Τεχνική αναφορά
Πληροφοριακό σύστημα Δικτύου Σχολικών Βιβλιοθηκών

https://github.com/andreasstamos/library-management-system

Ανδρέας Στάμος - Κωνσταντίνος Πίκουλας

Περιεχόμενα

Περιεχόμενα 1

1 Σχεδίαση εφαρμογής 1

2 Βάση δεδομένων 2
- 2.1 Διάγραμμα Οντοτήτων-Συσχετίσεων (Entity-Relationship Diagram) . . . 2
- 2.2 Σχεσιακό σχήμα (Relational Schema) . . . 2
- 2.3 Ευρετήρια . . . 3
- 2.4 Mock data . . . 4

3 Αυθεντικοποίηση (Authentication) - Εξουσιοδότηση (Authorization) 4
- 3.1 Αυθεντικοποίηση (Authentication) . . . 4
- 3.2 Εξουσιοδότηση (Authorization) . . . 5

Α΄ Βιβλιοθήκες και Frameworks που χρησιμοποιήθηκαν 5

Β΄ DDL Script 5

Γ΄ DML Script 8

Δ΄ Οδηγίες Εγκατάστασης 9
- Δ΄.1 Βάση Δεδομένων . . . 9
- Δ΄.2 Backend (API) . . . 9
- Δ΄.3 Web Application . . . 10

1 Σχεδίαση εφαρμογής

(α΄) PostgreSQL (β΄) Flask (γ΄) React (δ΄) Material UI

Σχήμα 1: Web Development Stack

Η εφαρμογή σχεδιάστηκε σε stack: PostgreSQL, Flask, React, Material UI (υλοποίηση του Google Material Design σε React Components). (βλ και σχήμα 1).

Με χρήση του Flask αναπτύχθηκε **RESTful API** (δεν διατηρείται πουθενά state για τα sessions), ενώ με χρήση React αναπτύχθηκε Web εφαρμογή που τρέχει αποκλειστικά στον client (σερβίρονται αποκλειστικά τα στατικά αρχεία JavaScript της εφαρμογής) και που με HTTP Requests επικοινωνεί με το API.

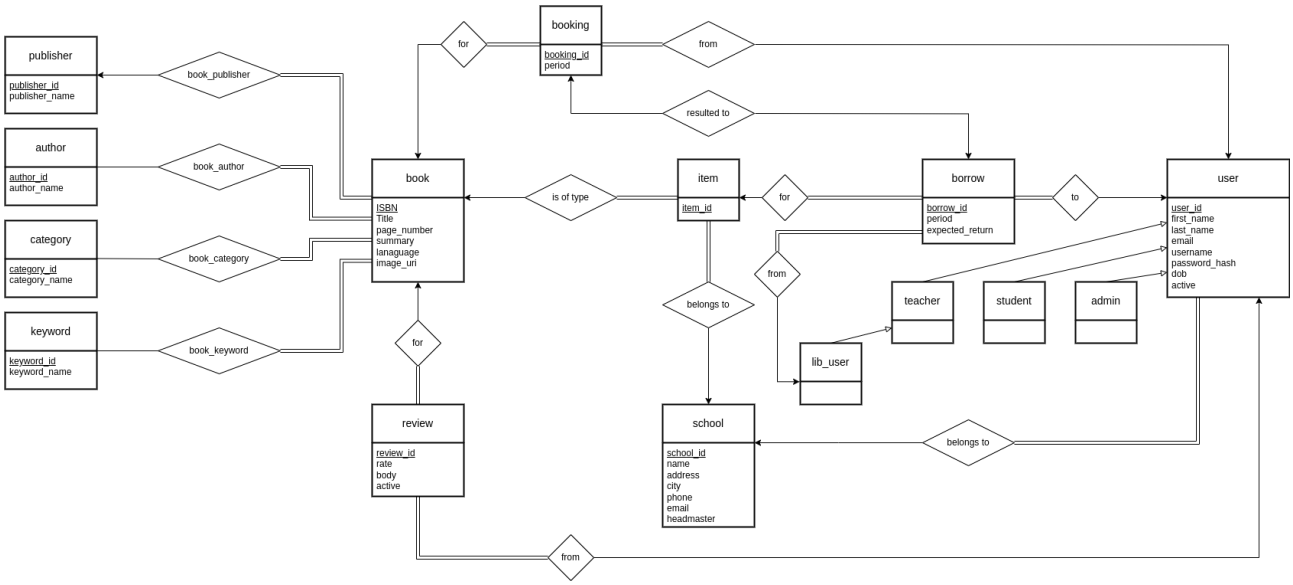
Στο API τα requests έχουν την μορφή JSON, γίνονται με χρήση HTTP POST και γίνονται validate ότι έχουν την σωστή μορφή με χρήση ενός **JSONSchema** για κάθε διαφορετικό endpoint.

Σε περιβάλλον παραγωγής η επικοινωνία με το API γίνεται με HTTPS.

