ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

LibraryDB

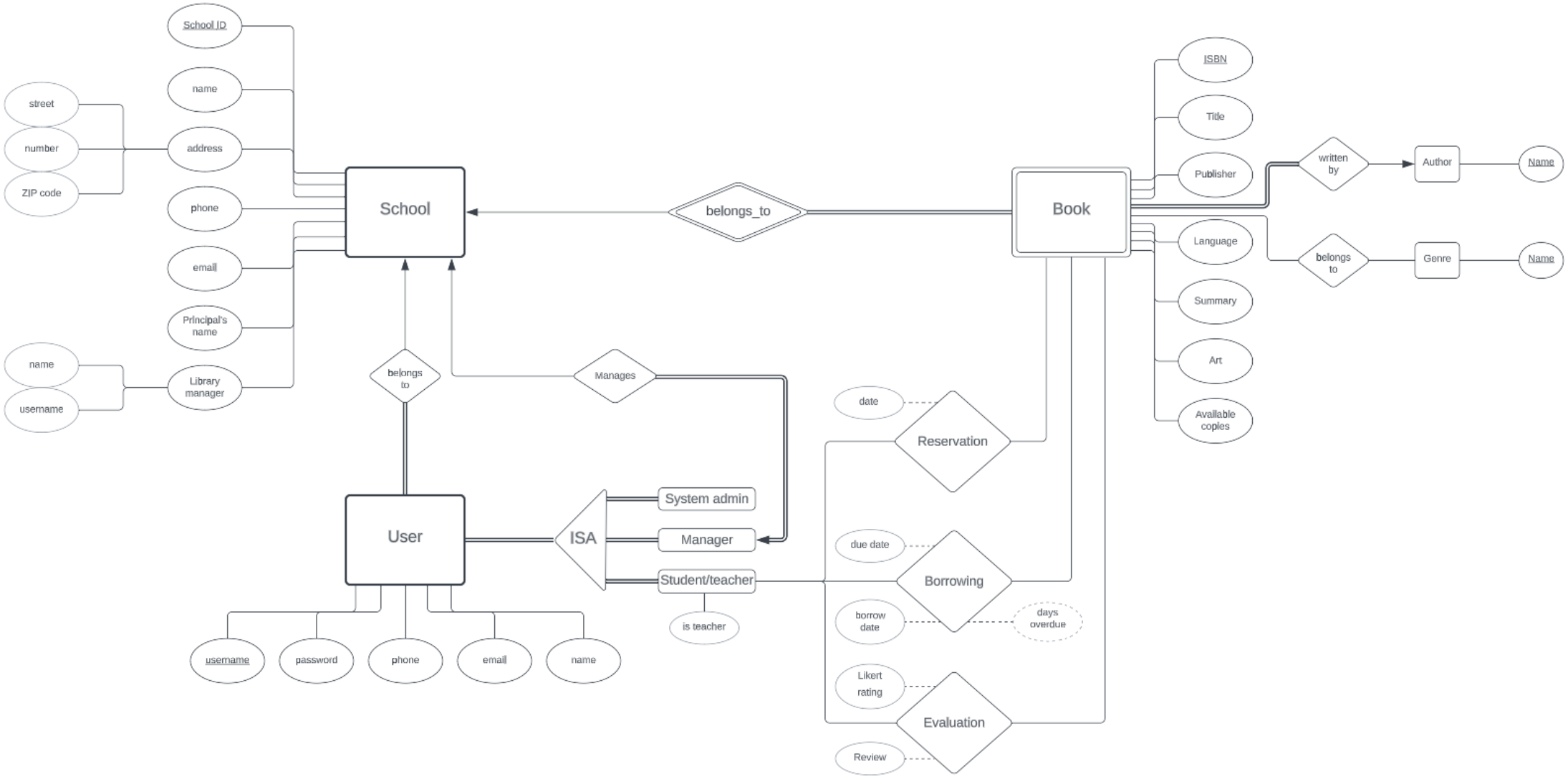
Περίληψη

Η παρούσα εργαστηριακή αναφορά αφορά την εξαμηνιαία εργασία του μαθήματος Βάσεων Δεδομένων, όπου ζητήθηκε η σχεδίαση και υλοποίηση συστήματος βάσης δεδομένων, καθώς και διεπαφής χρήστη, για την εξυπηρέτηση των πληροφοριακών αναγκών των σχολικών βιβλιοθηκών στα δημόσια σχολεία.

