Go language & Concurrent computing

ESTIAM Metz
5 September 2022

Lehajam Boujemaoui

Blockchain Backend Project

Suppose that we're building a backend web service that will allow mobile games's client to perform some operations on the blockchain, such as creating wallets or transfering tokens between wallets, etc.

A wallet is what allows users to interact with a blockchain, we call this paradigm web3 or decentralised internet:

- More info on web3 here: https://ethereum.org/en/web3/

In this project, you only need to implement 1 single **HTTP** API that will receive and process a request to create a new player account together with his/her blockchain wallet at once.

Inputs

1. The player must provide a **username**, **password**, and a **pin code** to protect his/her blockchain wallet.

- The player's **username** must be unique.
- The **username** must be a string of 3-100 characters, and must contain only lowercase English letters (a-z), digits (0-9) or underscore (_).
- The **password** must be a string of 6-32 characters.
- And **pin code** must be a string of exactly 6 digits (0-9).

API

2. The API should call an external service to create a wallet for the player on the blockchain.

- Note that we're not giving you a real service in this project, so you will have to mock it in your implementation.
- You don't have to call a real Wallet service
- Basically, you will have to send a HTTP **POST** request to this endpoint: **https://{BLOCKCHAIN_SERVICE_URL}/wallets/create**

API

- It expects a **JSON** body with these parameters:

```
{
    "blockchain": "ethereum", // the name of the blockchain, let's say we're only using ethereum for now
    "pin_code": "" // the player's provided pin code
}
```

- In case of success, the service will respond **200 OK** status code with this **JSON** body:

```
{
    "wallet_address": "", // the address of the created wallet on the blockchain
    "currency_code": "", // the currency of the blockchain, such as ETH
    "currency_balance": "" // the currency balance of the wallet
}
```

API

- In case of failure, the service will respond other status codes with an error message in its body:

```
{
    "error": "", // the error message
}
```

Storage

3. If the wallet is successfully created on the blockchain, its information should be stored in a **wallets** table in our database, and the player's account should be created and stored in another **players** table at the same time.

- Use json files for your database
- You should think of a way to design these 2 tables so that we can easily figure out which wallet belongs to which player. Let's say, to make it simple, we only allow 1 player to have exactly 1 blockchain wallet.
- You must make sure that there should be no wallets created without a player, or a player created without wallet. This is very important to ensure the consistency of our database.

Outputs

4. Once the **player** and **wallet** records have been successfully stored in the database, the API should send a successful response to the client with all the necessary information about the player and wallet.

- It's up to you to decide the schema of the database, and the structure of the request/response.
- We expect to see a good implementation with a well-written set of unit tests.
- You should include some performance benchmark for your API

Extra - This is optional for extra points

- You can use **PostgreSQL** for the database
- You can use goroutine in your app to increase performances. For example to process client requests, write to database, etc.
- You can serve requests with either **gRPC** or **HTTP**, or both (on 2 different ports).
- It would be great if there's a swagger documentation of the service as well.
- We also want to be able to run your code with 1 single **docker compose up** command.

Thank you

Lehajam Boujemaoui
lehajam@gmail.com (mailto:lehajam@gmail.com)
https://github.com/lehajam (https://github.com/lehajam)