2 Βάση δεδομένων

2.1 Διάγραμμα Οντοτήτων-Συσχετίσεων (Entity-Relationship Diagram)

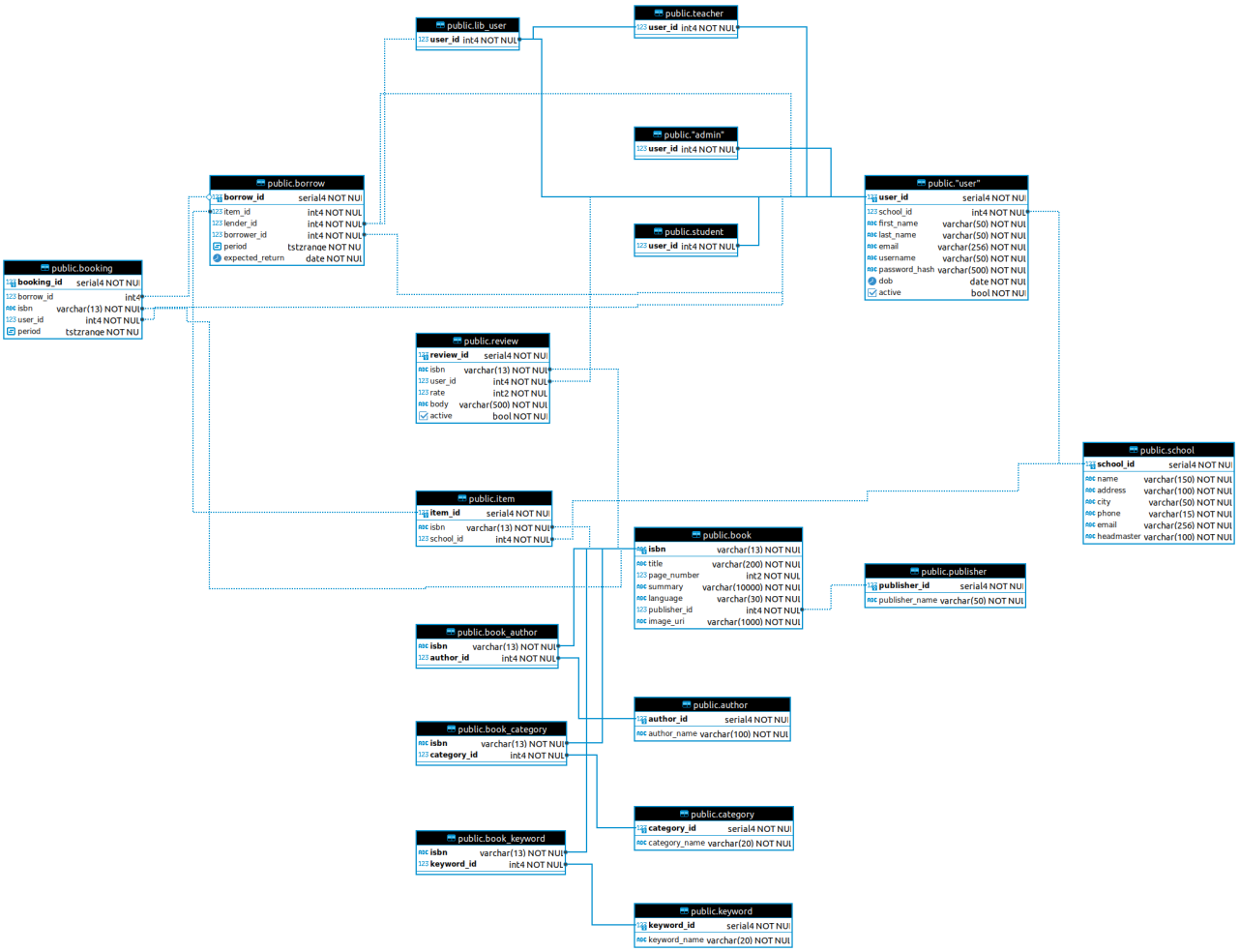
Αρχικά σχεδιάστηκε το σχήμα σε διάγραμμα οντοτήτων-συσχετίσεων (Entity-Relationship Diagram) όπως φαίνεται στην εικόνα 2.



Σχήμα 2: Διάγραμμα Οντοτήτων-Συσχετίσεων (Entity-Relationship Diagram)

2.2 Σχεσιακό σχήμα (Relational Schema)

Στην συνέχεια σχεδιάστηκε με βάση αυτό το relational σχήμα της βάσης δεδομένων σχεδιάστηκε σύμφωνα με την **κανονική μορφή Boyce-Codd (BCNF)** (κάθε σύνολο attributes προκύπτει μόνο από υπερκλειδί), προκειμένου να μην υπάρχει πλεονασμός δεδομένων. Το DDL script για την κατασκευή της βάσης υπάρχει στο παράρτημα B′ και το προκύπτον σχεσιακό διάγραμμα (relational diagram) φαίνεται στο σχήμα 3.



