Bài tập Module Programmer's Way

Người thực hiện: Nguyễn Đại Nghĩa

Yêu cầu

Mô phỏng tính năng gửi tiền tiết kiệm

Mô tả chi tiết:

- Giả sử user đã được verify trước đó (là user của ZLP)
- Kyc level của user có giá trị 1, 2, 3
- User có kyc level từ 2 trở lên được sử dụng tính năng này
- Nếu user rút tiền trước hạn, thì số tiền rút đi sẽ tính lãi là 0.1%/năm, số tiền còn lại tính theo mức lãi suất khi người dùng gửi
- Hạn mức cho kyc level là 50tr, kyc level 3 là 100tr
- Mức lãi suất như sau:

kyc level | kỳ hạn gửi | lãi suất

2 | 21 ngày | 3%/năm

2 | 3 tháng | 4%/năm

2 | 6 tháng | 5%/năm

2 | 12 tháng | 6%/năm

3 | 21 ngày | 3.5%/năm

3 | 3 tháng | 4.5%/năm

3 | 6 tháng | 5.5%/năm

3 | 12 tháng | 6.5%/năm

Yêu cầu:

- Mô phỏng phần xử lý ở server, demo bằng console đơn giản
- Vẽ siquence diagram cho các hàm xử lý
- Viết unit test cho các hàm xử lý
- Giải thích các design pattern và các nguyên tắc được áp dụng nếu có

