
# Database Design-CS 6360

## Term Project Report

## Bitcoin Trading Application

Team:

Deep Dimple Gosalia(dxg190036)

Hita Gangineni(hxg210013)

Bhavesh Prasad Pudi(bxp200022)

Shashank Kathavate(ssk210005)

# Table of Contents

# Design:

Based on the information provided, we are having a “IS A” hierarchy where the User entity is classified into 3 types, they are Clients, Traders and Manager. we are capturing the details like UserID, Username, Password, type of the user.

For each client there will be details assigned such as his first name, last name, age, phone number, email address, address details, his client ID, his BTC wallet balance and Fiat wallet balance.

For Traders, we will have the details like the name and id of the trader.

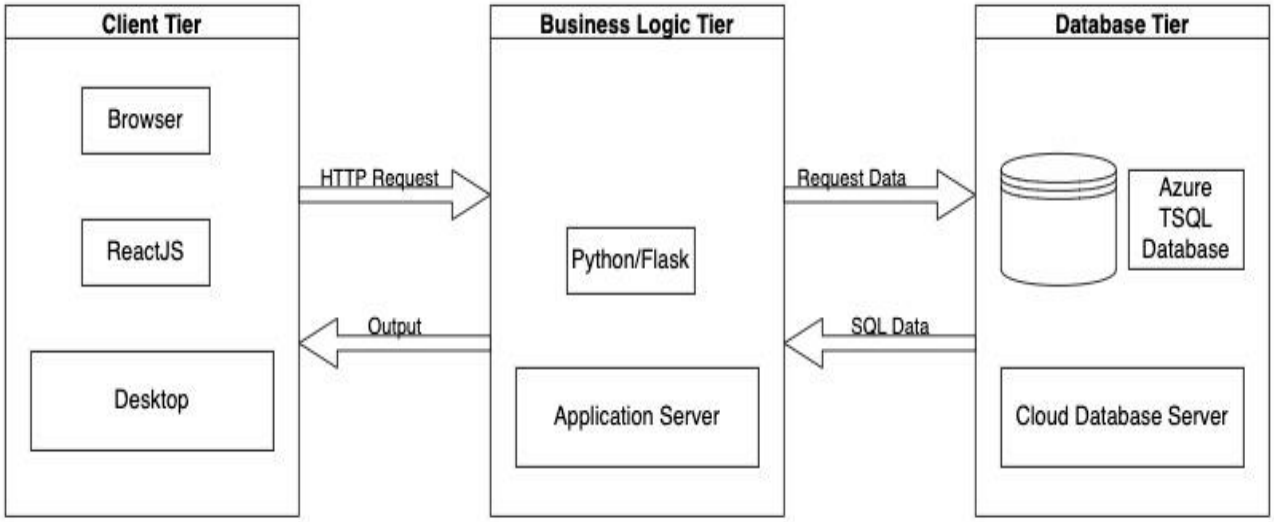
For Manager, we will have the details like the name and id of the manager.

For performing the transactions like buy/sell a bitcoin, either a client asks the trader to do a transaction on his behalf or he can do that on his behalf.

So, we have used a “Transacts” relation and we are storing the transactions carried out. In the transactions table we are capturing the details such as the Id of the transaction, Id of the client who has done it, Id of the trader, the commission applied for that transaction etc.

We have used a log table to note down the logs of the details about which transaction is made by whom and if the transaction is approved or rejected using the keys mentioned. Also, we have used another table called “cancellation logs” having the cancellation id and the transaction id of the cancelled or rejected transaction to capture the details.

## Software Architecture:




The architecture of our application is based on a typical three tier architectural model. Our client tier (View) is written in Javascript, HTML, and CSS, using ReactJS as the framework. This level of the architecture is what the user will interact with to access the features, perform certain predefined tasks based on his role in our application.

