CURRENCY BUYER USER MANUAL

MUKURU TECHNICAL TEST

JACQUES ARTGRAVEN
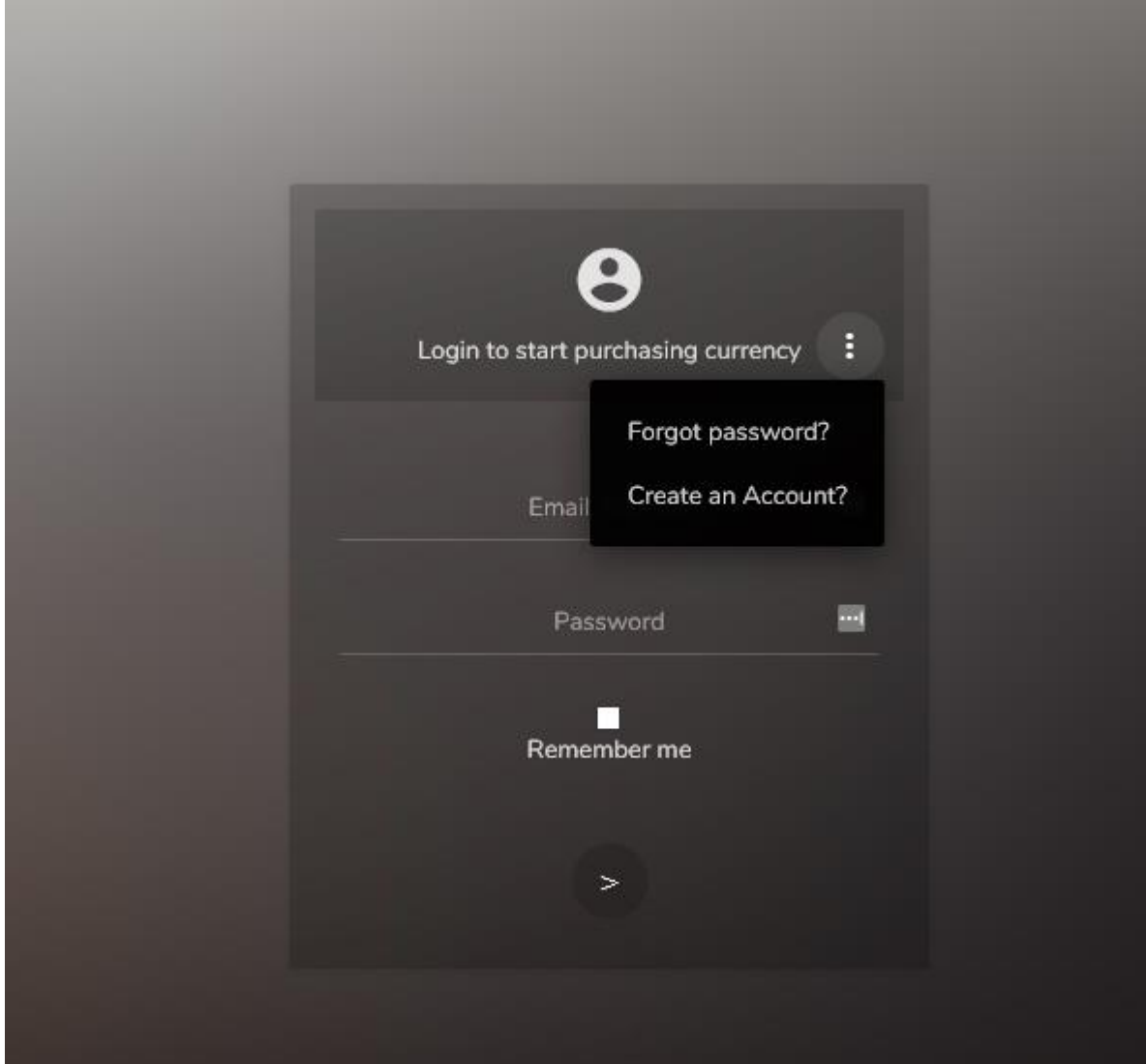
Contents

Login and registration

The public is able to register or login if they already have an account. But a user must be registered before they can login and purchase any currency.
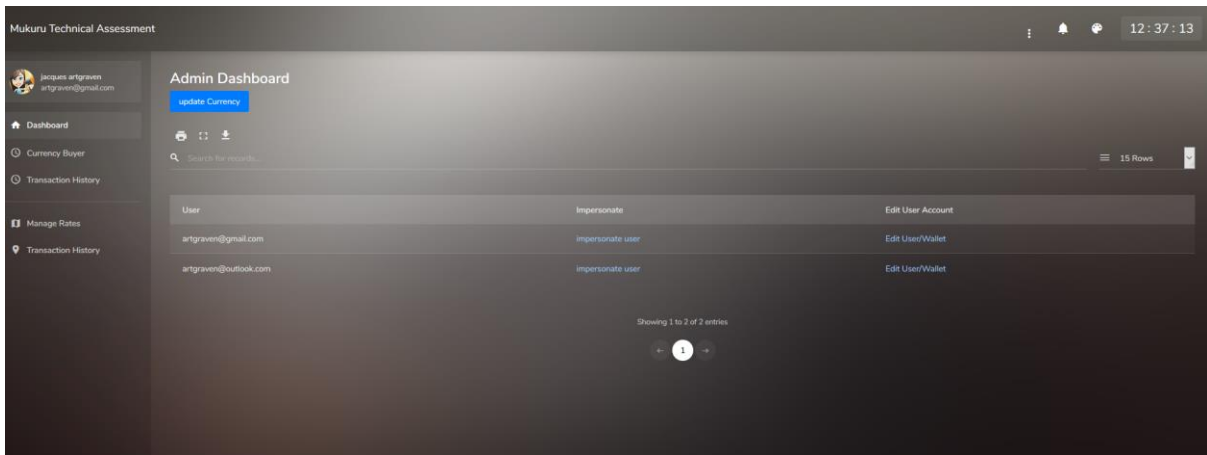
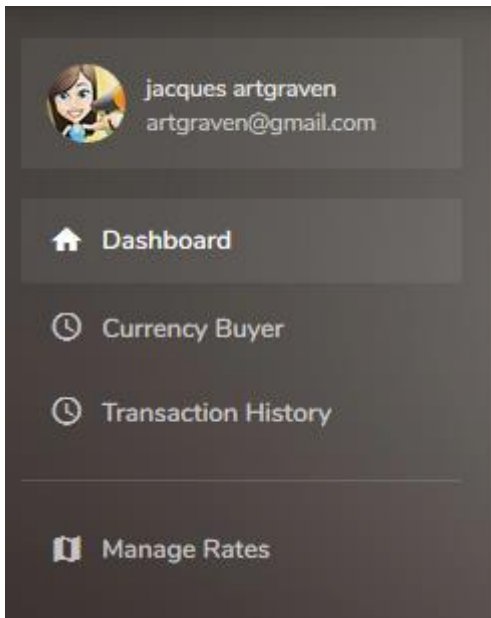We automatically award a new user with 100 credits in usd

In our first screen we can either login or we can recover our password or create a new account

