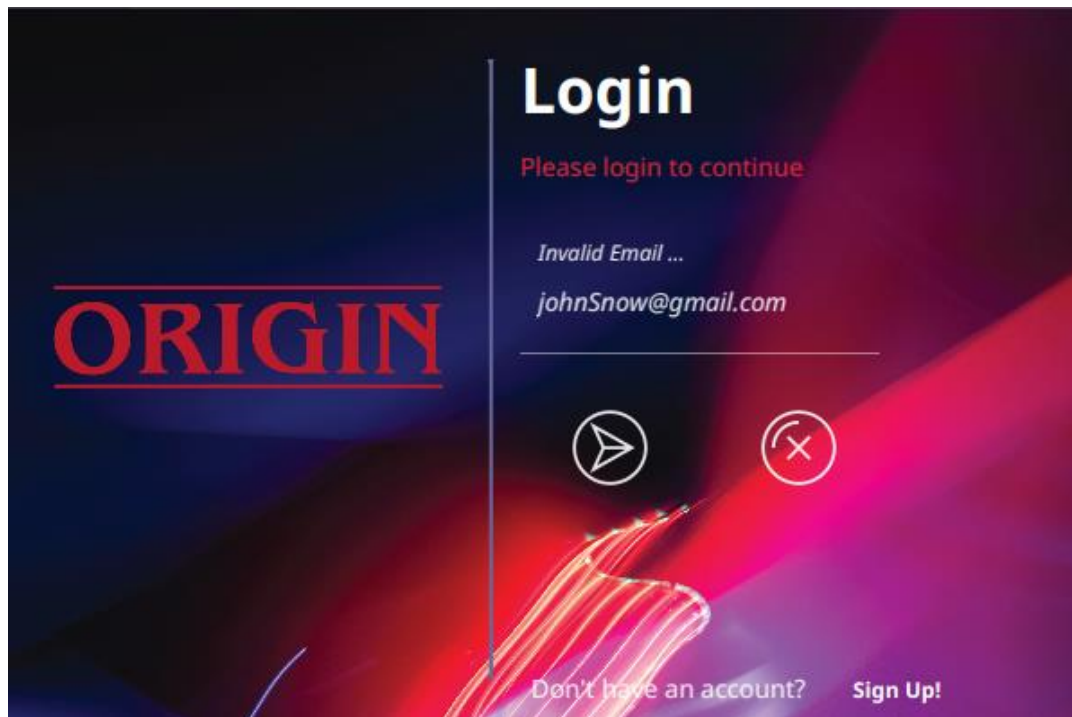
### Login Page:

Με στόχο τόσο ένα intuitive όσο και ένα καλαίσθητο αποτέλεσμα επιλέξαμε την χρήση μίας πιο μικρής και μινιμαλιστικής πρώτης εικόνας.

Στο δεξί μέρος του παραθύρου, παρατηρούνται τα περισσότερα στοιχεία που παρέχουν δυναμική αλληλεπίδραση. Στο κέντρο παρατηρείται ένα textfield που επιτρέπει στον χρήστη να παραθέσει το αναγνωριστικό του.



Με enter ή με την χρήση του button ( δεξιό βελάκι ) γίνεται προσπάθεια πρόσβασης στην βάση. Σε αποτυχημένη προσπάθεια ή σε περίπτωση που το email είναι λάθος εμφανίζεται κατάλληλο error message που ειδοποιεί τον χρήστη. Με το κουμπί ( X ) ο χρήστης μπορεί να κλείσει εύκολα από το πρόγραμμα.



Τα περισσότερα κουμπιά θα παρατηρήσετε πως σε όλη την διάρκεια της εφαρμογής έχουν hover και pressed effects για να γίνεται ορατή η αλληλεπίδραση μαζί τους από τον χρήστη. Συνήθως σε hover παρουσιάζεται κάποιο displacement της οθόνης ενώ on press περιβάλλονται από ένα κόκκινο τετραγωνάκι.

Στο κάτω δεξιά κομμάτι του παραθύρου με έντονα γράμματα υπάρχει η ένδειξη για sign-up η οποία με το πάτημα της οδηγεί σε μεταβολή του παραθύρου :



Δίνεται λοιπόν στον χρήστη η δυνατότητα να συμπληρώσει τα στοιχεία του, τα βελάκια πάνω δεξιά κάνουν αντίστοιχα submit ή cancel το sign-up. Προβλέπουμε λανθασμένη μορφή για email μέσω regex ενώ σε περίπτωση κάποιου άλλου λάθους παρουσιάζεται κατάλληλο μήνυμα στον χρήστη. Είναι υποχρεωμένος πάντα να συμπληρώσει όλα τα απαραίτητα στοιχεία του, παρά το κομμάτι του τύπου του. Καθώς το customer type είναι enum παρουσιάζουμε στον χρήστη μόνο τις συγκεκριμένες σωστές επιλογές σύμφωνα και με το ζητούμενο B2. Παρόμοιες ενέργειες με choice boxes χρησιμοποιούνται σε καθ' όλη την διάρκεια της υλοποίησης του project.



Με χρήση tabs ( ή shift+tabs ) πέρα από το ποντίκι φυσικά καθίσταται δυνατό το navigation μεταξύ των πεδίων. Στο τελευταίο πεδίο αν πατηθεί “enter” γίνεται αυτόματα η προσπάθεια submit του καινούργιου λογαριασμού. Έχει γίνει προσπάθεια τέτοια keyboard shortcuts να διατηρηθούν όσο γίνονται και στην υλοποίηση που ακολουθεί.

Τέλος με την εισχώρηση ορθού email καταλαβαίνουμε το είδος του χρήστη και τον προωθούμε σε καινούργιο παράθυρο, είτε ως customer, είτε ως admin, είτε ως employee. Το αρχικό παράθυρο login κλείνει.

◆ Ταχεία περιγραφή κώδικα :

Καθώς ο κώδικας είναι αρχικά περίπλοκος και υπερβολικά μεγάλος σε μέγεθος θα είναι πολύ δύσκολο να γίνει μία λεπτομερής ανάλυση. Θα αναφερθούμε δειγματικά και με βάση σημασίας για τον κώδικα στα key points. Κρίνουμε ωστόσο πως τόσο τα σχόλια στον κώδικα όσο και τα ονόματα των συναρτήσεων μας κάνουν αντιληπτή την λειτουργία του.

Αρχικά ο κορμός του maven archetype είναι τα πακέτα και το App.java. Σε αυτό παρατηρείται η δημιουργία του αρχικού primary stage καθώς και η χρήση του αντίστοιχου resource για το άνοιγμα του login Page. Είναι σημαντικό να αναφερθεί πως κάθε καινούργιο γραφικό παράθυρο απαιτεί μία καινούργια κλάση - έναν controller ο οποίος θα συσχετίσει τα γραφικά στοιχεία με συναρτήσεις και event handlers. Καθώς ένα καινούργιο fxml ανοίγει γίνεται instanced και η κλάση του controller του. Πέρα από τις συγκεκριμένες κλάσεις ωστόσο θα παρατηρήσουμε βοηθητικές κλάσεις όπως πχ κλάσεις για την αναπαράσταση των models και βοηθητικές κλάσεις όπως την DatabaseConnection.java.

Κρίνουμε σημαντικό να αρχίσουμε με την DatabaseConnection.java καθώς αυτή καλεί τον driver Manager jdbc και δημιουργεί ένα καινούργιο connection με την βάση δεδομένων μας σύμφωνα με το οποίο θα γίνει και κάθε αλληλεπίδραση της βάσης και του mariadb server. Το όνομα της βάσης, το username και ο κωδικός είναι παραμετροποιημένοι για να βοηθήσουν στο testing και στο flexibility του κώδικα. Οι πληροφορίες αυτές θα μπορούσαν να παρέχονται και ως execution arguments κατά την εκτέλεση του προγράμματος.

Επιστέφοντας στον κώδικα του LoginPageController αρχικά φαίνεται η χρήση του @FXML πριν από methods και members. Ο,τι ακολουθεί το συγκεκριμένο annotation συμπληρώνεται με metadata για το compilation και το runtime. Συνδέεται άμεσα με την javafx είτε ως στοιχείο πίνακα είτε ως συνάρτηση - event handler που χρησιμοποιείται από κάποιο στοιχείο on action. Η initialize συνάρτηση είναι μία ιδιαίτερη συνάρτηση καθώς εκτελείται για την αρχικοποίηση του controller και του αντίστοιχου fxml του. Συμπληρωματικά με τα βασικά στοιχεία της javafx χρησιμοποιούμε ορισμένα animations, με βασικότερο το duration στην εμφάνιση των μηνυμάτων λάθους.

Στον Login Page Controller διακρίνονται δύο βασικές συναρτήσεις, η `validateLogin()` και η `submitButtonOnAction()`. Η πρώτη εμφανίζεται στην διαδικασία επιβεβαίωσης ύπαρξης του χρήστη, έλεγχου του αν είναι ενεργός /ανενεργός, ή αν περιμένει έγκριση. Στην περίπτωση επιτυχημένης δυνατότητας εισόδου στην βάση ελέγχει το είδος του χρήστη που συνδέθηκε και τον μεταβαίνει στο παράθυρο που του αντιστοιχεί. Η δεύτερη συνάρτηση αφορά το sign-up ενός χρήστη. Δημιουργούνται όλα τα απαραίτητα records στην βάση ωστόσο το στοιχείο pending στον user είναι true. Ο συγκεκριμένος καινούργιος χρήστης φυσικά δεν μπορεί να κάνει login και πρέπει να περιμένει έγκριση. Μέσω της συνάρτησης γίνονται πολλοί έλεγχοι για το αν τα πεδία έχουν την απαραίτητη πληροφορία, ένα ορθό email κ.α. Σημαντικότερο εξ' αυτών είναι πως για λόγους αμυντικού προγραμματισμού, όταν επηρεάζουμε περισσότερους απο έναν πίνακες μετατρέπουμε το `autocommit` στο `connection` false. Σε περίπτωση που γίνει κάποιο λάθος στην συνέχεια της διαδικασίας καθώς καμία αλλαγή δεν έχει γίνει committed μπορούμε να κάνουμε rollback στο state του connection.

## Customer Home Page:

Μόλις ένας πελάτης συνδεθεί στην υπηρεσία, εμφανίζεται η αρχική του οθόνη:



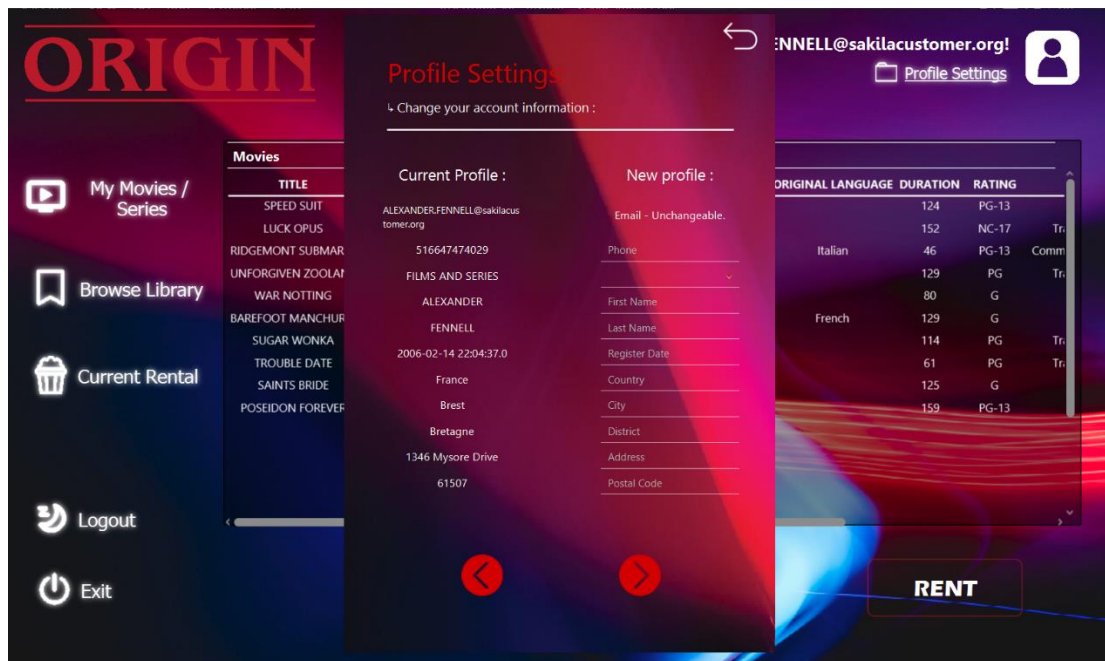
Όπως γίνεται αντιληπτό, το GUI του πελάτη χωρίζεται σε 5 βασικές ενότητες που ορίζουν τα κουμπιά στα αριστερά της οθόνης μαζί με το κουμπί στο κάτω μέρος της λεζάντας καλωσορίσματος:

- Ρυθμίσεις Προφίλ | Profile Settings
- Οι Ταινίες/Σειρές μου | My Movies/Series
- Αναζήτηση στη Βιβλιοθήκη | Browse Library
- Τρέχουσα Ενοικίαση | Current Rental
- Έξοδος από την υπηρεσία

Κάθε μία από τις παρακάτω ενότητες αναλύεται ξεχωριστά:

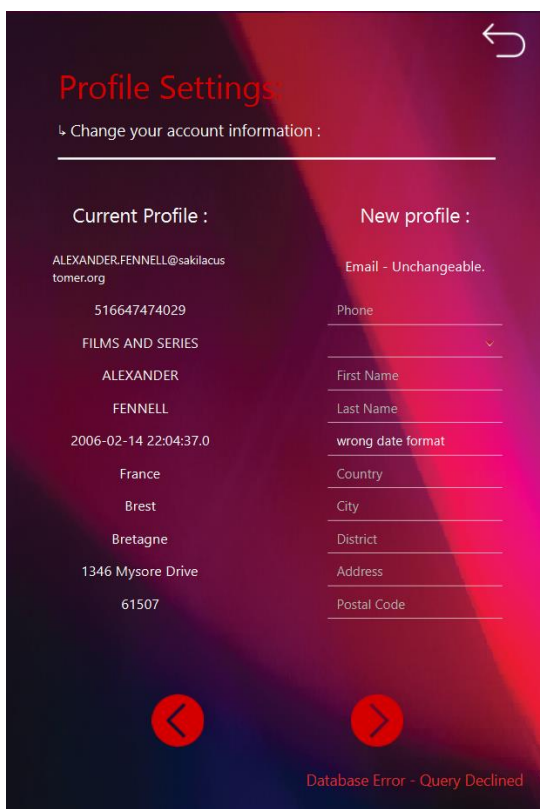
### Ρυθμίσεις Προφίλ | Profile Settings

Σε αυτή την ενότητα οδηγεί το κουμπί που βρίσκεται κάτω από την λεζάντα καλωσορίσματος, στο πάνω δεξιό μέρος της οθόνης. Χάρης σε αυτό το κουμπί, ο πελάτης έχει τη δυνατότητα να δει και να επεξεργαστεί τα στοιχεία του προφίλ του, τα οποία εμφανίζονται σε pop-up παράθυρο.



### Κύριο παράθυρο

Στα αριστερά φαίνονται τα τρέχοντα στοιχεία του πελάτη και στα δεξιά υπάρχουν πεδία για εισαγωγή των νέων στοιχείων. Παρατηρούμε ότι όλα τα στοιχεία εκτός από το email είναι διαθέσιμα προς επεξεργασία. Ο πελάτης με τον οποίο έχουμε συνδεθεί έχει *FILMS AND SERIES subscription*, που σημαίνει ότι δικαιούται να νοικιάσει τόσο ταινίες, όσο και επεισόδια σειρών. Το πεδίο αυτό είναι προφανώς διαθέσιμο προς επεξεργασία, για την περίπτωση που κάποιος πελάτης θελήσει να αλλάξει τον τύπο της συνδρομής του.



### Βέλη (κάτω μέρος της οθόνης)

Το βελάκι προς τα δεξιά επιβεβαιώνει τις αλλαγές εφόσον τα πεδία είναι έγκυρα. Σε περίπτωση μη έγκυρων νέων στοιχείων (πχ ημερομηνία με λάθος format), τυπώνεται μήνυμα λάθους κάτω δεξιά στην οθόνη, που δηλώνει απόρριψη του αιτήματος αλλαγής στοιχείων. Από την άλλη, αν το κουμπί πατηθεί με τα πεδία κενά, τότε διατηρούνται οι παλιές τιμές των πεδίων. Σε κάθε περίπτωση (εκτός από αυτή των λάθος πεδίων), γίνεται η ανανέωση των στοιχείων (έστω και αν αυτά παρέμειναν ίδια)



και το παράθυρο κλείνει.

Το βελάκι προς τα αριστερά κλείνει το παράθυρο επιστρέφει τον χρήστη στην αρχική οθόνη του πελάτη.

[Βέλος \(πάνω δεξιά μέρος της οθόνης\)](#)

Πρόκειται για βελάκι επιστροφής στην αρχική οθόνη του πελάτη, που κλείνει το παράθυρο.

### [Οι Ταινίες/Σειρές μου | My Movies/Series](#)

Σε αυτή την ενότητα οδηγεί το πρώτο κουμπί στο αριστερό μέρος της οθόνης. Το πάτημα αυτού του κουμπιού οδηγεί στην εμφάνιση πινάκων με τις ενοικιάσεις του πελάτη για ταινίες και επεισόδια.

PRICE	RENTAL DATE	TITLE	DURATION
0.4	2005-06-11 01:25:08.0	TROUBLE DATE	61
0.3	2022-08-23 01:22:27.0	WAR NOTTING	80
0.3	2022-08-23 01:22:27.0	BAREFOOT MANCHURIAN	129
0.3	2022-08-25 15:41:27.0	RIDGEMONT SUBMARINE	46

### [Κύριο παράθυρο](#)

Οι πίνακες ενοικιάσεων ταινιών και επεισοδίων εμφανίζονται σε ξεχωριστά παράθυρα, με πληροφορίες όπως, η τιμή που πλήρωσε ο πελάτης για να αποκτήσει το προϊόν, η ημερομηνία ενοικίασης, ο τίτλος και η διάρκεια της ταινίας/του επεισοδίου. Για τα επεισόδια παρέχονται επιπλέον πληροφορίες, όπως ο αριθμός του επεισοδίου, ο αριθμός της σεζόν και ο τίτλος της σειράς στην οποία αυτό ανήκει.

Movies		Episodes				
PRICE	RENTAL DATE ▼	TITLE	EPISODE NUM	DURATION	SEASON NUM	SERIES TITLE
0.05	2022-08-23 01:22:53.0	Chapter 2	2	44	2	The depressive film

Επισημαίνεται ότι οι στήλες έχουν συμπυκωθεί προκειμένου να φαίνεται όλη η πληροφορία στην εικόνα. Στην πραγματικότητα οι στήλες είναι πολύ πιο πλατιές και η εμφάνιση τους γίνεται με τη μετακίνηση του οριζόντιου scroll-bar στο κάτω μέρος του πίνακα.

#### Κουμπί “show info”

Επιλέγοντας ένα επεισόδιο (όπως έχουμε κάνει στην εικόνα παραπάνω), ή μία ταινία, ο πελάτης έχει τη δυνατότητα να δει περισσότερες πληροφορίες για το προϊόν, πατώντας το κουμπί “show info”. Το κουμπί αυτό τον μεταφέρει στην «βιβλιοθήκη» της υπηρεσίας και επιλέγει το προϊόν το οποίο ο πελάτης επιθυμεί να δει.

ORIGIN		Hello ALEXANDER.FENNELL@sakilacustomer.org!		Profile Settings	
EPISODE NO	TITLE	DURATION	RATING		
1	Chapter 1	42	G		
2	Chapter 2	44	PG		
3	Chapter 3	43	PG-13		

Από εκεί, ο πελάτης μπορεί να νοικιάσει την ταινία ή το επεισόδιο, ή να πλοηγηθεί στη βιβλιοθήκη, πάντα διατηρώντας επιλεγμένο το κατάλληλο προϊόν (στην περίπτωση της εικόνας το πλήκτρο back θα οδηγήσει σε πίνακα των season της σειράς στην οποία ανήκει το επεισόδιο και η σεζόν που περιέχει το επεισόδιο θα παρουσιάζεται επιλεγμένη) και σύμφωνα με τις δυνατότητες πλοήγησης που παρέχει η ενότητα «Αναζήτηση στη Βιβλιοθήκη», που αναλύεται αμέσως μετά.

### Αναζήτηση στη Βιβλιοθήκη | Browse Library

Η βιβλιοθήκη είναι αυτή που εμφανίζεται πρώτη στην αρχική οθόνη, με την είσοδο του πελάτη στην υπηρεσία και μπορεί να επανεμφανιστεί σε μεταγενέστερο χρόνο, με το πάτημα του ομώνυμου κουμπιού στο αριστερό τμήμα της αρχικής οθόνης. Ουσιαστικά πρόκειται για τους καταλόγους ταινιών και σειρών που προσφέρονται από την υπηρεσία.

ORIGIN

Hello ALEXANDER.FENNEL@sakilacustomer.org! [Profile Settings](#)

**My Movies / Series**

**Browse Library**

**Current Rental**

**Logout**

**Exit**

Movies		Series				
TITLE	DESCRIPTION	RELEASE YEAR	LANGUA...	ORIGINAL LANGUAGE	DURATION	RATING
SPEED SUIT	A Brilliant Display of a Frisbee And a Mad Scienti...	2006	English		124	PG-13
LUCK OPUS	A Boring Display of a Moose And a Squirrel who ...	2006	English		152	NC-17
RIDGEMONT SUBMARI...	A Unbelievable Drama of a Waitress And a Com...	2006	English	Italian	46	PG-13
UNFORGIVEN ZOOLAN...	A Taut Epistle of a Monkey And a Sumo Wrestler...	2006	English		129	PG
WAR NOTTING	A Boring Drama of a Teacher And a Sumo Westl...	2006	English		80	G
BAREFOOT MANCHURI...	A Intrepid Story of a Cat And a Student who mus...	2006	English	French	129	G
SUGAR WONKA	A Touching Story of a Dentist And a Database A...	2006	English		114	PG
TROUBLE DATE	A Lacklustre Panorama of a Foterste Psychologi...	2006	English		61	PG
SAINTS BRIDE	A Fateful Tale of a Technical Writer And a Comp...	2006	Italian		125	G
POSEIDON FOREVER	A Thoughtful Epistle of a Womanizer And a Mon...	2006	German		159	PG-13

**RENT**

Εικόνα BL1 – Ταινίες: Μετακινήστε το οριζόντιο scroll-bar στο κάτω μέρος του πίνακα για να εμφανίσετε όλες στήλες δεν φαίνονται.





Εικόνα BL2 - Σειρές

### Πίνακες

Οι κατάλογοι εμφανίζονται σε πίνακες που βρίσκονται σε ξεχωριστά Tabs και περιέχουν όλες τις πληροφορίες που υπάρχουν στη βάση για ταινίες και σειρές (εκτός από τους πεδία κωδικών (id), που δεν παρουσιάζονται γενικά στον πελάτη).

### Κουμπί "RENT" (Εικόνα BL1 - Ταινίες)

Εφόσον έχει επιλεγεί μία ταινία από τον πίνακα, το κουμπί αυτό προσθέτει την ταινία σε ένα πίνακα που διατηρεί τα προϊόντα της τρέχουσας ενοικίασης και ο οποίος φαίνεται στην ενότητα «Τρέχουσα ενοικίαση» που αναλύεται παρακάτω.

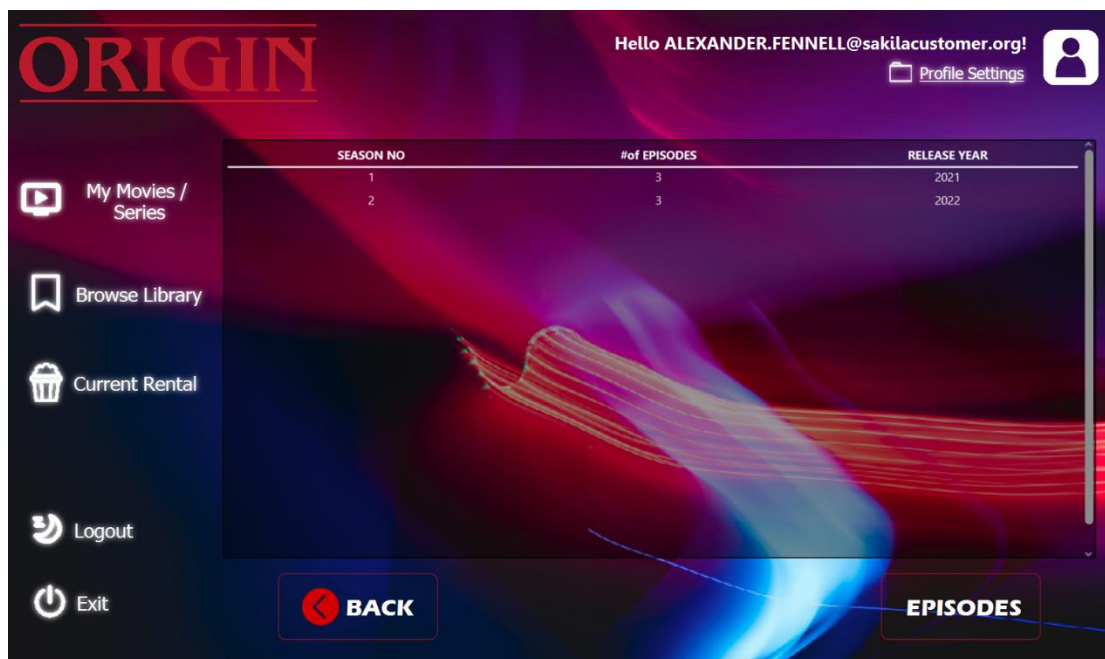
βλ. και Σημείωση για τα κουμπιά "RENT" .51

### Κουμπί "VIEW SEASONS" (Εικόνα BL2 - Σειρές)

Εφόσον έχει επιλεγεί μία σειρά από τον πίνακα, το κουμπί αυτό φανερώνει επόμενο πίνακα, με τις σεζόν της επιλεγμένης σειράς. Έστω ότι ο πελάτης επιλέγει την τρίτη σειρά και στη συνέχεια πατάει το κουμπί.