The Business Logic Tier will be written using NodeJs and Python3 using flask framework, and this tier represents the Application Server that will act as a bridge of communication for the Client Tier and Database Tier. This tier will serve HTML pages to the user's device and accept HTTP requests from the user and follow with the appropriate response.

Our Database Tier will be hosting on Microsoft Azure cloud. we are using the azure transact sql, this is where we will store all the crucial data our application needs to function.

## Overview of Frontend Code:

For the Frontend user interface, we have used reactjs to create pages which are more interactive and easier to access for the user.

We have used different packages and dependencies such as Grids, Textfields, Buttons, different CSS styles, Modals, Histograms etc.

From this the user will be able to perform actions based on the user role (Client, Trader & Manager), these actions will be captured and sent to the backend as the HTTP requests.

Example code snippet:

```
import './App.css';
import { Grid } from '@mui/material';
import styled from 'styled-components';
import TraderView from './components/TraderView';
import Signup from './components/Signup';
import Login from './components/Login';
import { useEffect, useState } from 'react';
import ManagerView from './components/Manager';
import ClientView from './components/ClientView';
import { Button } from '@tsamantanis/react-glassmorphism'

function App() {
  const [type, setType] = useState('')
  const logout = () => setType('')
  const [userObj, setUserObj] = useState([])

  useEffect(() => {
    console.log('hello', userObj)
  }, [userObj])

  return (
    <MainGrid container flex flexDirection={'row'} style={{ height: '100vh' }}>
      <Grid item md={12} lg={12}
        style={{
          backgroundColor: '#000000',
          backgroundImage: 'linear-gradient(147deg, #000000 0%, #04619f 74%)',
          border: '2px solid white',
        }}
      >
        {type !== '' && <Grid item md={12} lg={12} style={{ display: 'flex', alignItems: 'center', justifyContent: 'center' }}>
          <LogoutButton text={`Logout, ${userObj?.username}`} onClick={() => {
            logout()
          }} />
        </Grid>}
        {/* <TradingView /> */}
```

# Overview of Backend Code:

We have used a Python Flask framework backend to interface with the frontend and the azure tsql database we have created on cloud. We are using the “pyodbc” package which will allow us to connect the python3 with the Microsoft server. We created a config.py file which has the details of the server we need to connect to, the user credentials to connect to that server.

Example code snippet:

```
import pyodbc
import pandas as pd
azure ={
'server' : 'bitcoinserver.database.windows.net',
'database' : 'bitcoin',
'username' : 'hita123',
'password' : 'Hita@123',
'driver' : '{ODBC Driver 17 for SQL Server}'
}

def connect_to_azure():
    conn =  pyodbc.connect('DRIVER='+azure['driver']+';SERVER=tcp:'+azure['server']+';PORT=1433;DATABASE='+azure['database']+';UID='+azure['username']+';PWD='+
    return conn
```

So, in each API we have created we will be using this config file to connect to that server and execute the required operations

For Example, if a trader wants to look at the pending transactions that need to be approved then we will use a query like this qry = f"select * from transactions t where t.txstatus = 0 order by txid asc" in the API. Likewise, we have provisioned a distinct set of queries for each task like buying,selling bitcoins. Etc.