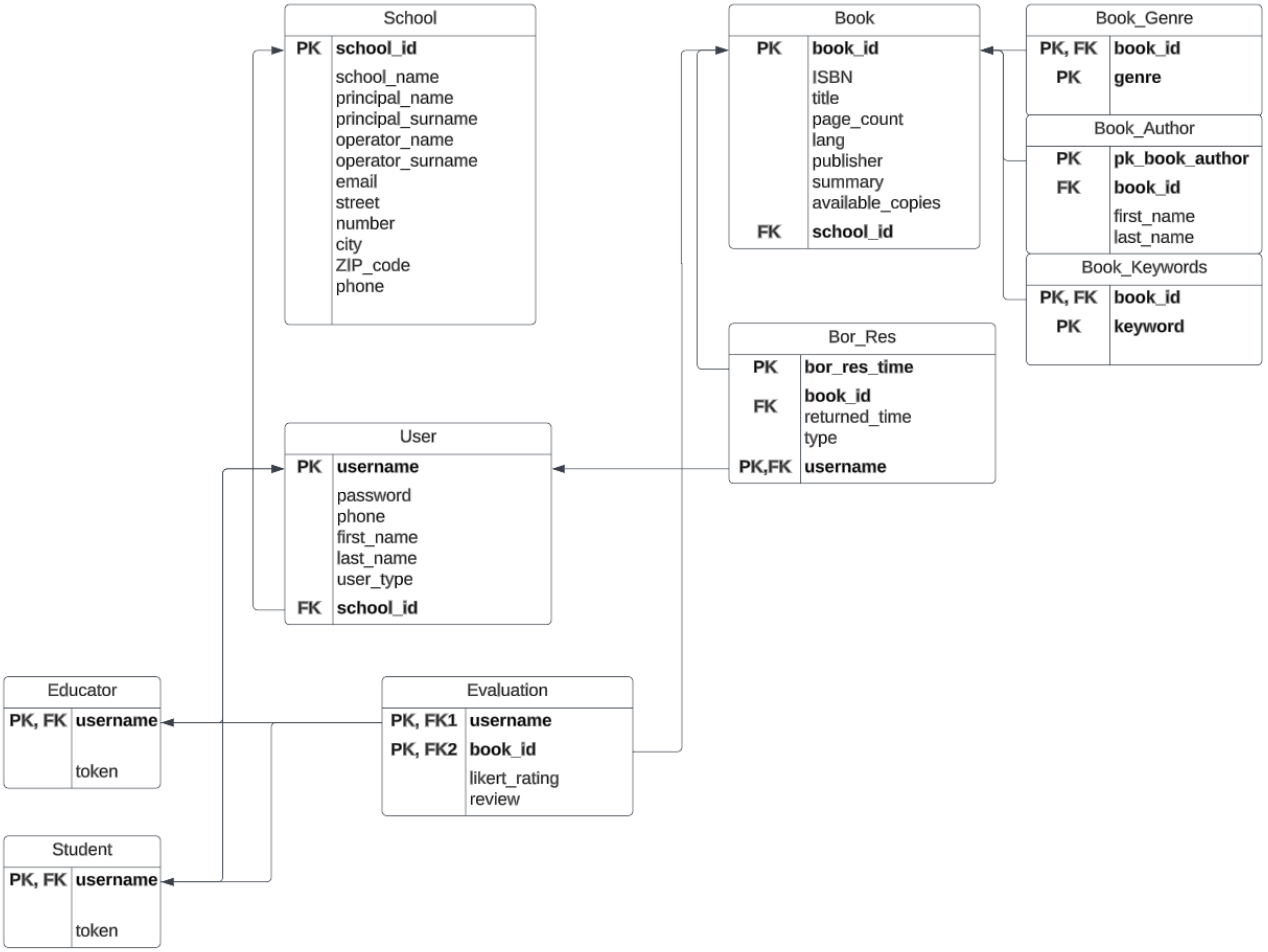
ΟΝΟΜΑΤΑ: Αικατερίνη Δουβή, Ευγένιος Καρδάσης
ΟΜΑΔΑ: 57
Α.Μ.: el20132, el20036
email: cathdouvi@gmail.com, edkardasis@gmail.com

