2. Principiile și condițiile de organizare a testelor

1. **Testare modulară și atomică**
   - Fiecare metodă de test (`[TestMethod]`) verifică un singur scenariu de funcționare sau de eroare.
   - Se urmărește ca, în cazul unui eșec pe parcursul unui test, diagnosticul să fie clar (de ex. mesajul de eroare returnat de metodele din DLL, excepția aruncată etc.).
2. **Izolarea resurselor și resetarea stării**
   - Pentru clasa `Database.Database`, la începutul fiecărui test se deschide o conexiune spre o bază de date „curată" (`ClearDatabase/biblioteca.db`) aflată în folderul de testare.
   - După fiecare test, conexiunea se închide în `[TestCleanup]` și, acolo unde a fost inserat ceva temporar (de ex. utilizator, carte, abonat), elementele sunt șterse explicit pentru a reveni la starea inițială.
   - Pentru clasa `Server.Server`, înainte de fiecare test se resetează instanța singleton `Database._database`, iar baza de date e adusă la starea de referință („ClearDatabase"). De asemenea, la final, serverul este oprit și utilizatorii inserați temporar (ex. `existingUser`) sunt șterși.
3. **Principiul „Arrange–Act–Assert"**
   - *Arrange*: se pregătește datele de intrare (de ex. se construiesc obiecte `Utilizator`, `Carte` sau `Abonat` cu atribute specifice).
   - *Act*: se apelează metoda din clasa under test (fie din `Database.Database`, fie printr-un mesaj transmis serverului).
   - *Assert*: se validează condițiile așteptate:
     - Dacă metoda trebuie să returneze `true`/`false`, se face `Assert.IsTrue(...)` sau `Assert.IsFalse(...)`.
     - Dacă metoda aruncă o excepție, se utilizează atributul `[ExpectedException]`.
     - Dacă metoda scrie ceva în consolă (ex. notificări de succes/eroare), capturăm output-ul în `StringWriter` și verificăm conținutul cu `StringAssert.Contains(...)`.
4. **Acoperirea scenariilor pozitive și negative**
   - Pentru fiecare funcționalitate de bază (inserare, ștergere, căutare, împrumut etc.) există cel puțin un test care validează comportamentul așteptat („happy path") și cel puțin un test care verifică comportamentul în situații eronate:
     - Date invalide (ex. `null`, stringuri nevalide, ID-uri negative).
     - Date care violează constrângeri de integritate (ex. dublarea unei chei unice).
     - Entități inexistente (ex. ștergerea unui utilizator inexistent, împrumuturi pe ID-uri invalide).
5. **Testare black-box**
   - Pentru clasa `Database`, testele se fac apelând exclusiv metodele publice (în cazul unor metode private cum este `IsCartedisponibil`, se folosește reflexie pentru a le invoca și a valida rezultatul).
   - Pentru clasa `Server`, testele trimit mesaje JSON către server (prin `TcpClient`) și verifică răspunsurile primite pe socket (astfel simulând cererile efective ale unui client).
   -

6. **Filtrarea pe baze de roluri și restricții**
    - În cazul gestionării rolurilor de utilizator, există teste care verifică inserarea unui `Utilizator` cu rol valid („bibliotecar”, „administrator”) și cu rol inexistent (trebuie să eșueze).
    - În cazul abonaților, se verifică inserarea și excepțiile în funcție de telefon unic sau `null`.

3. Obiectivele testării și ce se urmărește a fi verificat

- **Conectarea și gestiunea tranzacțiilor**
  Verificarea faptului că se poate deschide conexiunea către baza SQLite, că tranzacțiile pentru inserarea unei cărți sunt atomic executate, iar commit/rollback-urile funcționează în cazul erorilor.
- **Inserarea de entități cu date valide**
    - **Utilizator**: roluri valide („bibliotecar”, „administrator”) → metoda `InsertUser(...)` returnează `true` și afișează „Utilizator adaugat cu succes”.
    - **Isbn** (metoda `InsertIsbn`) → verificare format, inserare într-o singură tranzacție, afișare mesaj de succes.
    - **Carte** (metoda `InsertBook`) → inserare a înregistrării în tabelele `Isbn` și `Carte` concomitent, obținerea `id_carte`.
    - **Abonat** (metoda `InsertClient`) → inserarea unui abonat nou, valabilitate telefon unic.
- **Gestionarea excepțiilor și a condițiilor de eroare**
    - Inserare utilizator cu rol inexistent → metoda `InsertUser` returnează `false`, afișează „Rolul specificat nu exista în baza de date”.
    - Inserare dublă de utilizator cu același `nume_user` → aruncă o eroare SQL (constrângere unică), se prinde și se afișează un mesaj corespunzător („Eroare la inserarea utilizatorului în baza de date: ...”).
    - Inserare ISBN cu date invalide (`null` la câmp obligatoriu) → metoda `InsertIsbn` aruncă o excepție SQLite, se prinde și se returnează `false`.
    - Inserare Carte cu date invalide (ISBN incorect sau date lipsă) → metoda `InsertBook` returnează `-1`.
    - Inserare Client cu nume `null` → metoda `InsertClient` returnează `false` și afișează „Eroare la adăugarea unui abonat în baza de date: ...”.
    - Ștergere utilizator inexistent sau administrator → `DeleteUser` returnează `false`, afișează „Utilizatorul nu a fost găsit...”.
    - Ștergere carte cu `id_carte` invalid sau statut diferit de „disponibil” → `DeleteBook` returnează `false` și afișează „Nu a fost găsită nicio carte cu acest id.”
- **Funcționalități de căutare**
    - **GetCartiByIsbn**:
        - ISBN valid și carte disponibilă → se reîntoarce cel puțin un obiect `Carte` cu acele atribute.
        - ISBN invalid (sau tabelă goală) → metoda returnează listă goală.
    - **GetLoanedBooks**:
        - După efectuarea unui împrumut, `GetLoanedBooks` returnează lista de cărți nereturnate.
        - Pentru un abonat inexistent sau fără împrumuturi active → listă goală (sau `null`, tratat ca listă nepopulată).

