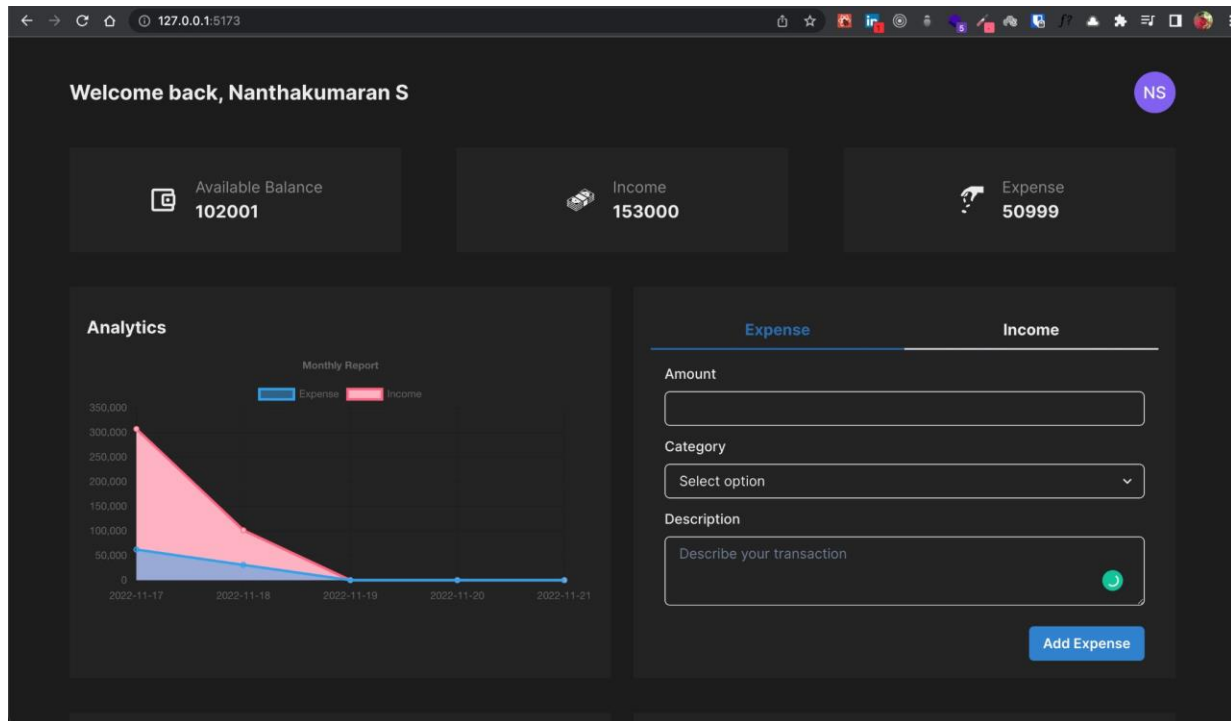


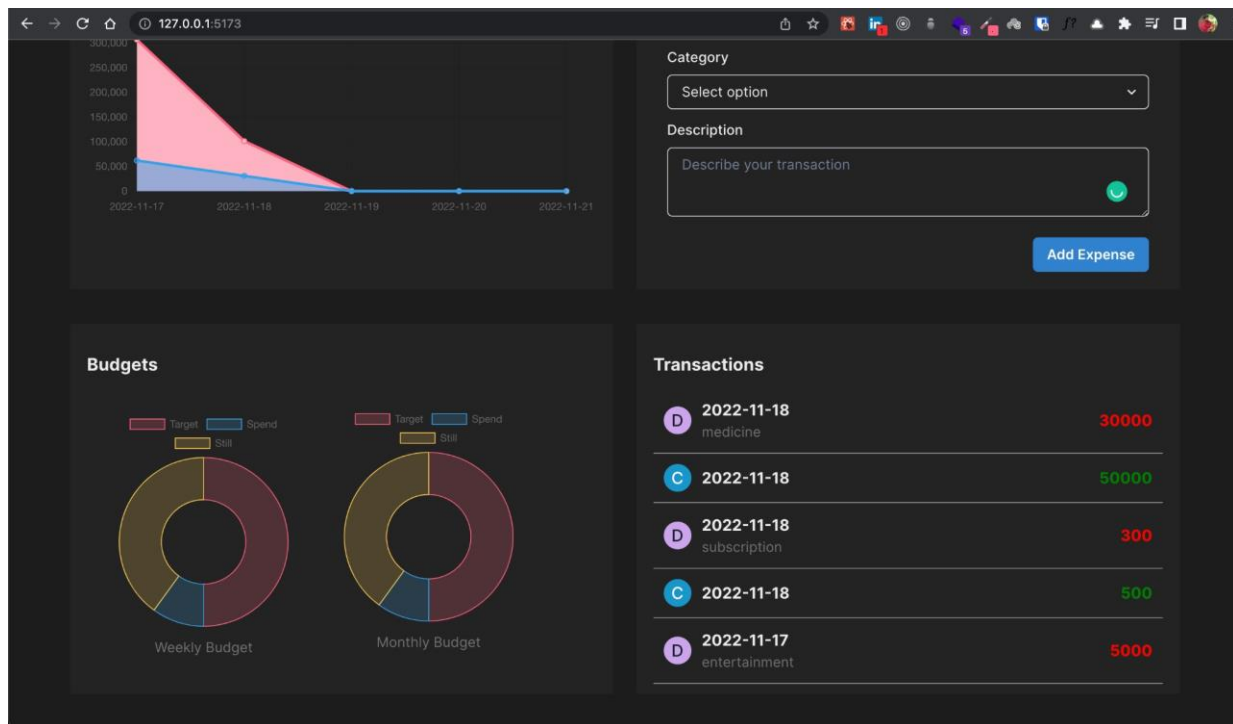
## Project Development phase - Sprint 3

Team ID	PNT2022TMID20278
Project Name	Personal Expense Tracker

### Analytics of expense and income



## Analytics of budget & Transaction History



## FrontEnd 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])
```

```
Code File Edit Selection View Go Run Terminal Window Help
ibm-personal-expense-tracker

EXPLORER
ibm-personal-expense-tracker
  node_modules
  public
  src
    components
    pages
    authenticate
    dashboard
      AdditionalTopCard.tsx
      Budget.tsx
      ChartSection.tsx 4, M
      Dashboard.tsx 3, M
      EntryPoint.tsx
      TopCard.tsx
      TransactionSection.tsx
    state
    state.ts
    utils
    constants.tsx
    PrivateRoute.tsx
    theme.ts
    App.tsx
    index.css
    main.tsx
    vite-env.d.ts
    .gitignore
    index.html
    package.json
    tsconfig.json
    tsconfig.node.json
    vite.config.ts
    yarn.lock

OUTLINE
TIMELINE
NPM SCRIPTS

Dashboard.tsx 3, M
ChartSection.tsx 4, M
TransactionSection.tsx M

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="498" bg="#232323" px={5} py={3}>
    <Text fontSize="xl" fontWeight="bold" mt={5} mb={3}>Analytics</Text>
    <line options={options} data={data} />
  </Flex>
)
```

```
Code File Edit Selection View Go Run Terminal Window Help
ibm-personal-expense-tracker

EXPLORER
ibm-personal-expense-tracker
  node_modules
  public
  src
    components
    pages
    authenticate
    dashboard
      AdditionalTopCard.tsx
      Budget.tsx
      ChartSection.tsx 4, M
      Dashboard.tsx 3, M
      EntryPoint.tsx
      TopCard.tsx
      TransactionSection.tsx
    state
    state.ts
    utils
    constants.tsx
    PrivateRoute.tsx
    theme.ts
    App.tsx
    index.css
    main.tsx
    vite-env.d.ts
    .gitignore
    index.html
    package.json
    tsconfig.json
    tsconfig.node.json
    vite.config.ts
    yarn.lock