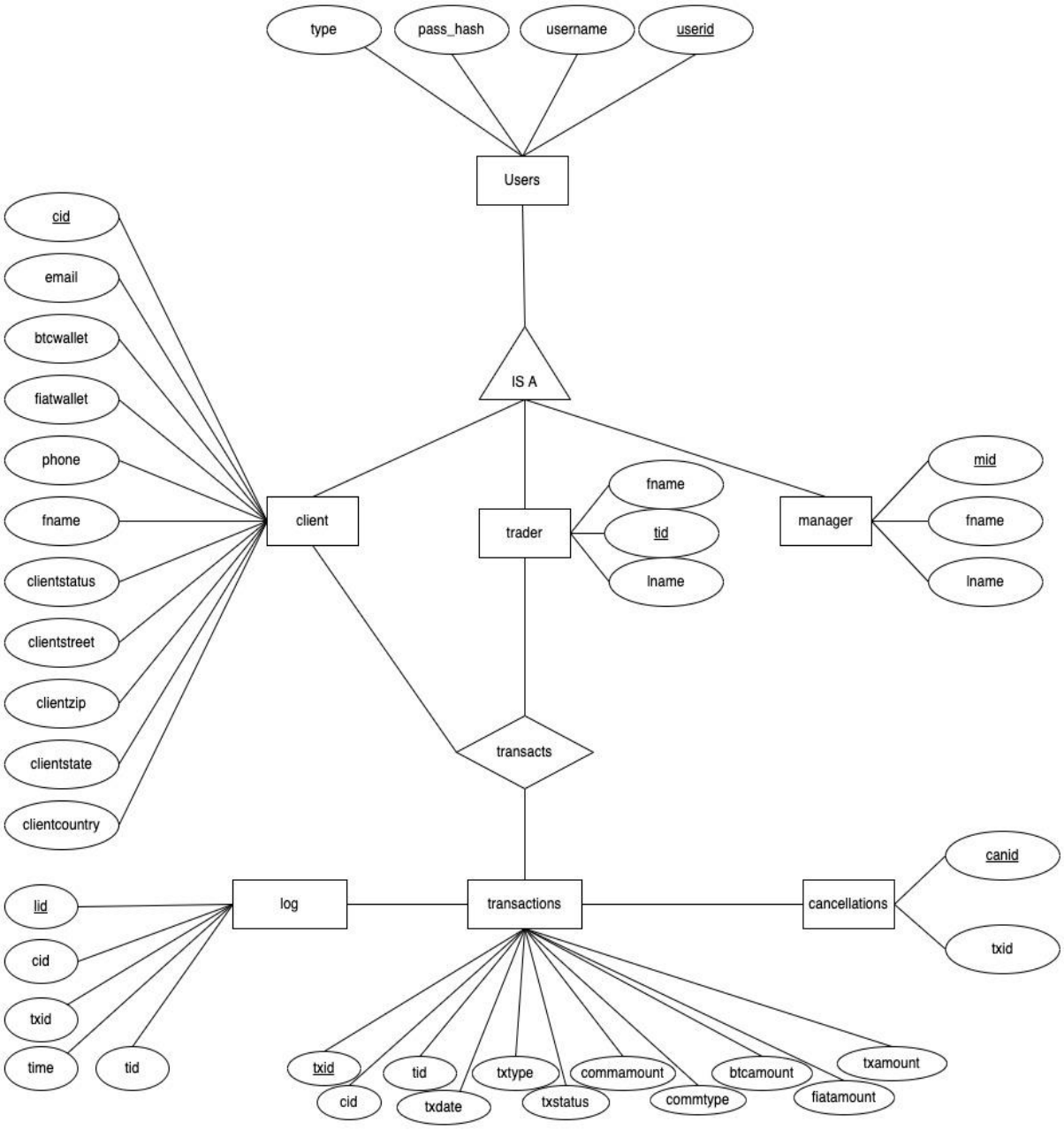
Example code snippet:

```
    def listTransactions(self):
        conn = cg.connect_to_azure()
        qry = f"SELECT * FROM [dbo].[transactions] ORDER BY txid DESC"
        df = pd.read_sql(qry, conn)
        json_user_data = df.to_json(orient="index")
        parsed_json = json.loads(json_user_data)
        return json.dumps(parsed_json)
```