- **Funcționalități de împrumut și returnare**
  - **InsertLoan**:
    - Dacă cartea este disponibilă și abonatul există (ID valid) → returnează `true`, setează `status = 'indisponibil'` și decrementarea limitei abonatului.
    - Dacă cartea e deja „indisponibil” sau abonatul nu există → returnează `false` și se afișează un mesaj corespunzător.
  - **ReturnBook**:
    - Dacă există un împrumut activ (Data_restituire = `null`), se actualizează `data_restituire`, se setează `status='disponibil'` și se incrementează limita abonatului. Returnează `true`.
    - Dacă nu există un împrumut activ pentru acea combinație (ID_invalid), se face rollback și returnează `false`, cu afișarea mesajului „Nu există un împrumut activ pentru această carte.”
- **Verificarea disponibilității cărților (metodă privată)**
  - `IsCartedisponibil(int idCarte)` (invocată prin reflexie):
    - ID valid și `status='disponibil'` în tabelă → `true`.
    - ID invalid sau `status!='disponibil'` sau eroare SQL → `false`.
- **Gestionarea statusului abonat**
  - **GetStatusAbonat(int idAbonat)**:
    - ID valid → returnează stringul „blocat”, „cu restrictii” sau „fara restrictii” (sau `null` dacă ID invalid).
  - **UpdateStatusAbonat(int idAbonat, string mesaj)**:
    - ID valid și mesaj fiabil („blocat”, „cu restrictii”, „fara restrictii”) → `true` și mesaj de succes.
    - ID invalid sau mesaj inexistent (în lipsa înregistrării) → `false`.
  - **UnRestrictAbonat**, **BlocareAbonat**, **RestrictAbonat**: reiau `UpdateStatusAbonat` cu mesajul corespunzător.
- **Detectarea abonaților întârziati sau cu restricții**
  - **CautareIntarziati()**: lista care cuprinde abonații cu:
    - fie status = „cu restrictii”,
    - fie cel puțin un împrumut cu `deadline < azi` și `data_restituire IS NULL`.
  - **CautareDoarIntarziati()**: listează doar abonații cu împrumuturi active ce au `deadline < azi` (fără să includă cei cu „cu restrictii” care nu au împrumut întârziat).
  - Testele verifică:
    - Prezența unui abonat cu restricții în CautareIntarziati (trebuie inclus).
    - Excluderea unui abonat fără nicio întârziere din CautareIntarziati (trebuie absent).
    - Dacă nu există întârziere, CautareDoarIntarziati → listă goală.
- **Funcționalități de server (componenta `Server.Server`)**
  - Se testează prin trimiterea mesajelor JSON structurate în obiecte `Message` și verificarea răspunsului textual primit:
    1. **login** (autentificarea angajatului):
       - Credențiale valide → „Login successful.”.
       - Credențiale invalide → „Login failed. Invalid data.”.
       - Al doilea login consecutive pe același `username` → al doilea apel eșuează („Login failed.”).
    2. **registerSubscriber**:

- Număr de telefon unic → „Subscriber Register successful.”.
- Număr de telefon duplicat → „Subscriber registration failed.”.

