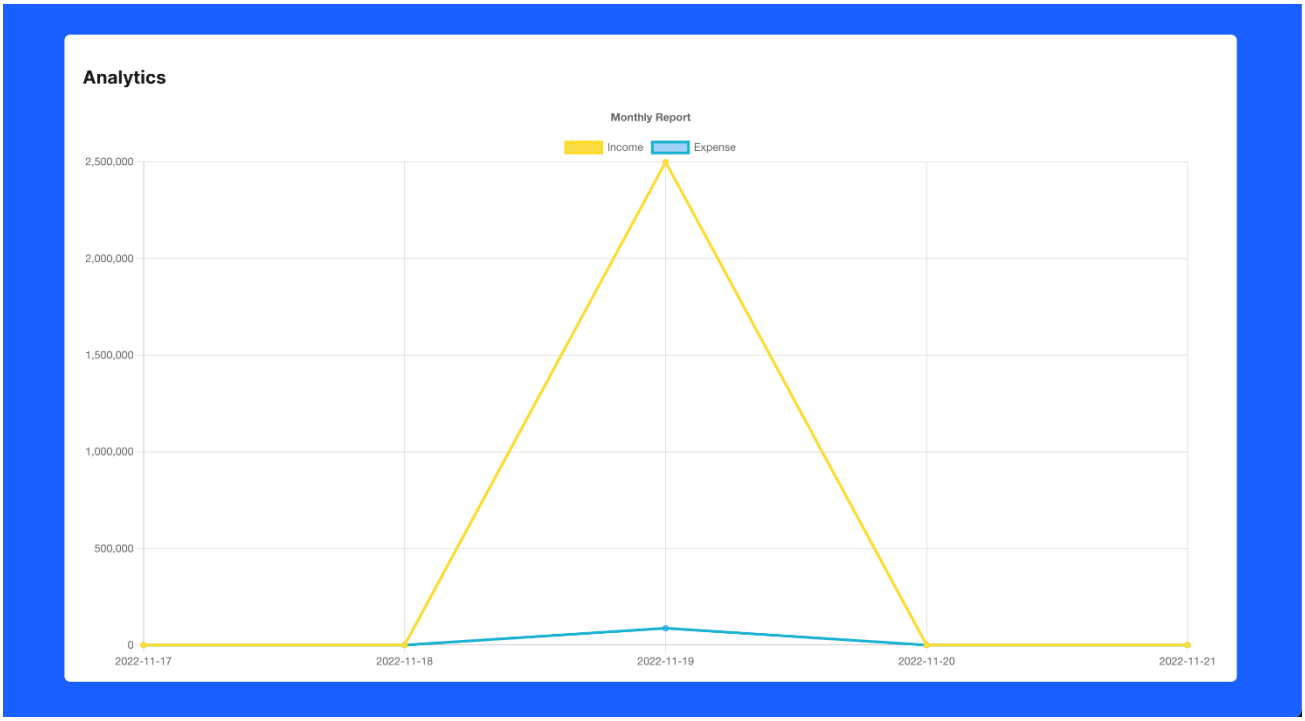
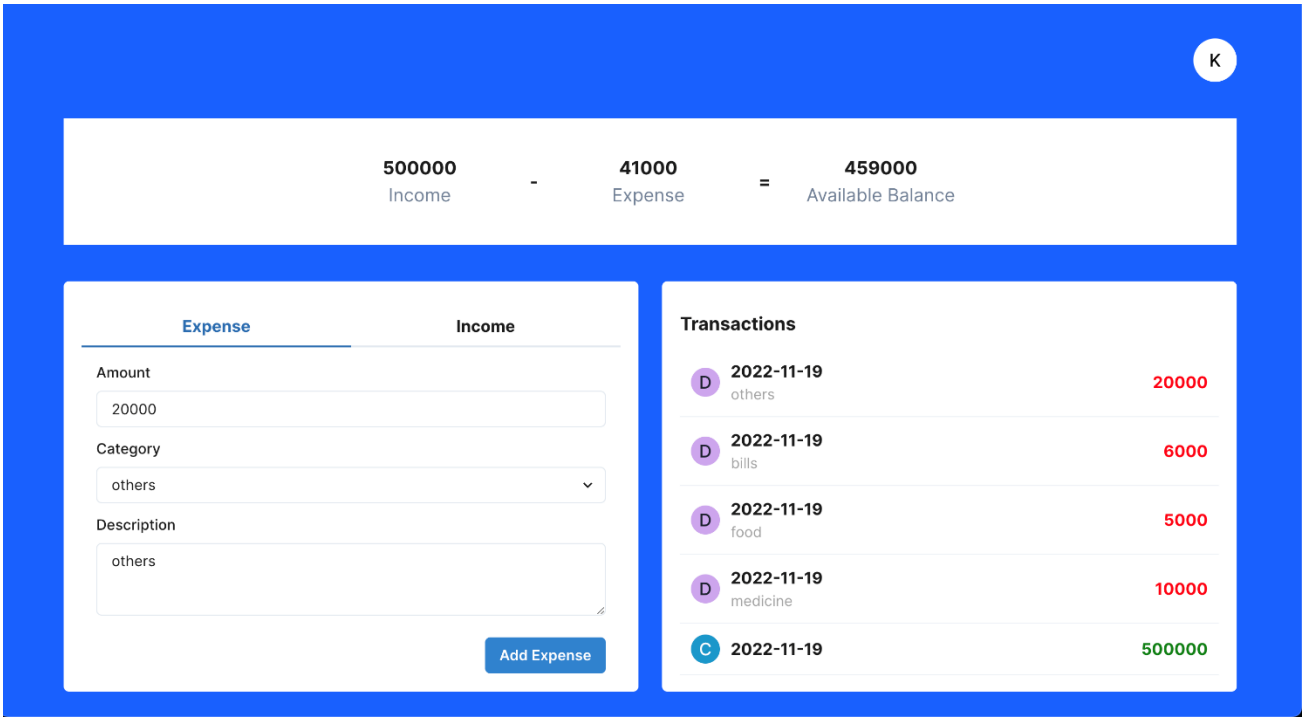
Project Development phase - Sprint 3

Team ID	PNT2022TMID04644
Project Name	Personal Expense Tracker

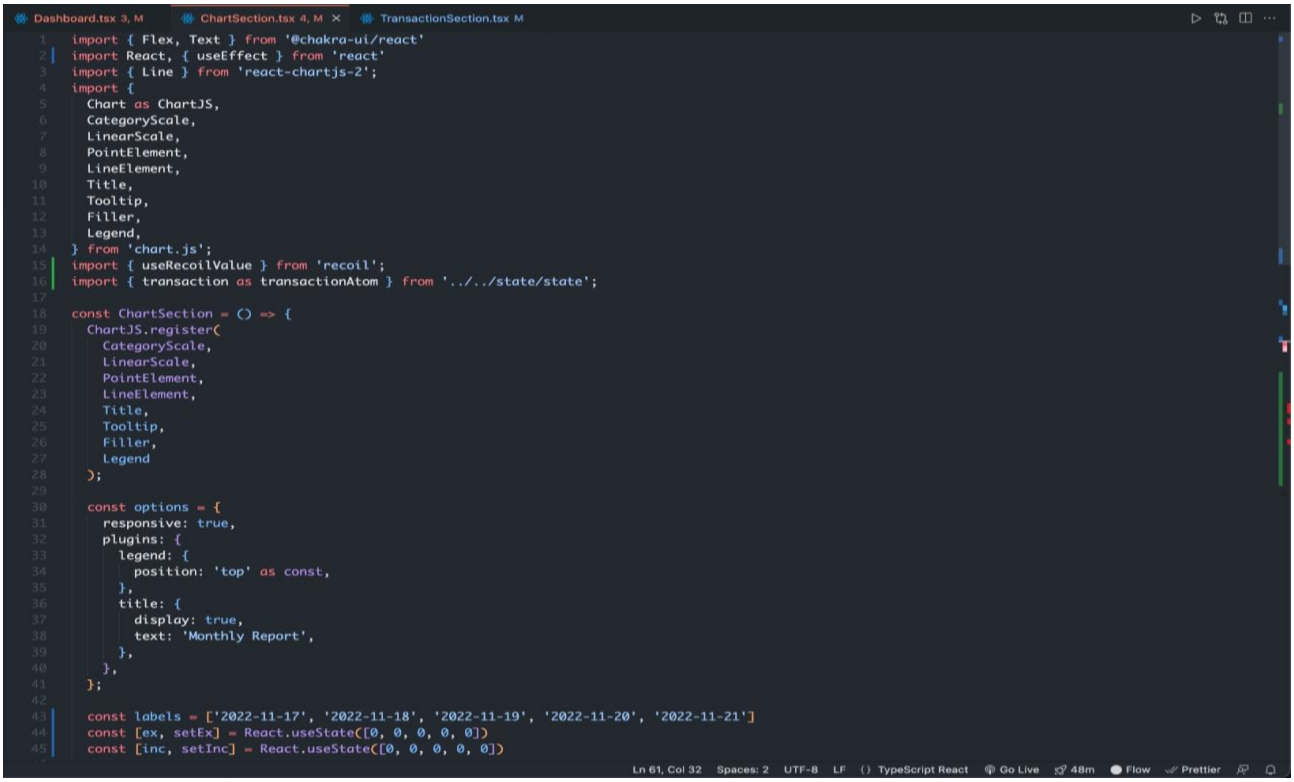
Analytics of expense and income



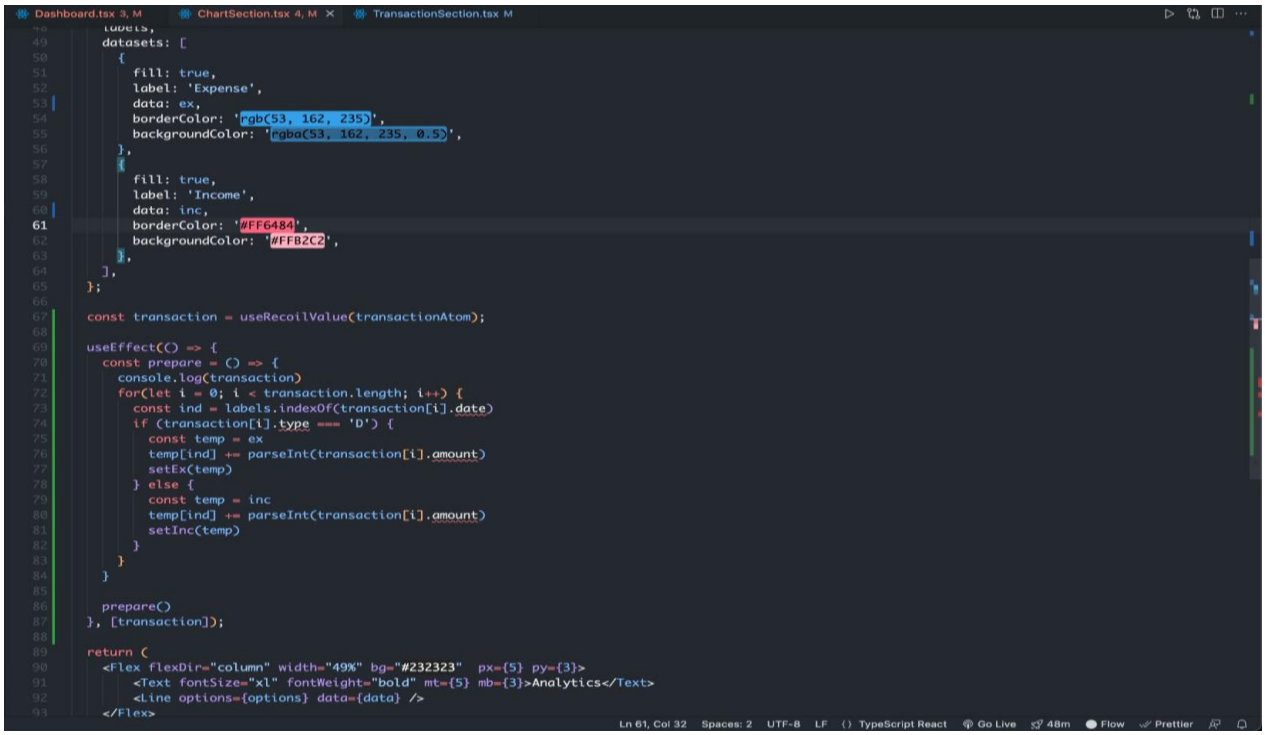
Analytics of budget & Transaction History

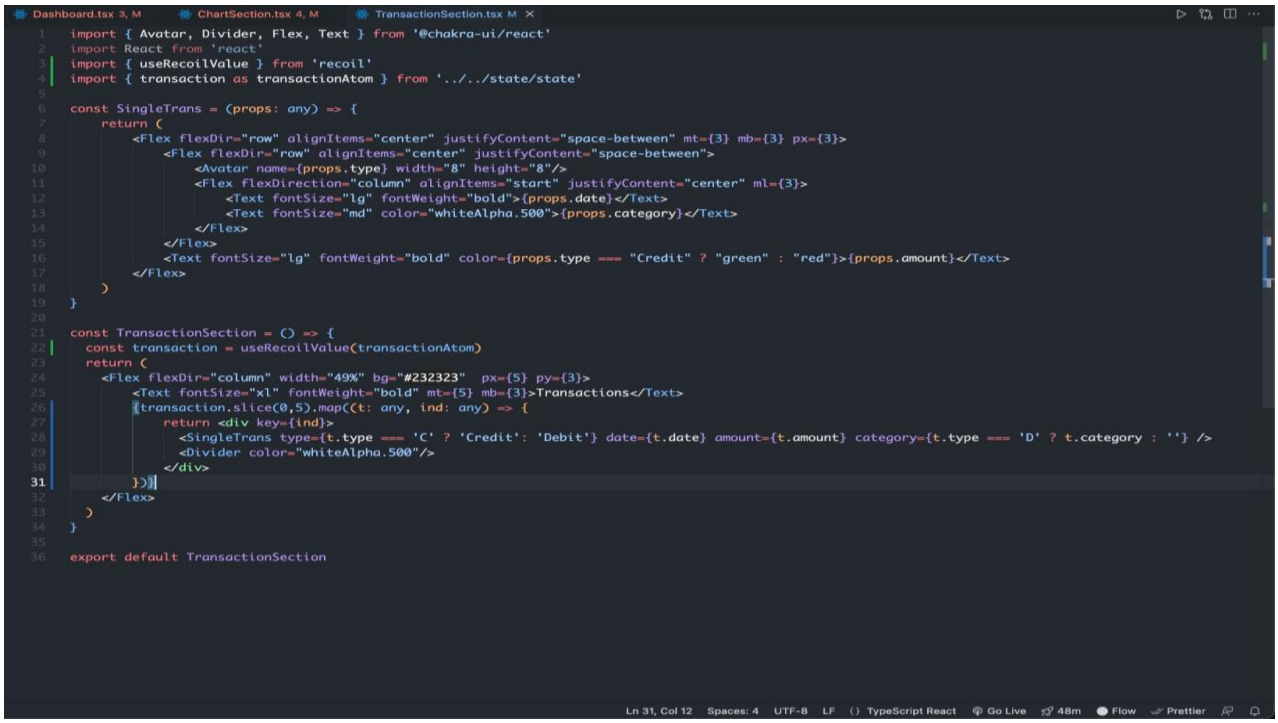


Front End Code



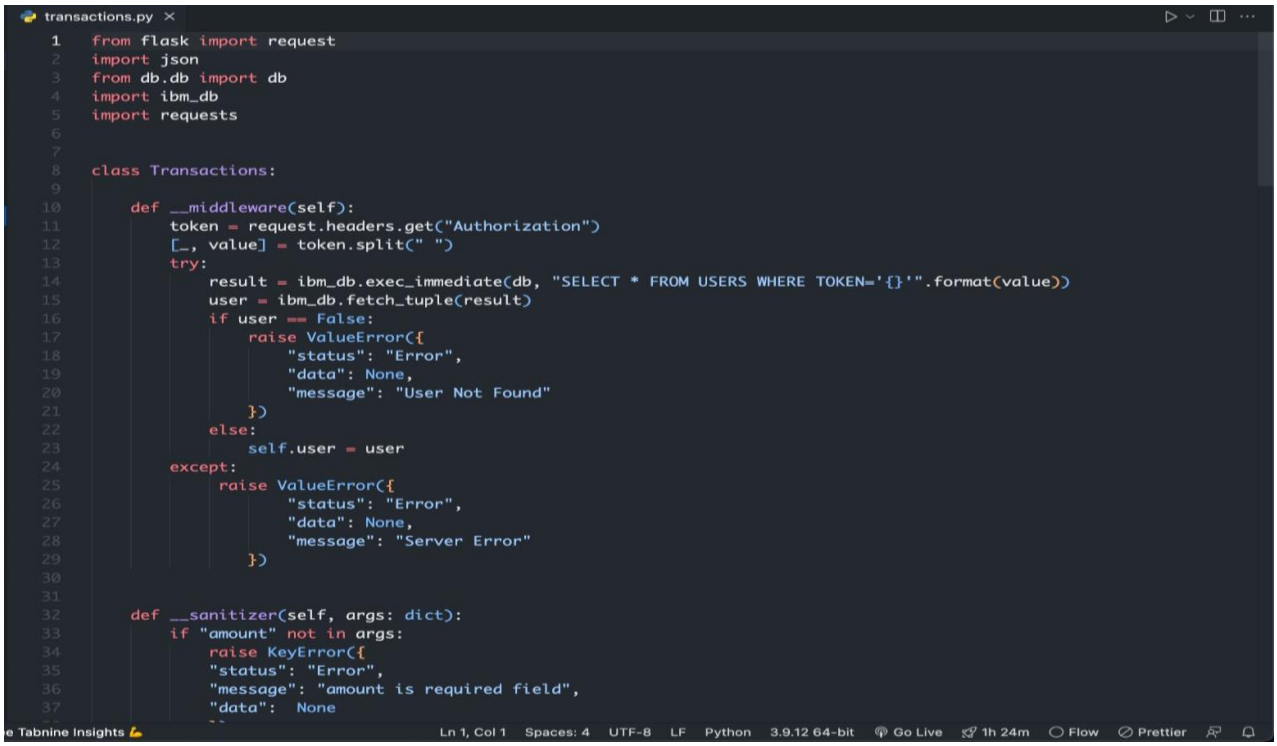
```
import { Flex, Text } from '@chakra-ui/react'
import React, { useEffect } from 'react'
import { Line } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Filler,
  Legend,
} from 'chart.js';
import { useRecoilValue } from 'recoil';
import { transaction as transactionAtom } from '../../state/state';

const ChartSection = () => {
  ChartJS.register(
    CategoryScale,
    LinearScale,
    PointElement,
    LineElement,
    Title,
    Tooltip,
    Filler,
    Legend
  );

  const options = {