When registering you will automatically be logged in

Navigation

When logged in as an admin I see a number of additional options not seen by normal users

Currency Buyer and Transaction history is the same for both user levels but the difference being Transaction History for a normal user shows only his or her transaction history and for admin will show all

Dashboard
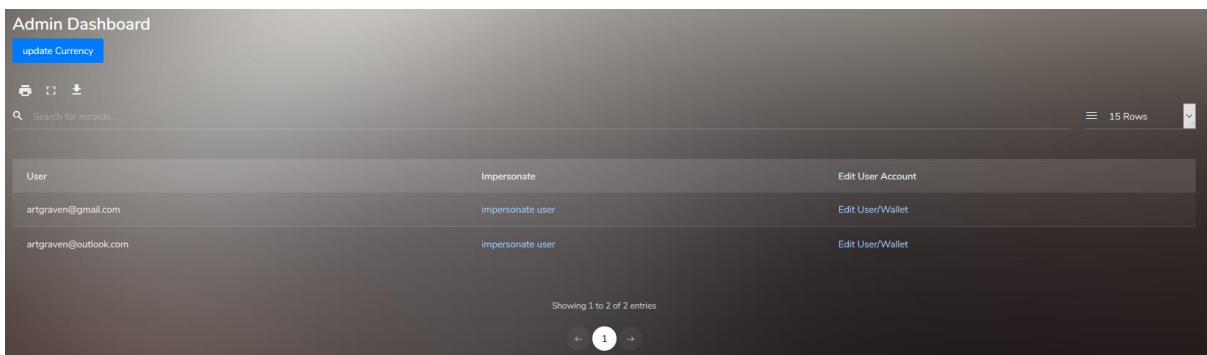
The admin dashboard has 2 key features

