Problema 1

1.Class diagram

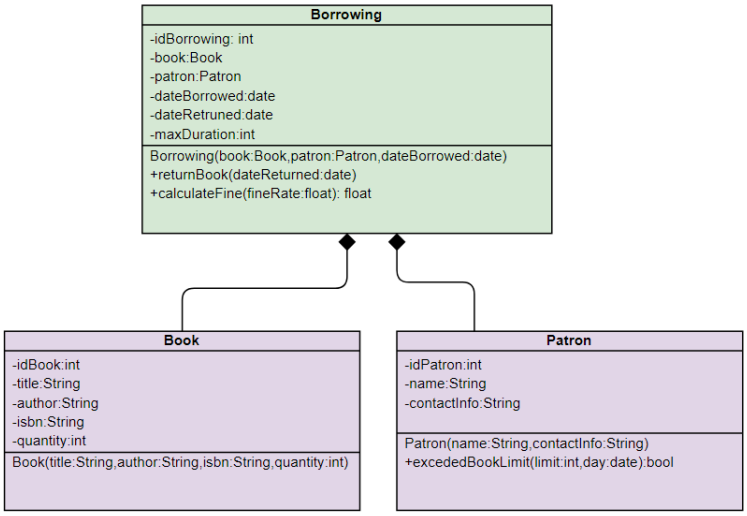


Diagrama de clase este o reprezentare vizuală a structurii și a relațiilor dintre clasele unui sistem.Diagrama de clase pe care am creat-o cuprinde 3 clase :Borrowing, Book și Patron.

Clasa Book:

- Atribute: idBook, title, author, isbn, quantity
- Metode: -constructorul clasei, care primește ca parametrii pe title, author, isbn și quantity . Nu primește id-ul ca parametru deoarece acesta ar trebui generat automat.

Clasa Patron

- Atribute: idPatron, name, contactInfo
- Metode : - constructorul clasei, care are ca parametrii pe name si contactInfo .

  -funcția `excededBookLimit (limit:int, day:date)` returnează "true" dacă o persoană (Patron) a depașit limita maximă de cărți pe care le poate împrumuta într-o zi.

Clasa Borrowing:

- Atribute: idBorrowing, book, patron, dateBorrowed, dateReturned, maxDuration
- Metode: -constructorul clasei, cu parametrii book, patron și dateBorrowed. Cu ajutorul acestui constructor se poate face un nou împrumut a cărții "book" de către persoana "patron" la data de "dateBorrowed".
  - `returnBook(dateReturned :date)` este funcția cu ajutorul căreia se face returnarea cărții .Ea trebuie să modifice atributul dateReturned al obiectului de tip Borrowing.
  - `calculateFine(fineRate:float)` calculează penalizarea în cazul în care diferența dintre dateReturned și dateBorrowed este mai mare decât maxDuration. În caz contrar returnează 0.

Relații:

Book și Patron au o relație de compozitie cu clasa Borrowing. Această relație indică faptul că o clasa de tip Borrowing conține o clasă de tip Book si una de tip Patron. Adică o carte poate fi împrumutată de un beneficiar, iar clasa "Borrowing" ține evidența detaliilor împrumutului, cum ar fi data împrumutului ,data returnării și numărul maxim de zile în care poate fi pastrată cartea.