Σχήμα 3: Σχεσιακό Διάγραμμα (Relational Diagram)

Προκειμένου να διασφαλίσουμε την σωστή χρονολογική σειρά των δανεισμών (/των κρατήσεων), δηλαδή ότι μια δεδομένη χρονική στιγμή κάθε αντίτυπο είναι δανεισμένο σε έναν μόνο χρήστη (/κάθε χρήστης έχει μόνο μια κράτηση για ένα βιβλίο) χρησιμοποιήσαμε τον τύπο δεδομένου TSTZRANGE της PostgreSQL και με χρήση EXCLUDE επιβάλαμε τον περιορισμό ότι δύο tuples που αναφέρονται στο ίδιο αντίτυπο (/αντίστοιχα στο ίδιο βιβλίο και χρήστη για κρατήσεις) δεν θα πρέπει να έχουν χρονική επικάλυψη. Η PostgreSQL με χρήση GIST ευρετηρίου μπορεί να επιβάλλει αποδοτικά τον περιορισμό αυτό.

Η κατανομή ρόλων στους χρήστες αποτυπώθηκε στην βάση με χρήση Specialization στην ER σχεδίαση. Στο relational σχήμα αυτό μεταφράστηκε σε ξεχωριστούς πίνακες για κάθε ρόλο που απλά περιέχουν τα user_id των χρηστών που κατέχουν τον αντίστοιχο ρόλο.

2.3 Ευρετήρια

Με δεδομένα τις ερωτήσεις που γίνονται στην βάση, δημιουργήσαμε και τα κατάλληλα ευρετήρια. Αξίζει να επισημανθεί ότι σε μερικά σημεία η αναζήτηση γίνεται σε ελεύθερο κείμενο, με χρήση των trigrams της PostgreSQL που υπολογίζουν μια μετρική similiarity μεταξύ συμβολοσειρών. Χρησιμοποιήθηκαν τα ειδικά ευρετήρια τύπου GIST για τους τελεστές των trigrams για τα συγκεκριμένα πεδία.

Επιπλέον αναφέρεται πως όπου επιβάλλεται στο σχήμα περιορισμός UNIQUE ούτως ή αλλώς δημιουργείται ευρετήριο για τα πεδία αυτά, οπότε δεν απαιτείται να δημιουργηθεί νέο αν αυτό απαιτηθεί για λόγους αναζήτησης.

Επισημαίνεται πως τα ευρετήρια έχουν σχεδιαστεί προσεγγιστικά με στόχο την βελτιστοποίηση της απόδοσης. Κατά την διάρκεια της ζωής της βάσης δεδομένων και σε βάθος χρόνου, θα πρέπει να μελετηθεί η συχνότητα εμφάνισης των διάφορων ερωτήματων και το κόστος υπολογισμού τους, προκειμένου να επαναξιολογηθούν τα απαιτούμενα ευρετήρια. Ενδέχεται σε μερικά σημεία, το κόστος συνεχούς ενημέρωσης του ευρετηρίου ενός πίνακα που ενημερώνεται συνεχώς να αποδειχθεί ότι δεν αξίζει για ένα ερώτημα που τελικά ερωτάται σπανίως.

Αναλυτικότερα, δημιουργήθηκαν ευρετήρια τύπου B+ tree στα:

- publisher_id του table book (χρήση στο φιλτράρισμα βιβλίων)
- email του table ”user” (χρήση στο “Ξέχασα τον κωδικό μου”)

- active του table "user" (ο χειριστής βιβλιοθήκης βλέπει τους inactive χρήστες, οι inactive χρήστες αναμένονται λίγοι)
- school_id του table "user" (γίνεται φιλτράρισμα σε πολλά σημεία να βρίσκονται οι χρήστες ενός συγκεκριμένου σχολείου)
- isbn του table item (στην σελίδα ενός βιβλίου εμφανίζονται τα αντίτυπα ενός βιβλίου)
- ischool_id του table item (σε πολλά ερωτήματα γίνεται ελέγχος ορθότητας-ασφάλειας πως ο χειριστής ενός σχολείου έχει δικαιώματα για πράξεις σε ένα συγκεκριμένο αντίτυπο)
- active του table review (ο χειριστής βιβλιοθήκης βλέπει τις inactive αξιολογήσεις, οι inactive αξιολογήσεις ανεμένονται λίγες)
- isbn του table review (στη σελίδα ενός βιβλίου εμφανίζονται οι αξιολογήσεις ενός βιβλίου, στην σελίδα όλων των βιβλίων εμφανίζονται μέσοι όροι αξιολογήσεων για όλα τα βιβλία)
- item_id του table borrow (στην φόρμα Δανεισμού/Επιστροφής γίνεται έλεγχος αν ένα αντίτυπο είναι δανεισμένο ή όχι)
- borrower_id του table borrow (ο χρήστης μπορεί να δει τους δανεισμούς του, επίσης ο χειριστής μπορεί να κάνει αναζήτηση δανεισμών ανά χρήστη)
- isbn του table booking (όταν γίνεται δανεισμός γίνεται ελέγχος για κρατήσεις στο ίδιο βίβλιο)
- user_id του table booking (ο χρήστης μπορεί να δει τις κρατήσεις του, επίσης ο χειριστής μπορεί να κάνει αναζήτηση κρατήσεων ανά χρήστη)

Επιπρόσθετα ευρετήρια τύπου B+ tree δημιουργούνται και για όλα τα primary keys καθώς και για όλα τα πεδία όπου υπάρχει περιορισμός UNIQUE. Πολλά από αυτά τα ευρετήρια είναι και χρήσιμα για διάφορα ερωτήματα που γίνονται, όμως αυτό δεν απαιτείται να μελετηθεί εφόσον το UNIQUE ούτως ή αλλώς είναι αναγκαίο να υπάρχει για εξασφάλιση ορθότητας.

Επιπλέον δημιουργήθηκαν ευρετήρια τύπου GIST που εξυπηρετούν τα trigram operators στα:

- author_name του table author (γίνεται αναζήτηση του author_name στο Autocomplete του search bar στα φίλτρα βιβλίων και στην εισαγωγή βιβλίου)
- publisher_name του table publisher (γίνεται αναζήτηση του publisher_name στο Autocomplete του search bar στα φίλτρα βιβλίων και στην εισαγωγή βιβλίου)
- category_name του table category (γίνεται αναζήτηση του category_name στο Autocomplete του search bar στα φίλτρα βιβλίων και στην εισαγωγή βιβλίου)
- keyword_name του table keyword (γίνεται αναζήτηση του keyword_name στο Autocomplete του search bar στα φίλτρα βιβλίων και στην εισαγωγή βιβλίου)
- συνδυασμό first_name, last_name του table "user" (γίνεται φιλτράρισμα βάσει ονοματεπώνυμου σε διάφορες σελίδες του χειριστή βιβλιοθήκης)
- title του table book (γίνεται αναζήτηση του title στην εμφάνιση λίστας βιβλίων, καθώς και στο Autocomplete του search bar στα φίλτρα βιβλίων).

2.4 Mock data

Για τους σκοπούς ελέγχου της λειτουργίας του Συστήματος, κατασκευάστηκαν δεδομένα ελέγχου με χρήση της βιβλιοθήκης Faker της Python. Ενδεικτικά αναφέρεται ότι εισάγονται 5 σχολεία, 500 χρήστες, 1000 βιβλία, 10000 αντίτυπα, 10000 αξιολογήσεις, 5000 δανεισμοί και 1500 κρατήσεις.

3 Αυθεντικοποίηση (Authentication) - Εξουσιοδότηση (Authorization)

3.1 Αυθεντικοποίηση (Authentication)

Η αυθεντικοποίηση των χρηστών γίνεται με χρήση username, password όπου το password γίνεται hash με χρήση *bcrypt* και κατόπιν salting. Το υπολογιστικά βαρύ bcrypt σε συνδυασμό με salting καθιστούν δύσκολη επίθεση bruteforce κατά των hash, ακόμα και αν γινόταν attack τύπου Rainbow Table.

Αφού αυθεντικοποιηθεί ο χρήστης, του χορηγείται ένα authentication token, υλοποιημένο ως JSON Web Token, ορισμένης χρονικής διάρκειας, που περιέχει πληροφορίες για το user_id, το school_id και τον ρόλο του χρήστη.

Το token έχει expiration time 5 ωρών και έπειτα απαιτείται επανασύνδεση. Η παραχάραξη του token αποκλείεται καθώς περιέχει Ψηφιακή Υπογραφή του API. Το authentication token χορηγείται σε κάθε request στο API ως HTTP Authentication Header και από την εγκυρότητά του εξασφαλίζεται ότι ο χρήστης που στέλνει το request είναι αυθεντικοποιημένος.

3.2 Εξουσιοδότηση (Authorization)

Στην βάση δεδομένων έχουν δημιουργηθεί χωριστοί πίνακες για κάθε ρόλο, που περιέχουν τα user_ids των χρηστών που διαθέτουν τον ρόλο αυτό. Όταν γίνει login, αφού γίνεται ελέγχος αυθεντικοποίησης, ελέγχεται αν ο χρήστης ανήκει σε κάποιον πίνακα και του απονέμεται ο αντίστοιχος ρόλος (δηλαδή ο αντίστοιχος ρόλος τοποθετείται ως role στο authentication token).

Σε επίπεδο API, το δικαίωμα ενός χρήστη για μια συγκεκριμένη πράξη ελέγχεται με βάση το role που λαμβάνεται στο authentication token μαζί με κάθε request.

Σε επίπεδο Web εφαρμογής, ελέγχεται σε κάθε route αν ο χρήστης διαθέτει το απαιτούμενο role πριν εμφανιστεί η σχετική σελίδα. Επισημαίνεται πως αυτό δεν προσφέρει πραγματική ασφάλεια, παρά μόνο εμφανίζει στον χρήστη τις σελίδες που μπορεί να έχει πρόσβαση. Αυτό διότι ούτως ή άλλως, ένας κακόβουλος χρήστης μπορεί να κάνει απευθείας requests στο API παρακάμπτοντας το Web App, και επίσης καθώς ο κώδικας της εφαρμογής εκτελείται πλήρως στον client, δηλαδή είναι πλήρως διαθέσιμος στον τελικό χρήστη, οπότε ένας κακόβουλος χρήστης θα μπορούσε απλά να αλλάξει τον κώδικα της εφαρμογής παρακάμπτοντας τους σχετικούς ελέγχους. (ακόμα και αν το κάνει αυτό το API δεν θα του δώσει δεδομένα).

Α΄ Βιβλιοθήκες και Frameworks που χρησιμοποιήθηκαν

1. PostgreSQL
2. Python
3. Βιβλιοθήκες Python:
   (α΄) Flask (και flask_cors, flask_jwt_extended)
   (β΄) Psycopg2
   (γ΄) jsonschema
   (δ΄) bcrypt
   (ε΄) latex (βιβλιοθήκη της python για παραγωγή pdf αρχείου από LaTeX, χρησιμοποιείται για την δημιουργία κάρτας βιβλιοθήκης)
   (στ΄) TeX Live - LaTeX + package “qrcode” (για την δημιουργία κάρτας βιβλιοθήκης)
4. React
5. React Router DOM
6. Βιβλιοθήκες JavaScript:
   (α΄) Axios
   (β΄) lodash
   (γ΄) jwt-decode
   (δ΄) dayjs
   (ε΄) file-saver
7. Material UI (+ Material Icons)

Β΄ DDL Script

Το DDL Script που κατασκευάζει τους πίνακες τις βάσεις (και τα σχετικά ευρετήρια) βρίσκεται στο αποθετήριο GitHub στο path:

`sql/schema.sql`

Επιπλέον ο διαδικτυακός υπερσύνδεσμος για το DDL Script απευθείας στο GitHub αποθετήριο είναι:
https://github.com/andreasstamos/library-management-system/blob/master/sql/schema.sql


```sql
DROP TABLE IF EXISTS booking;
DROP TABLE IF EXISTS borrow;
DROP TABLE IF EXISTS review;
DROP TABLE IF EXISTS item;
DROP TABLE IF EXISTS book_author;
DROP TABLE IF EXISTS book_category;
DROP TABLE IF EXISTS book_keyword;
DROP TABLE IF EXISTS book;
DROP TABLE IF EXISTS category;
DROP TABLE IF EXISTS keyword;
DROP TABLE IF EXISTS author;
DROP TABLE IF EXISTS publisher;
DROP TABLE IF EXISTS lib_user;
DROP TABLE IF EXISTS student;
DROP TABLE IF EXISTS teacher;
DROP TABLE IF EXISTS "admin";
DROP TABLE IF EXISTS "user";
DROP TABLE IF EXISTS school;

CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE EXTENSION IF NOT EXISTS btree_gist;

CREATE TABLE publisher (
        publisher_id SERIAL PRIMARY KEY,
        publisher_name VARCHAR(50) NOT NULL UNIQUE
);

CREATE INDEX index_publisher ON publisher USING GIST (publisher_name gist_trgm_ops);

CREATE TABLE book (
        isbn VARCHAR(13) NOT NULL PRIMARY KEY CHECK (isbn ~ '^[0-9]{13}'),
        title VARCHAR(200) NOT NULL,
        page_number SMALLINT NOT NULL CHECK (page_number > 0),
        summary VARCHAR(10000) NOT NULL,
        language VARCHAR(30) NOT NULL,
        publisher_id INTEGER NOT NULL REFERENCES publisher ON DELETE CASCADE,
        image_uri VARCHAR(1000) NOT NULL
);


CREATE INDEX index_book_title ON book USING GIST (title gist_trgm_ops);
CREATE INDEX index_book_publisher_id ON book (publisher_id);

CREATE TABLE author (
        author_id SERIAL PRIMARY KEY,
        author_name VARCHAR(100) NOT NULL UNIQUE
);

CREATE INDEX index_author ON author USING GIST (author_name gist_trgm_ops);

CREATE TABLE book_author (
        isbn VARCHAR(13) NOT NULL REFERENCES book ON UPDATE CASCADE ON DELETE CASCADE,
        author_id INTEGER NOT NULL REFERENCES author ON DELETE CASCADE,
        UNIQUE(isbn, author_id) --builts index
);

CREATE TABLE category (
        category_id SERIAL PRIMARY KEY,
        category_name VARCHAR(20) NOT NULL UNIQUE
);

```



```
62  CREATE INDEX index_category ON category USING GIST (category_name gist_trgm_ops);
63
64  CREATE TABLE book_category (
65          isbn VARCHAR(13) NOT NULL REFERENCES book ON UPDATE CASCADE ON DELETE CASCADE,
66          category_id INTEGER NOT NULL REFERENCES category ON DELETE CASCADE,
67          UNIQUE(isbn, category_id)
68  );
69
70  CREATE TABLE keyword (
71          keyword_id SERIAL PRIMARY KEY,
72          keyword_name VARCHAR(20) NOT NULL UNIQUE
73  );
74
75  CREATE INDEX index_keyword ON keyword USING GIST (keyword_name gist_trgm_ops);
76
77  CREATE TABLE book_keyword (
78          isbn VARCHAR(13) NOT NULL REFERENCES book ON UPDATE CASCADE ON DELETE CASCADE,
79          keyword_id INTEGER NOT NULL REFERENCES keyword ON DELETE CASCADE,
80          UNIQUE(isbn, keyword_id)
81  );
82
83  CREATE TABLE school (
84          school_id SERIAL PRIMARY KEY,
85          name VARCHAR(150) NOT NULL,
86          address VARCHAR(100) NOT NULL,
87          city VARCHAR(50) NOT NULL,
88          phone VARCHAR(15) NOT NULL CHECK (phone ~ '^\+?[0-9]+'),
89          email VARCHAR(256) NOT NULL UNIQUE CHECK (email ~
    ↪  '^[a-zA-Z0-9-]+@[a-zA-Z0-9-]+\.[a-z]{2,}$'),
90          headmaster VARCHAR(100) NOT NULL
91  );
92
93  CREATE TABLE "user" (
94          user_id SERIAL PRIMARY KEY,
95          school_id INTEGER NOT NULL REFERENCES school ON DELETE CASCADE,
96          first_name VARCHAR(50) NOT NULL,
97          last_name VARCHAR(50) NOT NULL,
98          email VARCHAR(256) NOT NULL UNIQUE CHECK (email ~
    ↪  '^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-z]{2,}$'),
99          username VARCHAR(50) NOT NULL UNIQUE,
100         password_hash VARCHAR(500) NOT NULL,
101         dob DATE NOT NULL,
102         active BOOLEAN NOT NULL DEFAULT FALSE
103 );
104
105 CREATE INDEX index_user_active ON "user" (active);
106 CREATE INDEX index_user_school_id ON "user" (school_id);
107 CREATE INDEX index_user_fullname ON "user" USING GIST (first_name gist_trgm_ops, last_name
    ↪  gist_trgm_ops);
108
109 CREATE TABLE "admin" (
110         user_id INT NOT NULL UNIQUE REFERENCES "user" ON DELETE CASCADE
111 );
112
113 CREATE TABLE teacher (
114         user_id INTEGER NOT NULL UNIQUE REFERENCES "user" ON DELETE CASCADE
115 );
116
117 CREATE TABLE lib_user (
118         user_id INTEGER NOT NULL UNIQUE REFERENCES "user" ON DELETE CASCADE REFERENCES
    ↪  teacher(user_id) ON DELETE CASCADE
```



```
);

CREATE TABLE student (
        user_id INTEGER NOT NULL UNIQUE REFERENCES "user" ON DELETE CASCADE
);

CREATE TABLE item (
        item_id SERIAL PRIMARY KEY,
        isbn varchar(13) NOT NULL REFERENCES book ON UPDATE CASCADE ON DELETE CASCADE,
        school_id INTEGER NOT NULL REFERENCES school ON DELETE CASCADE
);

CREATE INDEX index_item_isbn ON item (isbn);
CREATE INDEX index_item_school ON item (school_id);

CREATE TABLE review (
        review_id SERIAL PRIMARY KEY,
        isbn VARCHAR(13) NOT NULL REFERENCES "book" ON UPDATE CASCADE ON DELETE CASCADE,
        user_id INTEGER NOT NULL REFERENCES "user" ON DELETE CASCADE,
        rate SMALLINT NOT NULL CHECK (rate >= 1 AND rate <= 5),
        body VARCHAR(500) NOT NULL,
        active BOOLEAN NOT NULL DEFAULT FALSE,
        UNIQUE(user_id, isbn)
        -- One user can do only one review on a specific book
);

CREATE INDEX index_review_active ON review (active);
CREATE INDEX index_review_isbn ON review (isbn);

CREATE TABLE borrow (
        borrow_id SERIAL PRIMARY KEY,
        item_id INTEGER NOT NULL REFERENCES item ON DELETE CASCADE,
        lender_id INTEGER NOT NULL REFERENCES "user" ON DELETE CASCADE REFERENCES
 ↪  lib_user(user_id) ON DELETE CASCADE,
        borrower_id INTEGER NOT NULL REFERENCES "user" ON DELETE CASCADE,
        period TSTZRANGE NOT NULL DEFAULT TSTZRANGE(NOW(), NULL) CHECK (NOT ISEMPTY(period)),
        expected_return DATE NOT NULL CHECK (expected_return >= LOWER(period)::date),
        EXCLUDE USING GIST (item_id WITH =, period WITH &&)
);

CREATE INDEX index_borrow_item_id ON borrow (item_id);
CREATE INDEX index_borrow_borrower_id ON borrow (borrower_id);

CREATE TABLE booking (
         booking_id SERIAL PRIMARY KEY,
        borrow_id INTEGER REFERENCES borrow ON DELETE CASCADE UNIQUE,
         isbn VARCHAR(13) NOT NULL REFERENCES book ON UPDATE CASCADE ON DELETE CASCADE,
         user_id INTEGER NOT NULL REFERENCES "user" ON DELETE CASCADE,
         period TSTZRANGE NOT NULL DEFAULT (TSTZRANGE(NOW(), NOW() + INTERVAL '1 week'))
 ↪  CHECK (NOT ISEMPTY(period)),
        EXCLUDE USING GIST (user_id WITH =, isbn WITH =, period WITH &&)
);

CREATE INDEX index_booking_isbn ON booking (isbn);
CREATE INDEX index_booking_user_id ON booking (user_id);
```

Γ′ DML Script

Το DML Script που εισάγει τα Mock Data στην βάση, τα οποία δημιουργήθηκαν σύμφωνα με την διαδικασία που αναφέρθηκε στην ενότητα 2.4 βρίσκεται στο αποθετήριο GitHub στο path:

`fake_data/fake_data.sql`

Επιπλέον ο διαδικτυακός υπερσύνδεσμος για το DDL Script απευθείας στο GitHub αποθετήριο είναι:
https://github.com/andreasstamos/library-management-system/blob/master/fake_data/fake_data.sql

Το αρχείο αυτό δημιουργείται αυτόματα αν (αφού εγκαταστήσετε την βιβλιοθήκη Faker) τρέξετε την εντολή:

```
python3 create.py
```

Η γεννήτρια τυχαίων αριθμών έχει ρυθμιστεί με σταθερό seed προκειμένου τα αποτέλεσματα να είναι ντετερμινιστικά.

Δ΄ Οδηγίες Εγκατάστασης

Δ΄.1 Βάση Δεδομένων

Αρχικά πρέπει να εγκαταστήσετε την PostgreSQL (έκδοση 14+), να δημιουργήσετε μια βάση δεδομένων (πχ. με χρήση της εντολής `createdb`) και να δημιουργήσετε επιπλέον έναν χρήστη στην βάση δεδομένων, ο οποίος θα έχει δικαίωμα να δημιουργεί και κάνει drop πίνακες, καθώς επίσης και θα έχει δικαίωμα εγγραφής/ανάγνωσης σε αυτούς. (προκειμένου το backup/restore να λειτουργεί ο χρήστης θα πρέπει να έχει δικαιώματα να κάνει DROP τους πίνακες που θα κατασκευαστούν, καθώς και να τους δημιουργήσει).

Με χρήστη του συστήματος σας PostgresSQL που έχει δικαιώματα για αυτό, πρέπει έπειτα να εκτέλεσετε τα αρχεία `sql/schema.sql` και `sql/borrow.sql` που αντίστοιχα κατασκευάζουν τους πίνακες με τα ευρετήρια και εισάγουν μερικές απαραίτητες SQL Functions στην βάση.

Οι λεπτομέρειες εγκατάστασης της βάσης δεδομένων, ανάλογα πάντα και με το λειτουργικό σας σύστημα ξεφεύγουν του παρόντος οδηγού. Σας παραπέμπουμε στην επίσημη ιστοσελίδα της PostgreSQL για αναλυτικότερες οδηγίες.

Δ΄.2 Backend (API)

Αρχικά πρέπει να εγκαταστήσετε την python3. Επίσης πρέπει να εγκαταστήσετε το pdflatex του TexLive καθώς επίσης και να εξασφαλίσετε ότι το package qrcode του LaTex είναι εγκατεστημένο. Προτείνουμε την χρήση του Package Manager του λειτουργικού σας συστήματος αν δουλεύετε σε υπολογιστικό σύστημα Linux (για το LaTeX και το package του qrcode σε περιβάλλον Ubuntu/Debian τα πακέτα texlive-latex-recommended και texlive-pictures επαρκούν), διαφορετικά ακολουθήστε τις οδηγίες που προτείνονται στις επίσημες ιστοσελίδες των λογισμικών για την εγκατάστασή τους στο σύστημά σας.

Προτείνουμε η εγκατάσταση των βιβλιοθηκών Python να γίνει σε Virtual Environment και με χρήση pip3 για αποφυγή προβλημάτων συμβατότητας. Αν επιθυμείτε να εγκαταστήσετε με διαφορετικό τρόπο τις προαπαιτούμενες βιβλιοθήκες ή το σύστημα σας δεν υποστηρίζει Python Virtual Environment και pip3, σας παραπέμπουμε στο παράρτημα Α΄ καθώς και στο αρχείο `backend/requirements.txt` για τις σχετικές απαιτήσεις.

Το λειτουργικό σας σύστημα ενδεχομένως να έχει προεγκατεστήμενο το Python Virtual Envirionment. Διαφορετικά εγκαταστήστε τα με την εντολή: `pip3 install virtualenv`

Ακολουθήστε τα παρακάτω βήματα σε ένα τερματικό:

1. Εισέλθετε στον φάκελο του backend.

   ```
   cd backend
   ```

2. Με τον αγαπημένο σας editor (πχ. vi) επεξεργαστείτε το αρχείο `config.py`.

   Θέστε στις μεταβλητές DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASSWORD αντίστοιχα στην IP διεύθυνση του server του σύστηματος PostgrsSQL σας, στο TCP port του, στο όνομα της βάσης δεδομένων που θα χρησιμοποιήσει το παρόν σύστημα, στο όνομα χρήστη του σύστηματος PostgreSQL σας που θα χρησιμοποιήσει το παρόν σύστημα καθώς και στο password του. Αν δεν γνωρίζετε ένα ή περισσότερα από τα στοιχεία αυτά, σας παραπέμπουμε στην διαδικασία του παραρτήματος Δ΄.1 και στην επίσημη ιστοσελίδα της PostgreSQL για περισσότερες πληροφορίες.

   Επίσης θέστε στην μεταβλητή JWT_SECRET_KEY μια κατάλληλη κρυπτογραφικά τυχαία τιμή, η οποία θα χρησιμοποιηθεί για την Ψηφιακή Υπογραφή του authentication token που θα αποστέλλει το API όταν αυθεντικοποείται κάποιος χρήστης.

   Μην κοινοποιήσετε πουθένα την τιμή αυτή, όπως φυσικά και κανένα από τα υπόλοιπα στοιχεία.

3. Δημιουργήστε το Python Virtual Environment.

   ```
   python3 -m venv venv
   ```


4. Ενεργοποιήστε στο τερματικό που βρίσκεστε το Python Virtual Environment.

   ```
   source venv/bin/activate
   ```

5. Εγκαταστήστε τις προαπαιτούμενες βιβλιοθήκες.

   ```
   pip3 install -r requirements.txt
   ```

6. Εκτελέστε το backend με χρήση του Development Server του Flask:

   ```
   python3 run.py
   ```

ΠΡΟΣΟΧΗ/Disclaimer: Οι παραπάνω οδηγίες θα οδηγήσουν στο να τρέχει το backend στον τοπικό υπολογιστή με τον Development Server του Flask θέτοντας πιθανώς τον υπολογιστή σας σε κινδύνους ασφαλείας. Αυτός προφανώς αντιπροτείνεται για χρήση σε περιβάλλον παραγωγής. Την εγκατάσταση σε περιβάλλον παραγωγής θα πρέπει να αναλάβουν πεπειραμένοι χρήστες εγκαθιστώντας και ρυθμίζοντας ενδεικτικά αλλά όχι αποκλειστικά reverse proxy (πχ. nginx), ssl πιστοποιητικά, firewalls κλπ. Καμία ευθύνη δεν φέρουν οι developers σε καμία περίπτωση για πιθανά προβλήματα ασφαλείας.

Δ΄.3 Web Application

Αρχικά πρέπει να εγκαταστήσετε την python3. Προτείνουμε την χρήση του Package Manager του λειτουργικού σας συστήματος αν δουλεύετε σε υπολογιστικό σύστημα Linux, διαφορετικά ακολουθήστε τις οδηγίες που προτείνονται στις ιστοσελίδες τους για την εγκατάστασή τους στο σύστημα σας.

Αφού εισέλθετε στον φάκελο του Web Application με την εντολή:

```
cd client
```

εγκαταστήσετε εύκολα τα προαπαιτούμενα για το Web Application με την εντολή:

```
npm i
```

Μπορείτε να τρέξετε στον τοπικό υπολογιστή έναν Development Server που να σερβίρει την Web Application. Αυτός προφανώς αντιπροτείνεται για περιβάλλον παραγωγής. Ισχύει το ίδιο Disclaimer και η αποποίηση ευθύνης που αναφέρθηκε παραπάνω για το backend. Προκειμένου να τρέξετε τον Develpment Server τρέξτε την εντολή:

```
npm start
```

Αν επιθυμείτε να εγκαταστήσετε την Web Application σε έναν server με σκοπό την χρήση σε περιβάλλον παραγωγής και μόνο εφόσον είστε πεπειραμένος χρήστης και πάντα αποκλειστικά με δική σας ευθύνη, μπορείτε να κάνετε compile την εφαρμογή με χρήση της εντολής:

```
npm build
```

Στην συνέχεια τα περιεχόμενα του φακέλου `build/` θα πρέπει να σερβιριστούν ως στατικά αρχεία από τον αγαπημένο σας Διακομιστή Ιστού (πχ. Nginx/Apache) ή από το αγαπημένο σας Hosting Provider.