



ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ
ΑΝΑΦΟΡΑ ΕΞΑΜΗΝΙΑΙΑΣ ΕΡΓΑΣΙΑΣ
Εαρινό εξάμηνο 2023

Ομάδα 34

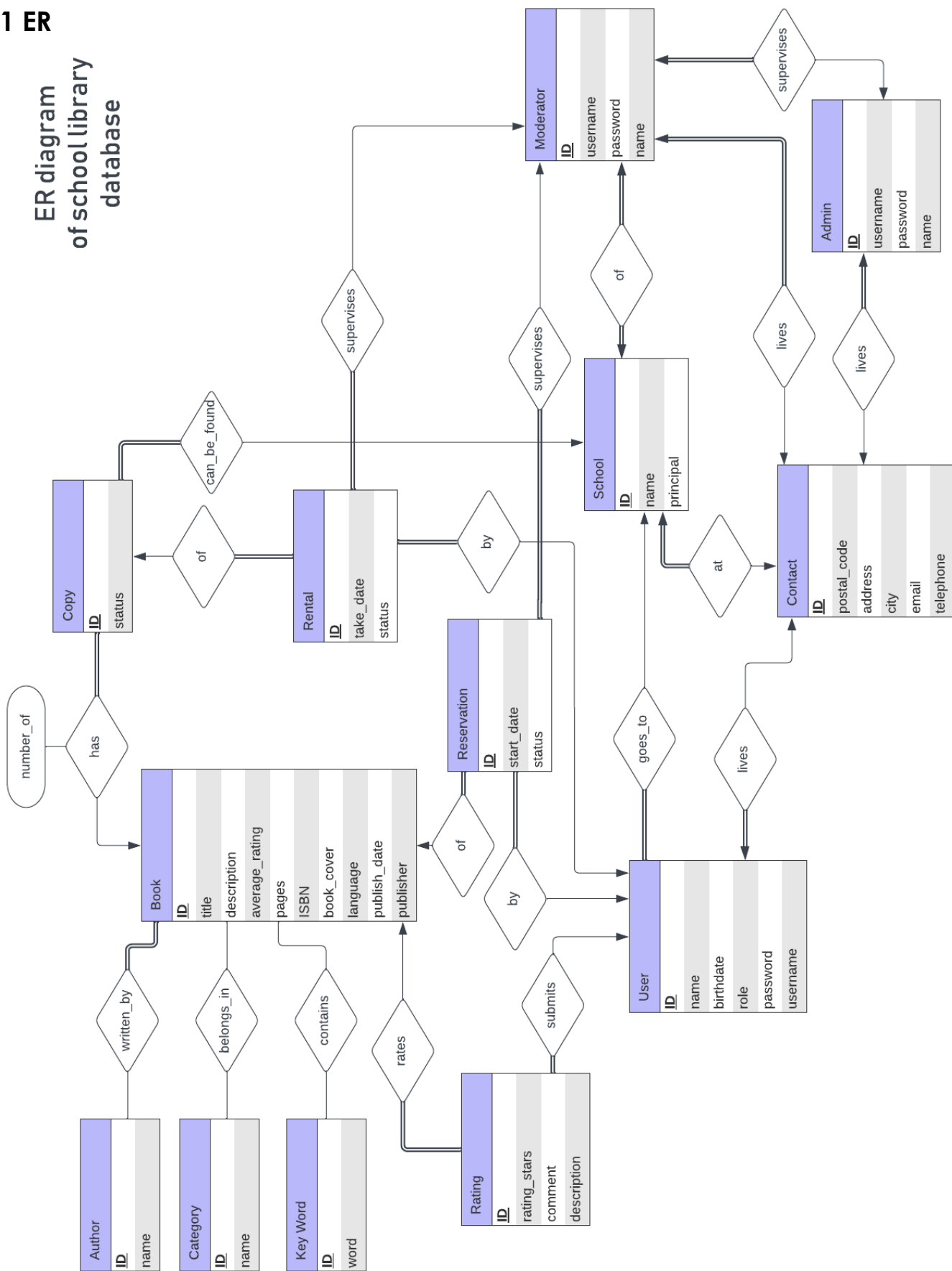
Γιάννης Μπουφίδης AM 03120162

Φωτεινή Κυριακοπούλου AM 03120182

Απόστολος Κυριακόπουλος AM

1.1 ER

ER diagram of school library database





1.2 Ανάπτυξη Βάσης

Στηριζόμενοι στο ER διάγραμμα και στις απαιτήσεις της εφαρμογής που θέλουμε να σχεδιάσουμε, υλοποιούμε τη βάση δεδομένων δημιουργώντας τα κατάλληλα tables στην MySQL.

Δομή βάσης

Ορίζουμε στη βάση τις κύριες οντότητες **book**, **user**, **reservation**, **rental** που αποτελούν το βασικό σκελετό του συστήματος αποθήκευσης και διαχείρισης πληροφοριών για τη λειτουργία της σχολικής βιβλιοθήκης.

Έπειτα, δημιουργούμε ξεχωριστές οντότητες για τα χαρακτηριστικά των βιβλίων, όπως τους συγγραφείς, τις κατηγορίες στις οποίες ανήκουν, τις λέξεις κλειδιά που περιέχουν και τα διαθέσιμα αντίτυπα. Προκειμένου, να υλοποιηθούν οι συνδέσεις μεταξύ τους ορίζουμε και τους κατάλληλους πίνακες.

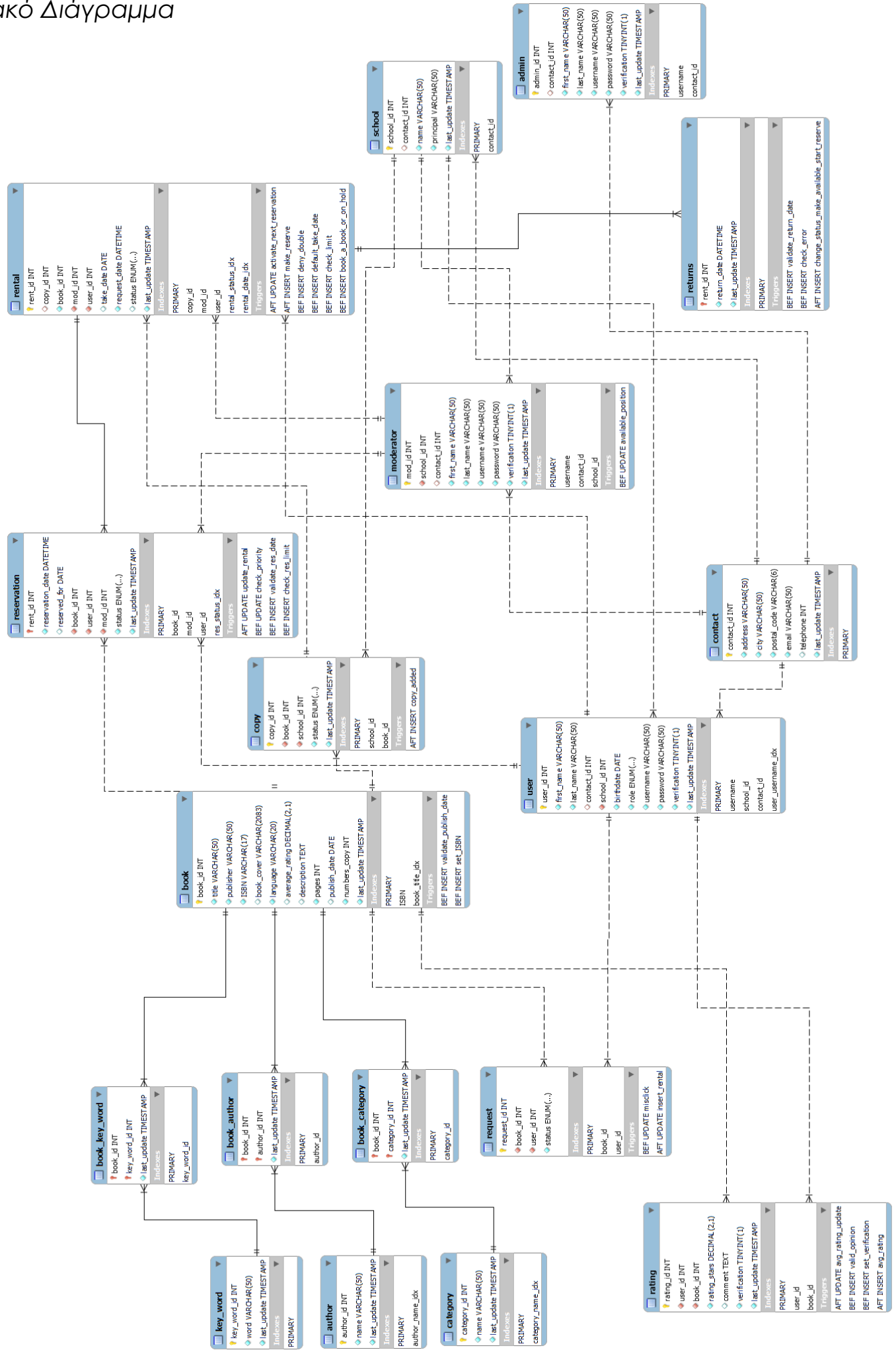
Όσον αφορά τους χρήστες της βάσης, διακρίνουμε τις οντότητες **user**, **moderator** και **admin** που αντιστοιχούν στους απλούς χρήστες της εφαρμογής, μαθητές ή εκπαιδευτικούς, στους χειριστές κάθε σχολικής μονάδας και στον κεντρικό διαχειριστή του δικτύου των σχολικών βιβλιοθηκών.