3. **loginSubscriber**:
   - Telefon existent → răspuns de forma „Subscriber Login successful|<IdAbonat>|<Status>|…”.
   - Telefon inexistent → „Subscriber Login failed”.
4. **searchBooks**:
   - Fără potriviri → „No books found.”.
   - Cu potriviri (există cel puțin o carte parțial corespondentă titlu/autor) → returnează un șir concatenat cu separator „~” și „|” (ex. `id~titlu~autor|…`).
5. **insertLoan**:
   - ID abonat și ID carte valide și carte disponibilă → „Inserted Loan successful.”.
   - ID-uri invalide sau carte indisponibilă → „Inserted Loan failed.”.
6. **getLoans**:
   - Pentru abonat cu împrumuturi active → șir concatenat similar celui de la `searchBooks` (ex. `idCarte~Titlu~Autor|…`).
   - Pentru abonat fără împrumuturi → „No loans found.”.
7. **returnBook**:
   - Pentru abonat/cartă cu împrumut activ → „Book returned successful.”.
   - În caz contrar („No active loan”) → „Book return failed.”.
8. **getStatusClient**:
   - ID abonat valid → „Status found: <status>”.
   - ID invalid → „Status not found.”.
9. **registerEmployee** (InsertUser):
   - Rol valid → „Employee registered successful.”.
   - Rol invalid → „Employee registration failed.”.
10. **deleteLibrarian** (DeleteUser):

- Utilizator non-administrator existent → „Librarian deleted successful.”.
- Utilizator inexistent sau administrator → „Librarian deletion failed.”.

11. **addBook** (InsertBook + InsertIsbn):

- Date valide → „Book added successful.”.
- Date invalide (ex. titlu `null`) → mesaj `null` (metoda nu mai trimite nimic – clientul primește `null response`).

12. **searchBook** (GetCartiByIsbn):

- ISBN valid → răspuns formatat `idCarte~Titlu~Autor~Editura|…`.
- ISBN inexistent → „No books found with the given ISBN.”.

13. **deleteBook** (DeleteBook):

- ID carte valid și carte disponibilă → „Book deleted successful.”.
- ID invalid → „Book deletion failed.”.

14. **searchSubscribers** (CautareIntarziati):

- Există abonați cu restricții/întârziere → răspuns formatat `id~nume~prenume~…|….`
- Nu există astfel de abonați → „No subscribers found with restrictions or blocked.”.

15. **updateStatus** (UpdateStatusAbonat):

- ID abonat valid → „Status updated successful.”.
- ID invalid → „Status update failed.”.

4. Grupurile de teste pentru clasa Database.Database

Pentru `Database.Database`, testele au fost împărțite în următoarele categorii (grupuri), conform funcționalităților implementate:

4.1. Teste pentru inserarea și autentificarea utilizatorilor

- **InsertUserSuccessfullyBibliotecar**
  - **Scop:** Verifică că un utilizator cu rol „bibliotecar” (rol valid) poate fi inserat cu succes.
  - **Ce se verifică:**
    - Metoda `InsertUser` returnează `true`.
    - Pe `Console.Out` apare șirul „Utilizator adaugat cu succes”.
    - La final, utilizatorul test (`TestUser5`) este șters.
- **InsertUserSuccessfullyAdministrator**
  - **Scop:** Verifică inserarea cu succes a unui utilizator cu rol „administrator” (rol valid).
  - **Ce se verifică:**
    - `InsertUser` returnează `true`.
    - Mesajul de consolă conține „Utilizator adaugat cu succes”.
- **InsertUserRoleNotFound**
  - **Scop:** Verificarea scenariului în care rolul specificat nu există în tabelă.
  - **Ce se verifică:**
    - `InsertUser` returnează `false`.
    - Mesajul de consolă conține „Rolul specificat nu exista in baza de date”.
- **InsertUserSqlError**
  - **Scop:** Verifică gestionarea situației în care se încearcă inserarea a doi utilizatori cu același `nume_user` (constrângere unică SQL).
  - **Ce se verifică:**
    - Primul `InsertUser(testUser)` returnează `true`.
    - Al doilea apel `InsertUser(testUser)` returnează `false`.
    - Mesajul de consolă indică „Eroare la inserarea utilizatorului in baza de date: ...”.
- **DeleteUserSuccessfully**
  - **Scop:** Ștergerea unui utilizator non-administrator existent.
  - **Ce se verifică:**

        - `InsertUser` pe `TestUser2` → `true`.
        - `DeleteUser("TestUser2")` → `true` (metoda `DeleteUser` returnează `true`).
- **DeleteUserUserNotFound**
    - **Scop:** Încercarea de ștergere a unui utilizator inexistent.
    - **Ce se verifică:**
        - `DeleteUser("User Not Found")` returnează `false`.
        - Mesajul de consolă conține „Utilizatorul nu a fost gasit sau a fost deja sters”.
- **DeleteUserSqlError**
    - **Scop:** Verifică scenariul în care se încearcă ștergerea unui administrator (constrângere: nu se poate șterge administrator).
    - **Ce se verifică:**
        - Se inserează `TestAdminDelete` cu rol „administrator”.
        - Apel `DeleteUser("TestAdminDelete")` → `false`.
        - Mesajul de consolă conține „Utilizatorul nu a fost gasit sau a fost deja sters sau este administrator.”.
- **LoginSuccessfully**
    - **Scop:** Autentificarea cu date valide pentru utilizator (rol „bibliotecar”).
    - **Ce se verifică:**
        - `InsertUser("TestLogin", ..., "bibliotecar")` → `true`.
        - Apel `Login(...)` → `true`.
        - La final, utilizatorul `TestLogin` este șters.
- **LoginUnsuccessfully**
    - **Scop:** Verificarea aruncării excepției `InvalidUserDataException` când datele de login sunt incorecte (parola greșită).
    - **Ce se verifică:**
        - Apel `Login("TestFailLogin", "gresit", "bibliotecar")` → aruncă `InvalidUserDataException`.

