**CASE 1 (Implemented approach to satisfy acceptance criteria)**:

Constraints:

1. Availability: The API should resolve under 100ms per request.
2. Consistency: The data returned from the api should be accurate (strictly consistent).

**Approach**:

1. Implemented a new POST API GetBalancesBatch which accepts a list of safe addresses and returns a list of token balances for the given safe addresses.
   1. Configured as a POST call as input safe addresses can be huge.
2. In GetBalancesBatch API, We loop through each address and check whether the safe address is available in the redis.
   1. If it is available then we fetch the data from redis and respond back to the user. Also, we increase the redis timeout to another 15 min for the current address.
   2. If it is not available then we make an async call to the main API /safes/{address}/balances/usd to fetch the balances. And store that value in the redis with the timeout 15 min.
   3. Here, the 15 min timeout is static. We can configure it based on the use case.
   4. For all the addresses which are not available in the redis, we are making parallel async calls to /safes/{address}/balances/usd to reduce the response time.
3. There will be a separate server running which gets all the available addresses from the redis and continuously polls the /safes/{address}/balances/usd API for each safe address to fetch the latest results.

**Pros**:

1. As we are fetching the balances from redis and avoiding server calls, we get the response time within 100ms for GetBalancesBatch API.
2. As we are continuously polling and having latest balances in the redis, the data will be strictly consistent.

**Cons**:

1. There will be a server overload due to continuous polling.
2. This approach is not suitable for the real world as we cannot accommodate server load to get high availability and consistency.

**Improvisations**:

1. We can find a pattern of addresses which are getting requested frequently and adjust the redis timeout accordingly.
2. We can analyse the responses and based on that, we can build ML models to know how frequently the DB gets updated. If we know how frequently DB gets updated, then we can improvise the approach by excluding the polling and setting the specified Redis timeout. Or we can schedule polls accordingly.

**Sample Input**:

{

"addresses": [

"0x5e14ed9dCeE22ba758E8de482301028b261c4a14",

"0xF1D8c2eED95D5fC2EaDe4E6Bb15a5969453E89a9"

]

}

**Sample Output**:

[

{

"tokenAddress": null,

"token": null,

"balance": "503768158000000000",

"ethValue": "1.0",

"timestamp": "2022-01-06T16:13:22.183324Z",

"fiatBalance": "1712.5343",

"fiatConversion": "3399.4493",

"fiatCode": "USD"

},

{

"tokenAddress": "0xeC5483804e637D45cDe22Fa0869656B64b5AB1ab",

"token": {

"name": "ACENT",

"symbol": "ACE",

"decimals": 18,

"logoUri": "https://gnosis-safe-token-logos.s3.amazonaws.com/0xeC5483804e637D45cDe22Fa0869656B64b5AB1ab.png"

},

"balance": "45000000000000000000",

"ethValue": "0.0",

"timestamp": "2022-01-06T16:13:22.188070Z",

"fiatBalance": "0.0",

"fiatConversion": "0.0",

"fiatCode": "USD"

}

]

**CASE 2**:

Constraints:

1. Availability: The API should resolve under 100ms per request.
2. Consistency: Eventual consistency or when we know how frequently data gets updated.

Approach:

1. In GetBalancesBatch API, for the first request, we call the main /safes/{address}/balances/usd API and cache the balances in the redis with the static timeout (it can be defined based on DB updates frequency).
2. If the request repeats again, then we will just lookup in the redis instead of making calls to the main API. So that we can achieve 100ms response time. But the data will be eventually consistent.
3. If we know the exact DB updates frequency, then it will be strictly consistent too.

**CASE 3**:

Constraints:

1. Availability: If the no. of hits for the API is huge and to reduce the load on the server.
2. Consistency: The data returned from the api should be accurate (strictly consistent).

Approach:

1. Implement request collapsing or request caching to make minimal calls to the server.
2. Data will be strictly consistent but response time will be single request response time i.e 250ms - 750 ms.
3. This approach is applicable when we need to reduce the load on the server. for example: Implementing Cricbuzz.

## **Questions**

Answer these questions specifically for the above use case (i.e. fetching safe balances)

1. What are the tradeoffs when it comes to optimizing for availability vs consistency?
   1. To fetch balances for 1 or more safe addresses, we can make async calls parallelly to reduce the response time. But the response time will be at least 1 request time i.e 250ms - 750 ms.
   2. If we want strong consistency then we have to call the main /safes/{address}/balances/usd API always to fetch the latest data (assuming we don’t have control over the server and don’t know when the DB gets updated). So, this will not reduce the response time.
   3. If we want response time within 100ms then we should maintain cache. And respond with the cache data instead of making calls to the server every time. So, data will be eventually consistent.
   4. So, overall we can have either high availability or high consistency in this use case but not both.
   5. If we want to have both, then we should follow the polling approach. But this will create a lot of load on the server. So, it’s not an ideal approach for a real world project.
2. In a real world scenario, if you only had to choose between availability and consistency, which criteria would you pick and why?
   1. We should optimize both availability and consistency. But based on the use case, we can do more optimization in either of them. We cannot choose between availability and consistency. This will be purely dependent on the use case. If my use case is to provide high consistency all the time, then I will make calls to the main API everytime. If my use case is to provide high availability, then will go with the eventual consistency.
   2. If I can get more details from the server or I have some control on it, then we can optimize both availability and consistency. For example, by listening to the DB updates or using sockets.
3. Is it possible to get both - a highly available and highly consistent data all the time? If not, then why? If yes, then how?
   1. It is possible to get highly available and highly consistent data but not all the time. There can be use cases where we should optimize one thing more than the other.
   2. For example, high consistency is not needed in calculating the number of likes/views in youtube videos. But if we try to optimize that then it may impact the response time.
   3. The given use case is also an example where we need to optimize one more than the other.