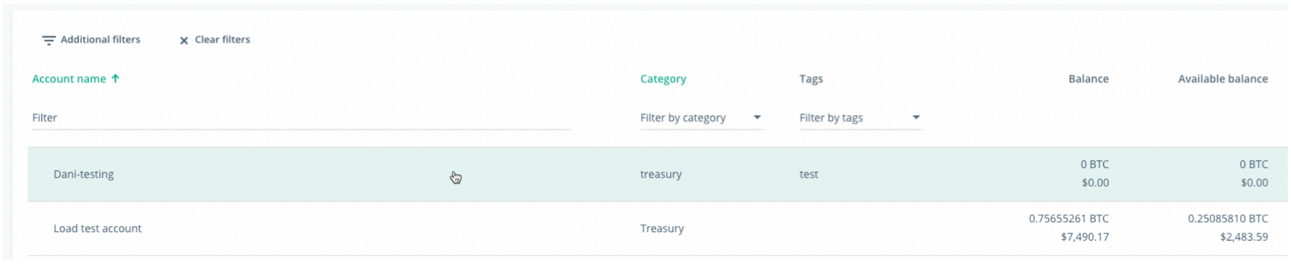
<u>Senior developer test</u>

The following test has been put together to try and test your skills has a SENIOR developer. These are some of the tasks that have already been done in our team.

Our stack is Angular and NestJS, a NodeJS framework (https://angular.io/ & https://nestjs.com/). It's highly recommended that you try to resolve the test using these technologies but if you can't, please, change Angular for React and NestJS for pure NodeJS with express – for example. Although always try to use Typescript over Javascript – our language of choice.

Part 1

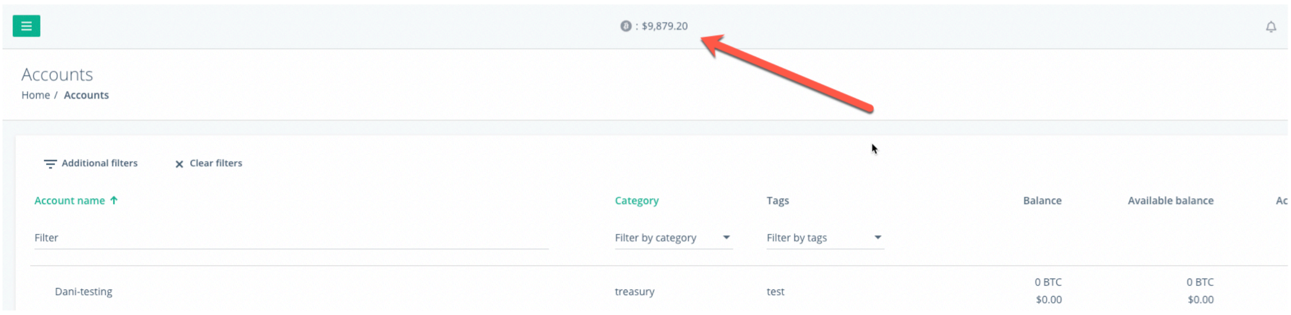
Create a web application that displays a list of accounts showing the balance and available balance in both BTC (bitcoin) and the equivalent in Dollars. There is an example below, please ignore the filtering. The column names are *Account Name, Category, Tag, Balance and Available balance*. *Category and Tag* are not important and return something at random. *Balance and Available Balance* store only BTC (Bitcoin), the dollar amount (displayed below the BTC) should be calculated using an exchange rate. You should return a minimum of 15 accounts (rows) so we have data to view.



Account name ↑	Category	Tags	Balance	Available balance
Filter	Filter by category	Filter by tags		
Dani-testing	treasury	test	0 BTC $0.00	0 BTC $0.00
Load test account	Treasury		0.75655261 BTC $7,490.17	0.25085810 BTC $2,483.59

Part 2

If you noticed above in the mockup, the balance and available balance are displayed both in BTC and Dollars. Your dataset only contains BTC, the value of the Dollar is calculated using an exchange rate. You should display an exchange rate on the screen, for example



Account name ↑	Category	Tags	Balance	Available balance	Ac
Filter	Filter by category	Filter by tags			
Dani-testing	treasury	test	0 BTC $0.00	0 BTC $0.00	

The backend (NestJS / NodeJS) should return the current exchange rate when the web page is first loaded. This is the value you should use when calculating the Balance and Available Balance. Simple – right ?

Now, lets make it more interesting, lets automatically push a new exchange rate every 15 seconds to the frontend (angular / react) from the backend (NestJS / NodeJS) using websockets – you can invent this exchange, it does not need to be real!

BONUS POINTS: NestJS has the ability to work directly with socket.io, otherwise if you have not used NestJS then use socket.io directly inside of NodeJS. As new values arrive at the frontend then the UI should update the exchange rate displayed at the top of the page (see above) and also re-calculate all balance and available balances inside the table.

Let’s also send, at a random interval, the BTC price for each row. The values should be random. The background color of the row should flash the following colors:

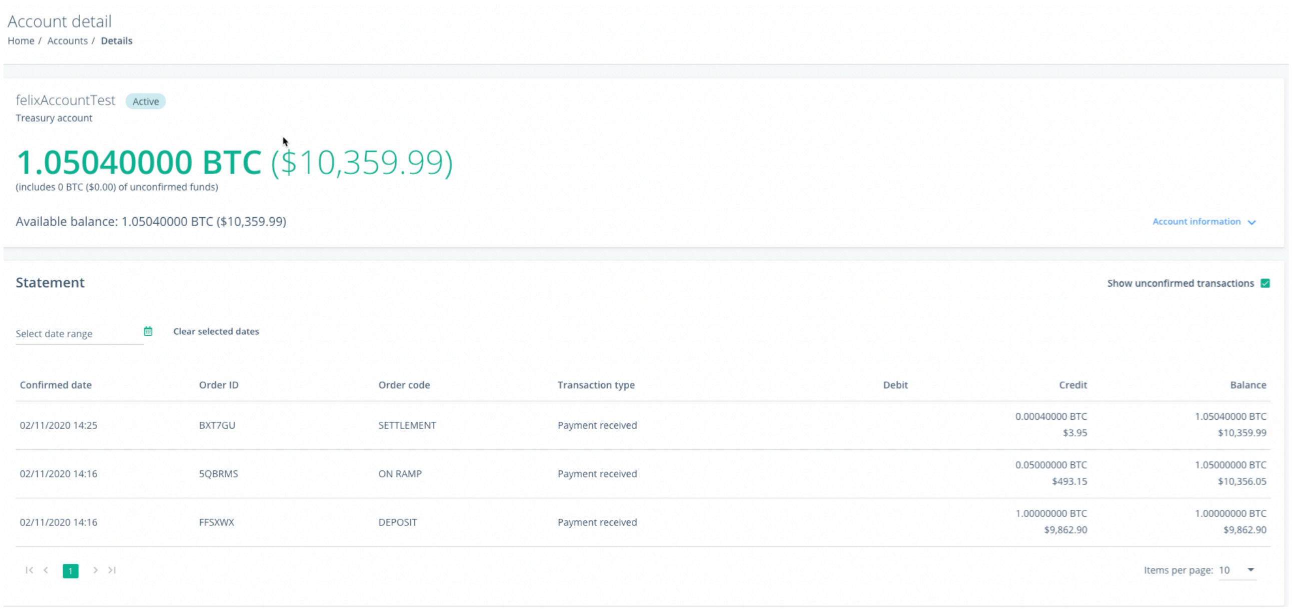
Red: The BTC price in Dollars is lower than the previous value

Green: The BTC price in Dollars is higher than the previous value

If the value does NOT change then do nothing with the background color.

Part 3

Finally clicking on a row should open a detailed page (keeping within the Single Page application) – the detailed page should show some transactions that belong to the specific account. Here is an example.



Account detail

Home / Accounts / Details

felixAccountTest Active

Treasury account

1.05040000 BTC ($10,359.99)

(includes 0 BTC ($0.00) of unconfirmed funds)

Available balance: 1.05040000 BTC ($10,359.99)

Statement

Show unconfirmed transactions ✓

Confirmed date	Order ID	Order code	Transaction type	Debit	Credit	Balance
02/11/2020 14:25	BXT7GU	SETTLEMENT	Payment received		0.00040000 BTC $3.95	1.05040000 BTC $10,359.99
02/11/2020 14:16	5QBRMS	ON RAMP	Payment received		0.05000000 BTC $493.15	1.05000000 BTC $10,356.05
02/11/2020 14:16	FFSXWX	DEPOSIT	Payment received		1.00000000 BTC $9,862.90	1.00000000 BTC $9,862.90

The transactions table should show an Order Code and/or Order Id – these should be unique values. Use the example mock above. The transaction should be related to the account – i.e. do not show transactions that belong to one account in another. This is a pure Master / Detail implementation. Master being the account list and details being the information of the account the user clicked on.

If you notice the debit / credit and balance fields above show BTC and Dollar – the transactions that you have are all in BTC so you must calculate the value of the Dollar amount – as you did before for the account page – exactly the same.
You could also flash the row RED / GREEN as you did in the account page, this shows good re-use of existing code. You should only flash RED / GREEN when new BTC prices arrives which you have already done above.

Recommendations

Again it's highly recommended to use our stack technologies: Angular and NestJS.

You are free to use what you want for your datasource, an in-memory collection or even better something like Mongo. Hint: NestJS supports MongoDB out of the box – the choice is yours! If you use MongoDB then you should provide us with a seeding script.

You should try and make use of good styling, using something like material design (angular material).

You should make use of dependency injection where available i.e. Angular and NestJS.

The Typescript <any> type makes us sad, please avoid it and always create types where you can. Feel free to make good use of OOP. The use of abstract classes, interfaces, properties and generics are all pluses!

All code should be tested and uploaded to a github repository with detailed instructions of how to execute it – step by step.