Problem Statement:

Write a Go program that implements a simple RESTful API for managing bank accounts and their transactions. The API should support the following operations:

Entities:

1. **Account:**
    - **ID:** Unique identifier for the account.
    - **Owner:** Name of the account holder.
    - **Balance:** Current balance of the account.
2. **Transaction:**
    - **ID:** Unique identifier for the transaction.
    - **AccountID:** Identifier of the account related to the transaction.
    - **Type:** Type of transaction (`deposit` or `withdrawal`).
    - **Amount:** Amount involved in the transaction.
    - **Timestamp:** Date and time when the transaction was made.

API Operations:

1. **Create a New Account**
    - **Endpoint:** `POST /accounts`
    - **Description:** Create a new bank account with an initial balance.
    - **Request Body:** JSON containing `owner` and `initial_balance`.
2. **Retrieve Account Details**
    - **Endpoint:** `GET /accounts/{id}`
    - **Description:** Retrieve details of a specific account by ID.
3. **List All Accounts**
    - **Endpoint:** `GET /accounts`
    - **Description:** Retrieve a list of all bank accounts.
4. **Create a Transaction**
    - **Endpoint:** `POST /accounts/{id}/transactions`
    - **Description:** Create a deposit or withdrawal transaction for a specific account.
    - **Request Body:** JSON containing `type` (`deposit` or `withdrawal`) and `amount`.
5. **Retrieve Transactions for an Account**
    - **Endpoint:** `GET /accounts/{id}/transactions`
    - **Description:** Retrieve all transactions associated with a specific account.
6. **Transfer Between Accounts**
    - **Endpoint:** `POST /transfer`
    - **Description:** Transfer funds from one account to another.
    - **Request Body:** JSON containing `from_account_id`, `to_account_id`, and `amount`.

Requirements:

- **HTTP Methods:** Use appropriate HTTP methods (GET, POST).
- **Data Format:** JSON for request and response bodies.
- **Concurrency:** Handle concurrent transactions safely to maintain data integrity.
- **Error Handling:** Gracefully handle errors such as insufficient funds, invalid account IDs, and invalid transaction types.
- **Code Quality:** Write clean, well-structured, and maintainable code.
- **Persistence:** In-memory storage is sufficient; a database is not required.
- **Instructions:** Provide clear instructions on how to run and test the application.