## Αναζήτηση στη Βιβλιοθήκη - Σεζόν | Browse Library – Seasons

Η συγκεκριμένη οθόνη εμφανίζεται με το πάτημα του κουμπιού “View Seasons” στην «Αναζήτηση στη Βιβλιοθήκη», στο Tab των σειρών.



Εικόνα BL3 – Σεζόν Επιλεγμένης Σειράς

### Πίνακας

Ο πίνακας παρουσιάζει τις πληροφορίες για τις σεζόν της σειράς που επιλέχθηκε και συγκεκριμένα τον αριθμό της σεζόν (1<sup>η</sup>, 2<sup>η</sup> κλπ), το πλήθος των επεισοδίων που αυτή περιέχει και το έτος κυκλοφορίας.

### Κουμπί “BACK”

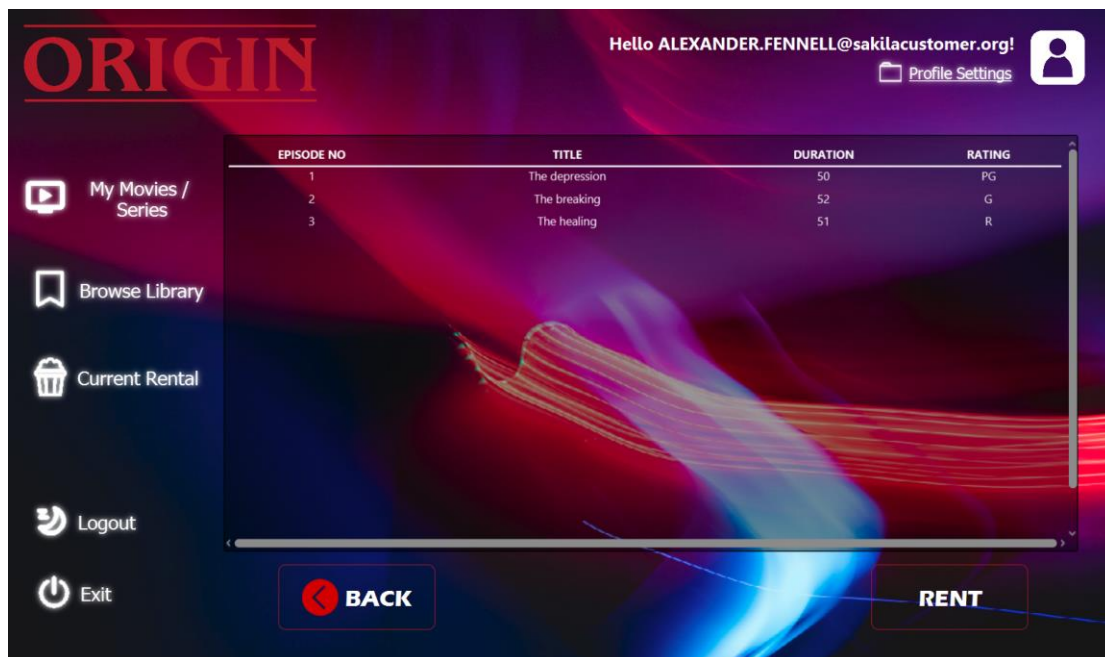
Το κουμπί αυτό προκαλεί επιστροφή στις σειρές, υπογραμμίζοντας τη σειρά που είχε επιλέξει ο χρήστης για να δει τις σεζόν.

### Κουμπί “EPISODES”

Το κουμπί αυτό φανερώνει τα επεισόδια μιας επιλεγμένης σεζόν. Έστω ότι ο πελάτης επέλεξε την πρώτη σεζόν της εικόνας BL3 και στη συνέχεια πάτησε το κουμπί.

## Αναζήτηση στη Βιβλιοθήκη - Επεισόδια | Browse Library – Episodes

Η υποενότητα αυτή ενεργοποιείται και το γραφικό περιβάλλον που την υλοποιεί εμφανίζεται με το πάτημα του κουμπιού “View Episodes” από το γραφικό περιβάλλον της υποενότητας «Αναζήτηση στη Βιβλιοθήκη - Σεζόν» (Εικόνα BL3 παραπάνω).



Εικόνα BL4 – Επεισόδια Επιλεγμένης Σεζόν

#### Πίνακας

Ο πίνακας παρουσιάζει τις πληροφορίες για τα επεισόδια της σεζόν που επιλέχθηκε όπως τον αριθμό επεισοδίου (1<sup>ο</sup>, 2<sup>ο</sup> επεισόδιο κλπ), τον τίτλο και τη διάρκεια του, καθώς και την σήμανση καταλληλότητας.

#### Κουμπί “BACK”

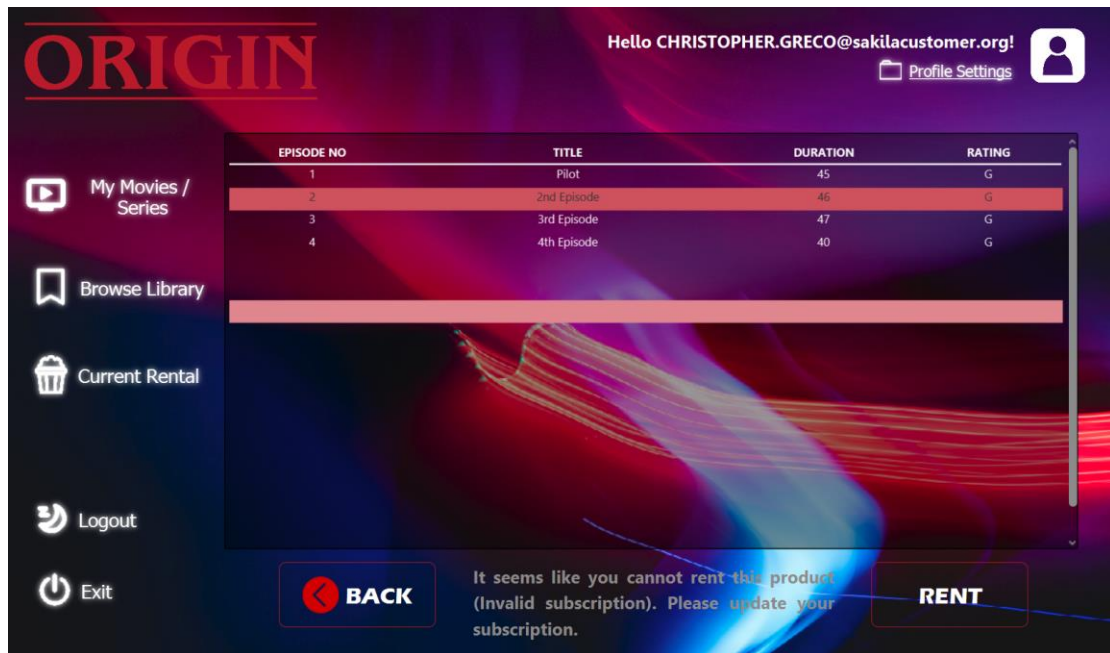
Το κουμπί αυτό προκαλεί επιστροφή στις σεζόν, υπογραμμίζοντας την επιλογή που είχε κάνει ο χρήστης για να δει τα επεισόδια.

#### Κουμπί “RENT”

Όπως και για τις ταινίες, το κουμπί αυτό προσθέτει ένα επιλεγμένο επεισόδιο στον πίνακα της τρέχουσας ενοικίασης, ενότητα η οποία αναλύεται παρακάτω.

#### Σημείωση για τα κουμπιά “RENT”

Η προσθήκη ενός προϊόντος στην «Τρέχουσα Ενοικίαση» επιτρέπεται μόνο εφόσον ο τύπος συνδρομής του πελάτη (“FILMS ONLY”, “SERIES ONLY”, “FILMS AND SERIES”), είναι σωστός, σύμφωνα με τις απαιτήσεις του project. Στην περίπτωση μας, έχουμε συνδεθεί με λογαριασμό πελάτη τύπου “FILMS AND SERIES” (όπως είδαμε και από τα Profile Settings) οπότε οι ενοικιάσεις όλων των προϊόντων είναι επιτρεπτές. Στην περίπτωση που η συνδρομή ήταν άλλου τύπου και ο πελάτης επιχειρούσε να νοικιάσει μη έγκυρο για τον τύπο του προϊόν, θα εμφανιζόταν στην οθόνη επεξηγηματικό μήνυμα, όπως φαίνεται στην εικόνα παρακάτω:



Σημειώνεται ότι έχουμε συνδεθεί με λογαριασμό άλλου πελάτη, τύπου "FILMS ONLY"

Ο νέος πελάτης είναι τύπου "FILMS ONLY" που επιχειρεί να νοικιάσει το επεισόδιο που φαίνεται επιλεγμένο στην εικόνα. Το μήνυμα που παρατηρείται ανακοινώνει στο χρήστη ότι η εγγραφή του δεν είναι έγκυρη για την ενοικίαση αυτού του προϊόντος και του ζητά να την αναβαθμίσει.

### Τρέχουσα Ενοικίαση | Current Rental

Τα γραφικά που υλοποιούν αυτή την ενότητα εμφανίζονται με το πάτημα του τρίτου κουμπιού στο αριστερό μέρος της οθόνης. Περιλαμβάνει όλα τα προϊόντα που επέλεξε να νοικιάσει ο πελάτης μέχρι να πατήσει το προαναφερόμενο κουμπί, λειτουργώντας σαν «καλάθι αγορών». Με το που εμφανίζεται η οθόνη τρέχουσας ενοικίασης, υπολογίζεται και το συνολικό ποσό χρέωσης "Price Total", σύμφωνα με τις τιμές που ισχύουν για τον τύπο του πελάτη "Customer Type". Οι πληροφορίες που αναφέρθηκαν τελευταίες φαίνονται στο δεξί μέρος της οθόνης, όπως φαίνεται στην επόμενη εικόνα.





Σημειώνεται ότι έχουμε επιστρέψει στο λογαριασμό του πρώτου πελάτη, τύπου “FILMS AND SERIES”

#### Πίνακας

Περιέχει τα προϊόντα προς ενοικίαση μαζί με πληροφορίες όπως το είδος τους (ταινία/επεισόδιο), τον τίτλο και την τιμή τους.

#### Κουμπί “SHOW INFO”

Μεταφέρει τον πελάτη στη βιβλιοθήκη, όπου υπογραμμίζει το επιλεγμένο από τον πελάτη προϊόν. Το κουμπί αυτό ουσιαστικά χρησιμεύει σε περίπτωση που ο πελάτης θέλει να δει πληροφορίες για ένα προϊόν που έχει επιλέξει να νοικιάσει ή να αναθεωρήσει την ενοικίαση του.

#### Κουμπί “REMOVE ITEM”

Διαγράφει το επιλεγμένο προϊόν από το καλάθι και ανανεώνει το συνολικό ποσό χρέωσης.

#### Κουμπί “CHECKOUT”

Επιβεβαιώνει την τρέχουσα ενοικίαση. Μόλις πατηθεί αυτό το κουμπί, γίνονται εισαγωγές στους πίνακες rental και payment της βάσης δεδομένων για κάθε προϊόν του πίνακα.

#### ΕΚΠΤΩΣΗ

Για κάθε τρίτη φορά που ο πελάτης νοικιάζει προϊόν μέσα στην ίδια μέρα, ο ίδιος δικαιούται έκπτωση 50% πάνω στην τιμή του προϊόντος. Παρατηρούμε ότι σε αυτή την περίπτωση ανανεώνεται το συνολικό ποσό χρέωσης και εμφανίζεται αντίστοιχο μήνυμα.

# ORIGIN

Hello ALEXANDER.FENNELL@sakilacustomer.org!

Profile Settings

My Movies / Series

Browse Library

Current Rental

Logout

Exit

TYPE	TITLE	PRICE
film	SUGAR WONKA	0.3
episode	Chapter 3	0.1
episode	This episode	0.1

SHOW INFO

REMOVE ITEM

Customer Type

FILMS AND SERIES

Price Total

0.45€

Discount Granted!

CHECKOUT

Ο πελάτης κανονικά θα έπρεπε να πληρώσει 0.5 €, όμως το τελευταίο επεισόδιο στον πίνακα είναι το τρίτο που επιλέγει προς ενοικίαση την ίδια μέρα. Έτσι αντί για 0.1 €, που ήταν η αρχική τιμή του, θα πληρώσει μόνο 0.05 €, λόγω της έκπτωσης. Έτσι προκύπτει και το νέο συνολικό ποσό.