2. Database Diagram:

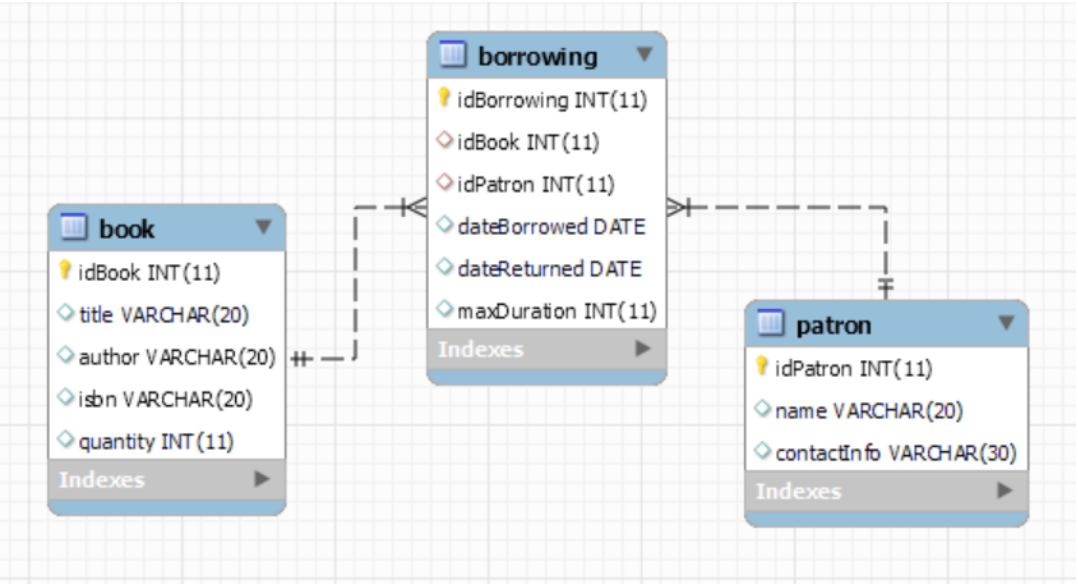
Diagrama bazei de date cuprinde 3 tabele care corespund cu clasele din diagrama de clase.

Primul atribut al fiecarei tabele (idBook,idBorrowing,idPatron) reprezintă cheia primară a acesteia, pe care am reprezentat-o cu un numar întreg.

Tabela Borrowing are și două chei străine , care reprezintă id-ul cărții împrumutate și id-ul persoanei care a facut împrumutul.

Între tabelele Book și Borrowing există o relație de one-to-many. Adică o carte poate apărea în mai multe împrumuturi, dar un împrumut poate conține o singură carte. Pentru a împrumuta mai multe cărți , se consideră că se face un împrumut diferit pentru fiecare carte .Am ales această variantă deoarece chiar dacă mai multe cărți pot fi împrumutate deodată nu înseamnă cu vor fi și returnate deodată.

Între tabelele Patron si Borrowing există , de asemenea o relație de tip one-to-many .O persoană poate face mai multe împrumuturi, în schimb un împrumut îi poate aparține unei singure persoane.



Problema 2

1.Logical Design:

Organizarea întrebarilor

Întrebarile au fost scrise direct într-un fișier JavaScript, denumit questions.js .
Acestea au urmatoarea structură. Fiecare întrebare are un id distinct, textul întrebarii, o listă de 4 variante de ales, și indexul variantei corecte din listă.

```
const questions = [
  {
    id:"0",
    question: "A green owl is the mascot for which app?",
    choices: ["Facebook", "Twitter", "Duolingo","Spotify"],
    correct: 2
  },
  {
   id:"1",
   question: "What is the capital of United Kingdom?",
   choices: ["London", "Paris", "Nairobi","San Francisco"],
   correct: 0
  },
  {
   id:"2",
   question: "How many days are there in a week?",
   choices: ["Five", "Three", "Eight","Seven"],
   correct: 3
  }]
```

Pseudocod pentru alegerea aleatoare fara repetitie a intrebarilor si calculul progresului si al scorului

```
id_intrebare <- 0;
questions_answered <- 0;
corect_answers <- 0;
contor[50] <- [0...0]

Cât timp (questions_answered < 50)
{
Cât timp (contor[id_intrebare] = 1)  {id_intrebare <- număr aleator între 0 si 49}
contor[id_intrebare] <- 1;
Se afișează întrebarea cu numărul id_intrebare;
Se așteaptă răspunsul utilizatorului
Dacă utilizatorul a răspuns corect atunci corect_answers++;
questions_answered++;
}
```

2. Algorithm Implementation (Partial Code)

- **Generate random questions .**

Se generează un numar aleator între 0 și 49. Acesta reprezintă id-ul întrebării care urmează să fie afișată. Pentru a ne asigura că nu se afișează o întrebare de două ori se folosește un șir cu rol de contor. Acesta a fost inițializat cu 0. De fiecare dată când algoritmul găsește un id nefolosit îl marchează cu 1. Dacă algoritmul găsește un id care a fost folosit deja, își va continua căutarea până când găsește un id nefolosit.

```
function generateRandomId() {
    id = Math.floor(Math.random() * 50);
    if (cnt[id] == 0) { cnt[id] = 1}
    else { generateRandomId(); }
}
```

- **Display questions on the interface**

Această funcție are rolul de a afișa o întrebare cu un anumit id pe interfața grafică. Prima linie din funcție afișează textul întrebării, iar a doua afișează variantele de răspuns.

```
function showQuestion() {
    document.getElementById("question-text").textContent = questions[id].question;
    document.querySelectorAll(".choice").forEach((choice, i)
                              => { choice.textContent = questions[id].choices[i]; });
    document.getElementById("feedback").textContent = "";
}
```

- **Receive user input, and calculate the score**

Aceasta funcție primește varianta aleasă de utilizator de la interfața grafică .Apoi verifică dacă este corectă și trimite înapoi soluția către interfața grafică sub forma unui feedback. Dacă răspunsul este corect atunci se incrementează variabila correctAnswers .
De asemenea, se incrementează variabila questionsAnswered, indiferent dacă răspunsul a fost corect sau nu. Această variabilă calculează progresul care se afișează pe interfața grafică printr-un progress bar. Feedbackul rămâne afișat timp de două secunde până când aplicația trece automat la următoarea întrebare , fără intervenția utilizatorului. Acest lucru se întâmplă doar dacă au mai rămas întrebări nerăspunse. Dacă au fost răspunse toate, se afișează doar punctajul final.

```
function verifyAnswer(selected) {
     const feedback = document.getElementById("feedback");
      if (selected == questions[id].correct) {
        correctAnswers++;
        feedback.textContent = "Correct!";
     } else {
        feedback.textContent = "Incorrect!" + "The correct answer is " +
questions[id].choices[questions[id].correct];
    }
    setTimeout(() => {
        questionsAnswered++;
        const progress = document.getElementById("progress");
        progress.value =  questionsAnswered;
```

```
 if (questionsAnswered < 50) { generateRandomId(); showQuestion(); }
    else {
        document.querySelector(".quiz-container").innerHTML
             = `<p>Correct answers:  ${correctAnswers} out of 50 questions.</p>`;
    }
}, 2000);
```

3. Class and Database Representation (Explanation Only):

Pentru reprezentarea bazei de date , aș folosi următoarele tabele :Question, User, UserProgress.

Pentru User avem atributele : id, nume

Atributele tabelei Question ar fi : id, text, choice1, choice2, choice3, choice4, correctChoice.

Pentru UserProgress: id, idUser, questionsAnswered, correctAnswers.

Înainte de începerea quizului ar trebui să existe deja un utilizator în tabela User, și mai multe intrebări în tabela Question.

La începutul unui quiz ,se creează o înregistrare nouă în tabela UserProgress, care are questionsAnswered=0 și correctAnswers=0. După fiecare întrebare la care răspunde utilizatorul , se incrementeaza questionAnswered , iar dacă răspunsul este corect se incrementează correctAnswers.

La finalul quizului, scorul utilizatorului se va afla in UserProgress.corectAnswers.

Daca utilizatorul începe un quiz nou , scorul nu se va suprascrie peste scorul din quizul trecut , ci se va scrie într-o înregistrare nouă a tabelei UserProgress.Din acest motiv ,între User si UserProgress ar trebui sa existe o relație one-to-many.