OUTLINE
TIMELINE
NPM SCRIPTS

Dashboard.tsx 3, M
ChartSection.tsx 4, M
TransactionSection.tsx M

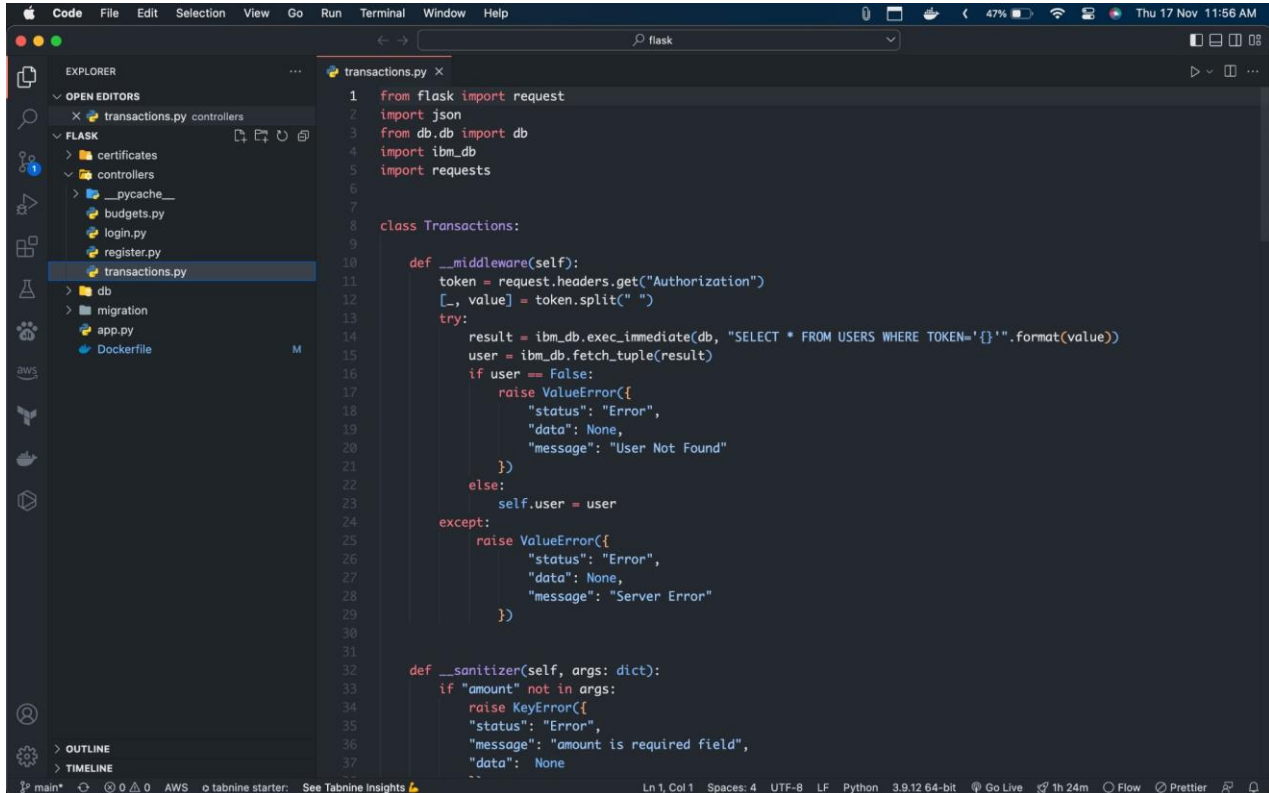
import { Avatar, Divider, Flex, Text } from 'chakra-ui/react'
import React from 'react'
import { useRecoilValue } from 'recoil'
import { transaction as transactionAtom } from '../state/state'

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 flexDir="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>
  )
}