Αν ο πελάτης αφαιρέσει ένα προϊόν από τον πίνακα τότε το συνολικό ποσό αλλάζει εκ νέου.

# ORIGIN

Hello ALEXANDER.FENNELL@sakilacustomer.org!

Profile Settings

My Movies / Series

Browse Library

Current Rental

Logout

Exit

TYPE	TITLE	PRICE
episode	Chapter 3	0.1
episode	This episode	0.1

SHOW INFO

REMOVE ITEM

Customer Type

FILMS AND SERIES

Price Total

0.20€

CHECKOUT

Κρίνεται χρήσιμο να αναφερθεί ο τρόπος υλοποίησης της παραπάνω διαδικασίας. Για την εφαρμογή της έκπτωσης, χρειάστηκε πρόβλεψη από τη java. Η συνάρτηση calculateTotal() του CustomerHomeController καλεί την sql procedure που υπολογίζει πόσες ενοικιάσεις έχουν γίνει την τρέχουσα ημερομηνία που ο χρήστης είναι συνδεδεμένος στην υπηρεσία. Ο αριθμός αυτός προσαυξάνεται κατά 1 για κάθε προϊόν στον πίνακα τρέχουσας ενοικιάσεις και αν το αποτέλεσμα είναι πολλαπλάσιο του 3, τότε δίνεται έκπτωση.

### Έξοδος από την Υπηρεσία

Πραγματοποιείται μέσω των κουμπιών “Logout”, που κλείνει το παράθυρο και επιστρέφει στην οθόνη σύνδεσης/εγγραφής και “Exit”, που κλείνει την εφαρμογή εξ’ ολοκλήρου. Και τα δύο αυτά κουμπιά βρίσκονται στο κάτω αριστερά μέρος της οθόνης.

### Γενικά για τον κώδικα

Όλες οι συναρτήσεις που κάνουν αλλαγές στις εγγραφές της βάσεις ανανεώνουν το log σύμφωνα με την εκφώνηση τόσο για τις επιτυχημένες όσο και για τις αποτυχημένες ενέργειες.

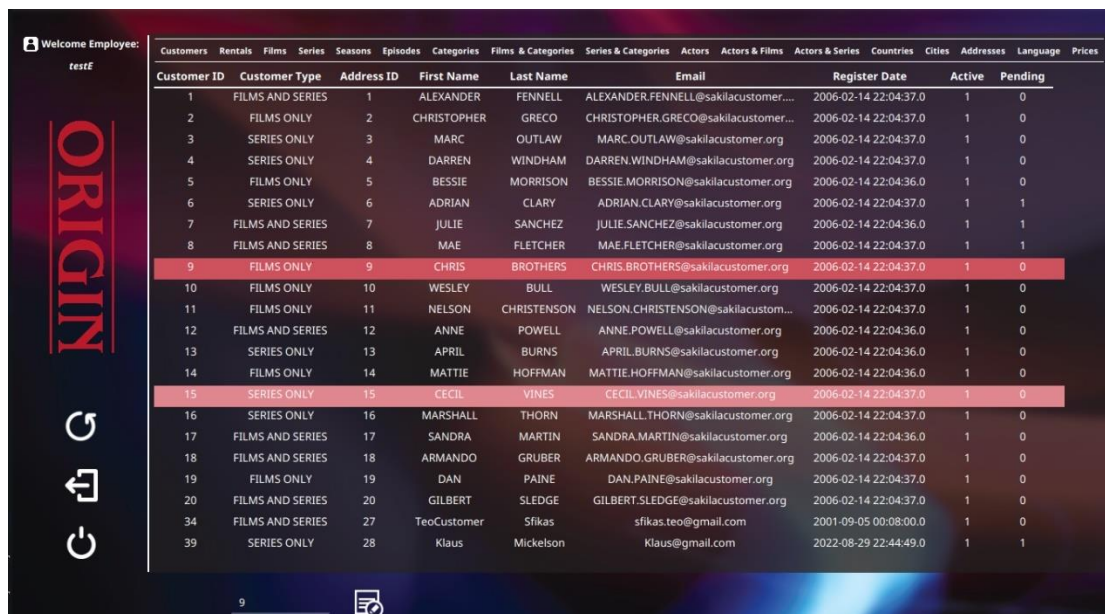
Σε ότι αφορά τον χειρισμό των Exceptions, η αντιμετώπιση τους είναι περισσότερο programmer-friendly, με λίγα μηνύματα προς το χρήστη της εφαρμογής και περισσότερα στην κοσνόλα (terminal) εξόδου.

Παρατηρούμε επίσης ότι, inserts, deletes και updates σε όλους τους πίνακες (πλην του πίνακα log) που αποτυγχάνουν αντιμετωπίζονται με αντίστοιχη εισαγωγή αποτυχημένης ενέργειας στο log. Οποιοδήποτε λάθος συμβεί σε αυτή την τελευταία εισαγωγή αντιμετωπίζεται με απλή εκτύπωση της εξαίρεσης στο terminal. Γενικά δεν καταγράφουμε καμία ενέργεια που αφορά τον πίνακα log, καθώς ο πίνακας θα γέμιζε με ενέργειες που εκτελέστηκαν πάνω του, πράγμα ανούσιο και κουραστικό για το χρήστη.

## Employee Home Page:

Με στόχο την υλοποίηση των λειτουργιών του Employee για τον σχεδιασμό του UI προσπαθήσαμε να παρέχουμε όσο το δυνατό τις περισσότερες πληροφορίες στο μικρότερο χώρο. Κρίνοντας πως η βασική δραστηριότητα του Employee είναι να διαχειρίζεται τα στοιχεία της βάσης έπρεπε να διαλέξουμε το πόσο αυστηρά το UI θα προσομοιάζει την βάση που δημιουργήσαμε. Με βασικό λοιπόν στόχο την μείωση των δυνατών λαθών και εν δυνάμει τυπογραφικών κατά την σύνδεση των πινάκων επιλέξαμε την χρήση των id πεδίων της βάσης αντί για την χρήση του ονόματος ενός βασικού πεδίου. Με αυτό τον τρόπο εκτελούμε και λιγότερα queries και αναζητήσεις στην βάση δεδομένων μας ενώ αποφεύγουμε πιθανά user errors.

Προσπαθώντας να παρέχουμε την περισσότερη δυνατή πληροφορία και ένα intuitive παράλληλα περιβάλλον εργασίας επιλέξαμε να μειώσουμε όσο το δυνατό πολλά αχρείαστα sub - menus και καταλόγους ενώ παράλληλα διατηρούμε παρόμοια τακτική επεξεργασίας για κάθε δυνατό πίνακα της βάσης.



The screenshot shows the Employee Home Page. On the left is a sidebar with a vertical navigation menu containing icons for Customers, Rentals, Films, Series, Seasons, Episodes, Categories, and a search icon. The main area displays a table with columns: Customer ID, Customer Type, Address ID, First Name, Last Name, Email, Register Date, Active, and Pending. The table contains 39 rows of customer data. The first 15 rows are highlighted in red. The table is titled 'Welcome Employee: testE'.

Customer ID	Customer Type	Address ID	First Name	Last Name	Email	Register Date	Active	Pending
1	FILMS AND SERIES	1	ALEXANDER	FENNELL	ALEXANDER.FENNELL@sakilacustomer....	2006-02-14 22:04:37.0	1	0
2	FILMS ONLY	2	CHRISTOPHER	GRECO	CHRISTOPHER.GRECO@sakilacustomer...	2006-02-14 22:04:37.0	1	0
3	SERIES ONLY	3	MARC	OUTLAW	MARC.OUTLAW@sakilacustomer.org	2006-02-14 22:04:37.0	1	0
4	SERIES ONLY	4	DARREN	WINDHAM	DARREN.WINDHAM@sakilacustomer.org	2006-02-14 22:04:37.0	1	0
5	FILMS ONLY	5	BESSIE	MORRISON	BESSIE.MORRISON@sakilacustomer.org	2006-02-14 22:04:36.0	1	0
6	SERIES ONLY	6	ADRIAN	CLARY	ADRIAN.CLARY@sakilacustomer.org	2006-02-14 22:04:37.0	1	1
7	FILMS AND SERIES	7	JULIE	SANCHEZ	JULIE.SANCHEZ@sakilacustomer.org	2006-02-14 22:04:36.0	1	1
8	FILMS AND SERIES	8	MAE	FLETCHER	MAE.FLETCHER@sakilacustomer.org	2006-02-14 22:04:37.0	1	1
9	FILMS ONLY	9	CHRIS	BROTHERS	CHRIS.BROTHERS@sakilacustomer.org	2006-02-14 22:04:37.0	1	0
10	FILMS ONLY	10	WESLEY	BULL	WESLEY.BULL@sakilacustomer.org	2006-02-14 22:04:37.0	1	0
11	FILMS ONLY	11	NELSON	CHRISTENSON	NELSON.CHRISTENSON@sakilacustom...	2006-02-14 22:04:37.0	1	0
12	FILMS AND SERIES	12	ANNE	POWELL	ANNE.POWELL@sakilacustomer.org	2006-02-14 22:04:36.0	1	0
13	SERIES ONLY	13	APRIL	BURNS	APRIL.BURNS@sakilacustomer.org	2006-02-14 22:04:36.0	1	0
14	FILMS ONLY	14	MATTIE	HOFFMAN	MATTIE.HOFFMAN@sakilacustomer.org	2006-02-14 22:04:36.0	1	0
15	SERIES ONLY	15	CECIL	VINES	CECIL.VINES@sakilacustomer.org	2006-02-14 22:04:37.0	1	0
16	SERIES ONLY	16	MARSHALL	THORN	MARSHALL.THORN@sakilacustomer.org	2006-02-14 22:04:37.0	1	0
17	FILMS AND SERIES	17	SANDRA	MARTIN	SANDRA.MARTIN@sakilacustomer.org	2006-02-14 22:04:36.0	1	0
18	FILMS AND SERIES	18	ARMANDO	GRUBER	ARMANDO.GRUBER@sakilacustomer.org	2006-02-14 22:04:37.0	1	0
19	FILMS ONLY	19	DAN	PAINE	DAN.PAINE@sakilacustomer.org	2006-02-14 22:04:37.0	1	0
20	FILMS AND SERIES	20	GILBERT	SLEDGE	GILBERT.SLEDGE@sakilacustomer.org	2006-02-14 22:04:37.0	1	0
34	FILMS AND SERIES	27	TeoCustomer	Sfikas	sfikas.teo@gmail.com	2001-09-05 00:08:00.0	1	0
39	SERIES ONLY	28	Klaus	Mickelson	Klaus@gmail.com	2022-08-29 22:44:49.0	1	1

Αρχικά για να αναλύσουμε το UI του employee θα θέλαμε να επικεντρώσουμε την προσοχή σας στο αριστερό μέρος του παραθύρου. Διακρίνεται αρχικά ένα welcome message με το email του current connected employee ενώ ακολουθούν το logo της εταιρίας και βασικά function keys. Το "refresh" κουμπί εκτελεί forced refresh σε όλους τους πίνακες της βάσης σε περίπτωση που ο employee το κρίνει απαραίτητο αν και με οποιαδήποτε αλλαγή γίνει στην βάση οι πίνακες γίνονται αυτόματα updated. Ακολουθεί το κουμπί για logout ( επιστροφή στο login screen ) και το Exit - Close ( έξοδος από την εφαρμογή ).



Μεταβαίνοντας στον πίνακα , στο πάνω μέρος μπορούμε να κάνουμε traverse κάθε πίνακα της βάσης πατώντας στο αντίστοιχο tab. Σχεδόν κάθε tab αντιστοιχεί ενεργά σε έναν πίνακα της βάσης δεδομένων μας ενώ φαίνονται συχνά και επιπλέον πληροφορίες από άλλα tables για την διευκόλυνση του employee ( Εξαίρεση αποτελεί το tab prices ) . Πατώντας σε κάθε Column μπορούμε να κάνουμε αναδιάταξη του πίνακα που βλέπουμε είτε με αλφαβητική σειρά είτε με αριθμητική ( ανάλογα με το είδος του πεδίου του column που πατάμε ). Το πεδίο του πίνακα που γίνεται hovered διακρίνεται με το κοραλί χρώμα ενώ το πεδίο που έχει γίνει pressed τελευταίο έχει ένα fuchsia background ( Η τιμή του ID / primary key πεδίου του διακρίνεται και στο textfield στο κάτω μέρος της εικόνας ). Ο employee ανάλογα με τις λειτουργίες του στον κάθε πίνακα στο προαναφερθές toolbar έχει και τα αντίστοιχα buttons. Μπορεί να επιλέξει το πεδίο που θα κάνει edit / delete είτε αλλάζοντας την τιμή στο textfield είτε πατώντας πάνω σε ένα record στον πίνακα ( Αν δεν υπάρχει τιμή στο textfield καμία από τις προαναφερθέντες λειτουργίες δεν εκτελούνται ). Ακόμα όπου μπορεί έχει την δυνατότητα να δημιουργήσει καινούργια records.