    responsive: true,
    plugins: {
      legend: {
        position: 'top' as const,
      },
      title: {
        display: true,
        text: 'Monthly Report',
      },
    },
  };

  const labels = ['2022-11-17', '2022-11-18', '2022-11-19', '2022-11-20', '2022-11-21']
  const [ex, setEx] = React.useState([0, 0, 0, 0, 0])
  const [inc, setInc] = React.useState([0, 0, 0, 0, 0])
```


Dashboard.tsx 3, M
ChartSection.tsx 4, M
TransactionSection.tsx M
labels,
datasets: [
{
fill: true,
label: 'Expense',
data: ex,
borderColor: 'rgb(53, 162, 235)',
backgroundColor: 'rgba(53, 162, 235, 0.5)',
},
{
fill: true,
label: 'Income',
data: inc,
borderColor: '#FF6484',
backgroundColor: '#FFB2C2',
},
],
};
const transaction = useRecoilValue(transactionAtom);
useEffect(() => {
const prepare = () => {
console.log(transaction)
for(let i = 0; i < transaction.length; i++) {
const ind = labels.indexOf(transaction[i].date)
if (transaction[i].type === 'D') {
const temp = ex
temp[ind] += parseInt(transaction[i].amount)
setEx(temp)
} else {
const temp = inc
temp[ind] += parseInt(transaction[i].amount)
setInc(temp)
}
}
}
prepare()
}, [transaction]);
return (
<Flex flexDir="column" width="49%" bg="#232323" px={5} py={3}>
<Text fontSize="xl" fontWeight="bold" mt={5} mb={3}>Analytics</Text>
<Line options={options} data={data} />
</Flex>
Ln 61, Col 32
Spaces: 2
UTF-8
LF
TypeScript React
Go Live
48m
Flow
Prettier

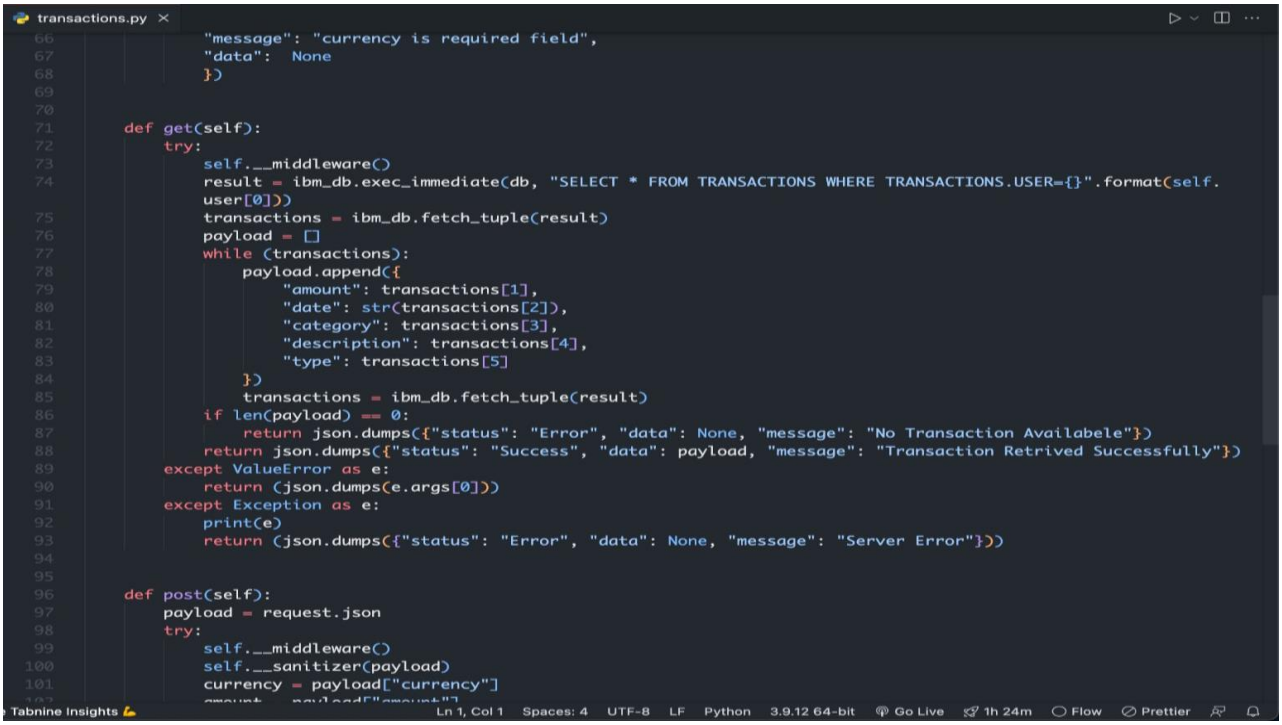

Dashboard.tsx 3, M
ChartSection.tsx 4, M
TransactionSection.tsx M
import { Avatar, Divider, Flex, Text } from '@chakra-ui/react'
import React from 'react'
import { useRecoilValue } from 'recoil'
import { transaction as transactionAtom } from '../../state/state'
const SingleTrans = (props: any) => {
return (
<Flex flexDir="row" alignItems="center" justifyContent="space-between" mt={3} mb={3} px={3}>