ER Diagram

Το ER διάγραμμα (σε Chen notation) δείχνει την αρχική ιδέα για τη δομή δεδομένων μας, και πώς αυτή εξελίχθηκε στη βάση δεδομένων που έχουμε τώρα.



Relational Schema



Το Schema είναι μια αφαιρετική αλλά ακριβής αναπαράσταση της δομής της βάσης μας.
Οι βασικοί πίνακες είναι το Σχολείο, ο Χρήστης και το Βιβλίο. Ως βιβλίο εδώ νοείται ένα ISBN σε μία βιβλιοθήκη, δηλαδή δύο αντίτυπα ενός βιβλίου σε ένα σχολείο είναι το ίδιο βιβλίο για εμάς (εξού και η μεταβλητή available_copies), ενώ δύο αντίτυπα σε διαφορετικά σχολεία, διαφορετικά. Το ζεύγος (book_id, ISBN) είναι μοναδικό. Αυτή η επιλογή ωφελεί στο ότι δεν καταλαμβάνει το χώρο που θα καταλάμβανε μία βάση που αποθηκεύει κάθε αντίτυπο, και επιτρέπει στο κάθε Χειριστή (Operator) να καθορίζει τα στοιχεία των βιβλίων (αν ο πίνακας είναι με unique ISBN, ποιός αποφασίζει λ.χ. σε ποιές κατηγορίες ανήκει το βιβλίο;).
Βοηθητικοί στην αποθήκευση των multivalued στοιχείων του βιβλίου είναι οι πίνακες Book_Genre, Book_Author, Book_Keywords.

Για τους χρήστες, primary key λαμβάνεται το username (= email). Χωρίζονται σε Μαθητές (Student), Εκπαιδευτικούς (Educator), Χειριστές (Operators) και Διαχειριστή (Administrator). Δεν εμφανίστηκε ανάγκη για ξεχωριστούς πίνακες για τους τελευταίους 2.
Οι σχέσεις των κρατήσεων και δανεισμών (Borrowing, Reservation) αποθηκεύονται σε ενιαίο πίνακα, με primary (κατ' επέκταση clustering) key το timestamp, ώστε να αποθηκεύονται σε χρονολογική σειρά. Τα όρια δανεισμού ανά εβδομάδα τηρούνται ορίζοντας τα token των χρηστών σε 1 και 2 αντίστοιχα. Token καταναλώνονται σε κάθε κράτηση και δανεισμό (εκτός όταν ο δανεισμός είναι από κράτηση. Κράτηση για βιβλίο που δεν είναι διαθέσιμο τίθεται σε αναμονή (Pending_Res), τα διαθέσιμα αντίτυπα του βιβλίου μειώνονται σε αρνητικό αριθμό,

και μόλις υπάρξει διαθέσιμο αντίτυπο, μετατρέπεται σε κράτηση (Reservation) κατά σειρά προτεραιότητας.

Υλοποίηση βάσης

Χρησιμοποιήθηκε η MariaDB και ως client το CLI της και το VSCode με τα κατάλληλα extensions.
Αρχικά έγινε η υλοποίηση των πινάκων, ακριβώς όπως περιγράφει το (τελικό) Schema, και επιπλέον των κατάλληλων constraints.

```
create table if not exists School (
    school_id int unsigned not null,
    school_name varchar(64) not null,
    principal_name varchar(64) not null,
    principal_surname varchar(64) not null,
    operator_name varchar(64) default null,
    operator_surname varchar(64) default null,
    email varchar(64) not null unique check (email like('%_@_%._%')),
    street varchar(64) not null,
    street_number smallint unsigned not null,
    city varchar(64) not null,
    zip_code mediumint unsigned not null check ( zip_code < 100000 ),
    phone_number bigint unsigned not null check ( phone_number < 10000000000
),
    primary key (school_id)
) engine = InnoDB;

create table if not exists Book (
    book_id int unsigned not null unique auto_increment,
    school_id int unsigned not null,
    ISBN char(13) not null check (ISBN rlike('[0-9]{13}')),
    title varchar(64) not null,
    page_count int unsigned not null,
    lang varchar(64) not null,
    publisher varchar(64) not null,
    summary varchar(2000) not null,
    available_copies int not null,
    primary key (school_id, ISBN),
    constraint fk_book_school_id foreign key (school_id) references School
(school_id) on delete restrict on update cascade
) engine = InnoDB;
.
.
.
```