4.2. Teste pentru gestionarea ISBN-urilor (tabela `Isbn`)

- **InsertIsbnSuccsessfully**
    - **Scop:** Inserarea cu succes a unui nou ISBN.
    - **Ce se verifică:**
        - Apel `InsertIsbn(testCarte)` cu `Isbn="0000000000009"` → `true`.
        - Mesajul de consolă conține „Noul isbn a fost adaugat cu succes.”
- **InsertIsbnAlreadyExisting**
    - **Scop:** Inserarea unui ISBN deja prezent în bază.
    - **Ce se verifică:**
        - Primul apel `InsertIsbn(testCarte)` cu `Isbn="0000000001"` → `true`.
        - Al doilea apel `InsertIsbn(testCarte)` → `true` (metoda recunoaște existența și afișează „Isbn-ul exista deja”).
        - Mesajul de consolă conține „Isbn-ul exista deja”.
- **InsertIsbnSqlError**
    - **Scop:** Inserarea unui ISBN cu date invalide (titlu `null`).
    - **Ce se verifică:**
        - Apel `InsertIsbn(testCarteCuTitluNull)` → `false`.

- Mesajul de consolă conține „Eroare la inserarea in tabela Isbn: ...”.

4.3. Teste pentru gestionarea cărților (tabela `Carte`)

- **InsertBookSuccessfully**
  - **Scop:** Inserarea cu succes a unei cărți noi (se presupune că ISBN-ul nu există în prealabil).
  - **Ce se verifică:**
    - Apel `InsertBook(testCarte)` → returnează `idCarte > 0`.
- **InsertBookSqlError**
  - **Scop:** Inserarea unei cărți cu date invalide (titlu `null`).
  - **Ce se verifică:**
    - Apel `InsertBook(testCarteCuTitluNull)` → returnează `-1`.
- **GetCartiByIsbnIsbnValid**
  - **Scop:** Obținerea cărților după un ISBN valid care tocmai a fost inserat.
  - **Ce se verifică:**
    - Se inserează `Carte testCarte` cu ISBN `"0000000000021"` → obținem `idCarte`.
    - Apel `GetCartiByIsbn("0000000000021")` → lista returnată are `Count > 0` și conține cel puțin un obiect cu `Titlu == "Carte Valid ISBN"`.
- **GetCartiByIsbnIsbnInvalid**
  - **Scop:** Apel `GetCartiByIsbn("ISBNINEXISTENT")` pentru un ISBN care nu există.
  - **Ce se verifică:**
    - Metoda returnează o listă cu `Count == 0`.

4.4. Teste pentru disponibilitatea cărților (metodă privată `IsCartedisponibil`)

- **IsCarteDisponibilaTrueCarte**
  - **Scop:** Pentru o carte proaspăt inserată cu `status="disponibil"`, `IsCartedisponibil(idCarte)` ar trebui să returneze `true`.
  - **Ce se verifică:**
    - Se apelează prin reflexie metoda privată `IsCartedisponibil(...)` și se obține `true`.
- **IsCarteDisponibilaFalse**
  - **Scop:** Dacă cartea a fost împrumutată (prin `InsertLoan`), `IsCartedisponibil(idCarte)` returnează `false`.
  - **Ce se verifică:**
    - Se inserează o carte cu `status="disponibil"`, apoi un abonat nou și se face `InsertLoan(...)` → cartea devine `indisponibil`.
    - Apel `IsCartedisponibil(idCarte)` → `false`.
- **IsCarteDisponibilaIdInvalid**
  - **Scop:** Apelul cu un `idCarte` invalid (negativ) trebuie să returneze `false`.
  - **Ce se verifică:**
    - Prin reflexie se apelează `IsCartedisponibil(-12345)` → `false`.