Στις οντότητες **school** και **contact** αποθηκεύουμε αντίστοιχα τα σχολεία στα οποία ανήκουν οι χρήστες καθώς και τα στοιχεία επικοινωνίας τους.

Προσθέτουμε τις οντότητες **request** και **returns** μέσω των οποίων ο χειριστής της κάθε σχολικής μονάδας μπορεί να έχει την εποπτεία των κρατήσεων και δανεισμών που πραγματοποιούν οι χρήστες.

Τέλος, δημιουργούμε την οντότητα **rating** που αναπαριστά τις αξιολογήσεις που καταχωρούν οι χρήστες για τα βιβλία της σχολικής τους βιβλιοθήκης.

Σχεσιακό Διάγραμμα





1.3 Constraints

Για να διατηρηθεί η ακεραιότητα και η συνέπεια των δεδομένων της βάσης μας ορίζουμε τους ακόλουθους περιορισμούς:

1. Primary key constraints → για κάθε οντότητα ορίζουμε μια στήλη ή έναν συνδυασμό στηλών που αναγνωρίζουν μοναδικά κάθε εγγραφή του πίνακα.

2. Foreign key constraints → ορίζουμε σχέσεις μεταξύ πινάκων, όπου το εξωτερικό κλειδί σε ένα πίνακα αναφέρεται στο πρωτεύον κλειδί ενός άλλου.

Έτσι συνδέουμε για παράδειγμα τον δανεισμό με τον χρήστη που τον πραγματοποίησε κ.ο.κ.

3. Unique constraints → επιλέγουμε κάποιες στήλες που επιθυμούμε να λαμβάνουν μοναδικές τιμές.

Για παράδειγμα απαιτούμε κάθε χρήστης να έχει μοναδικό username και email.

Επίσης αποτρέπουμε διπλότυπες τιμές π.χ. στη στήλη name του πίνακα category.

4. Not null constraints → όπου χρειάζεται επιβάλλουμε σε μια στήλη να μη παραμένει κενή κατά τη καταχώρηση εγγραφής στο πίνακα.

Για παράδειγμα, κατά την καταχώρηση ενός νέου βιβλίου στη βάση είναι απαραίτητο να συμπληρωθεί και ο τίτλος του.

5. Check constraints → ορίζουμε συνθήκες που πρέπει να ικανοποιούν τα δεδομένα σε μια στήλη

Π.χ. ο χρήστης να έχει καταχωρήσει στα στοιχεία επικοινωνίας έγκυρο email.

6. Default Constraint → σε ορισμένες στήλες ορίζουμε default τιμές κατά την εισαγωγή δεδομένων.

Για παράδειγμα, κατά την εισαγωγή νέων βιβλίων ορίζουμε το availability status των αντίτυπων να είναι «διαθέσιμο προς δανεισμό».



1.4 Ευρετήρια

Προκειμένου να βελτιώσουμε την απόδοση της εφαρμογής μας, ορίζουμε κατάλληλα ευρετήρια που μειώνουν το χρόνο και την κατανάλωση πόρων για την εκτέλεση των ερωτημάτων και αναζητήσεων πάνω στα δεδομένα της βάσης.

-Ευρετήρια για τα **IDs** των οντοτήτων: ορίζονται αυτόματα από την MySQL ευρετήρια για τα πρωτεύοντα κλειδιά των πινάκων.

-Ευρετήρια για τα **IDs** των σχέσεων μεταξύ των οντοτήτων: ορίζονται αυτόματα ευρετήρια για τα ξένα κλειδιά των πινάκων εφόσον αυτά αποτελούν με τη σειρά τους πρωτεύοντα κλειδιά των οντοτήτων-προέλευσης.

-Ευρετήρια που επιλέγουμε να ορίσουμε σύμφωνα με τις απαιτήσεις της εφαρμογής μας (τα ζητούμενα των queries και τα triggers που υλοποιούμε):

1. **book_title_idx** στο attribute name του πίνακα book καθώς πραγματοποιείται συχνή αναζήτηση τίτλων βιβλίων.

2. **category_name_idx** στο attribute name του πίνακα category καθώς πραγματοποιείται συχνή αναζήτηση των διάφορων κατηγοριών

3. **author_name_idx** στο attribute name του πίνακα author καθώς πραγματοποιείται συχνή αναζήτηση των ονομάτων των συγγραφέων

4. **user_username_idx** στο attribute username του πίνακα user καθώς πραγματοποιείται συχνή αναζήτηση των αναγνωριστικών των χρηστών

5. **rental_status_idx, rental_date_idx** στα attributes status και take_date του πίνακα rental καθώς πραγματοποιείται συχνή προσπέλασή της κατάστασης και ημερομηνίας καταχώρησης των δανεισμών, τόσο από τον διαχειριστή της σχολικής μονάδας, όσο και από τα διάφορα triggers που υλοποιούν τους ελέγχους. Αντίστοιχα, το ευρετήριο **reservation_status_idx** στο attribute status του πίνακα reservation καθώς πραγματοποιείται συχνή προσπέλαση της κατάστασης των καταχωρημένων κρατήσεων.



1.5 Triggers & Events

Για την ορθή λειτουργία της εφαρμογής μας υλοποιούμε τα ακόλουθα *triggers*:

- **validate_publish_date**: επαληθεύει άμα η ημερομηνία έκδοσης του βιβλίου είναι έγκυρη.
- **set_ISBN**: ορίζει ένα ISBN σε περίπτωση που στη καταχώρηση βιβλίου το αντίστοιχο πεδίο δε συμπληρωθεί.
- **copy_added**: ενημερώνει τον συνολικών αριθμό των αντίτυπων ενός βιβλίου σε περίπτωση καταχώρησης καινούργιου αντίτυπου.
- **set_verification**: για την καταχώρησης κριτικής η οποία συνοδεύεται από σχόλιο θέτει το attribute `verification = 1`, εάν ο χρήστης είναι εκπαιδευτικός, ή `verification = 0`, εάν ο χρήστης είναι μαθητής. Στη δεύτερη περίπτωση απαιτείται η έγκριση του χειριστή για τη δημοσίευση του σχολίου.
- **avg_rating**: υπολογίζει και ενημερώνει το μέσο όρο των αξιολογήσεων που έχουν αφήσει οι χρήστες για ένα συγκεκριμένο βιβλίο.
- **valid_opinion**: επαληθεύει αν ο χρήστης καταχωρεί αξιολόγηση σε βιβλίο που έχει δανειστεί.
- **insert_rental**: ύστερα από έγκριση του χειριστή στο αίτημα του χρήστη καταχωρεί τον δανεισμό / την κράτηση.
- **misclick**: επιτρέπει στον χειριστή να εγκρίνει μόνο αιτήματα που εκκρεμούν.
- **available_position**: υλοποιεί τον περιορισμό ότι κάθε σχολική μονάδα μπορεί να έχει έναν μοναδικό ενεργό χειριστή
- **validate_return_date**: ελέγχει την εγκυρότητα της ημερομηνίας επιστροφής του δανεισμού.
- **check_error**: ελέγχει άμα καταχωρείται επιστροφή βιβλίου που είχε δανειστεί.
- **change_status_make_available_start_reserve**: ελέγχει άμα καταχωρείται επιστροφή βιβλίου για το οποίο υπάρχει κράτηση και στη συνέχεια ενεργοποιεί

την κράτηση που έχει προτεραιότητα, διαφορετικά ενημερώνει τη διαθεσιμότητα του αντιτύπου.

- **book_a_book_or_on_hold, make_reserve:** ελέγχουν άμα υπάρχουν διαθέσιμα αντίτυπα προς δανεισμό, διαφορετικά καταχωρούν αίτημα κράτησης.
- **check_limit:** ελέγχει τους περιορισμούς των χρηστών όσον αφορά τα όρια δανεισμού
- **check_res_limit:** ελέγχει τους περιορισμούς των χρηστών όσον αφορά τα όρια κράτησης
- **deny_double:** εμποδίζει τον χρήστη να κάνει κράτηση ή δανεισμό βιβλίου που έχει στη κατοχή του.
- **default_take_date:** ενημερώνει την ημερομηνία διεκπεραίωσης του αιτήματος δανεισμού.
- **activate_next_reservation:** ελέγχει άμα ακυρώθηκε κράτηση και στη συνέχεια ενεργοποιεί (εάν υπάρχει) την επόμενη κράτηση που βρίσκεται σε προτεραιότητα
- **check_priority:** ελέγχει άμα η ενημέρωση των στοιχείων της κράτησης είναι έγκυρη
- **update_rental:** ενεργοποιεί το δανεισμό βιβλίου που ήταν σε κράτηση

Επίσης, υλοποιούμε τα ακόλουθα *events*:

- **check_for_latency:** ενημερώνει τους δανεισμούς βιβλίων των οποίων έχει καθυστερήσει η επιστροφή.
- **cancel_reservation:** ακυρώνει τις κρατήσεις που έχουν υπερβεί την μία εβδομάδα.