Ευρετήρια, στα περισσότερα σημεία όπου υπάρχουν foreign keys που δεν είναι primary, για τα queries που εξηγούνται κάτω:

```
-- when searching for books that belong to a school
create index fk_book_school_id_idx on Book (school_id);
-- when searching for a book by ISBN
create index fk_book_ISBN_idx on Book (ISBN);
--  when searching for the keywords, genres, authors of a book
create index fk_keyword_book_id_idx on Keywords (book_id);
create index fk_genre_book_id_idx on Book_Genre (book_id);
create index fk_author_book_id_idx on Book_Author (book_id);
-- when searching for the users of a school
create index fk_user_school_id_idx on Users (school_id);
-- When searching for the reservations of a user or book respectively
create index fk_borres_username_idx on Bor_Res (username);
create index fk_borres_book_id_idx on Bor_Res (book_id);
```

Μεγάλο κομμάτι της εργασίας μας ήταν η υλοποίηση triggers και events. Φτιάχτηκαν με τέτοιο τρόπο, ώστε η βάση να διατηρεί την ορθότητα των δεδομένων της όταν συμβαίνουν τα προβλεπόμενα queries (εισαγωγές, επικαιροποιήσεις, διαγραφές), διατηρώντας τους σωστούς αριθμούς token, available_copies κλπ, και απαγορεύοντας στο χρήστη να κάνει κρατήσεις και άλλες αλλαγές, όταν δεν πρέπει αυτό να επιτραπεί.

```
delimiter $
create trigger new_bor_res
before insert on Bor_Res
for each row
begin
    if (select count(1) from Users where username = new.username
        and school_id <> (select school_id from Book where book_id =
new.book_id)) then
        signal sqlstate '45000'
        set message_text = 'You picked the wrong school, fool';
    end if;
    -- has borrowing overdue
    if (select count(1) from Bor_Res
        where username = new.username
        and returned_time is null
        and timestampdiff(minute, bor_res_time, new.bor_res_time)>20160)
      then
        signal sqlstate '45000'
        set message_text = 'Return your old book first, you must';
    end if;
    -- no tokens
    if (select count(1) from
        (select username, token from Student where username = new.username
        union
        select username, token from Educator where username = new.username)
        as tokens
```

```
        where tokens.token = 0) then
        signal sqlstate '45000'
        set message_text = 'No tokens, see ya next week';
    end if;
    -- already has it
    if (select count(1) from Bor_Res
        where Bor_Res.username = new.username and Bor_Res.book_id =
new.book_id and returned_time is null
               and (Bor_Res.type <> 'Reservation' or new.type <> 'Borrowing'
)) then
        signal sqlstate '45000'
        set message_text = 'Double trouble';
    end if;
    if (new.type = 'Borrowing' and new.returned_time is null) then
        if ((select available_copies from Book where Book.book_id =
new.book_id) > 0) then
            update Book set available_copies = available_copies - 1 where
book_id = new.book_id;
        else
            signal sqlstate '45000'
            set message_text = 'No available copies';
        end if;
    else if (new.type = 'Reservation') then
        if ((select available_copies from Book where Book.book_id =
new.book_id) <= 0) then
            set new.type = 'Pending Res';
        end if;
        update Book set available_copies = available_copies - 1 where
book_id = new.book_id;
    end if;
    end if;
    if ((new.type <> 'Borrowing' or
        (select count(1) from Bor_Res where Bor_Res.username = new.username
and Bor_Res.book_id = new.book_id and Bor_Res.type = 'Reservation') = 0)
        and new.returned_time is null) then
        update Student set token = token - 1 where username = new.username;
        update Educator set token = token - 1 where username = new.username;
    end if;
end $
delimiter ;
.
.
.
```

Αυτός ο τρόπος σχεδιασμού επιτρέπει στον κώδικα backend να μπορεί να κάνει κάθε αναγκαία αλλαγή (insert, update, delete) με ένα μόνο query που έχει προβλεφθεί για αυτή, χωρίς να "ανησυχεί" για data integrity.