4.5. Teste pentru împrumuturi și returnări (tabelele `Imprumut` și `Carte`)

- **InsertLoanSuccessfully**
  - **Scop:** Inserarea cu succes a unui împrumut când cartea este disponibilă și abonatul există.
  - **Ce se verifică:**
    - Se inserează o carte disponibilă și un abonat nou (apel `InsertClient`).
    - `InsertLoan(idAbonat, idCarte, "acasa")` → `true`.
- **InsertLoanCarteIndisponibila**
  - **Scop:** Încercare de împrumut când cartea este deja marcată `status="indisponibil"`.
  - **Ce se verifică:**
    - Se inserează o carte cu `status="indisponibil"` manual (la crearea obiectului `Carte`).
    - Se creează un abonat și se face un `InsertLoan` → cartea devine „indisponibil” (procedeul intern nu permite `InsertLoan` pentru cartea deja indisponibilă).
    - Se face al doilea apel `InsertLoan(...)` → `false`.
- **InsertLoanAbonatInexistent**
  - **Scop:** Încercarea de împrumut cu un `idAbonat` invalid (nu există în tabelă).
  - **Ce se verifică:**
    - `InsertLoan(-12345, idCarteValabil, "acasa")` → `false`.
- **InsertLoanSqlError**
  - **Scop:** Încercarea de împrumut cu ambele ID-uri invalide (`-1, -1`).
  - **Ce se verifică:**
    - `InsertLoan(-1, -1, "acasa")` → `false`.
- **ReturnBookNoActiveLoanReturnsFalse**
  - **Scop:** Apelarea `ReturnBook` când nu există un împrumut activ pentru combinația dată.
  - **Ce se verifică:**
    - `ReturnBook(-1, -1)` → `false`.
- **ReturnBookAfterLoanReturnsTrue**
  - **Scop:** După ce s-a efectuat un împrumut, `ReturnBook` trebuie să returneze `true`.
  - **Ce se verifică:**
    - Se inserează carte și abonat, se face `InsertLoan(...)`, apoi `ReturnBook(...)` → `true`.

4.6. Teste pentru gestionarea clienților (tabela `Abonat`)

- **GetAbonatByPhoneValidPhone**
  - **Scop:** Obținerea unui abonat după telefon când acesta există.
  - **Ce se verifică:**
    - Se inserează un abonat cu telefon valid („0700765432”).
    - `GetAbonatByPhone("0700765432")` → obiect `Abonat` nenul și `Nume == "NumeValid"`.
- **GetAbonatByPhoneInvalidPhone**
  - **Scop:** Așteptare excepție `ClientNotFoundException` când telefonul nu există.

- **Ce se verifică:**
  - `GetAbonatByPhone("TELEFONINEXISTENT")` → aruncă `ClientNotFoundException`.
- **InsertClientSuccessfully**
  - **Scop:** Inserarea cu succes a unui abonat nou cu date valide.
  - **Ce se verifică:**
    - `InsertClient(abonatNouCuTelefonUnic)` → `true`.
    - Apoi se apelează `GetAbonatByPhone(...)` și se verifică că atributul `Nume` este corect („ClientTest”).
- **InsertClientSqlErrorNullName**
  - **Scop:** Inserarea unui abonat cu **nume = null** (câmp obligatoriu).
  - **Ce se verifică:**
    - `InsertClient(abonatCuNumeNull)` → `false`.
    - Mesajul de consolă conține „Eroare la adaugarea unui abonat in baza de date: ...”.

4.7. Teste pentru căutări și filtre (tabelele `Isbn`, `Carte`, `Imprumut`, `Abonat`)

- **CautareCartiPartialReturnsMatchingBooks**
  - **Scop:** Verifică căutarea parțială după titlu și autor.
  - **Ce se verifică:**
    - Se inserează trei cărți:
      1. `"AlphaTitle" + "AuthorMatch"`
      2. `"BetaTitle" + "AuthorMatch"`
      3. `"GammaTitle" + "OtherAuthor"`
    - Apel `CautareCartiPartial("Alpha", "Author")` → lista conține exact o carte (ID-ul lui `c1`).
    - Titlul și ID-ul se potrivesc cu `c1`.
- **CautareCartiPartialNoMatchesReturnsEmptyList**
  - **Scop:** Când nu există rezultate pentru cuvintele parțiale.
  - **Ce se verifică:**
    - `CautareCartiPartial("Nonexistent", "Nobody")` → listă cu `Count == 0`.
- **GetLoanedBooksAfterLoanReturnsBook**
  - **Scop:** După efectuarea unui împrumut, `GetLoanedBooks` returnează cartea împrumutată.
  - **Ce se verifică:**
    - Se inserează `Carte` și `Abonat`, apoi se face `InsertLoan`.
    - `GetLoanedBooks(idAbonat)` → listă nenulă și conține cartea cu `IdCarte` și `Titlu` potrivite.

4.8. Teste pentru statusul abonatului și abonații întârziati/with restrictions

- **UpdateStatusAbonatAbonatValidMesajValid**
  - **Scop:** Actualizarea statusului (`cu restrictii`) pentru un abonat existent.
  - **Ce se verifică:**
    - Se inserează un abonat nou.
    - `UpdateStatusAbonat(idAbonat, "cu restrictii")` → `true`.
- **UpdateStatusAbonatAbonatValidMesajInvalid**

- **Scop:** Dacă mesajul nu există (nu corespunde niciunei valori permise), metoda returnează `false`.
- **Ce se verifică:**
  - Se inserează un abonat, apoi se apelează `UpdateStatusAbonat(idAbonat, "statusInexistent")` → `false`.