2. DML dan DDL scripts

DDL Script

```
create table book (
  book_id int not null auto_increment,
  title varchar(50) not null,
  publisher varchar(50) not null,
  ISBN varchar(17) not null unique,
  book_cover varchar(2083) default null,
  language varchar(20) not null, check (language regexp '^[a-zA-Z\s]+$'),
  average_rating decimal(2,1) default null,
  description text default null, check (length(description) <= 1000),
  pages int not null, check (pages > 0),
  publish_date date default null,
  numbers_copy int not null default 0,
  last_update timestamp not null default current_timestamp on update current_timestamp,
  primary key (book_id)
);

create INDEX book_title_idx ON book(title);

create table author (
  author_id int not null auto_increment,
  name varchar(50) not null, check (name regexp '^[a-zA-Z\s.-]+'),
  last_update timestamp not null default current_timestamp on update current_timestamp,
  primary key (author_id)
);

create INDEX author_name_idx ON author(name);

create table category (
  category_id int not null auto_increment,
  name varchar(50) not null unique, check (name regexp '^[a-zA-Z\s-]+'),
  last_update timestamp not null default current_timestamp on update current_timestamp,
  primary key (category_id)
);

create INDEX category_name_idx ON category(name);

create table key_word (
  key_word_id int not null auto_increment,
  word varchar(50) not null unique, check (word regexp '^[a-zA-Z\s-]+'),
  last_update timestamp not null default current_timestamp on update current_timestamp,
  primary key (key_word_id)
);

create table book_category (
  book_id int not null,
  category_id int not null,
  last_update timestamp not null default current_timestamp on update current_timestamp,
  primary key (book_id, category_id),
  foreign key (book_id) references book (book_id) on update cascade on delete cascade ,
```

```

    foreign key (category_id) references category (category_id) on update cascade on delete cascade
);

create table book_author (
    book_id int not null,
    author_id int not null,
    last_update timestamp not null default current_timestamp on update current_timestamp,
    primary key (book_id, author_id),
    foreign key (book_id) references book (book_id) on update cascade on delete cascade,
    foreign key (author_id) references author (author_id) on update cascade on delete cascade
);

create table book_key_word (
    book_id int not null,
    key_word_id int not null,
    last_update timestamp not null default current_timestamp on update current_timestamp,
    primary key (book_id, key_word_id),
    foreign key (book_id) references book (book_id) on update cascade on delete cascade,
    foreign key (key_word_id) references key_word (key_word_id) on update cascade on delete cascade
);

create table contact (
    contact_id int not null auto_increment,
    address varchar(50) not null,
    city varchar(50) not null, check (city regexp '^[a-zA-Z\s-]+$'),
    postal_code varchar(6) not null, check (postal_code regexp '^[1-9][0-9]{2} [0-9]{2}$'),
    email varchar(50) not null, check (email regexp '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'),
    telephone int default null,
    last_update timestamp not null default current_timestamp on update current_timestamp,
    primary key (contact_id)
);

create table school (
    school_id int not null auto_increment,
    contact_id int null,
    name varchar(50) not null unique,
    principal varchar(50) not null, check (principal regexp '^[a-zA-Z -]+$'),
    last_update timestamp not null default current_timestamp on update current_timestamp,
    primary key (school_id),
    foreign key (contact_id) references contact (contact_id) on update cascade on delete set null
);

create table copy (
    copy_id int not null auto_increment,
    book_id int not null,
    school_id int not null,
    status enum('available', 'reserved', 'rented') not null default 'available',
    last_update timestamp not null default current_timestamp on update current_timestamp,
    primary key (copy_id),
    foreign key (school_id) references school (school_id) on update cascade on delete cascade,
    foreign key (book_id) references book (book_id) on update cascade on delete cascade
);

create table user (
    user_id int not null auto_increment,
    first_name varchar(50) not null, check (first_name regexp '^[a-zA-Z-]+$'),
    last_name varchar(50) not null, check (last_name regexp '^[a-zA-Z-]+$'),
    contact_id int not null,
    school_id int not null,
    birthdate date not null,
    role enum('teacher', 'student') not null,

```

```

username varchar(50) not null unique,
password varchar(50) not null,
verification bool not null default 0,
last_update timestamp not null default current_timestamp on update current_timestamp,
primary key (user_id),
foreign key (school_id) references school (school_id) on update cascade on delete cascade,
foreign key (contact_id) references contact (contact_id) on update cascade on delete cascade
);
create INDEX user_username_idx ON user(username);

create table rating (
rating_id int not null auto_increment,
user_id int not null,
book_id int not null,
rating_stars decimal(2,1) not null, check (rating_stars >= 0 and rating_stars <= 5),
comment text default null, check (length(comment) <= 500),
verification bool not null,
last_update timestamp not null default current_timestamp on update current_timestamp,
primary key (rating_id),
foreign key (user_id) references user (user_id) on update cascade on delete cascade,
foreign key (book_id) references book (book_id) on update cascade on delete cascade
);

create table moderator (
mod_id int not null auto_increment,
school_id int not null,
contact_id int null,
first_name varchar(50) not null, check (first_name regexp '^[a-zA-Z-]+$'),
last_name varchar(50) not null, check (last_name regexp '^[a-zA-Z-]+$'),
username varchar(50) not null unique,
password varchar(50) not null,
verification bool not null default 0,
last_update timestamp not null default current_timestamp on update current_timestamp,
primary key (mod_id),
foreign key (contact_id) references contact (contact_id) on update cascade on delete set null,
foreign key (school_id) references school (school_id) on update cascade on delete cascade
);

create table admin (
admin_id int not null auto_increment,
contact_id int null,
first_name varchar(50) not null, check (first_name regexp '^[a-zA-Z-]+$'),
last_name varchar(50) not null, check (last_name regexp '^[a-zA-Z-]+$'),
username varchar(50) not null unique,
password varchar(50) not null,
verification bool not null default 1,
last_update timestamp not null default current_timestamp on update current_timestamp,
primary key (admin_id),
foreign key (contact_id) references contact (contact_id) on update cascade on delete set null
);

create table rental (
rent_id int not null auto_increment,
copy_id int default null,
book_id int not null,
mod_id int not null,
user_id int not null,
take_date date,
request_date datetime not null default current_timestamp,
status enum('queued up', 'terminated', 'active', 'late', 'returned') default 'active',
last_update timestamp not null default current_timestamp on update current_timestamp,

```

```

    primary key (rent_id),
    foreign key (copy_id) references copy (copy_id) on update cascade on delete cascade,
    foreign key (mod_id) references moderator (mod_id) on update cascade on delete cascade,
    foreign key (user_id) references user (user_id) on update cascade on delete cascade
);
CREATE INDEX rental_status_idx ON rental (status);
CREATE INDEX rental_date_idx ON rental (take_date);

create table reservation (
    rent_id int not null,
    reservation_date datetime not null default current_timestamp,
    reserved_for date default null,
    book_id int not null,
    user_id int not null,
    mod_id int not null,
    status enum('on hold', 'active', 'terminated', 'completed') not null default 'on hold',
    last_update timestamp not null default current_timestamp on update current_timestamp,
    primary key (rent_id),
    foreign key (rent_id) references rental (rent_id) on update cascade on delete cascade,
    foreign key (book_id) references book (book_id) on update cascade on delete cascade,
    foreign key (mod_id) references moderator (mod_id) on update cascade on delete cascade,
    foreign key (user_id) references user (user_id) on update cascade on delete cascade
);
CREATE INDEX res_status_idx ON reservation (status);

create table returns (
    rent_id int not null,
    return_date datetime not null default current_timestamp,
    last_update timestamp not null default current_timestamp on update current_timestamp,
    primary key (rent_id),
    foreign key (rent_id) references rental (rent_id) on update cascade on delete cascade
);

create table request (
    request_id int not null auto_increment,
    book_id int not null,
    user_id int not null,
    status enum('pending', 'accepted', 'declined') not null default 'pending',
    primary key (request_id),
    foreign key (book_id) references book (book_id) on update cascade on delete cascade,
    foreign key (user_id) references user (user_id) on update cascade on delete cascade
);

/* -----triggers-----
-----*/

/*book triggers*/
delimiter $$
create trigger validate_publish_date
before insert on book
for each row
begin
    if new.publish_date > CURRENT_DATE() then
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Publish date cannot be in the future.';
    end if;
end$$
delimiter ;

delimiter $$
create trigger set_ISBN

```

```

before insert on book
for each row
begin
    if new.ISBN is null then
        set new.ISBN = generate_ISBN();
    end if;
end$$
delimiter ;

delimiter $$
create trigger copy_added
after insert on copy
for each row
begin
    DECLARE num_copies INT;
    SELECT COUNT(*) INTO num_copies FROM copy WHERE book_id = NEW.book_id;
    UPDATE book SET numbers_copy = num_copies WHERE book_id = NEW.book_id;
end$$
delimiter ;

/*moderator triggers*/
delimiter $$
create trigger available_position
before update on moderator
for each row
begin
    declare the_actives int;

    select count(*) into the_actives
    from moderator
    where school_id=new.school_id and verification=1;

    if (the_actives > 0 and new.verification = 1) then
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Try next time.';
    end if;
end$$
delimiter ;

/* rating triggers */
delimiter $$
create trigger set_verification before insert on rating
for each row
begin
    DECLARE user_role VARCHAR(50);
    SELECT role INTO user_role FROM user WHERE user_id = NEW.user_id;
    IF NEW.comment IS NULL THEN
        SET NEW.verification = 1;
    ELSE
        IF user_role = 'teacher' THEN
            SET NEW.verification = 1;
        ELSE
            SET NEW.verification = 0;
        END IF;
    END IF;
end $$
delimiter $$

delimiter $$
create trigger avg_rating after insert on rating
for each row
begin