Population

Αυτοματοποιήσαμε μερικώς την παραγωγή κώδικα για να παραγάγουμε insert queries που θα κάνουν populate τη βάση μας με dummy data. Ορισμένες αλλαγές έγιναν με το χέρι για να είναι πιο πλήρης η δοκιμή της βάσης μας.

Υλοποίηση διεπαφής χρήστη

Για το backend χρησιμοποιήθηκε η python 3.11 με το module Flask, ενώ κρατήσαμε το frontend λιτό (χωρίς JS) μόνο με html και css.
Υλοποιήθηκε login page, αρχικές σελίδες για κάθε είδος χρήστη, και επιπλέον σελίδες από όπου μπορούν να γίνουν τα ζητούμενα queries. Γίνεται session management μέσω του Flask-session, όπου αποθηκεύονται όποια δεδομένα πρέπει να υπάρχουν κοινά μεταξύ διαφόρων σελίδων. Αποθηκεύεται προσωρινά το username του χρήστη και γίνεται authentication στις endpoint σελίδες για να μην υπάρχει η δυνατότητα πρόσβασης σε αυτά κάποιου άνευ αδείας, μέσω του URL. Από τη μία σελίδα στην άλλη πάει κανείς με GET και POST requests, ανάλογα αν υπάρχει ανάγκη εξαγωγής πληροφοριών από το frontend.
Δυστυχώς δεν είχαμε το χρόνο να υλοποιήσουμε την λειτουργικότητα της σελίδας πέραν αυτής που αναφέρεται στα ερωτήματα 3.1 - 3.3, αλλά λόγω της φιλοσοφίας της εφαρμογής μας με το αυτοματοποιημένο data integrity, δεν απέχουμε πολύ. Αρκεί να τρέξει κανείς το αντίστοιχο query και επιτελείται πλήρως η αντίστοιχη λειτουργία.

Query 3.1.1

```
select School.school_name, count(*)
from ((Bor_Res join Users on Bor_Res.username = Users.username)
join School on Users.school_id = School.school_id)
where (type = 'Borrowing' and bor_res_time
like("2022-_____"))
group by School.school_id;
```

Query 3.1.2

```
select distinct first_name, last_name from Book_Author join
Book_Genre on Book_Author.book_id = Book_Genre.book_id where genre
= 'Fantasy' order by first_name;
select first_name, last_name from (Users join (Bor_Res join
Book_Genre on Bor_Res.book_id = Book_Genre.book_id) on
Users.username = Bor_Res.username) where user_type = 'Educator'
and Book_Genre.genre = 'Fantasy';
```

Query 3.1.3

```
SELECT first_name, last_name, COUNT(*) AS loans
FROM Bor_Res
JOIN Users ON Bor_Res.username = Users.username
WHERE user_type = 'Educator' AND TIMESTAMPDIFF(YEAR,
date_of_birth, CURRENT_DATE()) < 40
GROUP BY Users.username
ORDER BY loans DESC;
```

Query 3.1.4

```
select first_name, last_name
from (select first_name, last_name, count(Bor_Res.bor_res_time) as
loans
from (Bor_Res right join Book_Author on Bor_Res.book_id =
Book_Author.book_id and type = 'Borrowing')
group by first_name, last_name) as subquery where loans = 0;
```