Update Currency

This will do a run to the jsonrates api and get the current rates. It will either insert ones that don't exist or update existing ones.
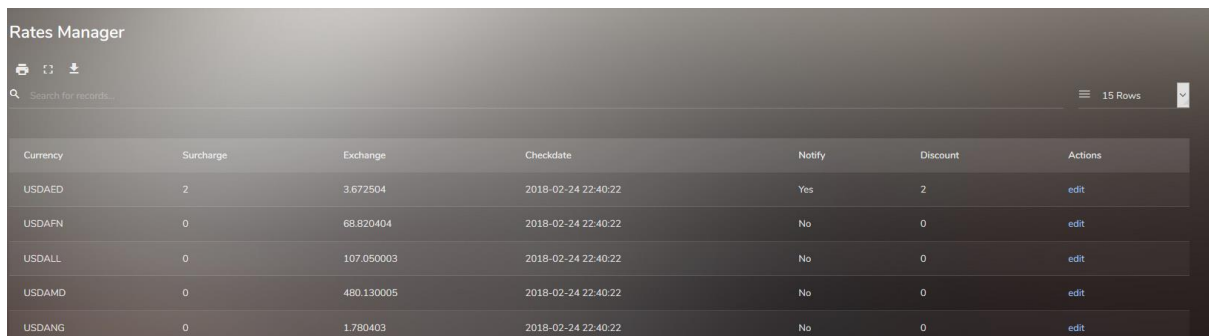
User management table

Note that all tables in the system has the ability to filter as well as export or print.

There are 2 essential features here

- Impersonate
  - The ability to impersonate the user and act on their behalf
  - Remember to exit impersonation when done
- Edit User/Wallet
  - The ability to modify funds in the users wallet as well as update details to their profile such as name, surname and so on

Manage Rates – admin only

This is a screen that shows all the current rates pulled from the api but allows us to update them with their rules

In the edit form for rates we can

Over write and update

- Surcharge
- Exhange rate
- Notify – whether a mail gets sent out as confirmation or not
- And the discount rate

Buyer Dashboard

the buyers dashboard allows the same data table we are already familiar with but now has the features of gettting a quote for the currency to usd and currency for usd
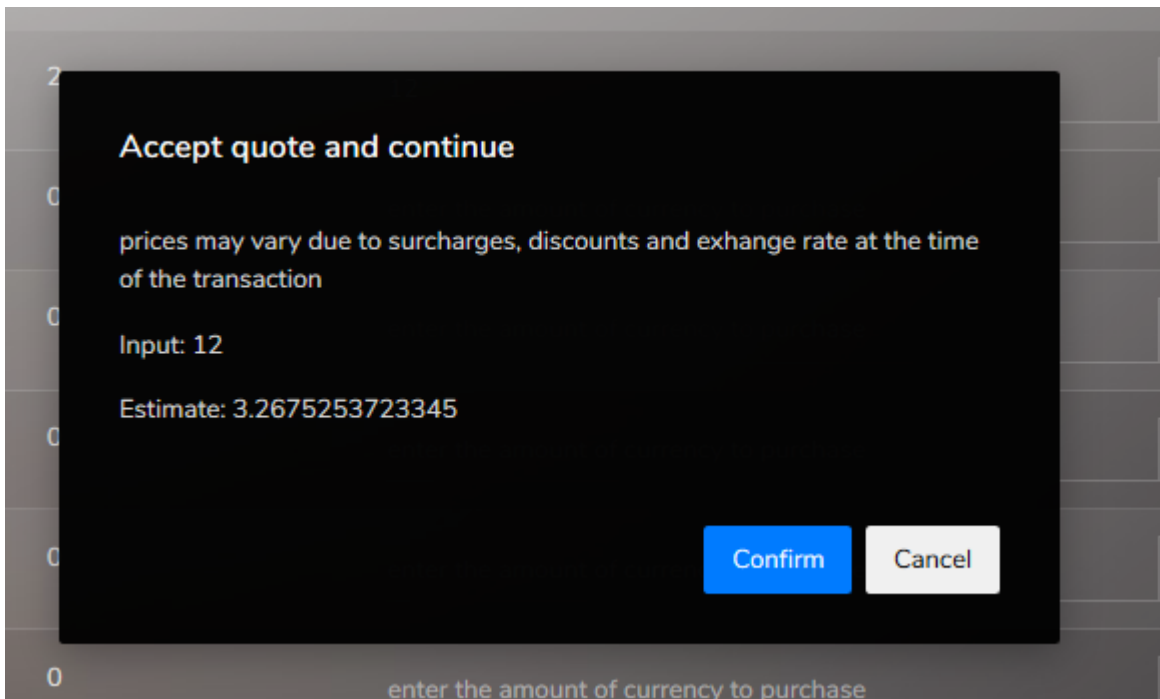
Once I enter a value in either field