- **UpdateStatusAbonatAbonatInvalid**
  - **Scop:** ID abonat invalid → `UpdateStatusAbonat(-88888, "cu restrictii")` → `false`.
- **UnrestrictAbonatAbonatValid**
  - **Scop:** Eliminarea restricțiilor pentru un abonat existent (`UnRestrictAbonat`).
  - **Ce se verifică:**
    - Se inserează un abonat cu `status="cu restrictii"`.
    - `UnRestrictAbonat(idAbonat)` → `true`.
- **UnrestrictAbonatAbonatInvalid**
  - **Scop:** ID invalid → `UnRestrictAbonat(-99999)` → `false`.
- **BlocareAbonatAbonatValid**
  - **Scop:** Blocarea unui abonat existent (`BlocareAbonat`).
  - **Ce se verifică:**
    - Se inserează un abonat cu `status="fara restrictii"`.
    - `BlocareAbonat(idAbonat)` → `true`.
- **BlocareAbonatAbonatInvalid**
  - **Scop:** ID invalid → `BlocareAbonat(-77777)` → `false`.
- **RestrictAbonatAbonatValid**
  - **Scop:** Aplicarea restricțiilor unui abonat existent (`RestrictAbonat`).
  - **Ce se verifică:**
    - Se inserează un abonat cu `status="fara restrictii"`.
    - `RestrictAbonat(idAbonat)` → `true`.
- **RestrictAbonatAbonatInvalid**
  - **Scop:** ID invalid → `RestrictAbonat(-33333)` → `false`.
- **GetStatusAbonatValidReturnsCorrectStatus**
  - **Scop:** Obținerea statusului unui abonat când ID-ul e valid.
  - **Ce se verifică:**
    - Se inserează un abonat cu `status="cu restrictii"`.
    - `GetStatusAbonat(idAbonat)` → returnează `"cu restrictii"`.
- **GetStatusAbonatInvalidIdReturnsNull**
  - **Scop:** ID invalid → `GetStatusAbonat(-9999)` → `null`.

4.9. Teste pentru căutarea abonaților întârziati/with restrictions

- **CautareIntarziatiRestrictedClientIncluded**
  - **Scop:** Verifică că un abonat cu `status="cu restrictii"` este returnat de `CautareIntarziati()`.
  - **Ce se verifică:**
    - Se inserează un abonat cu `status="cu restrictii"`.
    - `CautareIntarziati()` → lista conține cel puțin abonatul inserat.
- **CautareIntarziatiNoIssuesReturnsEmptyList**
  - **Scop:** Un abonat fără întârziere (status „fara restrictii”) nu trebuie returnat în `CautareIntarziati()`.
  - **Ce se verifică:**

        - Se inserează un abonat cu `status="fara restrictii"` și fără împrumuturi întârziate.
        - `CautareIntarziati()` → listă dar nu include abonatul inserat (se verifică `Exists(a => a.Telefon == ...) == false`).
- **CautareDoarIntarziatiNoOverdueReturnsEmptyList**
    - **Scop:** Pentru un abonat cu restricții dar fără întârziere efectivă, `CautareDoarIntarziati()` trebuie să returneze listă goală.
    - **Ce se verifică:**
        - Se inserează un abonat cu `status="cu restrictii"` care nu are împrumuturi.
        - `CautareDoarIntarziati()` → `Count == 0`.

5. Grupurile de teste pentru clasa Server.Server

Pentru `Server.Server`, testele au fost organizate în funcție de fiecare operațiune pe care serverul o primește și o procesează, prin mesaje JSON de tip `Message`. Implementarea serverului grupează cererile pe câmpul `operation`. Următoarele categorii de teste se regăsesc în `UnitTestServerUnitTests`:

5.1. Teste pentru autentificarea și gestionarea sesiunii de angajați

- **LoginValidCredentialsReturnsSuccess**
    - **Operă:** „login”
    - **Date de intrare:**

```
{
 "operation":"login",
 "data":[{"username":"existingUser","password":"correctHash","role":"bibliotecar"}]
}
```

    -
    - **Scop:** Autentificare angajat cu credențiale valide, rol „bibliotecar” pre-inserat (`existingUser`).
    - **Ce se verifică:**
        - Răspunsul serverului conține „Login successful.”.
- **LoginInvalidCredentialsReturnsFailInvalidData**
    - **Operă:** „login”
    - **Date de intrare:** „nonexistent” / „wrong” / „bibliotecar”.
    - **Scop:** Autentificare cu credențiale invalide.
    - **Ce se verifică:**
        - Răspunsul conține „Login failed. Invalid data.”.
- **LoginDuplicateAttemptReturnsFailOnSecond**
    - **Operă:** „login” (trimis de două ori consecutiv, cu aceleași date valabile).
    - **Scop:** Verifică că primul mesaj obține „Login successful.”, iar al doilea (pe aceeași conexiune) obține „Login failed.”.
    - **Ce se verifică:**
        - Răspunsul primului apel conține „Login successful.”.
        - Răspunsul celui de-al doilea apel conține „Login failed.”.

