**Neighborhood Library Service – Take-Home Test**

---

**Overview**
A small neighborhood library wants a new App to manage its members, books, and lending operations. You must build this service using:

- **Python** for the server implementation
- **REST or gRPC** with **Protocol Buffers** for the service interface
- **PostgreSQL** as the data store

---

**Functional Requirements**

- The library needs to track:
    - **Books** (title, author, etc.)
    - **Members** (name, contact info, etc.)
    - **Borrowing/Returning** operations (who borrowed what, when, etc.)

At a minimum, your App should allow the library staff to:

1. **Create/Update** records for books and members.
2. **Record** when a member borrows a book.
3. **Record** when a borrowed book is returned.
4. **Query** or list borrowed books in some fashion (e.g., all books a certain member has out).

You may include more functionality if you see fit (e.g., handling overdue books, tracking fines, due dates, etc.), but the above points are the core requirements.

---

**Your Task**

1. **Design the Database Schema**

    - Decide on the tables/entities you'll need, how they relate, and what fields/columns belong in each.

    - You'll store this data in PostgreSQL.

2. **Define the gRPC-Web Service(use REST if you are not comfortable with gRPC web)**

    - Create one or more `.proto` files with the necessary Protobuf message types and service definitions.

    - Determine which RPC methods (and their request/response messages) are needed to support the core library operations.

3. **Implement the Python gRPC or REST API Server**

    - Write Python code to spin up an API server.

    - In each API method, interact with your PostgreSQL database (create/read/update records, etc.).

4. **Documentation/Delivery**

    - Provide a short **README** detailing how to:

        - Set up the database (scripts or Docker instructions, etc.).

        - Generate or compile the `.proto` files (if needed).

        - Run the Python server (command-line steps, required libraries, environment variables).

    - Include any necessary files or folders so another developer can easily build, run, and test your service.

5. **Minimal Frontend**

    - Provide web frontend by using React (next.js preferably).

---

**Evaluation Criteria**

1. **Schema Design**
    - Clarity, normalization, and use of relationships in PostgreSQL.
2. **Service Interface**
    - How intuitive and well-structured your REST / gRPC  endpoints and Protobuf messages are.
3. **Code Quality**
    - Organization, readability, and maintainability of your code.
4. **Documentation**
    - Ease of setup and clear explanation of how to test the service.

---

**Tip (Optional)**

- Consider adding **error handling** for cases like trying to borrow a book that's already checked out.
- Implementing **validation** (e.g., ensuring valid inputs for new records).
- Show how a client might call your service (e.g., providing a sample script).

Good luck!