Welcome Employee: testE

Customers Rentals Films Series Seasons Episodes Categories Films & Categories Series & Categories Actors Actors & Films Actors & Series Countries Cities Addresses Language Prices

Episode ID	Title	Episode Num.	Episode Length	Rating	Season ID	Series ID	Series Title
1	Pilot	1	45	G	1	1	Hot Flowers
2	2nd Episode	2	46	G	1	1	Hot Flowers
3	3rd Episode	3	47	G	1	1	Hot Flowers
4	4th Episode	4	40	G	1	1	Hot Flowers
5	This episode	1	41	PG	2	2	Child of Gift
6	That episode	2	40	G	2	2	Child of Gift
7	This one episode	3	43	R	2	2	Child of Gift
8	That one episode	4	45	G	2	2	Child of Gift
9	That very one episode	5	48	G	2	2	Child of Gift
10	The depression	1	50	PG	3	3	The depressive film
11	The breaking	2	52	G	3	3	The depressive film
12	The healing	3	51	R	3	3	The depressive film
13	Chapter 1	1	42	G	4	3	The depressive film
14	Chapter 2	2	44	PG	4	3	The depressive film
15	Chapter 3	3	43	PG-13	4	3	The depressive film

Episode ID

Οι περισσότεροι πίνακες / tabs έχουν τα πεδία της εικόνας που προηγείται. Το κουμπί Edit record διακρίνεται από το μολυβάκι, το Delete record από το " x " ενώ το New record από το plus sign.

Εφόσον οι πίνακες συνδέονται στην βάση με την χρήση των ξένων κλειδιών κατά τις διαδικασίες edit και new ο employee πρέπει να έχει την δυνατότητα να κάνει traverse σε όλα τα records της βάσης με στόχο την αναζήτηση κάποιου id ή συγκεκριμένου ονόματος / πεδίου. Για αυτό τον σκοπό με το πάτημα ενός εκ των δύο αυτών κουμπιών εμφανίζεται ένα καινούργιο pop up window στο οποίο γίνεται είτε το edit του επιλεγμένου πεδίου είτε δημιουργούμε ένα καινούργιο record. Όλα τα

pop - up windows είναι παρόμοια και σε εμφάνιση και σε λειτουργία για αυτό παραδειγματικά θα επιδείξουμε 2 εξ αυτών.

Για την διαδικασία edit ενός record εμφανίζονται στα αριστερά τα στοιχεία του current record ενώ στα δεξιά παρουσιάζεται ένα textfield με καινούργιες τιμές. Στα πεδία της βάσης που είναι enum φαίνεται ένα choice box για την αποφυγή λαθών. Συγκεκριμένα για τον Customer ο Employee δεν μπορεί ούτε να τον διαγράψει ούτε να δημιουργήσει καινούργια αλλά μόνο να κάνει edit τα στοιχεία του. Στο καινούργιο edit-window δεν είναι ανάγκη να συμπληρωθεί κάθε πεδίο, όσα μείνουν κενά θα παραμείνουν οι προηγούμενες τιμές τους ώστε ο employee να μπορεί γρήγορα και εύκολα να αλλάξει συγκεκριμένα πεδία σε κάθε record. Τα καινούργια pop - ups είναι φυσικά μικρότερα παράθυρα και στα περισσότερα συστήματα μπορούν με μετακινηθούν στην οθόνη σας πατώντας το windows / command key σε συνδιασμό με mouse drag.

The screenshot shows a mobile application interface in 'EDIT MODE'. At the top, it says 'EDIT MODE:' in red, followed by 'Customer ID : 39'. Below this, there are two columns: 'Current Record:' and 'Edit New Values:'. The 'Current Record:' column lists various fields with their current values: Email (Klaus@gmail.com), Customer ID (39), Phone (6948275995), a choice box for 'SERIES ONLY', First Name (Klaus), Last Name (Mickelson), Register Date (2022-08-29 22:44:49.0), Active status (1), Country (USA), City (Mistic Falls), District (North Carolina), Address (Aubery Street 7), and Postal Code (26222). The 'Edit New Values:' column shows the same fields with input fields for new values. The 'Email' field is marked as 'Unchangeable'. The 'Phone' field has a checkmark. The 'Country' field has a dropdown arrow. The 'City' field has a checkmark. The 'District' field has a checkmark. The 'Address' field has a checkmark. The 'Postal Code' field has a checkmark. At the bottom, there are two red circular buttons with white arrows pointing left and right.

Current Record:	Edit New Values:
Klaus@gmail.com	Email - Unchangeable.
39	Customer ID
6948275995	Phone
SERIES ONLY	
Klaus	First Name
Mickelson	Last Name
2022-08-29 22:44:49.0	Register Date
1	Active
USA	Country
Mistic Falls	City
North Carolina	District
Aubery Street 7	Address
26222	Postal Code

Διακρίνονται 3 κουμπιά, δύο που κάνουν cancel την διαδικασία ( το βελάκι αριστερά και το undo στο πάνω μέρος του παραθύρου ) και το submit το οποίο εκτελεί τις αναγκαίες αλλαγές στην βάση και σε περίπτωση επιτυχίας μόνο κλείνει το παράθυρο. Σε περιπτώσεις αποτυχίας οπουδήποτε στην βάση υπάρχουν error labels που εμφανίζονται με το αναγκαίο error message. Όπως ακόμα φαίνεται ο employee δεν μπορεί να αλλάξει το email ενός πελάτη.

Αντίστοιχα λειτουργεί και το pop-up window για ένα καινούργιο record στην βάση μας, μόνο που είναι πιο απλουστευμένο καθώς δεν υπάρχουν τα πεδία για τα προηγούμενα values. Κάθε πεδίο πρέπει να συμπληρωθεί εκτός από τα primary keys ενός πίνακα τα οποία γίνονται auto incremented από την βάση. Αν τέτοια πεδία αφήθούν κενά η βάση θα βάλει το επόμενο ορθό id.

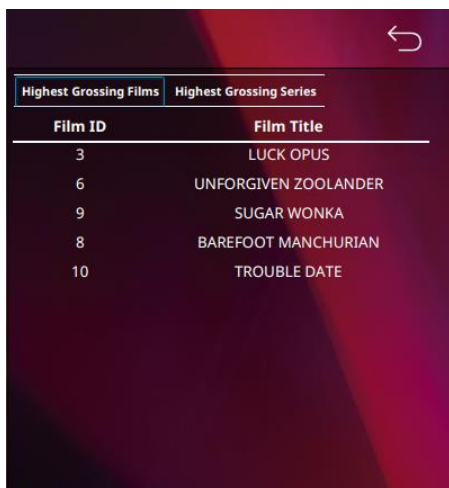
Ολοκληρώνοντας τα στοιχεία του UI οι πίνακες rental και prices διαφέρουν από όσα έχουμε εξηγήσει μέχρι στιγμής. Και στους δύο ο employee δεν έχει την δυνατότητα μεταβολής ή δημιουργίας ενός record. Στο Tab των rentals κάναμε intergrade το ερώτημα 4<sup>to</sup> bullet ( " 5 σειρές ή ταινίες με τις περισσότερες ενοικιάσεις τον τελευταίο μήνα " ). Αν τα textfields στο αντίστοιχο toolbar του tab των rentals μείνει κενό θα εκτελεστεί ακριβώς το default ερώτημα. Αν δωθούν τιμές μπορούμε να εστιάσουμε σε συγκεκριμένες ημερομηνίες και αριθμό records κάνοντας το ζητούμενο πιο δυναμικό. Με το πάτημα του κουμπιού "Highest Grossing" εμφανίζεται ένα καινούργιο pop-up το οποίο περιέχει τις ζητούμενες πληροφορίες.

Welcome Employee: testE

Customers Rentals Films Series Seasons Episodes Categories Films & Categories Series & Categories Actors Actors & Films Actors & Series Countries Cities Addresses Language Prices

Rental ID	Rental Date	Film ID	Series ID	Title	Customer ID	Customer Email
1	2005-06-11 01:25:08.0	10	0	TROUBLE DATE	1	ALEXANDER.FENNELL@sakilacustomer....
2	2005-06-15 03:42:27.0	0	2	2nd Episode	17	SANDRA.MARTIN@sakilacustomer.org
3	2005-06-17 16:47:32.0	3	0	LUCK OPUS	4	DARREN.WINDHAM@sakilacustomer.org
4	2005-06-20 16:10:19.0	6	0	UNFORGIVEN ZOOLANDER	8	MAE.FLETCHER@sakilacustomer.org
5	2005-07-13 20:30:32.0	3	0	LUCK OPUS	9	CHRIS.BROTHERS@sakilacustomer.org
6	2005-07-15 08:41:38.0	10	0	TROUBLE DATE	8	MAE.FLETCHER@sakilacustomer.org
7	2005-08-17 07:12:31.0	3	0	LUCK OPUS	16	MARSHALL.THORN@sakilacustomer.org
8	2005-08-22 15:52:57.0	9	0	SUGAR WONKA	5	BESSIE.MORRISON@sakilacustomer.org
9	2005-08-27 15:34:01.0	8	0	BAREFOOT MANCHURIAN	9	CHRIS.BROTHERS@sakilacustomer.org
10	2005-08-31 12:34:12.0	6	0	UNFORGIVEN ZOOLANDER	12	ANNE.POWELL@sakilacustomer.org
11	2005-09-01 03:42:27.0	0	7	This one episode	17	SANDRA.MARTIN@sakilacustomer.org
12	2005-09-02 03:42:27.0	0	1	Pilot	13	APRIL.BURNS@sakilacustomer.org
13	2005-09-03 03:42:27.0	0	1	Pilot	5	BESSIE.MORRISON@sakilacustomer.org
14	2005-09-04 03:42:27.0	0	3	3rd Episode	9	CHRIS.BROTHERS@sakilacustomer.org
15	2005-09-05 03:42:27.0	0	8	That one episode	20	GILBERT.SLEDGE@sakilacustomer.org
16	2005-09-06 03:42:27.0	0	3	3rd Episode	8	MAE.FLETCHER@sakilacustomer.org
17	2005-09-07 03:42:27.0	0	11	The breaking	13	APRIL.BURNS@sakilacustomer.org
18	2005-09-08 03:42:27.0	0	8	That one episode	6	ADRIAN.CLARY@sakilacustomer.org
19	2005-09-09 03:42:27.0	0	8	That one episode	4	DARREN.WINDHAM@sakilacustomer.org
20	2005-09-10 03:42:27.0	0	8	That one episode	1	ALEXANDER.FENNELL@sakilacustomer....
21	2005-09-11 03:42:27.0	0	13	Chapter 1	20	GILBERT.SLEDGE@sakilacustomer.org
22	2005-09-12 03:42:27.0	0	12	The healing	18	ARMANDO.GRUBER@sakilacustomer.org
23	2005-09-13 03:42:27.0	0	12	The healing	13	APRIL.BURNS@sakilacustomer.org

Start Date End Date Number Highest Grossing



Film ID	Film Title
3	LUCK OPUS
6	UNFORGIVEN ZOOLANDER
9	SUGAR WONKA
8	BAREFOOT MANCHURIAN
10	TROUBLE DATE

Διακρίνονται 2 tabs, ένα για τις περισσότερες σε ενοικιάσεις ταινίες και ένα για τις αντίστοιχες σειρές στο χρονικό Διάστημα που προαναφέραμε. Το βελάκι πάνω δεξιά κλείνει ουσιαστικά το pop - up παράθυρο. Τέλος θα θέλαμε να επισημάνουμε πως ο employee έχει πρόσβαση μόνο στον πίνακα των ενοικιάσεων των πελατών και όχι στον πίνακα των πληρωμών λόγω σχεδιασμού και ερμηνείας της εκφώνησης.

#### ◆ Ταχεία περιγραφή κώδικα employee :

Ο κώδικας που αφορά τον employee controller και τα αντίστοιχα edit και new παράθυρα είναι υπερβολικά μεγάλος αλλά επίσης επαναλαμβανόμενος. Θα μπορούσε από την μεριά μας να είχε γίνει καλύτερη παραμετροποίηση του κώδικα με στόχο την μείωση του και την ευκολότερο μελλοντικό sustainability. Παρουσιάζονται παρόμοιες διαδικασίες για κάθε tab όπως ο κώδικας για κάθε κουμπί που παραμένει παρόμοιος, το initialization των πινάκων και τέλος η λειτουργία και ανάθεση των περισσότερων fxmls. Κάθε καινούργιο pop-up εφόσον σχεδιάστηκε στατικά με την χρήση των fxml resources απαιτεί καινούργιο controller και ένα ακόμα καινούργιο αρχείο. Ωστόσο καθώς κάθε πίνακας / tab έχει διαφορετικά στοιχεία, μεγέθη στηλών και queries στην βάση η υλοποίηση ενός πραγματικά δυναμικού κώδικα αν και καλύτερη αποδείχθηκε υπερβολικά δύσκολη.