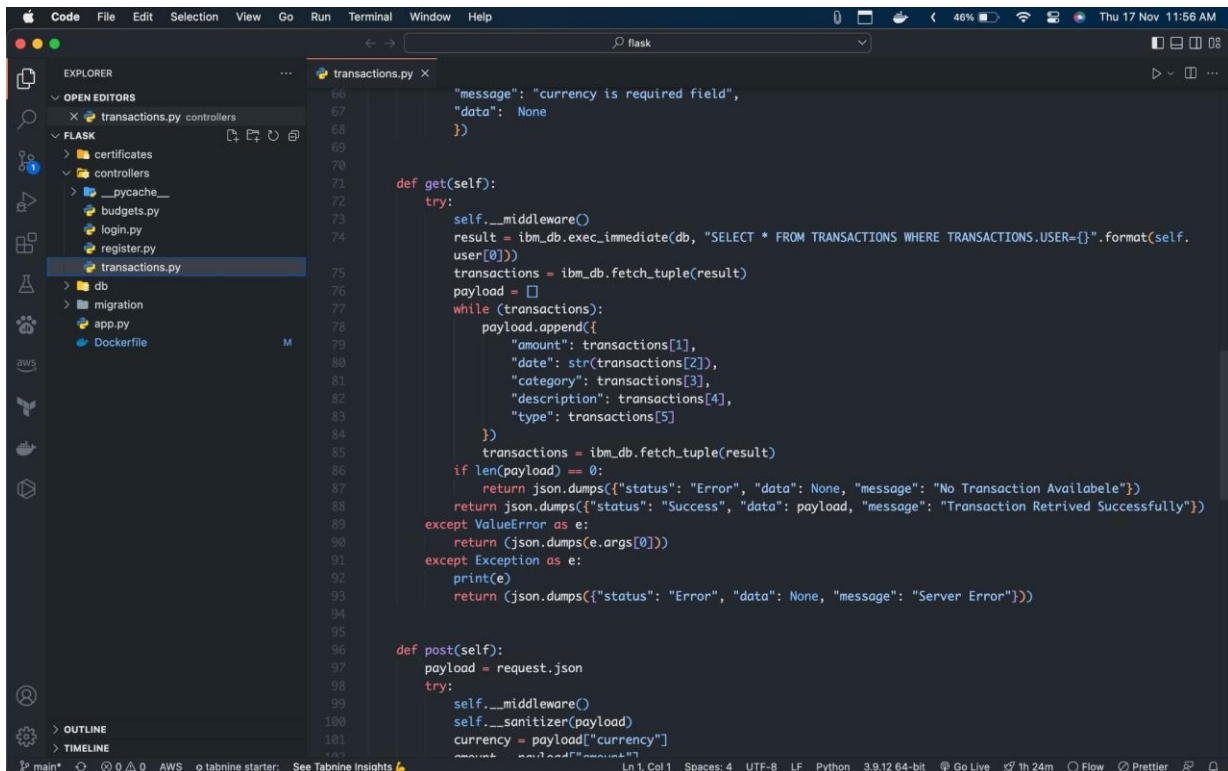
const TransactionSection = () => {
  const transaction = useRecoilValue(transactionAtom)
  return (
    <Flex flexDir="column" width="498" bg="#232323" px={5} py={3}>
      <Text fontSize="xl" fontWeight="bold" mt={5} mb={3}>Transactions</Text>
      <transaction.slice(0, 5).map((t: any, ind: any) => {
        return <div key={ind}>
          <SingleTrans type={t.type === 'C' ? 'Credit' : 'Debit'} date={t.date} amount={t.amount} category={t.type === 'D' ? t.category : ''} />
          <Divider color="whiteAlpha.500"/>
        </div>
      )}>
    </Flex>
  )
}