Query 3.1.5

```
select School.operator_name, School.operator_surname, plithos_a
from
(select a.school_id as col1, b.school_id, plithos_a from
(select school_id, plithos_a from
(select Book.school_id, count(school_id) as plithos_a from
Bor_Res join Book on Bor_Res.book_id = Book.book_id
where timestampdiff(day, bor_res_time, current_timestamp) < 365
group by school_id) as a
) as a
join (select school_id, plithos_b from
(select Book.school_id, count(school_id) as plithos_b from
Bor_Res join Book on Bor_Res.book_id = Book.book_id
where timestampdiff(day, bor_res_time, current_timestamp) < 365
group by school_id) as b
) as b
on a.plithos_a = b.plithos_b
where plithos_a >= 20 and a.school_id != b.school_id)
as pinakara
join School on pinakara.col1 = School.school_id;
```

Query 3.1.7

```
select name, surname, count(*) as book_no from
(
    SELECT DISTINCT ISBN, Book_Author.first_name AS name,
Book_Author.last_name AS surname
    FROM Book_Author
    JOIN Book ON Book_Author.book_id = Book.book_id
  ) AS subsubquery
  GROUP BY name, surname
having book_no <=
((SELECT MAX(plithos) FROM
(
  SELECT name, surname, COUNT(*) AS plithos
  FROM
  (
    SELECT DISTINCT ISBN, Book_Author.first_name AS name,
Book_Author.last_name AS surname
    FROM Book_Author
```

```
    JOIN Book ON Book_Author.book_id = Book.book_id
  ) AS subsubquery
  GROUP BY name, surname
) AS subquery)
-5);
```

Query 3.2.1

```
select title, first_name, last_name, available_copies
from (Book left join Book_Author on Book.book_id =
Book_Author.book_id) left join Book_Genre on Book_Genre.book_id =
Book.book_id
where school_id = 551003 and title like('%') and
CONCAT(first_name, ' ', last_name) like('%%') and genre like('%%')
group by Book.book_id order by available_copies asc;
```

Query 3.2.2

```
;
select Users.username, first_name, last_name, timestampdiff(day,
bor_res_time, current_timestamp)
from Users
join Bor_Res on Users.username = Bor_Res.username
where (returned_time is null
and timestampdiff(hour, bor_res_time, current_timestamp) > (14)*24
and Users.first_name like('%%')
and Users.last_name like('%%')
and Users.school_id = 920305
);
```

Query 3.2.3

```
;
select Users.username, first_name, last_name,
round(avg(Likert_rating), 1) from
(Users join Evaluation on Users.username = Evaluation.username) as
ev_usr
join Book_Genre on ev_usr.book_id = Book_Genre.book_id
```

```
where Users.username like('%%') and Book_Genre.genre like('%%')
group by Users.username, first_name, last_name;

SELECT Users.username, first_name, last_name,
ROUND(AVG(Likert_rating), 1)
FROM Users
JOIN Evaluation ON Users.username = Evaluation.username
JOIN Book_Genre ON Evaluation.book_id = Book_Genre.book_id
WHERE Users.username LIKE '%%' AND Book_Genre.genre LIKE '%%'
AND Users.school_id = 920305
GROUP BY Users.username, first_name, last_name;
```

Query 3.3.1

```
;
SELECT bookid, title, authors, summary, pc, ac, genres,
ROUND(average_rating, 1) FROM
(
SELECT books.book_id as bookid, books.title as title,
books.author_names as authors, books.summary as summary,
       books.page_count as pc, books.available_copies as ac,
genres.genre_names as genres,
       AVG(Evaluation.Likert_rating) AS average_rating,
books.school_id as sch
FROM (
  SELECT Book.book_id, Book.title, Book.summary, Book.page_count,
Book.available_copies, Book.school_id,
         GROUP_CONCAT(CONCAT(Book_Author.first_name, ' ',
Book_Author.last_name) SEPARATOR ', ') AS author_names
  FROM Book
  JOIN Book_Author ON Book.book_id = Book_Author.book_id
  GROUP BY Book.book_id, Book.title, Book.summary,
Book.page_count, Book.available_copies
) AS books
JOIN (
  SELECT Book.book_id,
         GROUP_CONCAT(DISTINCT Book_Genre.genre SEPARATOR ', ') AS
genre_names
  FROM Book
```

```
  JOIN Book_Genre ON Book.book_id = Book_Genre.book_id
  GROUP BY Book.book_id
) AS genres ON books.book_id = genres.book_id
LEFT JOIN Evaluation ON books.book_id = Evaluation.book_id
GROUP BY books.book_id, books.title, books.author_names,
books.summary, books.page_count, books.available_copies,
genres.genre_names
ORDER BY books.book_id
) as books
WHERE books.title LIKE '%%' AND books.authors LIKE '%%' AND
books.genres LIKE '%%' AND books.sch = {920305};
```