<Flex flexDir="row" alignItems="center" justifyContent="space-between">
<Avatar name={props.type} width="8" height="8"/>
<Flex flexDirection="column" alignItems="start" justifyContent="center" ml={3}>
<Text fontSize="lg" fontWeight="bold">{props.date}</Text>
<Text fontSize="md" color="whiteAlpha.500">{props.category}</Text>
</Flex>
</Flex>
<Text fontSize="lg" fontWeight="bold" color={props.type === "Credit" ? "green" : "red"}>{props.amount}</Text>
</Flex>
)
}
const TransactionSection = () => {
const transaction = useRecoilValue(transactionAtom)
return (
<Flex flexDir="column" width="49%" bg="#232323" px={5} py={3}>
<Text fontSize="xl" fontWeight="bold" mt={5} mb={3}>Transactions</Text>
{transaction.slice(0,5).map((t: any, ind: any) => {
return
<SingleTrans type={t.type === 'C' ? 'Credit': 'Debit'} date={t.date} amount={t.amount} category={t.type === 'D' ? t.category : ''} />
<Divider color="whiteAlpha.500"/>

})}
</Flex>
)
}
export default TransactionSection
Ln 31, Col 12
Spaces: 4
UTF-8
LF
TypeScript React
Go Live
48m
Flow
Prettier

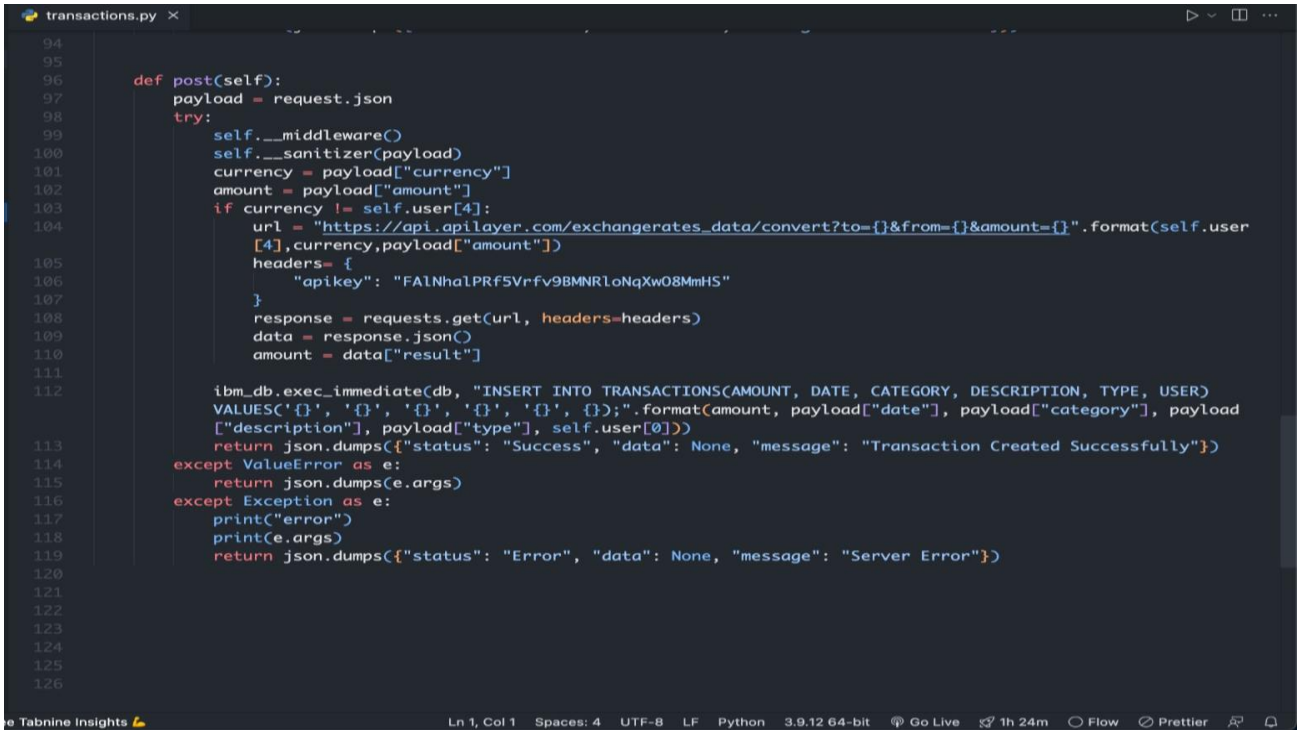
Flask Code



```python
from flask import request
import json
from db.db import db
import ibm_db
import requests


class Transactions:

    def __middleware(self):
        token = request.headers.get("Authorization")
        [_, value] = token.split(" ")
        try:
            result = ibm_db.exec_immediate(db, "SELECT * FROM USERS WHERE TOKEN='{}'".format(value))
            user = ibm_db.fetch_tuple(result)
            if user == False:
                raise ValueError({
                    "status": "Error",
                    "data": None,
                    "message": "User Not Found"
                })
            else:
                self.user = user
        except:
             raise ValueError({
                    "status": "Error",
                    "data": None,
                    "message": "Server Error"
