GOHIGHLEVEL - Backend Dev HIRING CHALLENGE 2

Problem Statement:

Design and Implement a backend service for Wallet system supporting

- Setup wallet
- Credit / Debit transactions
- Fetching transactions on wallet
- Get wallet details

Description:

Create the following APIs for implementing a wallet system.

1. **Initialise wallet**
    1. Setup a new wallet with initial balance.
    2. API endpoint → /setup
    3.

Request [POST]	Response

{ balance, name } e.g. { balance: 20, name: ‘Hello world’ }	Status 200, Response body: { id, balance, transactionId: ‘4349349843’, name: ‘Hello world’, date: <JS Date obj>}

4. Requested balance can be decimal upto 4 precision points. E.g. 20.5612

2. **Credit/Debit amount**

    1. This API should credit/debit the requested amount to the wallet.

    2. API endpoint → /transact/:walletId

    3.

Request [POST]	Response

Param: walletId → id of wallet
Body: { amount, description }
e.g. { amount: 10, description: ‘Recharge’ }
	Status: 200, Response body:
	{ balance: 30, transactionId: ‘8328832323’ }

4. For Credit the amount will be a positive number, for Debit it will be a negative number.

5. Amount can be decimal upto 4 precision points e.g 4.1203, 0.321, 1.0045

3. Fetch transactions

   1. Given the wallet id, fetch the recent transactions on it.
   2. API endpoint → /transactions?walletId={walletId}&skip={skip}&limit={limit}
   3.

Request [GET]	Response
Query: {walletId, skip, limit} E.g walletId='2434343'&skip=10&limit=25	Status: 200, Response body: [{ id, walletId: string, amount: number, balance: number, description: string, date: <JS Date obj>, type: 'CREDIT'/'DEBIT' }, …...]

4. The response for this API should be an array of transactions where each transaction object consists of mentioned properties.

    1. _id: Transaction id

    2. walletId: Id of wallet

    3. amount: Transaction amount

    4. balance: Balance of wallet after transaction

    5. date: Timestamp of transaction

4. **Get wallet**

    1. Given wallet id, fetch the wallet details.

    2. API endpoint → /wallet/:id

    3.

Request [GET]	Response

Params: id	Status: 200, Response body:
	{
	  id,
	  balance,
	  name,
	  date
	}

Example:

Consider the following set of operations that can be operated for the task.

1. POST /setup { balance: 10, name: ‘Wallet A’ }.
    1. New wallet is created with balance as 10 and name as Wallet A
    2. Response: { id: ‘1243434’, balance: 10, name: ‘Wallet A’, date: <JS Date obj> }
2. POST /transact/1243434 --> { amount: 2.4, description: ‘Recharge’’}
    1. Wallet should be credited 2.4.
    2. New transaction doc should be created with following details

        1. id - system generated id
        2. walletId - id of wallet
        3. amount - transaction amount
        4. description - description for transaction
        5. balance - balance of wallet after transaction
    3. New wallet balance: 12.4

3. GET /transactions?walletId=1243434&skip=0&limit=10
    1. Returns following response

```
[
  {
              id: ‘343434’,
              walletId: ‘1243434’,
              amount: 2.4,
              balance: 12.4,
              description: ‘Recharge’,
              date,
              type: ‘CREDIT’
        },
                {
                    id: ‘544521”,
                      walletId: ‘1243434’,
                      amount: 10,
                      balance: 10,
                      description: ‘Setup’,
                      date,
                      type: ‘CREDIT’
                }
            ]
```

4. GET /wallet/1243434

    1. Returns

```
{
  id:’1243434’,
  balance: 12.4,

      name: ‘Wallet A’
      date
     }
```

Optional UI:

Create a simple 2 page web app to show wallet balance and transactions.

1. In page 1, initially if no wallet is configured, show a simple input field asking for User name, optional initial balance field and Submit button.
   When the user clicks the button, use the data provided to initialise a new wallet.

2. Save the wallet id in local storage which can be used next time if the user revisits the page.
   Use the wallet id to show the wallet balance and name.

3. Once we have the wallet initialized, show a section for doing transactions

   1. An input field for transaction amount

   2. A toggle for indicating whether its a CREDIT or DEBIT transaction

   3. And a submit button to execute the transaction

   4. Once the transaction is completed, the wallet balance should automatically update to reflect the transaction

4. Show a link in page 1 to visit page 2, Wallet transactions

5. In page 2, show a table of all the transactions available for the wallet

6. The transaction table should support following

    1. Pagination of data

    2. Ability to sort transactions by date and amount

    3. Export CSV file of all transactions available

Languages and Databases:

As we use Node.JS primarily for our backend services, we would prefer if you used the same.
For the database, you can use as per your preference but following are our recommendations with priority.

- MongoDB

- MySQL

- Firebase

For the frontend you can use any popular JS frameworks or none at all. Feel free to use any JS/CSS libraries as you seem fit.

Notes:

1. All ids to be system generated.
2. Date refers to JS Date object
3. For transaction amount and wallet balance, use 4 decimal places as precision and keep in mind while doing floating point operations
4. Design your transactions while keeping race conditions in check on wallet document
5. Please follow the exact URL structure and request, response params naming convention.
6. Publish the code in github and maintain the repo as **private.**
    1. Provide README for APIs and implementation.
7. Deploy your code in any online hosting services and mention the URL for the service.
    1. Please don’t include auth on your hosted APIs
8. Prepare a short screen recording to explain the task and code implementation.
9. Add following username in your github repo as collaborators:

1. dev-highlevel