Query 3.3.2

```
;
select Book.book_id, Book.title, substring(bor_res_time, 1, 16),
substring(returned_time, 1, 16) from Book join Bor_Res on
Book.book_id = Bor_Res.book_id where username =
"fred.weasley@gmail.com" and type = 'Borrowing';
select Book.book_id, Book.title, substring(bor_res_time, 1, 16)
from Book join Bor_Res on Book.book_id = Bor_Res.book_id where
username = {"fred.weasley@gmail.com"} and {type = 'Reservation'};
```

User Manual

Η εφαρμογή της βάσης δίνει τη δυνατότητα στον χρήστη, ανάλογα με την θέση του να δει την κατάλληλη σελίδα για αυτόν.

Για αρχή, στο landing page του ζητείται το username (email) και ο κωδικός του λογαριασμού του στο site. Σε περίπτωση που δεν έχει λογαριασμό, μπορεί να δει τι πρέπει να κάνει για να αποκτήσει.
Αφού εισάγει τις ζητούμενες πληροφορίες, οδηγείται στο κατάλληλο home page.

Όλοι οι χρήστες μπορούν να δουν στην αρχική τους σελίδα κάποιες καρτέλες, οι οποίες αντιστοιχούν στα queries που ζητούνται για τον καθένα, και έχουν την επιλογή να δουν τα αποτελέσματα τους με το πάτημα ενός κουμπιού.
Σε πολλές από αυτές τις καρτέλες υπάρχει δυνατότητα φιλτραρίσματος με την χρήση των textboxes και dropdowns που παρέχονται.

Σε μαθητές και δασκάλους υπάρχει η δυνατότητα να κάνουν ηλεκτρονική κράτηση ενός βιβλίου. Στην αρχική τους σελίδα μπορούν να δουν αν τους επιτρέπεται να κάνουν κράτηση (δύο κρατήσεις ανά εβδομάδα οι μαθητές και μία οι δάσκαλοι) και εφόσον γίνεται, μπορούν να δουν τα βιβλία που παρέχει το σχολείο τους και να επιλέξουν αυτό που θέλουν να δανειστούν. Αν η κράτηση είναι επιτυχής, θα εμφανιστεί το κατάλληλο μήνυμα. Σε περίπτωση που η κράτηση αποτύχει θα βρεθούν στο landing page μαζί με ένα μήνυμα που θα εξηγεί τι πήγε λάθος.

Installation Guide

το repository
https://github.com/kdouvi/librarydb

step 1

Αρχικά πρέπει να γίνει clone τοπικά το git repo της εφαρμογής. Αυτό γίνεται μέσω της εφαρμογής GitHub Desktop ή με την εντολή git clone https://github.com/kdouvi/librarydb μέσω κάποιου terminal στο επιθυμητό directory.

step 2

Για την εγκατάσταση της βάσης χρειάζεται ένα DBMS και ένας sql server. Μόλις δημιουργηθεί σύνδεση με τον server πρέπει να τρέξουμε τα αρχεία Schema.sql και Population.sql που βρίσκονται στο φάκελο dbsetup (αναγκαστικά με αυτή τη σειρά).

step 3

Αρχικά εγκαθιστούμε τις απαραίτητες βιβλιοθήκες που αναφέρονται στο αρχείο requirements.txt με την εντολή pip install -r requirements.txt. Για να τρέξουμε την εφαρμογή εκτελούμε την εντολή python app.py και ανοίγουμε έναν browser στη διεύθυνση localhost:3000. Μετά από αυτό θα έπρεπε να εμφανίζεται η αρχική σελίδα της εφαρμογής.