```

```

declare average decimal(2,1);
select avg(rating_stars) into average from rating r
where r.book_id = new.book_id and r.verification = true;

update book b
set b.average_rating = average
where b.book_id = new.book_id;
end $$
delimiter $$

delimiter $$
create trigger avg_rating_update after update on rating
for each row
begin
declare average decimal(2,1);
select avg(rating_stars) into average from rating r
where r.book_id = new.book_id and r.verification = true;

update book b
set b.average_rating = average
where b.book_id = new.book_id;
end $$
delimiter $$

delimiter $$
create trigger valid_opinion before insert on rating
for each row
begin
DECLARE the_reads int;

select count(*) into the_reads
from rental
where user_id=new.user_id and book_id=new.book_id and status in ('active', 'late', 'returned');

if the_reads=0 then
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Give it a shot first.';
end if;
end $$
delimiter $$

/* request triggers */
delimiter $$
create trigger insert_rental
after update on request
for each row
begin
if new.status = 'accepted' then
    insert into rental (book_id, user_id) values (new.book_id, new.user_id);
end if;
end $$
delimiter ;

delimiter $$
create trigger misclick
before update on request
for each row
begin
if old.status <> 'pending' then
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You misclicked.';
end if;
end $$

```



```

delimiter ;

/* returns triggers */
delimiter $$
CREATE TRIGGER validate_return_date
BEFORE INSERT ON returns
FOR EACH ROW
BEGIN
    declare the_day date;

    select r.take_date into the_day
    from rental r
    where r.rent_id = new.rent_id;

    IF NEW.return_date > CURRENT_DATE() THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Return date cannot be in the future.';
    END IF;

    IF NEW.return_date < the_day THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You went back to the future?';
    END IF;
END$$
delimiter ;

delimiter $$
CREATE TRIGGER check_error
BEFORE INSERT ON returns
FOR EACH ROW
BEGIN
    declare the_status enum('queued up', 'terminated', 'active', 'late', 'returned');

    select r.status into the_status
    from rental r where r.rent_id = new.rent_id;

    IF the_status not in ('active', 'late') THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You sure you can return that?';
    END IF;
END$$
delimiter ;

delimiter $$
create trigger change_status_make_available_start_reserve after insert on returns
for each row
begin
    declare the_copy int;
    declare the_book int;
    declare the_reservations int;
    declare the_reservation int;
    declare the_school int;

    select u.school_id into the_school
    from rental r
    join user u on u.user_id = r.user_id
    where r.rent_id = new.rent_id;

    select r.copy_id into the_copy
    from rental r where r.rent_id = new.rent_id;

    select c.book_id into the_book
    from copy c where c.copy_id = the_copy;

```

```

update rental set status = 'returned' where rent_id = new.rent_id;

select count(*) into the_reservations
from reservation res
    join user u on u.user_id = res.user_id
where res.book_id = the_book and res.status = 'on hold' and u.school_id = the_school;

if the_reservations <> 0 then
    select res.rent_id into the_reservation
    from reservation res
    join user u on u.user_id = res.user_id
    where res.book_id = the_book and status = 'on hold' and u.school_id = the_school
    order by reservation_date limit 1;
    update reservation set status = 'active', reserved_for = curdate() where rent_id = the_reservation;
    update copy set status = 'reserved' where copy_id = the_copy;
else
    update copy set status = 'available' where copy_id = the_copy;
end if;

end$$
delimiter ;

/* rental-reservation triggers */
delimiter $$
create trigger book_a_book_or_on_hold before insert on rental
for each row
begin
    declare available_copies int;
    declare total_copies int;
    declare the_school int;
    declare the_mod int;

    select school_id into the_school
    from user where user_id = new.user_id;

    select mod_id into the_mod
    from moderator m where m.school_id = the_school and m.verification = true;

    select count(*) into available_copies from copy c
    where c.book_id = new.book_id and c.school_id = the_school and c.status = 'available';

    select count(*) into total_copies from copy c
    where c.book_id = new.book_id and c.school_id = the_school;

    set new.mod_id = the_mod;

    if available_copies > 0 then
        set new.copy_id = (select min(c.copy_id)
        from copy c
        where c.book_id = new.book_id and c.school_id = the_school and c.status = 'available'
        order by c.copy_id);
        set new.take_date = date(new.request_date);
        update copy set status = 'rented' where copy_id = new.copy_id;
    elseif total_copies > 0 then
        set new.status = 'queued up';
        set new.take_date = null;
    else
        signal sqlstate '45000' set message_text = 'Book not available';
    end if;
end$$
delimiter ;

```

```

delimiter $$
create trigger make_reserve after insert on rental
for each row
begin
    if new.status = 'queued up' then
        insert into reservation (rent_id, book_id, user_id, mod_id, reservation_date)
        values (new.rent_id, new.book_id, new.user_id, new.mod_id, new.request_date);
    end if;
end$$
delimiter ;

delimiter $$
create trigger check_limit before insert on rental
for each row
begin
    declare rentals int;
    declare user_role enum('teacher', 'student');

    select role INTO user_role from user u where u.user_id = NEW.user_id;
    select count(*) into rentals
    from rental r
    where r.user_id = NEW.user_id and r.take_date > date_sub(now(), interval 1 week) and r.copy_id is not null;

    if user_role = 'student' then /*student case*/

        if rentals >= 2 then
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'You have reached the limit of rentals for the week';
        end if;
    elseif user_role = 'teacher' then /*teacher case*/
        if rentals >= 1 then
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'You have reached the limit of rentals for the week';
        end if;
    end if;
end$$
delimiter ;

delimiter $$
create trigger deny_double before insert on rental
for each row
begin
    declare double_rent int;

    select count(*) into double_rent
    from rental where user_id = new.user_id and book_id = new.book_id
    and status in ('active', 'queued up', 'late');

    if double_rent > 0 then
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You have an active rental/reservation of this
book!';
    end if;
end$$
delimiter ;

delimiter $$
create trigger default_take_date before insert on rental
for each row
begin
    if new.take_date is null and new.status = 'active' then set new.take_date = curdate();

```

```

        end if;
end$$
delimiter;

delimiter $$
CREATE TRIGGER validate_res_date
BEFORE INSERT ON reservation
FOR EACH ROW
BEGIN
    IF NEW.reserved_for < CURRENT_DATE() and new.reserved_for is not null THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Reservation date cannot be in the past.';
    END IF;
END$$
delimiter ;

delimiter $$
create trigger check_res_limit before insert on reservation
for each row
begin
    declare reservations int;
    declare late_rental int;
    declare book_rented int;
    declare book_reserved int;
    declare user_role enum('teacher', 'student');

    select role into user_role from user where user_id = new.user_id;
    select count(*) into reservations
    from reservation
    where user_id = NEW.user_id
    and status in ('active', 'on hold') and date_sub(now(), interval 1 week) <= reservation_date;

    if user_role = 'student' then /*student case*/
        if reservations >= 2 then
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'You have reached the limit of reservations for the week';
        end if;
    elseif user_role = 'teacher' then /* teacher case */

        if reservations >= 1 then
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'You have reached the limit of reservations for the week';
        end if;
    end if;

    select count(*) into book_rented
    from rental
    where user_id = NEW.user_id and book_id = NEW.book_id and status in ('active', 'late');

    select count(*) into book_reserved
    from reservation
    where user_id = NEW.user_id and book_id = NEW.book_id and status in ('on hold', 'active');

    if book_rented <> 0 and book_reserved <> 0 then
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You have rented this book already!';
    end if;

    select count(*) into late_rental
    from rental
    where user_id = NEW.user_id and status = 'late';

```

```

if late_rental <> 0 then
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'You have a rental that is late!';
end if;
end$$
delimiter ;

delimiter $$
create trigger activate_next_reservation
after update on rental
for each row
begin
    declare the_status enum('on hold', 'active', 'terminated', 'completed');
    declare the_school int;
    declare the_copy int;
    DECLARE the_next_reservation INT;

    select status into the_status
    from reservation where rent_id = new.rent_id;

    select school_id into the_school
    from user where user_id = new.user_id;

    select min(c.copy_id) into the_copy
    from copy c
    where c.book_id = new.book_id and c.school_id = the_school and c.status = 'reserved'
    order by c.copy_id;

    if (the_status = 'active' and NEW.status = 'terminated') then
        set the_next_reservation = (
            select r.rent_id
            from reservation r
            join user u on u.user_id = r.user_id
            where r.book_id = new.book_id and r.status = 'on hold' and u.school_id = the_school
            order by r.reservation_date asc
            limit 1
        );
        update reservation set status = 'terminated' where rent_id = new.rent_id;

        if the_next_reservation is not null then
            update reservation
            set status = 'active', reserved_for = curdate()
            where rent_id = the_next_reservation;
        else
            update copy set status = 'available' where copy_id = the_copy;
        end if;
    end if;
end $$
delimiter ;

delimiter $$
create trigger update_rental
after update on reservation
for each row
begin
    declare the_school int;
    declare the_copy int;

    select school_id into the_school
    from user where user_id = new.user_id;