export default TransactionSection
```

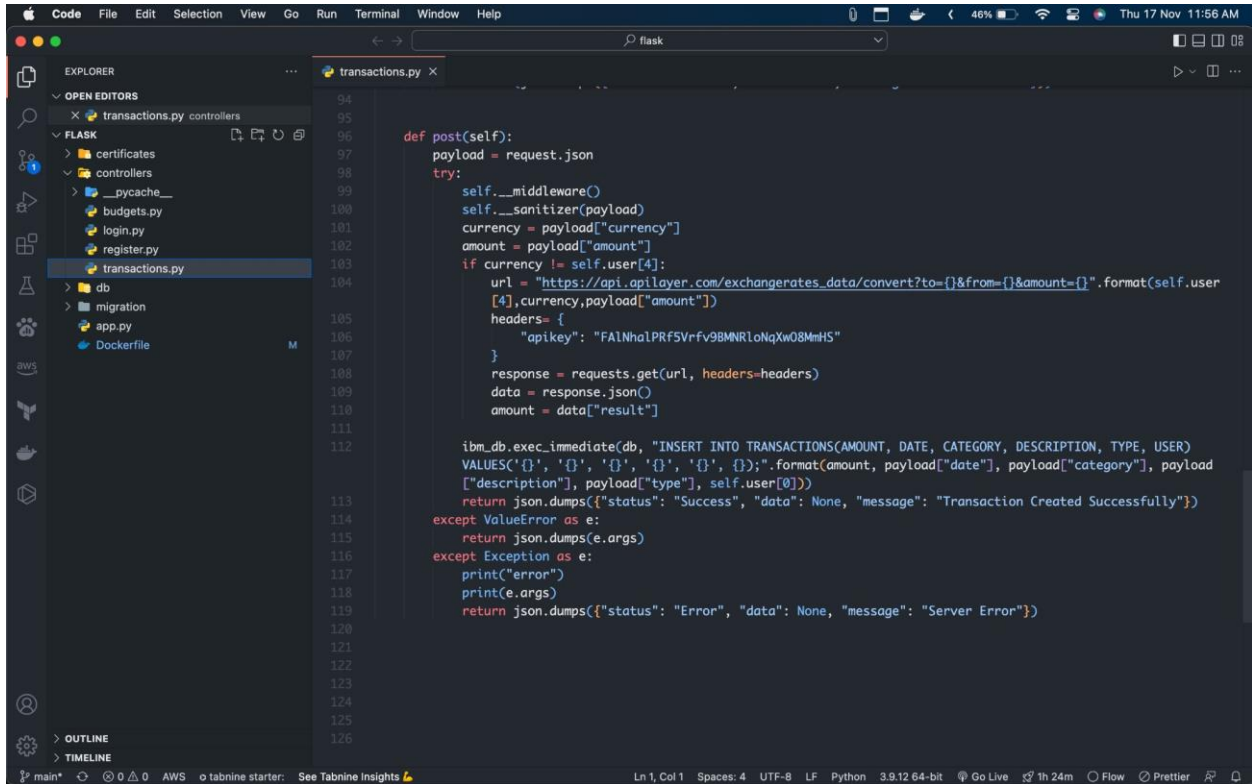
## Flask Code



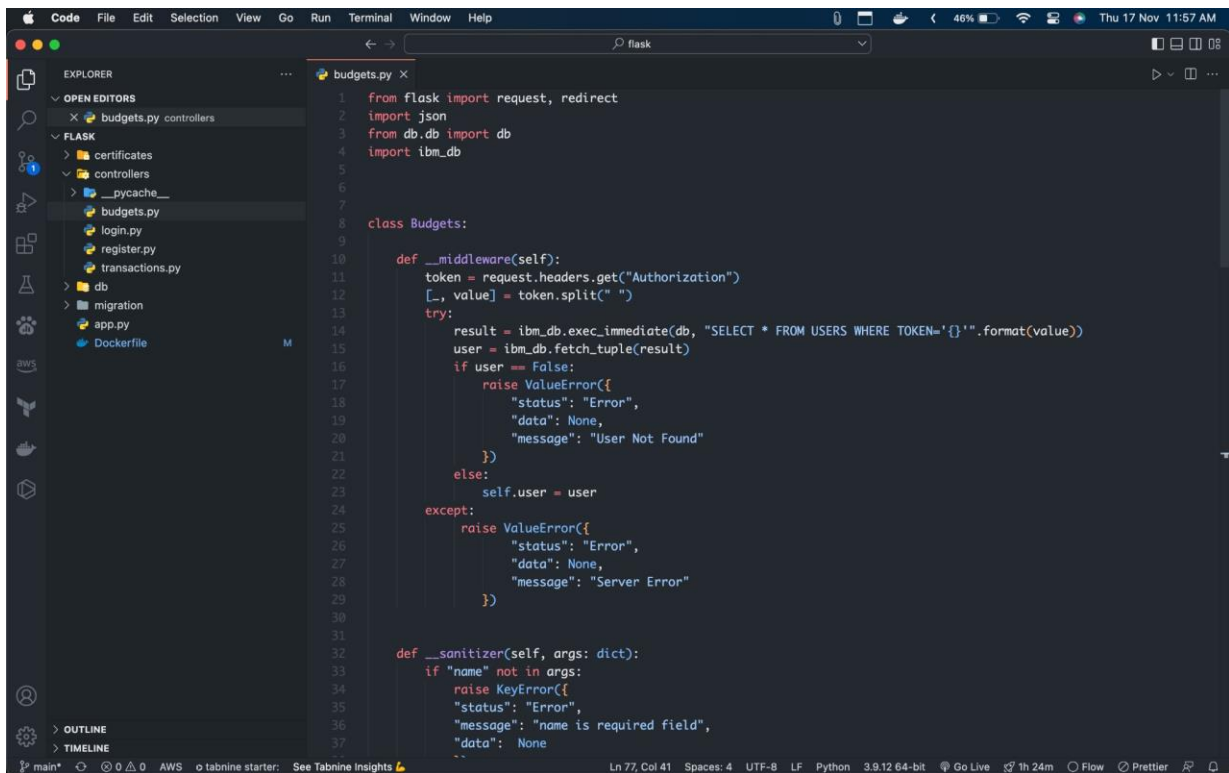
```
1 from flask import request
2 import json
3 from db.db import db
4 import ibm_db
5 import requests
6
7
8 class Transactions:
9
10     def __middleware(self):
11         token = request.headers.get("Authorization")
12         [_, value] = token.split(" ")
13         try:
14             result = ibm_db.exec_immediate(db, "SELECT * FROM USERS WHERE TOKEN='{}'".format(value))
15             user = ibm_db.fetch_tuple(result)
16             if user == False:
17                 raise ValueError({
18                     "status": "Error",
19                     "data": None,
20                     "message": "User Not Found"
21                 })
22             else:
23                 self.user = user
24         except:
25             raise ValueError({
26                 "status": "Error",
27                 "data": None,
28                 "message": "Server Error"
29             })
30
31     def __sanitizer(self, args: dict):
32         if "amount" not in args:
33             raise KeyError({
34                 "status": "Error",
35                 "message": "amount is required field",
36                 "data": None
37             })
```



