Task

1. Create a private GitHub repository. This is where all the code will reside. Keep the name of your repo as iitk-coin. Add me (bhuvansingla) as a collaborator. Add an entry to this sheet: https://bit.ly/3vvoYLh.

2. Set up your local Go project. You must have read that in Go conventionally you keep all the code at $GOPATH/src. You should be following that convention. Ideally, your code will be residing at $GOPATH/src/github.com/yourusername/iitk-coin/. Initialize your Go module.

3. Write a program that connects to a database. The database that we will be working with is SQLite. On executing the program it should create a new table (say, User with two fields rollno and name). Create a function that takes in new user details as arguments and adds it to the database. Your program should not take any input from command line. Aim to write clean and structured code.

Now we'll start working on the coin related endpoints:

Task

1. Create an endpoint that accepts a POST request and awards coins to a user. The body will have the rollno of the user and the number of coins to be given.

2. Create an endpoint that accepts a POST request to transfer coins between two users. The body will have the rollnos of the particpating users and the number of coins to transfer.

3. Create an endpoint that accepts a GET request and returns the coin balance of a user. The body will have the roll no of the user.

Notes

1. The server handles API requests concurrently by design. By no way should your endpoints create or destroy coins when it is not intended. You need to take care that all the steps of your transactions either complete successfully or don't happen at all.

2. Make sure that there is no such possible interleaving between two concurrent transactions that can cause unwanted behavior. To simulate and test different interleavings you can make use of sleep timers between lines of your code.

3. Some of the endpoints that you will be creating would not be for all users but just admins, but you can ignore that for now. We'll take up permission levels a bit later. You can keep these APIs public for now i.e. no authorization required.

4. Take care of as many edge cases as you can. Also, handle errors everywhere. Feel free to ask questions (but only after you have Googled yourself before).

Now we'll start working on the coin related endpoints:

Task

1. Create an endpoint that accepts a POST request and awards coins to a user. The body will have the rollno of the user and the number of coins to be given.

2. Create an endpoint that accepts a POST request to transfer coins between two users. The body will have the rollnos of the particpating users and the number of coins to transfer.

3. Create an endpoint that accepts a GET request and returns the coin balance of a user. The body will have the roll no of the user.

Notes

1. The server handles API requests concurrently by design. By no way should your endpoints create or destroy coins when it is not intended. You need to take care that all the steps of your transactions either complete successfully or don't happen at all.

2. Make sure that there is no such possible interleaving between two concurrent transactions that can cause unwanted behavior. To simulate and test different interleavings you can make use of sleep timers between lines of your code.

3. Some of the endpoints that you will be creating would not be for all users but just admins, but you can ignore that for now. We'll take up permission levels a bit later. You can keep these APIs public for now i.e. no authorization required.

4. Take care of as many edge cases as you can. Also, handle errors everywhere. Feel free to ask questions (but only after you have Googled yourself before).