I press get quote for the quote

Which will create a modal popup with the option to cancel or confirm the transaction

Please note you must have enough funds in your wallet or it wont let you proceed
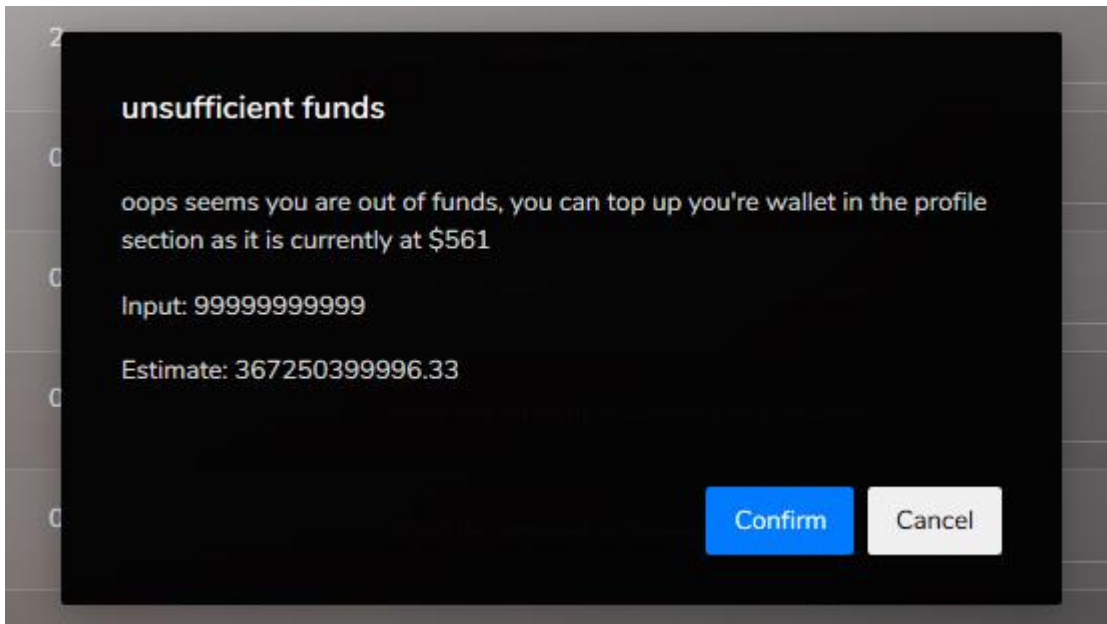
if I have enough funds and I press the confirm we will pass this to the transaction controller

and if all is well there we will be redirected to our confirmation page

please note the quote generated at first is based on the exhange rate

but the invoice following is the final amount after discounts and surcharges have been applied as it will be outlined in the table.

Transaction History

This is a table view of all my transactions in the past. Each one with a link to the original invoice
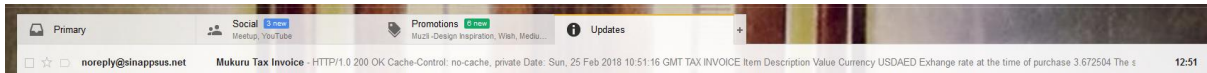

Mail Confirmation

Note that if the currency had a yes for notify then I will receive a mail

Which looks like this

Closing Notes

Unfortunantly due to limited time I was unable to get to some of the components.

The calculations on surcharge isnt right yet

I would have loved to do a lot more testing

As to the api part. I didn't write a full fledged api but simply used serialization to output json and used ajax calls to run to process that in the front.

Ideally I would have added an authentication key based on the user id and salt and written a method to verify this then added a page outlining the calls and their paramaters. I would however have used a library to help read my comments or fosRestBundle to help me do this and deal with cross site scripting issues

I added simple rules for methods such as post delete and get but these can be much more refined.

Given the simplicity of the application I didn't do a database diagram and instead used an orm that makes the logic pretty readable but a visual view would be useful if I had to share this with a team

More commenting. I had added some good commenting but I would have liked to do more

I would have used googles location api and added the address as part of the registration and used that to determine the currency based on the address

I would have added more validation, such as forcing numbers only where needed and handling transactions notifications when we the wallet goes into arrears.

Next I started writing a console application to run the currency update in doing this I would have converted the current call out of the controller and into a service object that I could call from there.

It has been added and you can see it if you run: bin/console currency:update

Right now it will log out that it worked but it wont actually do anything as I didn't have time to implement the service object.

When deploying it I would have implemented a better deployment strategy and I would have locked down the javascript and css files permissions from writing to avoid js/agent attacks