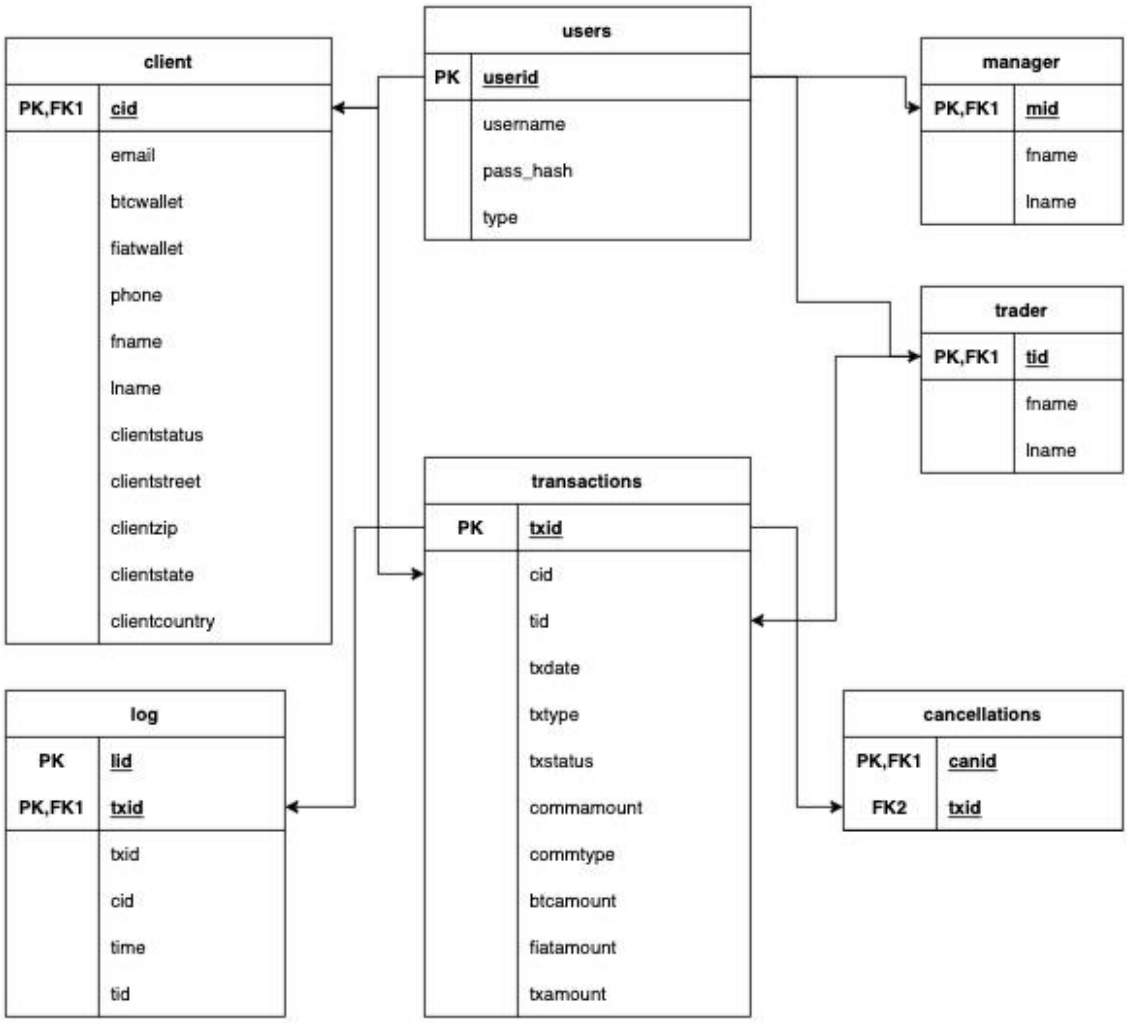
We have provided the sql queries as the prepared statements in order to prevent the sql injection attacks.