                })


    def __sanitizer(self, args: dict):
        if "amount" not in args:
            raise KeyError({
            "status": "Error",
            "message": "amount is required field",
            "data":  None
```



```python
            "message": "currency is required field",
            "data":  None
            })


    def get(self):
        try:
            self.__middleware()
            result = ibm_db.exec_immediate(db, "SELECT * FROM TRANSACTIONS WHERE TRANSACTIONS.USER={}".format(self.
            user[0]))
            transactions = ibm_db.fetch_tuple(result)
            payload = []
            while (transactions):
                payload.append({
                    "amount": transactions[1],
                    "date": str(transactions[2]),
                    "category": transactions[3],
                    "description": transactions[4],
                    "type": transactions[5]
                })
                transactions = ibm_db.fetch_tuple(result)
            if len(payload) == 0:
                return json.dumps({"status": "Error", "data": None, "message": "No Transaction Availabele"})
            return json.dumps({"status": "Success", "data": payload, "message": "Transaction Retrived Successfully"})
        except ValueError as e:
            return (json.dumps(e.args[0]))
        except Exception as e:
            print(e)
            return (json.dumps({"status": "Error", "data": None, "message": "Server Error"}))


    def post(self):
        payload = request.json
        try:
            self.__middleware()
            self.__sanitizer(payload)
            currency = payload["currency"]
```


transactions.py
def post(self):
payload = request.json
try:
self.__middleware()
self.__sanitizer(payload)
currency = payload["currency"]
amount = payload["amount"]
if currency != self.user[4]:
url = "https://api.apilayer.com/exchangerates_data/convert?to={}&from={}&amount={}".format(self.user[4],currency,payload["amount"])
headers= {
"apikey": "FAlNhalPRf5Vrfv9BMNRloNqXwO8MmHS"
}
response = requests.get(url, headers=headers)
data = response.json()
amount = data["result"]
ibm_db.exec_immediate(db, "INSERT INTO TRANSACTIONS(AMOUNT, DATE, CATEGORY, DESCRIPTION, TYPE, USER) VALUES('{}', '{}', '{}', '{}', '{}', {});".format(amount, payload["date"], payload["category"], payload["description"], payload["type"], self.user[0]))
return json.dumps({"status": "Success", "data": None, "message": "Transaction Created Successfully"})
except ValueError as e:
return json.dumps(e.args)
except Exception as e:
print("error")
print(e.args)
return json.dumps({"status": "Error", "data": None, "message": "Server Error"})
Tabnine Insights
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Python
3.9.12 64-bit
Go Live
1h 24m
Flow
Prettier