```

```

        select min(c.copy_id) into the_copy
        from copy c
        where c.book_id = new.book_id and c.school_id = the_school and c.status = 'reserved'
        order by c.copy_id;

    if (old.status = 'active' and NEW.status = 'completed') then
        update rental r
        set r.status = 'active', r.take_date = curdate(), r.copy_id = the_copy
        where r.rent_id = new.rent_id;
        update copy set status = 'rented' where copy_id = the_copy;
    end if;
end $$
delimiter ;

delimiter $$
create trigger check_priority
before update on reservation
for each row
begin
    IF (old.status <> 'active' and new.status in ('terminated','completed'))
        or (old.status in ('terminated','completed')) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You misclicked!';
    END IF;
end $$
delimiter ;

/* -----events-----
-----*/

delimiter $$
create event cancel_reservation
on schedule every 12 hour
starts CURRENT_TIMESTAMP + interval 5 minute/*INTERVAL 12 HOUR/*12 at night*/
do
begin
    update reservation res
    join rental r on res.rent_id = r.rent_id
    set res.status = 'terminated', r.status = 'terminated'
    where res.status = 'active' and res.reserved_for <= date_sub(now(), interval 1 week);
end $$
delimiter ;
ALTER EVENT cancel_reservation ENABLE;
SET GLOBAL event_scheduler = ON;

delimiter $$
create event check_for_latency
on schedule every 1 day
starts current_timestamp() + interval 5 minute
do
begin
    update rental set status = 'late' where status = 'active'
    and date_add(take_date, interval 7 day) <= curdate();
end$$
delimiter ;
ALTER EVENT check_for_latency ENABLE;
SET GLOBAL event_scheduler = ON;

/* -----functions-----
-----*/
DELIMITER //
CREATE FUNCTION generate_ISBN() RETURNS VARCHAR(17) deterministic
BEGIN

```



```

DECLARE core INT;
DECLARE prefix VARCHAR(3);
DECLARE str_core VARCHAR(9);
DECLARE check_digit INT;
DECLARE sum INT;
DECLARE check_digit_str VARCHAR(1);
DECLARE core_first VARCHAR(3);
DECLARE core_sec VARCHAR(4);
DECLARE core_third VARCHAR(2);
DECLARE isbn_mock VARCHAR(12);
DECLARE isbn VARCHAR(17);
DECLARE i INT;

SET prefix = '978';
SET core = floor(rand() * 999999999) + 1;
SET str_core = cast(core as char);
SET str_core = LPAD(str_core, 9, '0');

SET isbn_mock = CONCAT(prefix, str_core);

SET sum = 0;
SET i = 1;
WHILE i <= 12 DO
    IF mod(i,2) = 1
        THEN SET sum = sum + CAST(SUBSTR(isbn_mock, i, 1) as signed);
    ELSE
        SET sum = sum + 3*CAST(SUBSTR(isbn_mock, i, 1) as signed);
    END IF;
    SET i = i + 1;
END WHILE;

SET check_digit = 10 - mod(sum,10);

IF check_digit = 10 THEN
    SET check_digit_str = 'X';
ELSE
    SET check_digit_str = CAST(check_digit AS CHAR);
END IF;

SET isbn = CONCAT(prefix, '-', SUBSTR(isbn_mock, 4, 3), '-',
    SUBSTR(isbn_mock, 7, 4), '-', SUBSTR(isbn_mock, 11, 2), '-', check_digit_str);

RETURN isbn;
END//
delimiter ;

```

DDL Script

Το κομμάτι του κώδικα που αφορά την εισαγωγή δεδομένων στη βάση βρίσκεται στο σύνδεσμο του git.

3. User manual

Ακολουθεί ο οδηγός χρήσης της εφαρμογής για τους μαθητές, τους εκπαιδευτικούς και τους χειριστές των σχολείων καθώς και για τον διαχειριστή της βάσης.

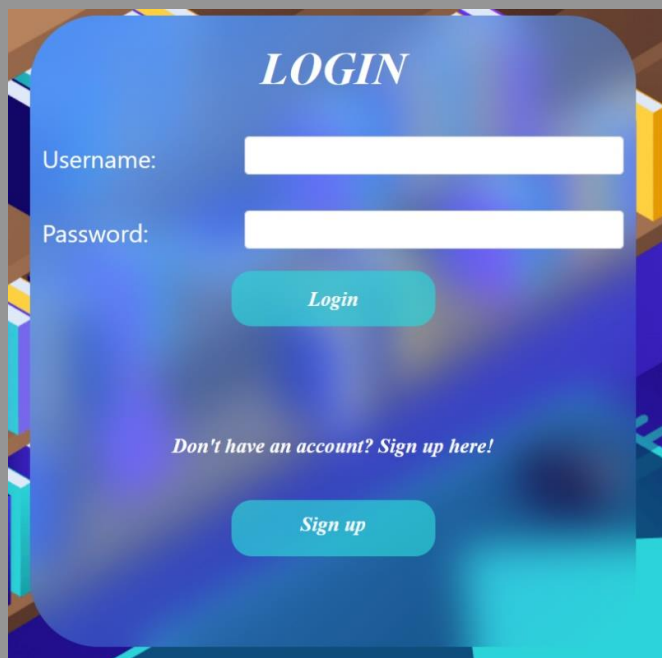

Πανελλήνια Σχολική Βιβλιοθήκη



ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ

Αθήνα 2023

□ Είσοδος στην εφαρμογή



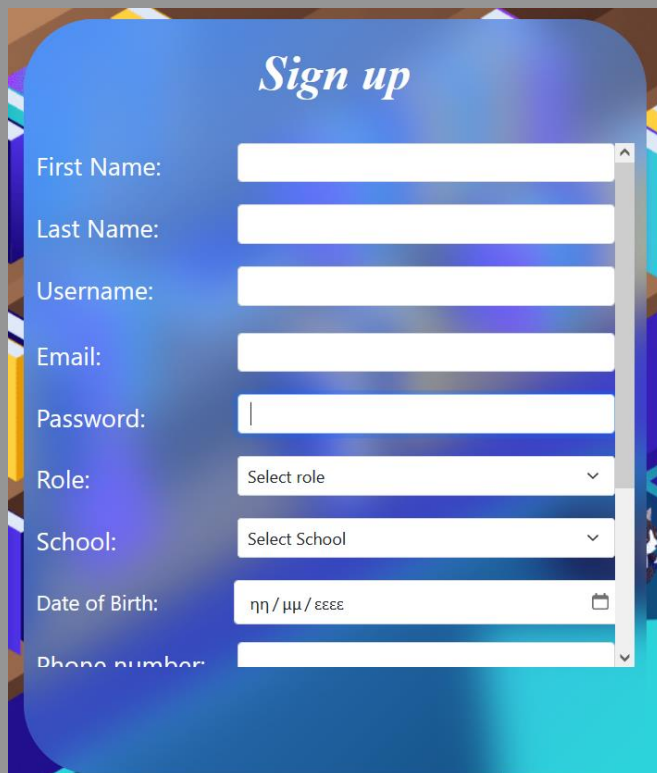
Για την είσοδο στην εφαρμογή πρέπει να εισάγεις το “Username” και το “Password” του λογαριασμού σου στα κατάλληλα πεδία και να κάνεις κλικ στο κουμπί “Login”.

Σε περίπτωση που δεν έχεις λογαριασμό μπορείς να δημιουργήσεις έναν με το πάτημα του κουμπιού Sign up.

Για τη δημιουργία νέου λογαριασμού απλού χρήστη πρέπει να συμπληρώσεις το όνομα, την ιδιότητα (student / teacher), την σχολική μονάδα στην οποία ανήκεις καθώς επίσης και τα στοιχεία επικοινωνίας σου. Στη συνέχεια, ολοκληρώνεις την εγγραφή σου με το πάτημα του κουμπιού Sign up.

Θα πρέπει να περιμένεις την έγκριση του λογαριασμού σου από τον χειριστή του σχολείου σου.

Θα λάβεις ενημερωτικό email για την επιτυχία της εγγραφής σου.



Εάν είσαι εκπαιδευτικός μπορείς να στείλεις αίτημα στον Διαχειριστή να γίνεις χειριστής της βιβλιοθήκης του σχολείου σου εισάγοντας τα στοιχεία σου και έπειτα την ιδιότητα moderator.