Αρχικά καθώς παρατηρούμε τον Employee Controller φαίνονται όλα τα πεδία τα οποία αντιστοιχίζονται σε fx objects και το private πεδίο editID στο οποίο αποθηκεύονται οι τιμές των textfields των records προς διαγραφή ή επεξεργασία. Στην βασική συνάρτηση initialize() καλούνται οι παρόμοιες υποσυναρτήσεις για το initialization των πινάκων / tabs που παρατηρούμε στο gui. Γίνεται μία σύνδεση με την βάση όπου εκτελούμε το απαραίτητο query και αποθηκεύουμε τα αποτελέσματα του σε ένα Result Set. Ο ευκολότερος τρόπος για να αρχικοποιήσουμε πίνακες αποδείχθηκε η χρήση των FxCollections και των observable list. Κάθε record του παραγόμενου Result Set χρησιμοποιείται για να δημιουργήσει ένα ένα instance μίας βοηθητικής model κλάσης η οποία ουσιαστικά αντικατοπτρίζει κάθε tab. Όλα τα instances αποθηκεύονται σε ένα observable list σύμφωνα με το οποίο παρέχονται τιμές στο

αντίστοιχο tableView. Στο υπόλοιπο του Employee Controller για κάθε tab υπάρχουν μαζεμένες οι συναρτήσεις και μέθοδοι για την λειτουργία του επισημασμένα με σχόλια.

Οι συναρτήσεις \*OnClickPressed ορίζουν την τιμή του textField κάθε tab με βάση το στοιχείο που πατήθηκε στο tableView. Η συνάρτηση refresh καλεί εξ αρχής την συνάρτηση initialize για κάθε πίνακα εξατομικευμένα, ενώ η συναρτήσεις edit και new αφού ελέγξουν για την ύπαρξη τιμής στα αντίστοιχα textfields δημιουργούν τα edit και new pop-up windows που προαναφέραμε. Τέλος παρατηρείται η κατάλληλη συνάρτηση DeleteRecord για κάθε πίνακα η οποία διαγράφει το record από την βάση και έπειτα κάνει refresh τον πίνακα.

Είναι αναγκαία επίσης η αναφορά στην χρήση του log. Στο κομμάτι της βάσης υπάρχουν τα αναγκαία triggers after insert , update και delete τα οποία επιβεβαιώνουν την ανανέωση του log ύστερα από κάθε επιτυχημένο query. Σε περίπτωση αποτυχίας δεν μπορεί να υπάρξει ένα trigger που να αναλάβει την ίδια διαδικασία. Για αυτό το σκοπό κάθε φορά που γίνεται μία διαδικασία Insert , Delete ή Update και αναφερθεί Error στο try\_catch block πέρα από την ανανέωση του αντίστοιχου ErrorLabel κάνουμε και ένα insert στο log για αποτυχημένη προσπάθεια επεξεργασίας της βάσης μας.

Τέλος οι αναγκαίοι controllers που δημιουργήσαμε κρίνονται απλοί με στοιχεία κώδικα τα οποία έχουμε ήδη καλύψει πέρα από αυτό του EditCustomerController. Με στόχο την μερική επαναχρησιμοποίηση του κώδικα στο ζητούμενο του edit Customer Profile και στην διαδικασία του Sign-up δημιουργήθηκε η απαίτηση για την απόκρυψη των πεδίων IDs κατά την διαδικασία του edit. Με στόχο την αποφυγή redundant πληροφορίας γίνεται έλεγχος στο αν η χώρα και η πόλη υπάρχουν ήδη στην βάση μας για την χρησιμοποίηση των συγκεκριμένων records. Στην περίπτωση που δεν υπάρχουν δημιουργούνται καινούργιες εγγραφές και έπειτα γίνονται οι διεργασίες στους πίνακες user και customer. Εφόσον παρουσιάζεται η αλληλένδετη σχέση 5 πινάκων μετατρέπουμε προσωρινά το auto\_commit του jdbc σε false. Μόνο αν γίνουν ορθά οι διαδικασίες σε όλους τους εμπλεκόμενους πίνακες γίνεται το final commit στην βάση. Σε περίπτωση κάποιου λάθους λοιπόν σιγουρευόμαστε πως δεν θα έχουν εκτελεστεί αλλαγές στους μισούς πίνακες και πως δεν θα έχουμε race conditions κατά την εκτέλεση των queries σε ένα multi-client μοντέλο βάσης.



## Admin Home Page:

Με την είσοδο ενός διαχειριστή στην υπηρεσία, εμφανίζεται η αρχική του οθόνη:

ID	TYPE	ADDRESS	FIRST NAME	LAST NAME	EMAIL
1	FILMS AND SERIES	1346 Mysore Drive	ALEXANDER	FENNELL	ALEXANDER.FENNELL@sakilacustomer.org
2	FILMS ONLY	1740 Le Mans Loop	CHRISTOPHER	GRECO	CHRISTOPHER.GRECO@sakilacustomer.org
3	SERIES ONLY	1386 Yangor Avenue	MARC	OUTLAW	MARC.OUTLAW@sakilacustomer.org
4	SERIES ONLY	391 Callao Drive	DARREN	WINDHAM	DARREN.WINDHAM@sakilacustomer.org
5	FILMS ONLY	442 Rae Bareli Place	BESSIE	MORRISON	BESSIE.MORRISON@sakilacustomer.org
9	FILMS ONLY	430 Alessandria Loop	CHRIS	BROTHERS	CHRIS.BROTHERS@sakilacustomer.org
10	FILMS ONLY	1792 Valle de la Páscua Place	WESLEY	BULL	WESLEY.BULL@sakilacustomer.org
11	FILMS ONLY	68 Molodetno Manor	NELSON	CHRISTENSON	NELSON.CHRISTENSON@sakilacustomer.org
12	FILMS AND SERIES	692 Joliet Street	ANNE	POWELL	ANNE.POWELL@sakilacustomer.org
13	SERIES ONLY	1101 Bucuresti Boulevard	APRIL	BURNS	APRIL.BURNS@sakilacustomer.org
14	FILMS ONLY	127 Purnea (Purnia) Manor	MATTIE	HOFFMAN	MATTIE.HOFFMAN@sakilacustomer.org
15	SERIES ONLY	231 Kaliningrad Place	CECIL	VINES	CECIL.VINES@sakilacustomer.org
16	SERIES ONLY	1224 Huejutla de Reyes Boulevard	MARSHALL	THORN	MARSHALL.THORN@sakilacustomer.org
17	FILMS AND SERIES	1 Valle de Santiago Avenue	SANDRA	MARTIN	SANDRA.MARTIN@sakilacustomer.org
18	FILMS AND SERIES	379 Lublin Parkway	ARMANDO	GRUBER	ARMANDO.GRUBER@sakilacustomer.org

Μετακινήστε την οριζόντια μπάρα στο κάτω μέρος του πίνακα για να δείτε όλες τις στήλες του πίνακα

Όπως φαίνεται από την εικόνα, το GUI του διαχειριστή μπορεί να χωριστεί σε 5 ενότητες:

- Διαχείριση Λογαριασμών | Account Registry
- Ιστορικό Εσόδων | Income History
- Χρεώσεις | Pricing List
- Πληροφορίες Log | Log Info
- Έξοδος από την υπηρεσία

Κάθε μία από αυτές αναλύεται ξεχωριστά:

### Διαχείριση Λογαριασμών | Account Registry

Πρόκειται για το Tab “Account Registry” που φαίνεται ανοιχτό στην παραπάνω εικόνα και παρέχει πληροφορίες για όλους τους χρήστες της βάσης:

#### Tab Πελατών (Customers)

Τόσο αυτό, όσο και τα υπόλοιπα Tabs του Account Registry έχουν συγκεκριμένη δομή:

1. Πίνακες με δεδομένα για κάθε κατηγορία χρήστη

2. Toolbar με κουμπιά που υλοποιούν συγκεκριμένες λειτουργίες για επιλεγμένους από το διαχειριστή χρήστες και τις οποίες θα εξηγήσουμε

Για το Tab πελατών (που φαίνεται στην εικόνα της αρχικής του διαχειριστή), προφέρεται η δυνατότητα εισαγωγής νέου πελάτη, με το πάτημα του κουμπιού “Insert Customer”, που οδηγεί το διαχειριστή σε νέο pop-up παράθυρο, για την εισαγωγή των απαραίτητων στοιχείων.

The screenshot shows the ORIGIN system interface. On the left, there's a sidebar with 'Search in Log based on:' (All Tables, Email, Starting Date, Ending Date), 'Show Log Info', 'Logout', and 'Exit'. The main area is divided into three sections: 'NEW CUSTOMER (\* : necessary field)', 'NEW ADDRESS', and 'Pending Requests'. The 'NEW CUSTOMER' form has fields for Customer ID, Type\*, First Name\*, Last Name\*, Email\*, and Active (0 or 1: any other value is considered as 1). The 'NEW ADDRESS' form has fields for Country\*, City\*, District\*, Address\*, Postal Code\*, and Phone\*. The 'Pending Requests' table lists customers with their last names and emails. At the bottom, there are 'CANCEL' and 'CONFIRM' buttons.

LAST NAME	EMAIL
FENNELL	ALEXANDER.FENNELL@sakilacustomer.org
GRECO	CHRISTOPHER.GRECO@sakilacustomer.org
OUTLAW	MARC.OUTLAW@sakilacustomer.org
WINDHAM	DARREN.WINDHAM@sakilacustomer.org
MORRISON	BESSIE.MORRISON@sakilacustomer.org
BROTHERS	CHRIS.BROTHERS@sakilacustomer.org
BULL	WESLEY.BULL@sakilacustomer.org
CHRISTENSON	NELSON.CHRISTENSON@sakilacustomer.org
POWELL	ANNE.POWELL@sakilacustomer.org
BURNS	APRIL.BURNS@sakilacustomer.org
HOFFMAN	MATTIE.HOFFMAN@sakilacustomer.org
VINES	CECIL.VINES@sakilacustomer.org
THORN	MARSHALL.THORN@sakilacustomer.org
MARTIN	SANDRA.MARTIN@sakilacustomer.org
GRUBER	ARMANDO.GRUBER@sakilacustomer.org

This is a close-up of the 'NEW CUSTOMER' form. It shows the fields for Customer ID, Type\*, First Name\*, Last Name\*, Email\*, and Active (0 or 1: any other value is considered as 1). Below the form, a message states: 'Missing Values. Please fill all the fields with the \* symbol.' The 'NEW ADDRESS' section is also visible, with fields for Country\*, City\*, District\*, Address\*, Postal Code\*, and Phone\*. At the bottom, there are 'CANCEL' and 'CONFIRM' buttons.

Το συγκεκριμένο παράθυρο, έχει υποχρεωτικά πεδία, τα οποία σημειώνονται με αστερίσκο. Αν κάποιο από αυτά παραμείνει κενό, όταν πατηθεί το κουμπί “CONFIRM”, που επιβεβαιώνει την εισαγωγή νέου πελάτη με τα αναγραφόμενα στοιχεία και επιστρέφει στην αρχική, τότε εμφανίζεται μήνυμα λάθους και η εισαγωγή δεν ολοκληρώνεται (Εικόνα ARC1).

Εικόνα ARC1: Απαραίτητα πεδία κενά

**NEW CUSTOMER ( \* : necessary field)**

8

SERIES ONLY

maria

papadopoulou

demonstration1@gmail.com

Active (0 or 1: any other value is considered as 1)

Invalid Id: Id will be changed automatically.

**NEW ADDRESS**

Greece



Patras

Rio

Panepistimiou 20

26443

123456789

 CANCEL CONFIRM 

Εικόνα ARC2: Μη έγκυρο Id

Κατά το πάτημα του κουμπιού μπορεί επίσης να εμφανιστεί μήνυμα σε περίπτωση εισαγωγής κωδικού (id) που αντιστοιχεί ήδη σε κάποιο χρήστη της βάσης. Αν ο διαχειριστής εισάγει τέτοιο κωδικό, τότε πέρα από το επεξηγηματικό μήνυμα, προκαλείται και η απενεργοποίηση του πεδίου καθώς το id αποφασίζεται αυτόματα από τη βάση (εισαγωγή user με id = NULL). Με αντίστοιχο τρόπο ο κωδικός πελάτη παίρνει τιμή στη βάση, ακόμα και αν το πεδίο παραμένει κενό, γι' αυτό δεν έχει επισημανθεί ως απαραίτητο στοιχείο (Εικόνα ARC2).

**NEW CUSTOMER ( \* : necessary field)**

8

SERIES ONLY

maria

papadopoulou

up1072494@upnet.gr

Active (0 or 1: any other value is considered as 1)

Email is already being used. Please change your email.

**NEW ADDRESS**

Greece



Patras

Rio

Panepistimiou 20

26443

123456789

 CANCEL CONFIRM 

Τέλος, αν το email που έχει εισαχθεί χρησιμοποιείται ήδη, προβάλλεται αντίστοιχο μήνυμα και η εισαγωγή πελάτη δεν ολοκληρώνεται (Εικόνα ARC3).

Στο παράθυρο υπάρχει και το κουμπί "CANCEL", για ακύρωση της ενέργειας και επιστροφή στην αρχική οθόνη διαχειριστή.

