Software Developer Assignment

Objective

Develop a full-fledged CRUD application that manages a small library of books. The frontend should be built using React/Next.js and TypeScript, and the backend should use Golang with a database of your choice.

Frontend (React/Next.js & TypeScript)

User Interface

- Implement a dashboard that lists all books, with options to add, edit, view in detail, and delete books.
- Create forms for adding new books and editing existing ones.
- Each book entry should display basic information such as title, author, and year of publication.
- Use modal dialogs for form submissions to enhance user experience.

State Management

- Utilize Context API to manage state across components.
- Form Handling
- Include client-side validation with visual feedback for required fields.
- Use controlled components for form inputs to handle form data.

Routing

- Set up dynamic routing for viewing individual book details.

Error Handling

- Present user-friendly error messages for network issues and form errors.

Backend (Golang)

The backend consists of two parts that need to be implemented together no need to create different repositories for this.

Part 1 - RESTful API

Develop endpoints for managing books

- GET /books – Retrieve all books.
- POST /books – Add a new book.
- GET /books/{id} – Get a single book by ID.
- PUT /books/{id} – Update a book by ID.
- DELETE /books/{id} – Delete a book by ID.

Database Integration (Database of your choice)

- Integrate any database of your choice for data persistence.
- Implement queries within the Golang application to interact with the database.

Validation

- Enforce backend validations ensuring that all fields meet certain criteria before being processed (e.g., the title is not empty).

Logging

- Implement detailed logging, especially for API requests and error handling.

Part 2 - URL Cleanup and Redirection Service

Objective

Refine a backend service that performs URL cleanup and redirection based on specific rules and operation types. The service should accept JSON input containing the original URL and the designated operation type, process the URL accordingly, and return the cleaned or redirected URL in JSON format.

Task Description

Your task is to implement an HTTP POST endpoint that receives a JSON object containing a URL string and an operation type. This URL will be processed by your service according to the defined criteria and the specified operation:

1. **Operation Types:**
    a. redirection: Modify the URL for redirection purposes.
    b. canonical: Clean up the URL to its canonical form.
    c. all: Both of the types above.

2. **Basic Requirements:**
    a. For a canonical URL operation, the service should process the request by cleaning up the URL. This involves removing query parameters and trailing slashes.
    b. For redirection, in addition to ensuring the domain is www.byfood.com, convert the entire URL to lowercase.
    c. For the third option it should conduct both of the above requirements and return the result.

3. **Input/Output Format:**
    a. **Input:** A JSON object with keys named url and operation, where url points to the string that needs to be processed and the operation defines the type of processing to be applied.
    b. **Output:** A JSON object with a key named processed_url which has the processed URL string as its value.

Implementation Details:

- Validate the input rigorously and handle any errors with appropriate messages.
- Accommodate both operation types within the same endpoint, distinguishing behavior by reading the operation key.
- Include instructions on how to initialize and run your service, and execute the provided test cases.

Example request and response

Example 1:
Request:

```
Unset
{
  "url": "https://BYFOOD.com/food-EXPeriences?query=abc/",
  "operation": "all"
}
```

Expected Response:

```
Unset
{
  "processed_url": "https://www.byfood.com/food-experiences"
}
```

Example 2:
Request:

```
Unset
{
  "url": "https://BYFOOD.com/food-EXPeriences?query=abc/",
  "operation": "canonical"
}
```

Expected Response:

```
Unset
{
  "processed_url": "https://BYFOOD.com/food-EXPeriences"
}
```

The following requirements are for both backend parts:

API Documentation

- Document each endpoint using tools like Swagger to provide an interactive API reference.

Deliverables

- The codebase must be hosted in a version control system with a clear commit history.
- A comprehensive README.md file detailing setup instructions, project structure, endpoint usage, and how to run tests.
- Screenshots of the working application should be included in the documentation.
- All installation and config scripts should ensure smooth local setup for development and testing purposes.
- Ensure comprehensive unit tests for APIs, focusing on different edge cases, and providing test cases for the operations, by showing request and response bodies.

Good luck and don’t hesitate to show off your skills!