- Περιήγηση στην εφαρμογή
 - › Αρχική σελίδα απλού χρήστη



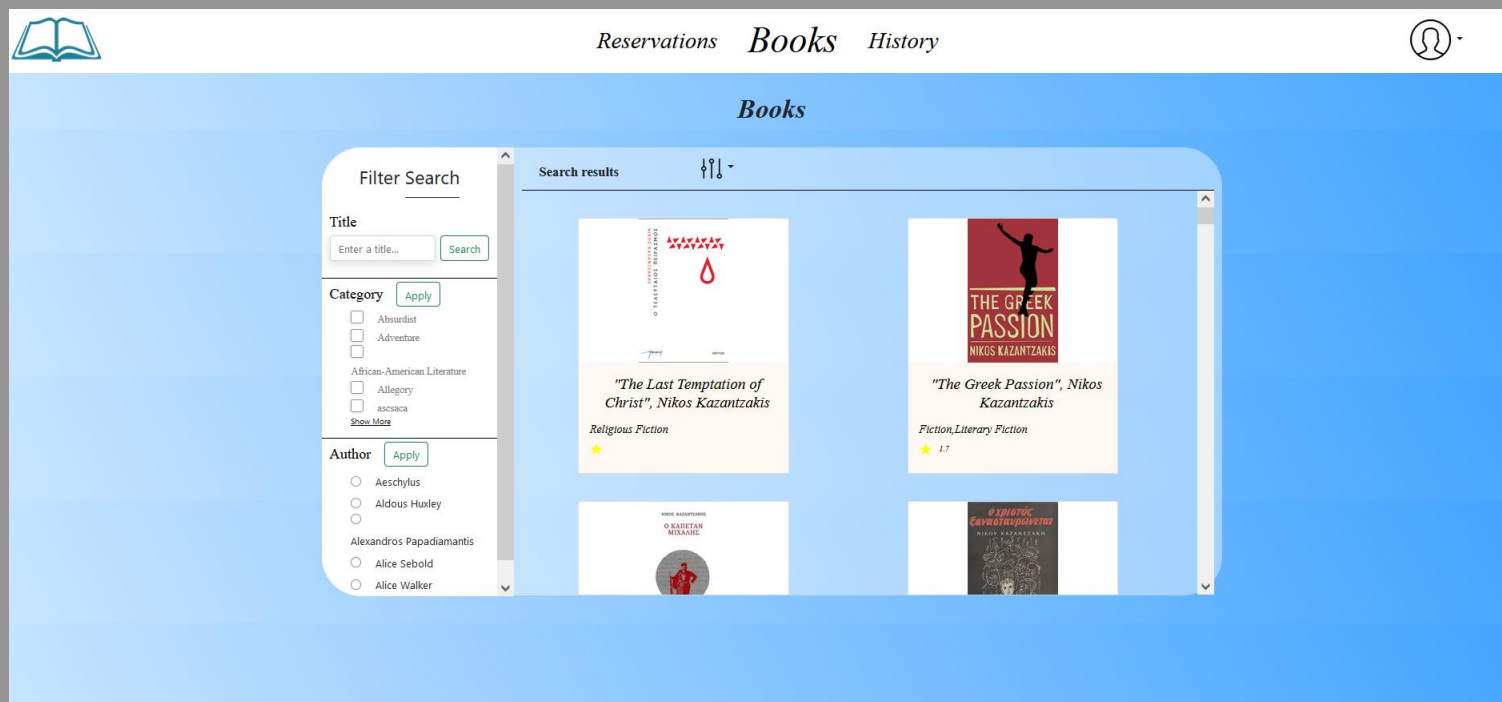
- Κάνοντας κλικ πάνω στο πλαίσιο με τίτλο “Books” μπορείς να μεταβείς στη λίστα των βιβλίων που προσφέρει η σχολική σου βιβλιοθήκη.
- Κάνοντας κλικ πάνω στο πλαίσιο με τίτλο “History” μπορείς να μεταβείς στο ιστορικό των δανεισμών που έχεις καταχωρήσει.
- Με κλικ στο πάνω δεξί εικονίδιο σου δίνεται η δυνατότητα να δεις τα στοιχεία του λογαριασμού σου ή να αποσυνδεθείς από την εφαρμογή.

Στο άνω πλαίσιο της οθόνης σου υπάρχουν συντομεύσεις που σε οποιοδήποτε μέρος της εφαρμογής μπορούν να σε επαναφέρουν στην αρχική σελίδα, στη λίστα των βιβλίων του σχολείου σου, στη λίστα με τις ενεργές κρατήσεις καθώς και στο ιστορικό δανεισμού του λογαριασμού σου.



□ Περιήγηση στην εφαρμογή

› Λίστα με τα διαθέσιμα βιβλία της σχολικής σου βιβλιοθήκης



Στη τρέχουσα σελίδα μπορείς να περιηγηθείς στη βιβλιοθήκη του σχολείου σου, να δεις τους τίτλους και τους συγγραφείς των διαθέσιμων βιβλίων, καθώς επίσης και το μέσο όρο αξιολογήσεων που έχουν αφήσει χρήστες που τα έχουν διαβάσει!

Ακόμη, μπορείς να αναζητήσεις τίτλους βιβλίων και να φιλτράρεις περαιτέρω την αναζήτησή σου ανάλογα με την κατηγορία που σε ενδιαφέρει ή τον συγγραφέα της επιλογής σου.

Σε περίπτωση που ένα βιβλίο τραβήξει τη προσοχή σου, κάνοντας κλικ σε αυτό θα μεταφερθείς στη σελίδα του βιβλίου όπου μπορείς να διαβάσεις περισσότερες πληροφορίες γι' αυτό και να καταχωρήσεις άμα επιθυμείς αίτημα για το δανεισμό/την κράτηση του.



□ Περιήγηση στην εφαρμογή

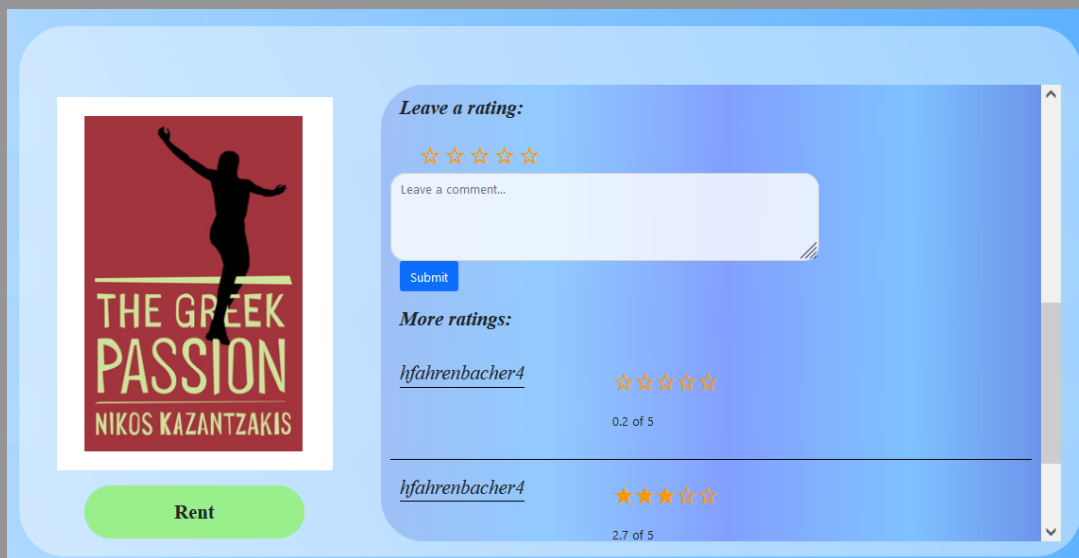
› Στοιχεία βιβλίου

Στη σελίδα αυτή μπορείς να δεις τα ακόλουθα στοιχεία:

- Τη περιγραφή του βιβλίου
- Τον αριθμό των σελίδων
- Τον εκδότη
- Τη γλώσσα στην οποία διατίθεται.



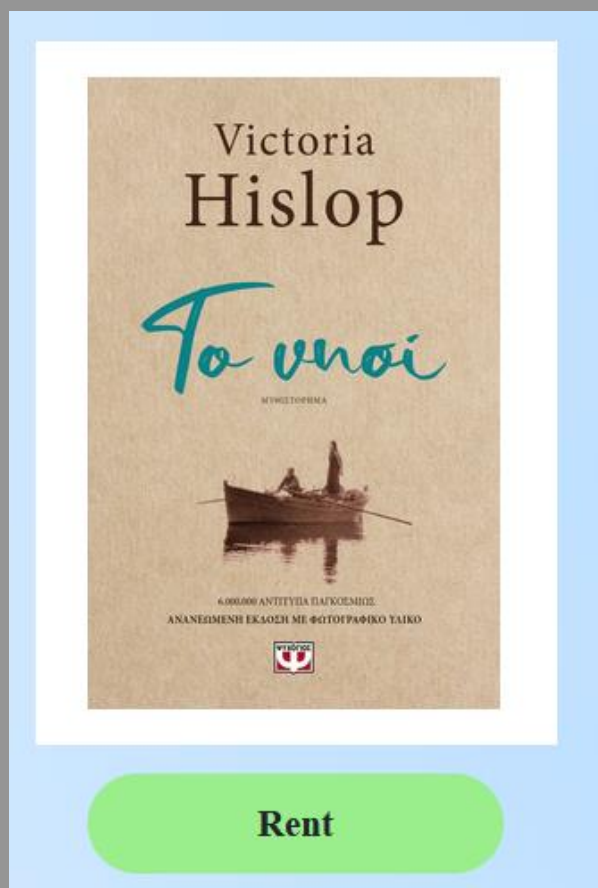
Ακόμη, με πλοήγηση στο κάτω μέρος της σελίδας μπορείς να διαβάσεις τις αξιολογήσεις που έχουν αφήσει χρήστες ή να καταχωρήσεις τη δική σου άποψη για το βιβλίο!





□ Δανεισμός

› Καταχώρηση αιτήματος



Για να καταχωρήσεις αίτημα δανεισμού βιβλίου κάνεις κλικ στο κουμπί “Rent”.

Το αίτημα σου καταχωρείται και κατόπιν έγκρισης του από τον χειριστή του σχολείου σου μπορείς να παραλάβεις το βιβλίο της επιλογής σου.

Ο δανεισμός σου θα είναι σε ισχύ για μια εβδομάδα και με το πέρας της θα πρέπει να πραγματοποιηθεί η επιστροφή του αντιτύπου.