# Appendix:

## ER Diagram:

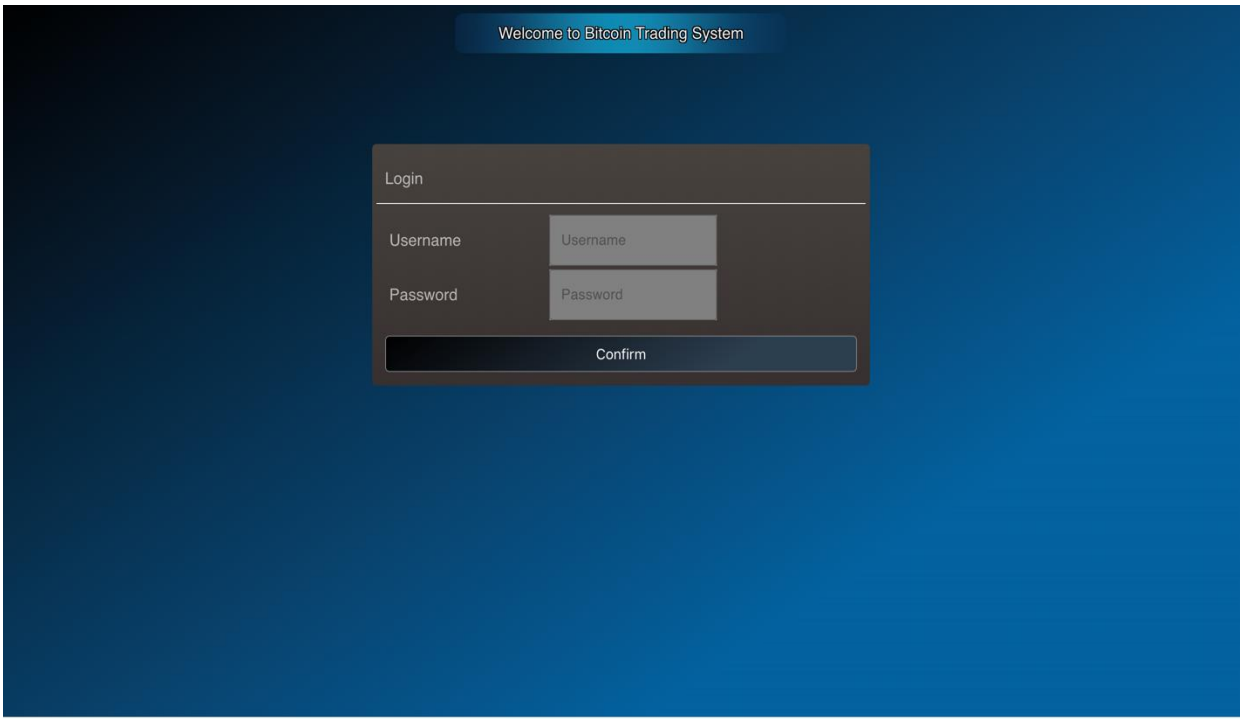


# Relational Schema:

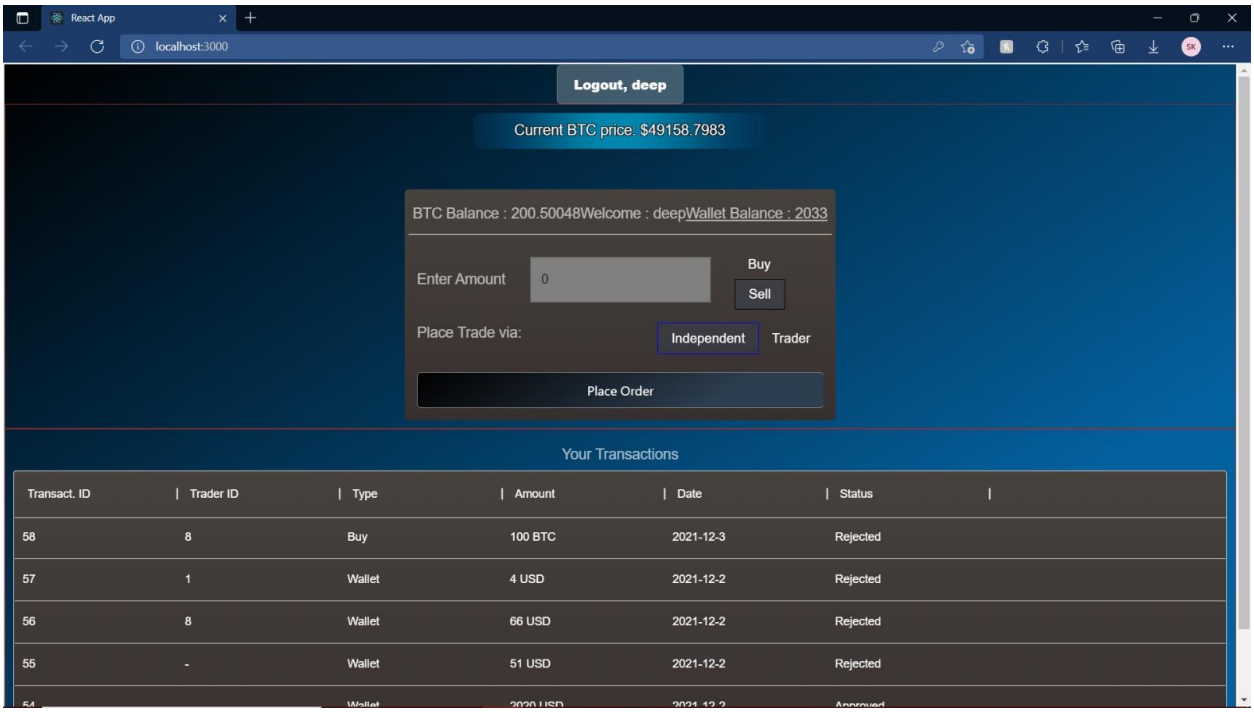


# User Interface for each page:

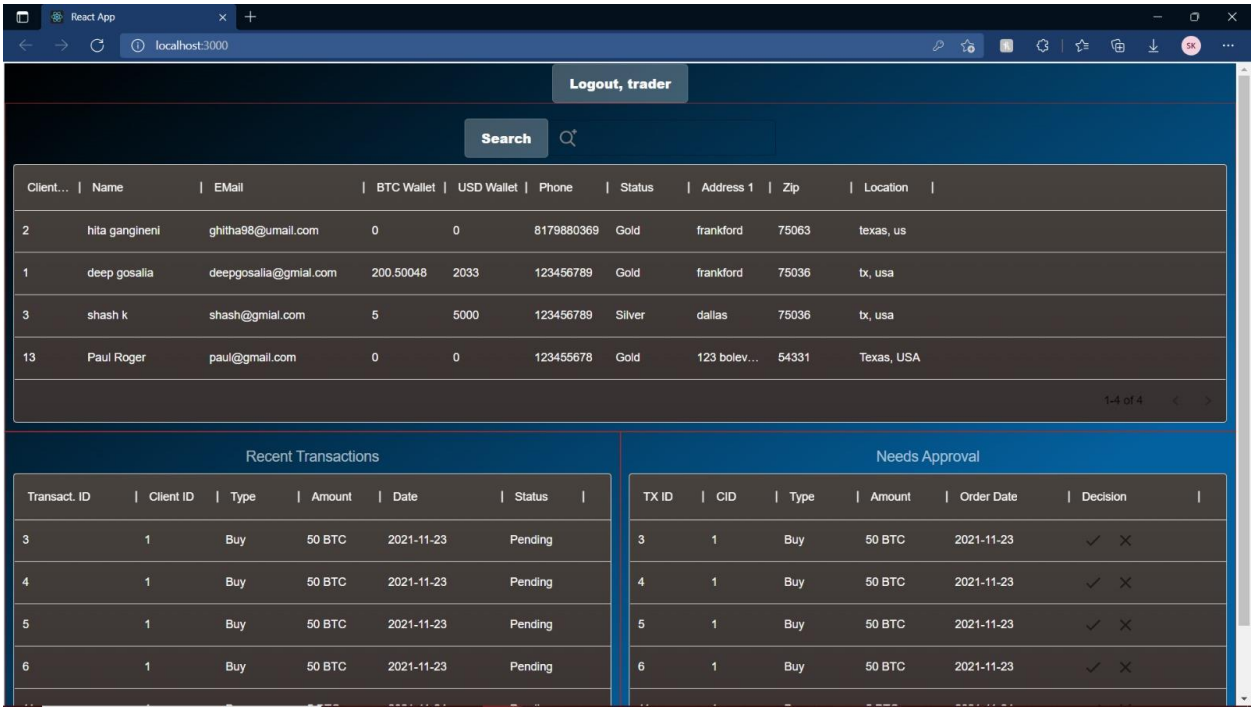
## Login Page:



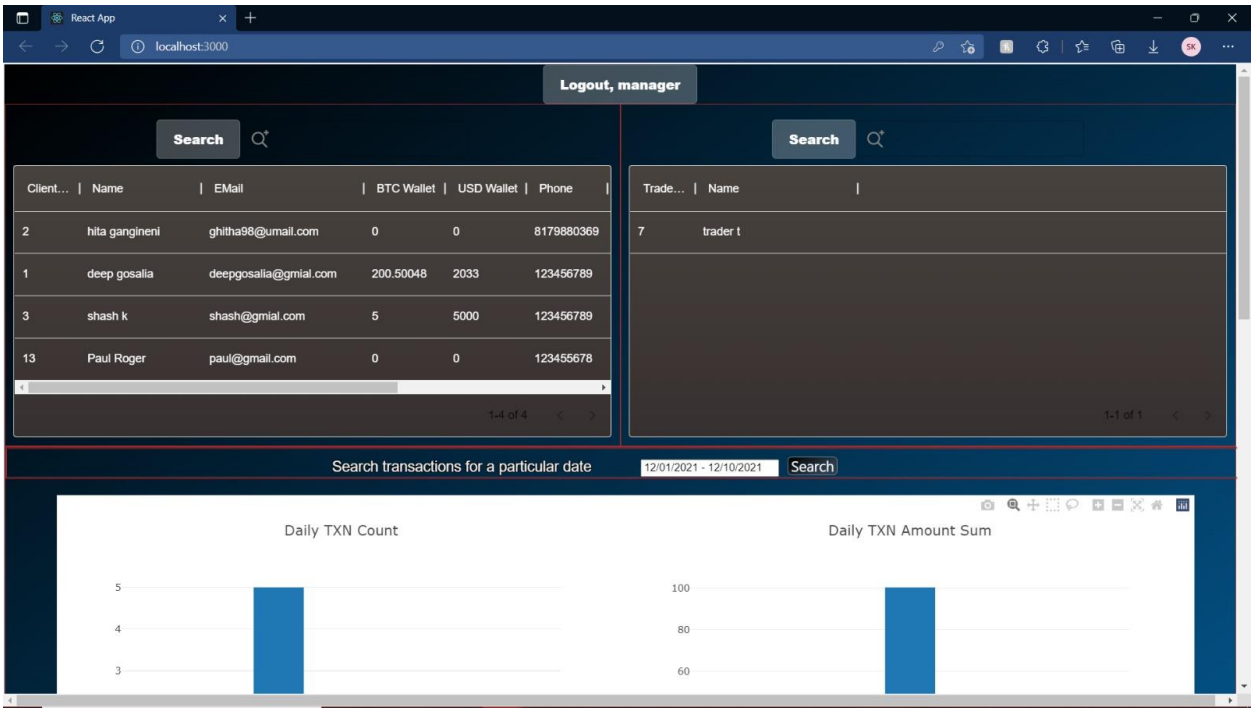

## Client UI:



Trader UI:




Manager View:



Histograms to view average transactions: