Full-stack developer exam

We are going to create a web application for crowdfunding.

The application has two kinds of users:

Users that create a crowdfunding campaign (the Campaign Owner) and users that donate the crowdfunding (the Donator).

For this task, we won't be implementing a UI for the Campaign Owners. Instead, we will populate the DB with test Campaign Owners using the MySQL script created as a part of the task.

Each Campaign Owner has one or more campaigns in the DB.

The Campaign Owners' data should include at least the following:

- User ID
- Username
- Crypto wallet address

The Donator visits the website anonymously, no authentication mechanisms are required for them, and no data is being stored about them.

The UI for the Donator user should include the Campaigns list page.

The donations history for each campaign should be recorded in the DB, but it should not have a UI.

After entering the application website, the Donator should be able to see the full list of active campaigns.

As the user scrolls the campaigns list, the records should gradually fade in and out. Enough test donations must be loaded into the database to demonstrate this feature.

The Campaign should have at least the following information in the DB:

- Unique alphanumeric ID
- Name
- Description
- The goal amount in USD (how much money the campaign should collect to be successful)
- The expiration date

- The status - can be 'active', 'expired', 'fraud', 'successful'

Campaign Statuses

1. 'active'

The campaign is marked as active by default when added.

2. 'successful'

The campaign is marked as successful when the crowdfunding campaign reaches the goal (collects enough money).

3. 'expired'

The campaign is marked as expired when its expiration time is over.

A periodic event in the database should be implemented which is executed every 10 seconds and marked the campaigns as 'expired' if their expiration date has passed.

4. 'fraud'

The campaign should be marked as fraud, if a Partner Authority Service calls our system API (named MarkCampaignAsFraud - we should implement this API) with the unique campaign ID. In this case, the system should mark the Campaign Owner user as fraud and all their campaigns as fraud, and all the donations made for these campaigns should be marked as 'invalid'.

New donation

To donate, the Donator user should click on the campaign record on the list and see a popup window. There they can enter the amount of money (in US dollars) they want to donate and their nickname which may contain English letters, digits, and underscores.

The data for the donation should include at least the following information:

- Amount in US dollars
- The nickname of the donator
- State - can be 'valid' (when new) or 'invalid' when the campaign is marked as 'fraud'

(The actual financial transaction is not in the scope of this task. The donation that the user makes cannot be undone).

General considerations

The solution needs to be implemented using a NodeJS server with a MySQL database and a React website.

SOLID principles must be used while developing it.

The client-side validation of the amount of input must be implemented.

The number of Donator users is expected to grow, and we want to keep the system as responsive as it can be, so the number of calls to the database and external services must be minimized. Add caching when applicable.

Malicious users can be present on this platform who want to send invalid requests to overload the server or to steal the data by MySQL injection. Therefore, all data must be strictly validated on the server side and the business logic must be validated in the database procedures.

It might be possible that new currencies of the campaign goals such as EUR are also added in the future, so the database structure should be designed in a way that it's easy to add the new currencies. The users and crowdfunding campaigns also might have some new statuses added in the future, so they must be stored in a way that it's easy to add the new statuses. Also, all database logic called from the application should be encapsulated into database procedures, and database transactions should be used where applicable.

Mandatory deliverables

The SQL script for loading test Campaign Owners and crowdfunding campaigns, NodeJS server code, and the ReactJS code with all required running instructions must be delivered.

Bonus question

In addition to entering the donation amount in USD and the nickname, the Donator should pick one of the crypto currencies available in the popup when donating.

The system has a list of available cryptocurrencies prepopulated in the DB.

The corresponding value in USD should be taken from some 3rd party service [for example https://api.coinbase.com/v2/exchange-rates?currency=ETH].

The system stores the donation in the DB after the rate calculation.

The data for the donation should additionally include the following information:

- Selected Crypto Currency
- The Crypto amount as calculated at the time of the donation

The system is intended to grow, and new cryptocurrencies will be added to it, so the database structure must be designed in a way that it's easy to add them (not in UI).