Όρια δανεισμού:

- Αν είσαι μαθητής μπορείς να δανειστείς έως και δύο βιβλία/ εβδομάδα
- Αν είσαι εκπαιδευτικός μπορείς να δανειστείς έως και ένα βιβλίο/ εβδομάδα.

› Ιστορικό δανεισμού

Στη σελίδα “History” μπορείς να παρακολουθείς το status των δανεισμών που έχεις καταχωρήσει.

#	Book	Date of Rental	Status
1	Your Name	2022-12-01	returned



□ Κράτηση

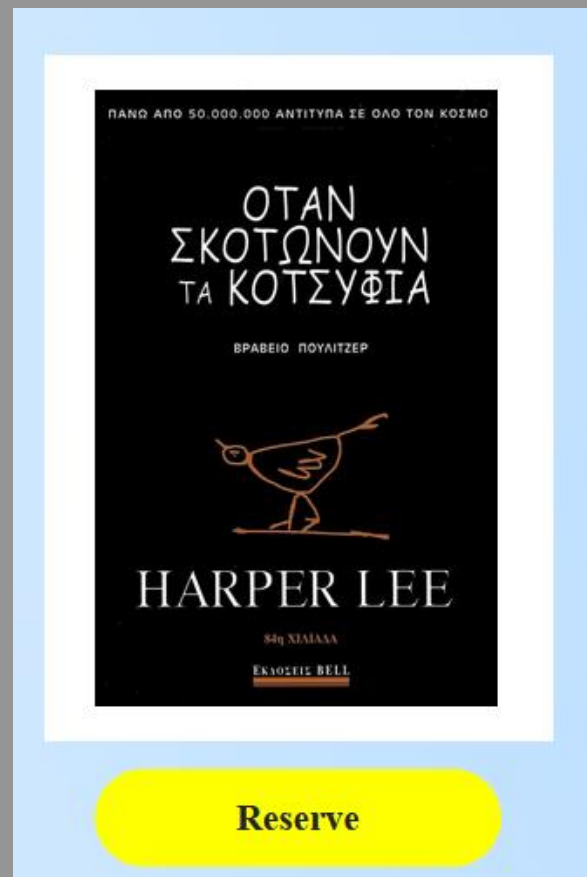
› Καταχώρηση αιτήματος

Σε περίπτωση που δεν υπάρχει κάποιο διαθέσιμο αντίτυπο μπορείς να κάνεις κλικ στο κουμπί “Reserve” και να μπεις σε προτεραιότητα για το δανεισμό του βιβλίου της επιλογής σου.

Κατόπιν έγκρισης του αιτήματος η κράτηση σου θα είναι σε ισχύ για μια εβδομάδα.

Όρια κράτησης:

- Αν είσαι μαθητής μπορείς να κάνεις δύο κρατήσεις/εβδομάδα, ενώ αν είσαι εκπαιδευτικός μπορείς να κάνεις έως και μια κράτηση/εβδομάδα.
- Δεν επιτρέπεται να καταχωρήσεις κράτηση αν έχεις καθυστερήσει την επιστροφή ενός βιβλίου ή έχεις εν ισχύ δανεισμό του ίδιου τίτλου.



› Ενεργές κρατήσεις

Στη σελίδα “Reservations” μπορείς να παρακολουθείς το status των κρατήσεων που έχεις καταχωρήσει και να ακυρώσεις άμα επιθυμείς κάποια τρέχουσα κράτησή σου.

#	Book	Activation Date	Status
1	The Works and Days	2023-06-04	active

Cancel



□ Λογαριασμός

› Προεπισκόπηση

A screenshot of a web application's 'Profile' page. The page has a light blue background. At the top, there is a navigation bar with a book icon, the text 'Reservations Books History', and a user profile icon. Below the navigation bar, the word 'Profile' is centered. The main content area contains a 'Change password' section with two input fields labeled 'Enter current password' and 'Enter password', and a 'Check' button. Below this is a 'User info' section with an 'Edit' button. The 'User info' section contains several input fields for 'First Name', 'Last Name', 'Email', 'Username', 'School', 'Address', 'City', and 'Postal Code'.

▪ Αλλαγή κωδικού πρόσβασης

Στο πλαίσιο “Change password” μπορείς να αλλάξεις τον κωδικό πρόσβασης σου.

Εφόσον εισάγεις τον τρέχων σου κωδικό, θα εμφανιστούν δύο πεδία εισαγωγής κειμένου στα οποία μπορείς να πληκτρολογήσεις τον νέο σου κωδικό.

▪ Τροποποίηση στοιχείων

Αν είσαι χρήστης με την ιδιότητα «Εκπαιδευτικός» μπορείς με κλικ στο κουμπί “Edit” να τροποποιήσεις τα στοιχεία του λογαριασμού σου.



□ Περιήγηση στην εφαρμογή

› Αρχική σελίδα Διαχειριστή



Ο διαχειριστής του δικτύου της σχολικής βιβλιοθήκης έχει τη δυνατότητα μέσω της εφαρμογής να επιβλέπει:

- Με κλικ στο “Rentals per Month” → τη λίστα με το συνολικό αριθμό δανεισμών ανά σχολείο (για κάθε μήνα του έτους).
- Με κλικ στο “Category Stats” → για δεδομένη κατηγορία βιβλίου που επιλέγει τη λίστα των συγγραφέων που ανήκουν σε αυτή και τη λίστα των εκπαιδευτικών που έχουν δανειστεί βιβλία της κατηγορίας αυτής το τελευταίο έτος.
- Με κλικ στο “Top 3 Category Pairs” → τα τρία κορυφαία ζεύγη που εμφανίστηκαν σε δανεισμούς.
- Με κλικ στο “Young Readers” → τους νέους εκπαιδευτικούς ηλικίας < 40 ετών που έχουν δανειστεί τα περισσότερα βιβλία καθώς και των αριθμό των βιβλίων.
- Με κλικ στο “More than 20” → τους χειριστές που σε διάστημα ενός έτους έχουν πραγματοποιήσει περισσότερους από 20 δανεισμούς.



- Με κλικ στο “No Rental Authors” → τους συγγραφείς των οποίων κανένα βιβλίο δεν έχει τύχει δανεισμού.
- Με κλικ στο “5 To The Top” → τους συγγραφείς που έχουν γράψει τουλάχιστον 5 βιβλία λιγότερα από τον συγγραφέα με τα περισσότερα βιβλία.

Επίσης, έχει τη δυνατότητα:

- Με κλικ στο “Moderator Handling” → να διαχειρίζεται τους χειριστές των σχολικών μονάδων.
- Με κλικ στο “New School Entry” → να προσθέτει στο δίκτυο σχολικής βιβλιοθήκης νέα σχολεία.

Η αποσύνδεση από την εφαρμογή γίνεται με κλικ στο κουμπί “Logout” στο άνω αριστερό μέρος της αρχικής σελίδας.

□ Διαχείριση χειριστών

Home Page		Mods				
#	Name	E-mail	School	Telephone	Address	
1	Dimitra Chronopoulos	ikunze8@squidoo.com	1o Geniko Lykeio Kozanis	2109012345	Aristoteleous Street 9	Cancel
2	Antonis Petropoulos	antoine@gmail.com	1o Geniko Lykeio Kozanis	2109847837	Vorra 24	
3	Despina Papadakis	jglannotti4@mozilla.org	34o Gymnasio Athinon	2105678901	Marathonos Avenue 5	Cancel
4	Alexandros Symeonidis	fscrippsk@php.net	Anatoliko Lykeio Thessalonikis	2106765432	Koral Street 21	Cancel
5	Andreas Aggelopoulos	pcasboit5@t.co	Geniko Lykeio Naxiou	2106789012	Leoforos Voulagmenis 6	Verify
6	Konstantinos Antoniadis	ikon5@eats.com	Geniko Lykeio Naxiou	2107645937	Apollonios 35	Verify
7	Eleftheria Stavropoulos	tdumand3@github.io	Geniko Lykeio Patron	2104567890	Voulagmenis Avenue 4	Cancel

Στη τρέχουσα σελίδα, ο Διαχειριστής μπορεί να απενεργοποιεί λογαριασμούς Χειριστών (κουμπί “Cancel”) ή να εγκρίνει τις αιτήσεις εγγραφής τους (κουμπί “Verify”).