5.2. Teste pentru înregistrarea și autentificarea abonaților

- **RegisterSubscriberNewPhoneReturnsSuccess**
  - **Operă:** „registerSubscriber”
  - **Date de intrare:** abonat cu număr de telefon unic (gen „099900123456”).
  - **Scop:** Înregistrare abonat nou.
  - **Ce se verifică:**
    - Răspunsul conține „Subscriber Register successful.”.
- **RegisterSubscriberDuplicatePhoneReturnsFail**
  - **Operă:** „registerSubscriber”
  - **Date de intrare:** abonat cu telefon deja existent în bază (`0700111222`).
  - **Scop:** Înregistrare abonat cu telefon duplicat.
  - **Ce se verifică:**
    - Răspunsul conține „Subscriber registration failed.”.
- **LoginSubscriberExistingPhoneReturnsSuccessData**
  - **Operă:** „loginSubscriber”
  - **Date de intrare:** telefonul `0700765432`, deja inserat în `[TestInitialize]`.
  - **Scop:** Autentificarea abonatului după telefon.
  - **Ce se verifică:**
    - Răspunsul conține șirul „Subscriber Login successful|<IdAbonat>|<Status>|…”.
- **LoginSubscriberInvalidPhoneReturnsFail**
  - **Operă:** „loginSubscriber”
  - **Date de intrare:** telefon inexistent (`0000000000`).
  - **Scop:** Autentificare eșuată pentru abonat.
  - **Ce se verifică:**
    - Răspunsul conține „Subscriber Login failed”.

5.3. Teste pentru căutarea și afișarea cărților

- **SearchBooksNoMatchReturnsNoBooks**
  - **Operă:** „searchBooks”
  - **Date de intrare:** `{"title":"ZZZNonexistentTitle","author":"Nobody"}`.
  - **Scop:** Căutare fără potriviri.
  - **Ce se verifică:**
    - Răspunsul conține „No books found.”.
- **SearchBooksMatchExistsReturnsList**
  - **Operă:** „searchBooks”
  - **Date de intrare:** `{"title":"Alpha","author":"AuthorMatch"}` (având cel puțin o carte în baza de date cu aceste parțialități).
  - **Scop:** Căutare cu potriviri.
  - **Ce se verifică:**
    - Răspunsul conține caracterul „~” (adică a fost returnat un șir formatat `id~Titlu~Autor|…`).

5.4. Teste pentru împrumuturi (operațiunea „insertLoan”)

- **InsertLoanInvalidIdsReturnsFail**
  - **Operă:** „insertLoan”
  - **Date de intrare:** `{"subscriberId":"-1","bookId":"-1","selectedLocation":"acasa"}`.
  - **Scop:** Încercarea de inserare a unui împrumut cu ID-uri invalide.
  - **Ce se verifică:**
    - Răspunsul conține „Inserted Loan failed.”.
- **InsertLoanValidReturnsSuccess**
  - **Etape:**
    1. **addBook**: se trimite mesaj „addBook” pentru a crea o carte temporară cu `ISBN="TEMPISBN123"`.
       - Răspuns: „Book added successful.”
    2. **registerSubscriber**: se înregistrează un abonat temporar (telefon unic, ex. „099900…”).
       - Răspuns: „Subscriber Register successful.”
    3. **searchBook**: se caută cartea după `isbn="TEMPISBN123"`.
       - Răspuns: `"<bookId>~TempTitle~TempAuthor~TempPub|"` → extragem `bookId`.
    4. **loginSubscriber**: se autentifică abonatul folosind telefonul temporar.
       - Răspuns: `"Subscriber Login successful|<subscriberId>|…"` → extragem `subscriberId`.
    5. **insertLoan**: se trimite `{"subscriberId": "<subscriberId>", "bookId":"<bookId>", "selectedLocation":"acasa"}`.
       - **Scop:** Verifică împrumutul în scenariul normal (abonat și carte valide).
       - **Ce se verifică:** Răspunsul conține „Inserted Loan successful.”.

5.5. Teste pentru obținerea împrumuturilor (operațiunea „getLoans”)

- **GetLoansNoActiveReturnsNoLoans**
  - **Operă:** „getLoans”
  - **Date de intrare:** `{"subscriberId":"-1"}` (abonat inexistent).
  - **Scop:** Când nu există împrumuturi active pentru abonatul specificat.
  - **Ce se verifică:** Răspunsul conține „No loans found.”.