Source code bài làm và Sequence diagram

Đường dẫn repo: Bài nộp

Ngôn ngữ và công nghệ sử dụng

Ngôn ngữ sử dụng: Go

API viết bằng gRPC, sử dụng protobuf

Thư viện và công cụ hỗ trợ viết unit tests: testing, gomock, mockgen

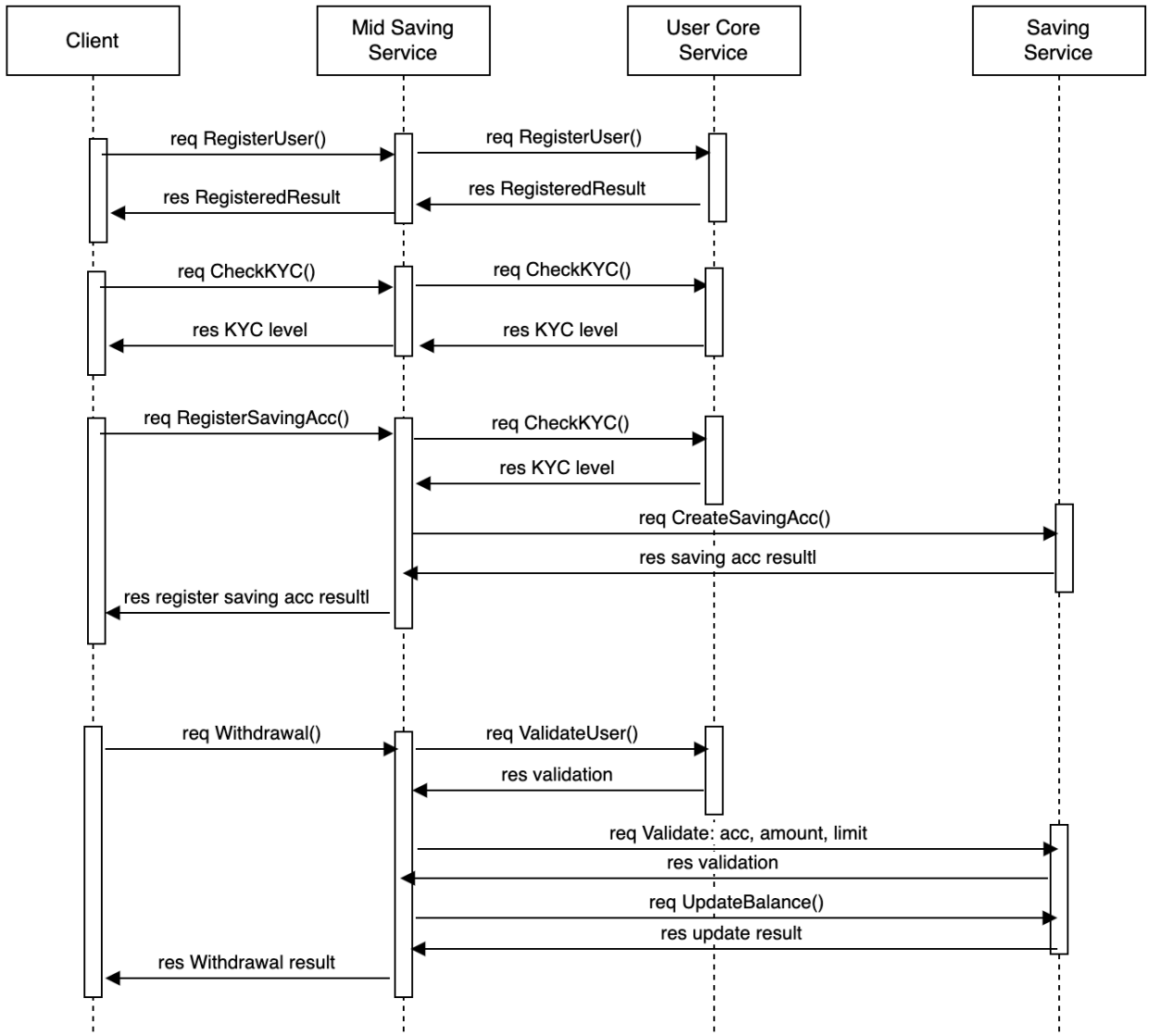
Mô phỏng hệ thống

- Hệ thống gồm **client** và 3 services là **mid_saving**, **user_core** và **saving_core**. Hiện tại thực hiện demo đơn giản nên sử dụng localhost
- Client sẽ gửi tất cả requests liên quan đến mid_saving.
- Mid_saving xử lý các logic, gọi các services user_core và saving_core để lấy các thông tin cần thiết. Cuối cùng trả respones về cho client
- Các services user_core và saving_core quản lý dữ liệu, thực hiện các truy vấn liên quan đọc, ghi, cập nhật, xóa dữ liệu, trả về kết quả cho mid_saving

Các chức năng mà client có thể sử dụng gồm:

- Đăng ký người dùng: RegisterUser
- Kiểm tra KYC level: GetCurrentKYC
- Mở tài khoản tiết kiệm: OpenSavingAccount
- Truy vấn thông tin tài khoản: AccountInquiry
- Rút tiền: Withdrawal

Sequence Diagram



Định nghĩa các message và service

Giữa client và mid_saving (chi tiết được trình bày trong source code): `service MidSavingService { rpc RegisterUser(RegisterUserRequest) returns (RegisterUserResponse); rpc GetCurrentKYC(GetCurrentKYCRequest) returns (GetCurrentKYCResponse); rpc OpenSavingsAccount(OpenSavingsAccountRequest) returns (OpenSavingsAccountResponse); rpc Withdrawal(WithdrawalRequest) returns (WithdrawalResponse); rpc AccountInquiry(AccountInquiryRequest) returns (SavingAccount); }`

Giữa mid_saving user_core: `service UserService { rpc RegisterUser(RegisterUserRequest) returns (RegisterUserResponse); rpc GetCurrentKYC(GetCurrentKYCRequest) returns (GetCurrentKYCResponse); }`

Giữa mid_saving và saving_core: `service SavingsService { rpc OpenSavingsAccount(SavingAccount) returns (SavingAccount); rpc AccountInquiry(AccountInquiryRequest) returns (SavingAccount); rpc UpdateBalance(WithdrawalRequest) returns (SavingAccount); }`

Business logic và các hàm phụ trợ

Các hàm xử lý hiện tại được gọi ở mid_saving: Các hàm xử lý về thời gian, tính lãi suất,... được triển khai trong file helper.go

Viết Unit Test

Sử dụng thư viện **testing** để viết unit test

- Unit test cho tất cả các hàm xử lý quan trọng của các file **helper.go** và **interest_strategy.go** đã được viết và PASS.
- Unit test cho các grpc method ở các files **mid_saving_logic.go**, **user.go**, **saving.go** cần sử dụng mock để giả định kết quả trả về qua gRPC API, so sánh với kết quả kỳ vọng:
    - Sử dụng thư viện gomock
    - Sử dụng tool mockgen để generate template mock cho các interfaces cần thiết

Mẫu thiết kế hướng đối tượng (Design Pattern) được sử dụng

Trong yêu cầu đề bài, khi thực hiện rút tiền, tùy vào thời gian đáo hạn ước tính (dueDate) so với thời điểm rút tiền mà cách tính lãi sẽ thay đổi, dựa vào lãi suất kỳ hạn lúc gửi tiền hay dựa vào lãi suất không kỳ hạn (0.01). Việc này phù hợp với việc sử dụng **Strategy Pattern**.

Thực hiện:

- Tạo interface IInterestCalculator chứa method CalculateInterest().
- Cài đặt 2 concrete class tương ứng với tính lãi đúng hạn OnTimeInterestCalculator và tính lãi sớm EarlyInterestCalculator, 2 classes này implement phương thức CalculateInterest(), cài đặt phù hợp với mức lãi và cách tính đã ràng buộc.
- Cuối dùng, thêm 1 field Calculator có kiểu IInterestCalculator vào trong struct của đối được tài khoản tiết kiệm SavingAccountPT. field này có thể được gán để giữ tham chiếm tới 1 trong hai đối tượng tính toán lãi, thay đổi linh hoạt dựa vào ngày tính lãi là sớm hay đúng đáo hạn