□ Προσθήκη σχολικών μονάδων στο δίκτυο

The screenshot shows a web application interface with a dark sidebar on the left and a main content area. The sidebar contains a 'Home Page' link at the top and several icons below it, including a school building icon labeled 'New School'. The main content area has a title 'Add a New School' and a green button labeled 'Add this school'. Below the title is a form with the following fields:

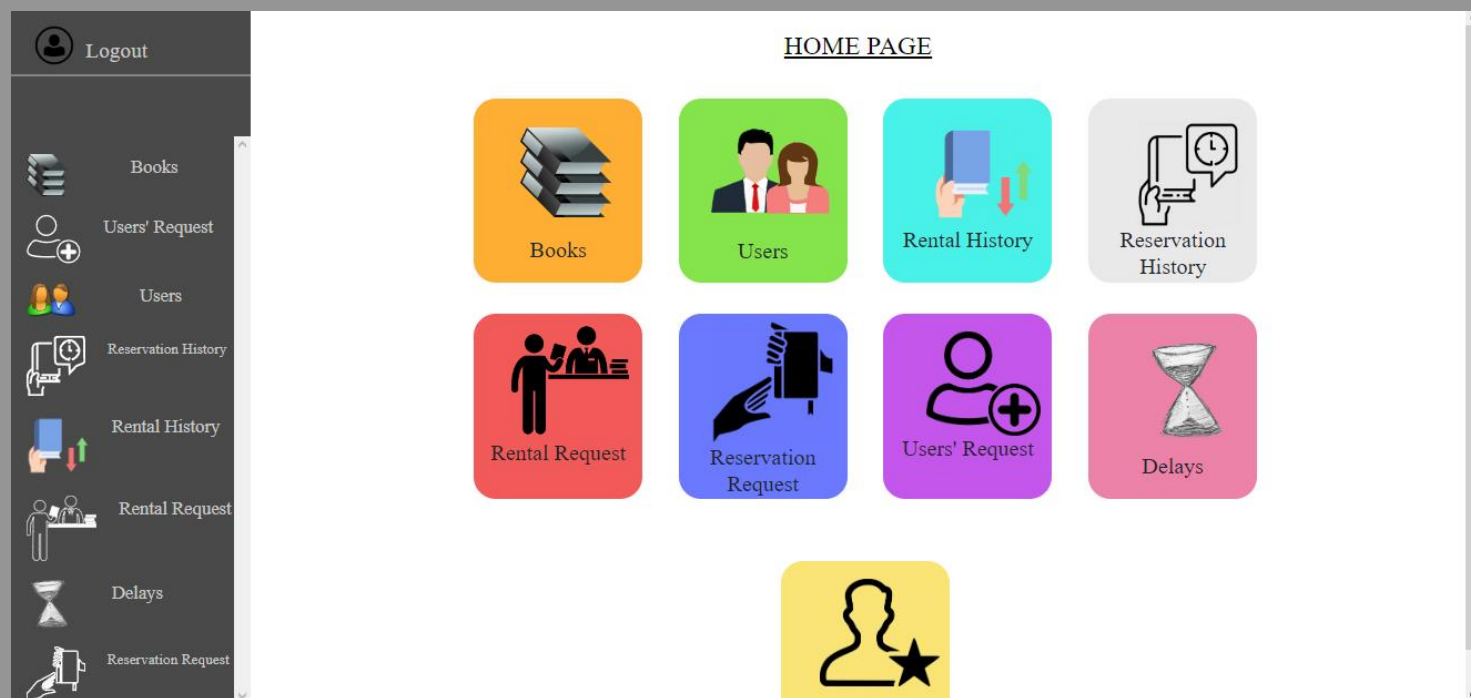
- School Name:
- School Address:
- City:
- Postal Code:
- Telephone:
- Email:
- Principal's Name:

Ο Διαχειριστής συμπληρώνει τα στοιχεία της σχολικής μονάδας που θέλει να προσθέσει στο δίκτυο και στη συνέχεια καταχωρεί την εγγραφή στη βάση με το πάτημα του κουμπιού "Add this school".



□ Περιήγηση στην εφαρμογή

› Αρχική σελίδα Χειριστή



Ο χειριστής της σχολικής βιβλιοθήκης έχει τη δυνατότητα μέσω της εφαρμογής:

- Με κλικ στο “Books” → να επιβλέπει τη λίστα των βιβλίων στο σύστημα του σχολείου, να προσθέτει νέα βιβλία ή να επεξεργάζεται ήδη υπάρχοντα
- Με κλικ στο “Users” → να απενεργοποιεί τους λογαριασμούς ήδη εγγεγραμμένων χρηστών.
- Με κλικ στο “Rental History” → να έχει την εποπτεία των δανεισμών
- Με κλικ στο “Reservation History” → να έχει την εποπτεία των κρατήσεων
- Με κλικ στο “Rental Request” → να εγκρίνει τις αιτήσεις δανεισμού που υποβάλλουν οι χρήστες



- Με κλικ στο “Users’ Request” → να εγκρίνει τις αιτήσεις εγγραφής νέων χρηστών
- Με κλικ στο “Delays” → να αναζητά χρήστες που έχουν καθυστερήσει την επιστροφή ενός βιβλίου
- Με κλικ στο “Ratings” → να βλέπει τον μέσο όρο αξιολογήσεων ανά δανειζόμενο και ανά κατηγορία
- Με κλικ στο “Reservation Request” → να εγκρίνει τις αιτήσεις κράτησης που υποβάλλουν οι χρήστες

☐ **Επισημάνσεις**

Εάν ύστερα από κάποιο αίτημα στην εφαρμογή δε γίνεται επαναφόρτιση ή ανακατεύθυνση σε άλλη σελίδα το αίτημα δεν πραγματοποιείται επιτυχώς.

4. Ο σύνδεσμος για το git repo της εφαρμογής μας

<https://github.com/ApostolosAK/ProjectDB>




5. Οδηγίες εγκατάστασης της εφαρμογής

Για το node.js:

Από τον σύνδεσμο <https://nodejs.org/en/download> επιλέγοντας τον κατάλληλο installer για τον υπολογιστή σας κατεβάζετε το αντίστοιχο αρχείο setup της node.js . Ακολουθήστε τα βήματα προκειμένου να εγκαταστήσετε την node.js στον υπολογιστή σας.

Latest LTS Version: 18.16.0 (includes npm 9.5.1)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Current Latest Features	
 Windows Installer node-v18.16.0-x64.msi	 macOS Installer node-v18.16.0.pkg	 Source Code node-v18.16.0.tar.gz

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit / ARM64	
macOS Binary (.tar.gz)	64-bit	ARM64
Linux Binaries (x64)	64-bit	
Linux Binaries (ARM)	ARMv7	ARMv8
Source Code	node-v18.16.0.tar.gz	

Στην συνέχεια κατεβάστε τα αρχεία τα οποία βρίσκονται στο git repo που βρίσκεται στον παραπάνω σύνδεσμο (**ενότητα 4** της αναφοράς)

Αυτό μπορείτε να το κάνετε πατώντας το πράσινο κουμπί με την αναγραφή “Code” και στην συνέχεια επιλέγοντας το “Download Zip” κατεβάστε τα αρχεία σε μορφή zip. Αφού κατεβάσετε τα αρχεία και τα κάνετε unzip, κάντε unzip τους φακέλους **views.rar** και **node_modules.rar** μέσα στον φάκελο που βρίσκονται όλα τα υπόλοιπα αρχεία που κατεβάσατε.

Μεταφέρετε το αρχείο [DigiCertGlobalRootCA.crt.pem](#) σε ένα directory του οποίου το path περιέχει αποκλειστικά λατινικούς χαρακτήρες και στην συνέχεια

ανοίγοντας το αρχείο server.js σε έναν IDE Editor της επιλογής σας κάντε την εξής αλλαγή:

Στην σειρά 38 του κώδικα αντικαταστήστε το επιλεγμένο κομμάτι που φαίνεται στην παρακάτω εικόνα με το path με τους λατινικούς χαρακτήρες που οδηγεί στον φάκελο που αποθηκεύσατε το [DigiCertGlobalRootCA.crt.pem](#).

```
28
29
30 // Create a MySQL connection
31 var connection = mysql.createConnection({
32   host: 'booksdb.mysql.database.azure.com',
33   user: 'GOAT1',
34   password: 'QWqw1!ERer2@',
35   database: 'booklibrary',
36   port: 3306,
37   ssl: {
38     ca: fs.readFileSync('C:/Users/Giannis Boufidis/Desktop/here/DigiCertGlobalRootCA.crt.pem')
39   }
40 });
41
42 // Connect to the database
43 connection.connect(function(err) {
44   if (err) {
45     console.error('Error connecting to the database:', err);
46     return;
47   }
48 });
```

Προσοχή!!! : Τα directories του path πρέπει να χωρίζονται με '/' και όχι με '\' .

Ανοίγοντας το command window σας (πληκτρολογήστε cmd στην μπάρα αναζήτησης του λογισμικού σας) εκτελέστε τις εντολές που φαίνονται στην παρακάτω εικόνα και περιμένετε για την εγκατάσταση των πακέτων σας

```
Γραμμή εντολών
Microsoft Windows [Version 10.0.19044.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Giannis Boufidis>node -v
v18.16.0

C:\Users\Giannis Boufidis>npm -v
9.5.1

C:\Users\Giannis Boufidis>npm install bcryptjs cookie-parser dotenv ejs express jsonwebtoken mysql2 nodemon
```

Στην συνέχεια ανοίγοντας τον φάκελο που βρίσκονται τα unzipped αρχεία που κατεβάσατε στην αρχή στο command window τρέξτε τις παρακάτω εντολές:

```
Γραμμή εντολών - node server.js
Microsoft Windows [Version 10.0.19044.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Giannis Boufidis>D:
D:\>cd D:\Γιάννης\6ο εξαμηνο\Ροή Λ\Βάσεις Δεδομένων\server\ProjectDB-main
D:\Γιάννης\6ο εξαμηνο\Ροή Λ\Βάσεις Δεδομένων\server\ProjectDB-main>node server.js
Server running on http://localhost:${port}
Connected to the database successfully!
```

Είστε έτοιμοι να χρησιμοποιήσετε την εφαρμογή. Το μόνο που έχετε να κάνετε είναι να συνδεθείτε στον localhost μέσω του browser της επιλογής σας πληκτρολογώντας <http://localhost:3000> στην μπάρα αναζήτησης.