5.6. Teste pentru returnarea unei cărți (operațiunea „returnBook”)

- **ReturnBookNoActiveReturnsFail**
  - **Operă:** „returnBook”
  - **Date de intrare:** `{"subscriberId":"-1","bookId":"-1"}` (împrumut inexistent).
  - **Scop:** Încercarea de returnare când nu există împrumut.
  - **Ce se verifică:** Răspunsul conține „Book return failed.”.

5.7. Teste pentru interogarea statusului clientului (operațiunea „getStatusClient”)

- **GetStatusClientInvalidIdReturnsNotFound**

- **Operă:** „getStatusClient”
- **Date de intrare:** `{"subcriberId":"-1"}`.
- **Scop:** Solicitare status pentru abonat inexistent.
- **Ce se verifică:** Răspunsul conține „Status not found.”.

5.8. Teste pentru gestionarea angajaților prin server (operațiunile „registerEmployee” și „deleteLibrarian”)

- **RegisterEmployeeInvalidRoleReturnsFail**
  - **Operă:** „registerEmployee”
  - **Date de intrare:** `{"username":"tempEmp","password":"pw","role":"nonexistentRole"}`.
  - **Scop:** Înregistrare angajat cu rol inexistent.
  - **Ce se verifică:** Răspunsul conține „Employee registration failed.”.
- **DeleteLibrarianInvalidUserReturnsFail**
  - **Operă:** „deleteLibrarian”
  - **Date de intrare:** `{"username":"noSuchUser"}`.
  - **Scop:** Ștergerea unui bibliotecar care nu există sau este administrator (deci nu poate fi șters).
  - **Ce se verifică:** Răspunsul conține „Librarian deletion failed.”.

5.9. Teste pentru adăugarea și ștergerea cărților prin server (operațiunile „addBook”, „searchBook”, „deleteBook”)

- **AddBookInvalidDataReturnsNullResponse**
  - **Operă:** „addBook”
  - **Date de intrare:** `{"isbn":"INVALIDISBN","title":null,"author":"A","genre":"G","publisher":"P"}`.
  - **Scop:** Inserare carte cu date invalide (`title=null`).
  - **Ce se verifică:**
    - Serverul scrie în consolă „Book addition failed.”, dar **nu trimite nicio linie** către client → din perspectiva testului, `SendSingle(...)` returnează `null`.
- **SearchBookNoMatchReturnsNoBooks**
  - **Operă:** „searchBook”
  - **Date de intrare:** `{"isbn":"0000000000"}` (ISBN inexistent).
  - **Scop:** Căutare carte după ISBN inexistent.
  - **Ce se verifică:** Răspunsul conține „No books found with the given ISBN.”.
- **DeleteBookInvalidIdReturnsFail**
  - **Operă:** „deleteBook”
  - **Date de intrare:** `{"idBook":"-9999"}`.
  - **Scop:** Ștergerea unei cărți cu `id_carte` invalid.
  - **Ce se verifică:** Răspunsul conține „Book deletion failed.”.

5.10. Teste pentru căutarea abonaților cu restricții/întârziere (operațiunea „searchSubscribers”)

- **SearchSubscribersNoneMatchReturnsNoSubscribers**
  - **Operă:** „searchSubscribers”
  - **Date de intrare:** listă goală (nu se transmit date suplimentare).
  - **Scop:** Nu există abonați cu restricții sau întârziere.
  - **Ce se verifică:** Răspunsul conține „No subscribers found with restrictions or blocked.”.

5.11. Teste pentru actualizarea statusului abonatului (operațiunea „updateStatus”)

- **UpdateStatusInvalidSubscriberReturnsFail**
  - **Operă:** „updateStatus”
  - **Date de intrare:** `{"subscriberId":"-1","status":"cu restrictii"}`.
  - **Scop:** Încercarea de actualizare a statusului pentru un abonat inexistent.
  - **Ce se verifică:** Răspunsul conține „Status update failed.”.

6. Concluzii

- Toate funcționalitățile principale (CRUD, autentificare, împrumuturi, căutări) sunt acoperite de cel puțin un test pozitiv („happy path”) și un test negativ (eroare SQL, date invalide sau entități inexistente).
- Testele unitare asigură robustețea codului, validând că în situații de eroare nu există scurgeri de date, tranzacții neterminate sau state inconsistente.
- Prin abordarea black-box (pentru subclass-le private) și simularea traficului client–server (pentru `Server.Server`), s-a asigurat că API-ul expus de librării funcționează conform așteptărilor, iar clienții pot interpreta corect răspunsurile.
- Structura testelor, organizată pe grupuri de funcționalități (Utilizatori, ISBN, Cărți, Abonați, Împrumuturi, Căutări, Server), permite întreținerea ușoară: la adăugarea unei noi metode în DLL, e simplu de identificat în ce categorie va trebui adăugat testul.