Εικόνα ARC3: Email ήδη χρησιμοποιείται



Εκτός από το να εισάγει νέο πελάτη, ο διαχειριστής μπορεί και να διαγράψει πελάτες, πατώντας το κουμπί “Delete Customer”, που φαίνεται στο κάτω μέρος του Tab πελατών, αφού επιλέξει κάποιο πελάτη, όπως δείχνει το παράδειγμα με τις δύο επόμενες εικόνες.

**ORIGIN** Hello up1072494@upnet.gr!

Search in Log based on: **Account Registry** Income History Pricing List

All Tables **CHRIS.BROTHERS@saki** Starting Date Ending Date Show Log Info Logout Exit

ID	TYPE	ADDRESS	FIRST NAME	LAST NAME	EMAIL
1	FILMS AND SERIES	1346 Mysore Drive	ALEXANDER	FENNELL	ALEXANDER.FENNELL@sakilacustor
2	FILMS ONLY	1740 Le Mans Loop	CHRISTOPHER	GRECO	CHRISTOPHER.GRECO@sakilacustor
3	SERIES ONLY	1386 Yangor Avenue	MARC	OUTLAW	MARC.OUTLAW@sakilacustomer.i
4	SERIES ONLY	391 Callao Drive	DARREN	WINDHAM	DARREN.WINDHAM@sakilacustome
5	FILMS ONLY	442 Rae Bareli Place	BESSIE	MORRISON	BESSIE.MORRISON@sakilacustome
9	FILMS ONLY	430 Alessandria Loop	CHRIS	BROTHERS	CHRIS.BROTHERS@sakilacustomer
10	FILMS ONLY	1792 Valle de la Pascua Place	WESLEY	BULL	WESLEY.BULL@sakilacustomer.oi
11	FILMS ONLY	68 Molodetno Manor	NELSON	CHRISTENSON	NELSON.CHRISTENSON@sakilacusti
12	FILMS AND SERIES	692 Joliet Street	ANNE	POWELL	ANNE.POWELL@sakilacustomer.c
13	SERIES ONLY	1101 Bucuresti Boulevard	APRIL	BURNS	APRIL.BURNS@sakilacustomer.oi
14	FILMS ONLY	127 Purnea (Purnia) Manor	MATTIE	HOFFMAN	MATTIE.HOFFMAN@sakilacustome
15	SERIES ONLY	231 Kaliningrad Place	CECIL	VINES	CECIL.VINES@sakilacustomer.or
16	SERIES ONLY	1224 Huejutla de Reyes Boulevard	MARSHALL	THORN	MARSHALL.THORN@sakilacustome
17	FILMS AND SERIES	1 Valle de Santiago Avenue	SANDRA	MARTIN	SANDRA.MARTIN@sakilacustome
18	FILMS AND SERIES	379 Lublin Parkway	ARMANDO	GRUBER	ARMANDO.GRUBER@sakilacustome

Insert Customer Delete Customer

**ORIGIN** Hello up1072494@upnet.gr!

Search in Log based on: **Account Registry** Income History Pricing List

All Tables **BESSIE.MORRISON@saki** Starting Date Ending Date Show Log Info Logout Exit

ID	TYPE	ADDRESS	FIRST NAME	LAST NAME	EMAIL
1	FILMS AND SERIES	1346 Mysore Drive	ALEXANDER	FENNELL	ALEXANDER.FENNELL@sakilacustor
2	FILMS ONLY	1740 Le Mans Loop	CHRISTOPHER	GRECO	CHRISTOPHER.GRECO@sakilacusti
3	SERIES ONLY	1386 Yangor Avenue	MARC	OUTLAW	MARC.OUTLAW@sakilacustomer.i
4	SERIES ONLY	391 Callao Drive	DARREN	WINDHAM	DARREN.WINDHAM@sakilacustome
5	FILMS ONLY	442 Rae Bareli Place	BESSIE	MORRISON	BESSIE.MORRISON@sakilacustome
10	FILMS ONLY	1792 Valle de la Pascua Place	WESLEY	BULL	WESLEY.BULL@sakilacustomer.oi
11	FILMS ONLY	68 Molodetno Manor	NELSON	CHRISTENSON	NELSON.CHRISTENSON@sakilacusti
12	FILMS AND SERIES	692 Joliet Street	ANNE	POWELL	ANNE.POWELL@sakilacustomer.c
13	SERIES ONLY	1101 Bucuresti Boulevard	APRIL	BURNS	APRIL.BURNS@sakilacustomer.oi
14	FILMS ONLY	127 Purnea (Purnia) Manor	MATTIE	HOFFMAN	MATTIE.HOFFMAN@sakilacustome
15	SERIES ONLY	231 Kaliningrad Place	CECIL	VINES	CECIL.VINES@sakilacustomer.or
16	SERIES ONLY	1224 Huejutla de Reyes Boulevard	MARSHALL	THORN	MARSHALL.THORN@sakilacustome
17	FILMS AND SERIES	1 Valle de Santiago Avenue	SANDRA	MARTIN	SANDRA.MARTIN@sakilacustome
18	FILMS AND SERIES	379 Lublin Parkway	ARMANDO	GRUBER	ARMANDO.GRUBER@sakilacustome
19	FILMS ONLY	1926 Gingoog Street	DAN	PAINE	DAN.PAINE@sakilacustomer.oi

Insert Customer Delete Customer

Παρατηρούμε ότι πλέον, ο πελάτης με κωδικό 9, τον οποίο επιλέξαμε και διαγράψαμε, δεν υπάρχει.

## Tab Υπαλλήλων (Employees)

Το Tab της διαχείρισης λογαριασμών για υπαλλήλους είναι παρόμοιο με αυτό των πελατών, όπως φαίνεται στην επόμενη εικόνα.

**ORIGIN** Hello up1072494@upnet.gr!

Search in Log based on: All Tables

Email  
Starting Date  
Ending Date

Show Log Info

Logout  
Exit

Account Registry Income History Pricing List

ID	FIRST NAME	LAST NAME	EMAIL	REGISTER DATE	ACTIVE	PHONE
21	MARKO	POLO	employee660@sakilaemployee.org	2020-03-27 15:00:00.0	1	69420
22	RACHEL	GREEN	employee661@sakilaemployee.org	2020-03-27 15:00:00.0	1	69420
23	FRANK	OMEGLE	employee662@sakilaemployee.org	2018-01-01 10:00:00.0	0	69420
24	OCTOBER	FEST	employee663@sakilaemployee.org	2019-05-12 12:00:00.0	1	69420
25	MELENE	RACCOON	employee664@sakilaemployee.org	2019-01-22 15:00:00.0	1	69420
26	OCTAPUS	HOST	employee665@sakilaemployee.org	2019-11-25 15:00:00.0	1	69420
27	SAM	MC DONALDS	employee666@sakilaemployee.org	2020-12-05 15:00:00.0	0	69420
84	maria tests id field emp...	regdat field empty, acti...	testE1	2022-08-21 03:14:25.0	1	
85	maria tests id exists (x2)	email exists, reg date n...	testE2	2012-03-15 19:20:00.0	1	69420
86	maria test regdate sucks	active 0	testE3	2022-08-21 03:19:34.0	0	
87	maria tests id exists	+all normal	testE4	2022-08-21 03:20:34.0	1	
88	maria tests id exists to disable, acti...		testE5	2022-08-21 03:25:25.0	0	
89	no id, label missing fiel...	sos fields ok, reg date s...	testE6	2022-08-21 03:30:01.0	1	
91	maria tests active is string		testE7	2022-08-21 03:31:34.0	1	
92	maria tests auto_increment value		testE8	2022-08-21 03:49:49.0	1	
93	maria tests auto_increment value		testE9	2022-08-21 03:50:49.0	1	

Insert Employee Delete Employee Make Admin

Οι διαδικασίες εισαγωγής και διαγραφής λειτουργούν με τον τρόπο που περιγράψαμε και για τον πελάτη, με τις εξής διαφορές στην εισαγωγή:

- ❖ Κάθε φορά που η εισαγωγή δεν ολοκληρώνεται (έλλειψη απαραίτητων στοιχείων, μη έγκυρο id/email), θεωρείται αποτυχημένη και το log ανανεώνεται κατάλληλα.
- ❖ Αν κατά την εισαγωγή του νέου υπαλλήλου στη βάση συμβεί κάποιο λάθος, τότε το σύστημα προσπαθεί μία ακόμα φορά να εισάγει τον υπάλληλο με ημερομηνία εγγραφής την τρέχουσα και έπειτα ανανεώνει το log ανάλογα με το εάν η δεύτερη προσπάθεια ήταν επιτυχημένη ή όχι. Η λειτουργία αυτή προστέθηκε για λόγους πρόληψης λάθους στη μορφή της ημερομηνίας που εισάγει ο διαχειριστής κατά την δημιουργία του νέου λογαριασμού για τον υπάλληλο.

Με το τρίτο και τελευταίο κουμπί του Toolbar στο Tab των Υπαλλήλων, "Make Admin" ο διαχειριστής μπορεί να μετατρέψει έναν επιλεγμένο υπάλληλο σε διαχειριστή. Η διαδικασία αυτή χωρίζεται σε 3 βήματα σε επίπεδο βάσης:

Βήμα 1. Διαγραφή του χρήστη από τον πίνακα υπαλλήλων

Βήμα 2. Προσθήκη του χρήστη στον πίνακα διαχειριστών

Βήμα 3. Ανανέωση του τύπου του χρήστη στον πίνακα χρηστών



Ωστόσο, θεωρείται ατομική διαδικασία. Αν κάποιο βήμα αποτύχει, είναι σαν να έχουν αποτύχει όλα και γίνονται οι κατάλληλες ενημερώσεις του log. Αν όλα τα βήματα εκτελεσθούν επιτυχώς, τότε η διαδικασία είναι επιτυχημένη και ο χρήστης βλέπει μια ομοιόμορφη αλλαγή στους πίνακες του γραφικού περιβάλλοντος:

**ORIGIN** Hello up1072494@upnet.gr!

Search in Log based on: All Tables

Account Registry Income History Pricing List

Customers		Employees	Admins	Pending Requests		
ID	FIRST NAME	LAST NAME	EMAIL	REGISTER DATE	ACTIVE	PHONE
21	MARKO	POLO	employee660@sakilaemployee.org	2020-03-27 15:00:00.0	1	69420
22	RACHEL	GREEN	employee661@sakilaemployee.org	2020-03-27 15:00:00.0	1	69420
23	FRANK	OMEGLE	employee662@sakilaemployee.org	2018-01-01 10:00:00.0	0	69420
24	OCTOBER	FEST	employee663@sakilaemployee.org	2019-05-12 12:00:00.0	1	69420
25	MELENE	RACoon	employee664@sakilaemployee.org	2019-01-22 15:00:00.0	1	69420
26	OCTAPUS	HOST	employee665@sakilaemployee.org	2019-11-25 15:00:00.0	1	69420
27	SAM	MC DONALDS	employee666@sakilaemployee.org	2020-12-05 15:00:00.0	0	69420
84	maria tests id field emp...	regdat field empty, acti...	testE1	2022-08-21 03:14:25.0	1	
85	maria tests id exists (x2)	email exists, reg date n...	testE2	2012-03-15 19:20:00.0	1	69420
86	maria test regdate sucks	active 0	testE3	2022-08-21 03:19:34.0	0	
87	maria tests id exists	+all normal	testE4	2022-08-21 03:20:34.0	1	
88	maria tests	id exists to disable, acti...	testE5	2022-08-21 03:25:25.0	0	
89	no id, label missing fiel...	sos fields ok, reg date s...	testE6	2022-08-21 03:30:01.0	1	
91	maria tests	active is string	testE7	2022-08-21 03:31:34.0	1	
92	maria tests	auto_increment value	testE8	2022-08-21 03:49:49.0	1	
95	maria tests	wrong reg date	an email for testing	2022-08-24 17:06:40.0	1	
96	maria test2	wrong reg date type	rvn	2022-08-24 17:08:42.0	1	

Logout Exit

Insert Employee Delete Employee Make Admin

Όπως φαίνεται από τις εικόνες, ο υπάλληλος με κωδικό 26 δεν υπάρχει πλέον στον πίνακα υπαλλήλων.

Account Registry		Income History		Pricing List	
Customers		Employees		Admins	
ID	FIRST NAME	LAST NAME	EMAIL	REGISTER DATE	ACTIVE
26	OCTAPUS	HOST	employee665@sakilaemployee.org	2019-11-25 15:00:00.0	1
28	THODORIS	SFIKAS	up1072550@upnet.gr	2022-01-11 20:46:50.0	1
29	NTOUROU	STELLA	up1059651@upnet.gr	2022-01-11 20:48:10.0	1
30	PAPADOPOULOU	MARIA	up1072494@upnet.gr	2022-01-11 20:48:50.0	1

Έχει προστεθεί όμως στον πίνακα διαχειριστών.

#### Tab Διαχειριστών (Admins)

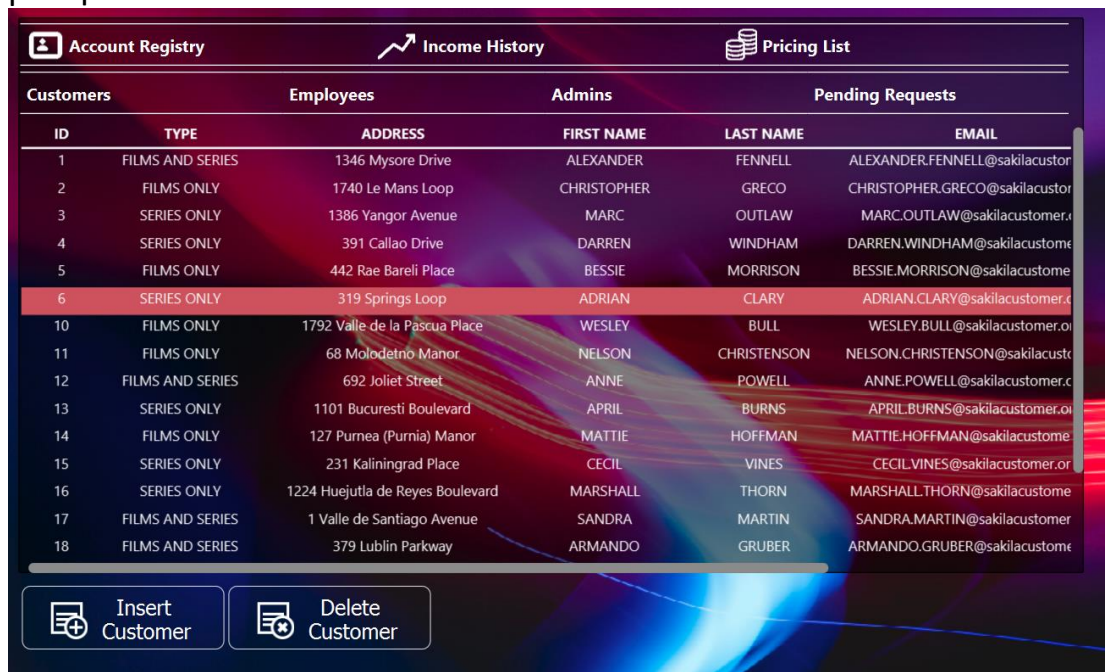
Είναι αυτό που φαίνεται στην ακριβώς προηγούμενη εικόνα. Ο διαχειριστής έχει τη δυνατότητα να διαγράψει διαχειριστή εξ' ολοκλήρου (κουμπί "Delete Admin") ή να τον μετατρέψει σε υπάλληλο (κουμπί "Make Employee"). Οι λειτουργίες αυτές είναι αντίστοιχες των προηγούμενων (η διαγραφή όπως στις δύο προηγούμενες κατηγορίες χρηστών, και η μετατροπή σε υπάλληλο είναι ακριβώς αντίστροφη της μετατροπής υπαλλήλου σε διαχειριστή, με την ατομικότητα της διαδικασίας να παραμένει).

#### Tab Εκκρεμών Αιτήσεων (Pending Requests)

Σε αυτόν τον πίνακα συγκεντρώνονται όσες αιτήσεις έχουν γίνει από πελάτες για να εγγραφούν στην υπηρεσία.



Προσφέρονται δύο λειτουργίες: αποδοχή (κουμπί "Accept") και απόρριψη (κουμπί "Decline") επιλεγμένου αιτήματος. Στην πρώτη περίπτωση, ο χρήστης σταματά να θεωρείται εκκρεμές αίτημα, οπότε είναι πλέον εμφανής στον πίνακα του Tab πελατών και μπορεί να συνδεθεί στην υπηρεσία, ενώ στη δεύτερη διαγράφεται εντελώς από τη βάση.



Ο χρήστης με κωδικό 6 είναι πλέον εμφανής στον πίνακα πελατών, διότι ο διαχειριστής αποδέχτηκε το αίτημα του.



## Ιστορικό Εσόδων | Income History

Παρουσιάζει σε πίνακα τα έσοδα της υπηρεσίας για ταινίες και επεισόδια ανά μήνα:



TOTAL INCOME		
MONTH	FILMS	EPISODES
2005-06	1.2	0.4
2005-07	0.2	0.0
2005-08	0.8	0.0
2005-09	0.4	5.0
2022-08	0.9	0.05

Ο διαχειριστής μπορεί μόνο να έχει πρόσβαση σε αυτές τις πληροφορίες, χωρίς να μπορεί να τις επεξεργαστεί.

## Χρεώσεις | Pricing List

Δείχνει στον διαχειριστή τις τρέχουσες τιμές των προϊόντων, ανάλογα με τον τύπο συνδρομής του πελάτη, όπως φαίνεται στην επόμενη εικόνα:

# ORIGIN

Hello up1072494@upnet.gr!

Search in Log based on:

Account Registry

Income History

Pricing List

All Tables

Email

Starting Date

Ending Date

Show Log Info

Logout

Exit

These are the current prices for both films and episodes according to customer categories:

FILMS ONLY:	0.4	€
SERIES ONLY:	0.2	€
FILMS AND SERIES (FILMS):	0.3	€
FILMS AND SERIES (SERIES):	0.1	€

Ο διαχειριστής μπορεί να αλλάξει τις τιμές ταινιών και επεισοδίων για κάθε κατηγορία συνδρομής, πληκτρολογώντας την νέα τιμή στο text field που περιέχει την τρέχουσα. Αν το πεδίο μείνει κενό, ή δοθεί μη έγκυρη τιμή, τότε η τιμή του προϊόντος παραμένει ίδια, όπως φαίνεται και από το επεξηγηματικό μήνυμα που εμφανίζεται στην οθόνη. Έστω ότι ο διαχειριστής δίνει τιμή για τύπο συνδρομής “SERIES ONLY” 0.2.3 €.

Account RegistryIncome HistoryPricing List

These are the current prices for both films and episodes according to customer categories:

FILMS ONLY:	0.4	€
SERIES ONLY:	0.2	€
FILMS AND SERIES (FILMS):	0.3	€
FILMS AND SERIES (SERIES):	0.1	€

Invalid price inserted. No changes made.

## Πληροφορίες Log | Log Info

Πρόκειται για παράθυρο που περιέχει πίνακα με τις καταγεγραμμένες ενέργειες στον πίνακα log της βάσης. Το παράθυρο αυτό είναι pop-up και εμφανίζεται μόλις πατηθεί το κουμπί “Show Log Info” στο αριστερό μέρος της αρχικής οθόνης διαχειριστή. Ο διαχειριστής μπορεί να επιλέξει ποιες ενέργειες θέλει να δει με βάση τα «φίλτρα αναζήτησης» που υπάρχουν επίσης στο αριστερό μέρος της αρχικής, ακριβώς πάνω από το προαναφερόμενο κουμπί.

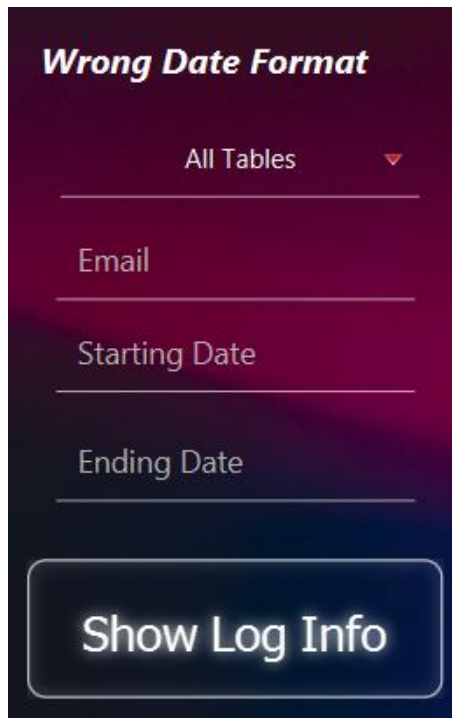
ID	TYPE	ADDRESS	FIRST NAME	LAST NAME	EMAIL
1	FILMS AND SERIES	1346 Mysore Drive	ALEXANDER	FENNELL	ALEXANDER.FENNELL@sakilacustomer.com
2	FILMS ONLY	1740 Le Mans Loop	CHRISTOPHER	GRECO	CHRISTOPHER.GRECO@sakilacustomer.com
3	SERIES ONLY	1386 Yangor Avenue	MARC	OUTLAW	MARC.OUTLAW@sakilacustomer.com
4	SERIES ONLY	391 Callao Drive	DARREN	WINDHAM	DARREN.WINDHAM@sakilacustomer.com
5	FILMS ONLY	442 Rae Bareli Place	BESSIE	MORRISON	BESSIE.MORRISON@sakilacustomer.com
6	SERIES ONLY	319 Springs Loop	ADRIAN	CLARY	ADRIAN.CLARY@sakilacustomer.com
10	FILMS ONLY	1792 Valle de la Pascua Place	WESLEY	BULL	WESLEY.BULL@sakilacustomer.com
11	FILMS ONLY	68 Molodetno Manor	NELSON	CHRISTENSON	NELSON.CHRISTENSON@sakilacustomer.com
12	FILMS AND SERIES	692 Joliet Street	ANNE	POWELL	ANNE.POWELL@sakilacustomer.com
13	SERIES ONLY	1101 Bucuresti Boulevard	APRIL	BURNS	APRIL.BURNS@sakilacustomer.com
14	FILMS ONLY	127 Purnea (Purnia) Manor	MATTIE	HOFFMAN	MATTIE.HOFFMAN@sakilacustomer.com
15	SERIES ONLY	231 Kaliningrad Place	CECIL	VINES	CECIL.VINES@sakilacustomer.com
16	SERIES ONLY	1224 Huejutla de Reyes Boulevard	MARSHALL	THORN	MARSHALL.THORN@sakilacustomer.com
17	FILMS AND SERIES	1 Valle de Santiago Avenue	SANDRA	MARTIN	SANDRA.MARTIN@sakilacustomer.com
18	FILMS AND SERIES	379 Lublin Parkway	ARMANDO	GRUBER	ARMANDO.GRUBER@sakilacustomer.com

Τα φίλτρα που μπορούν να χρησιμοποιηθούν είναι:

1. **Επιλογή Πίνακα:** από προεπιλογή, θα εμφανιστούν πληροφορίες του log για όλους τους πίνακες. Ο διαχειριστής μπορεί να επιλέξει κάποιον από τους υπάρχοντες πίνακες στη βάση και να δει τις ενέργειες του log που αφορούν μόνο τον συγκεκριμένο πίνακα. Προφανώς, εφόσον στο log δεν καταγράφονται ενέργειες που αφορούν τον ομώνυμο πίνακα, δεν υπάρχει επιλογή για εμφάνιση πληροφοριών log σε αυτόν.
2. **Επιλογή χρήστη:** κάθε φορά που ο διαχειριστής επιλέγει έναν χρήστη από τους πίνακες πελατών, υπαλλήλων και διαχειριστών, το πεδίο Email ανανεώνεται αυτόματα, παίρνοντας το email του επιλεγμένου χρήστη. Έτσι στο Log θα εμφανιστούν μόνο οι ενέργειες που έχει εκτελέσει ο συγκεκριμένος χρήστης, σε όποιον πίνακα έχει επιλεγεί. Ο διαχειριστής μπορεί να σβήσει το email από το πεδίο φίλτρου, δηλώνοντας ότι θέλει να δει τις ενέργειες όλων των χρηστών από τον επιλεγμένο πίνακα.



3. Επιλογή χρονικού διαστήματος: πρόκειται για τα δύο τελευταία πεδία, τα οποία είναι υποχρεωτικά και περιορίζουν το πλήθος των ενεργειών που θα εμφανιστούν στο συγκεκριμένο χρονικό διάστημα. Αν τα πεδία δεν έχουν το format της ημερομηνίας που χρησιμοποιεί η SQL, τότε εμφανίζεται επεξηγηματικό μήνυμα.



Εδώ τα φίλτρα χρονικού διαστήματος είναι κενά, οπότε δεν έχουν την επιθυμητή μορφή. Ως επέκταση θα μπορούσε να υλοποιηθεί ένας extra έλεγχος, ώστε σε αυτή την περίπτωση να εμφανίζονται οι πληροφορίες του επιλεγμένου πίνακα για τον επιλεγμένο χρήστη, από την αρχή καταγραφής των ενεργειών μέχρι την τρέχουσα ημερομηνία.

### Έξοδος από την Υπηρεσία

Υλοποιείται από τα δύο τελευταία κουμπιά στο αριστερό μέρος της αρχικής διαχειριστή. Το κουμπί “Logout”, αποσυνδέει τον χρήστη και επιστρέφει στην οθόνη σύνδεσης, ενώ το κουμπί “Exit” κλείνει εξ’ ολόκληρου την εφαρμογή

### Γενικά για τον κώδικα

Η διαφορά σε σχέση με τον Customer είναι στον χειρισμό των Exceptions, που πλέον έχουν γίνει περισσότερο user-friendly, με περισσότερα μηνύματα προς τον χρήστη.