```
66         "message": "currency is required field",
67         "data": None
68     })
69
70     def get(self):
71         try:
72             self.__middleware()
73             result = ibm_db.exec_immediate(db, "SELECT * FROM TRANSACTIONS WHERE TRANSACTIONS.USER={} ".format(self.user[0]))
74             transactions = ibm_db.fetch_tuple(result)
75             payload = []
76             while (transactions):
77                 payload.append({
78                     "amount": transactions[1],
79                     "date": str(transactions[2]),
80                     "category": transactions[3],
81                     "description": transactions[4],
82                     "type": transactions[5]
83                 })
84                 transactions = ibm_db.fetch_tuple(result)
85             if len(payload) == 0:
86                 return json.dumps({"status": "Error", "data": None, "message": "No Transaction Available"})
87             return json.dumps({"status": "Success", "data": payload, "message": "Transaction Retrived Successfully"})
88         except ValueError as e:
89             return (json.dumps(e.args[0]))
90         except Exception as e:
91             print(e)
92             return (json.dumps({"status": "Error", "data": None, "message": "Server Error"}))
93
94     def post(self):
95         payload = request.json
96         try:
97             self.__middleware()
98             self.__sanitizer(payload)
99             currency = payload["currency"]
100             payload["currency"] = currency
```



```
94
95
96 def post(self):
97     payload = request.json
98     try:
99         self.__middleware()
100         self.__sanitizer(payload)
101         currency = payload["currency"]
102         amount = payload["amount"]
103         if currency != self.user[4]:
104             url = "https://api.apilayer.com/exchangerates_data/convert?to={}&from={}&amount={}".format(self.user
105             [4], currency, payload["amount"])
106             headers = {
107                 "apikey": "FA1Nh1PRf5Vrfv9BMNR1oNqXw08MmHS"
108             }
109             response = requests.get(url, headers=headers)
110             data = response.json()
111             amount = data["result"]
112
113             ibm_db.exec_immediate(db, "INSERT INTO TRANSACTIONS(AMOUNT, DATE, CATEGORY, DESCRIPTION, TYPE, USER)
114             VALUES('{}', '{}', '{}', '{}', '{}', {})".format(amount, payload["date"], payload["category"], payload
115             ["description"], payload["type"], self.user[0]))
116             return json.dumps({"status": "Success", "data": None, "message": "Transaction Created Successfully"})
117         except ValueError as e:
118             return json.dumps(e.args)
119         except Exception as e:
120             print("error")
121             print(e.args)
122             return json.dumps({"status": "Error", "data": None, "message": "Server Error"})
123
124
125
126
```



```
1 from flask import request, redirect
2 import json
3 from db.db import db
4 import ibm_db
5
6
7
8 class Budgets:
9
10     def __middleware(self):
11         token = request.headers.get("Authorization")
12         [_, value] = token.split(" ")
13         try:
14             result = ibm_db.exec_immediate(db, "SELECT * FROM USERS WHERE TOKEN='{}'".format(value))
15             user = ibm_db.fetch_tuple(result)
16             if user == False:
17                 raise ValueError({
18                     "status": "Error",
19                     "data": None,
20                     "message": "User Not Found"
21                 })
22             else:
23                 self.user = user
24         except:
25             raise ValueError({
26                 "status": "Error",
27                 "data": None,
28                 "message": "Server Error"
29             })
30
31
32     def __sanitizer(self, args: dict):
33         if "name" not in args:
34             raise KeyError({
35                 "status": "Error",
36                 "message": "name is required field",
37                 "data": None
38             })
39
```

```
58 def get(self):
59     try:
60         self.__middleware()
61         result = ibm_db.exec_immediate(db, "SELECT * FROM BUDGETS WHERE BUDGETS.USER={}".format(self.user[0]))
62         budgets = ibm_db.fetch_tuple(result)
63         payload = []
64         while (budgets):
65             payload.append({
66                 "name": budgets[0],
67                 "date": str(budgets[1]),
68                 "range": budgets[2],
69                 "limit": budgets[3]
70             })
71             budgets = ibm_db.fetch_tuple(result)
72         print(payload)
73         if len(payload) == 0:
74             return json.dumps({"status": "Error", "data": None, "message": "No Budget Available"})
75         return json.dumps({"status": "Success", "data": payload, "message": "Budget Retrieved Successfully"})
76     except ValueError as e:
77         return json.dumps(e.args[0])
78     except Exception as e:
79         print(e)
80         return json.dumps({"status": "Error", "data": None, "message": "Server Error"})
81
82
83 def post(self):
84     payload = request.json
85     try:
86         self.__middleware()
87         self.__sanitizer(payload)
88         ibm_db.exec_immediate(db, "INSERT INTO BUDGETS(NAME, CREATED_AT, RANGE, LIMIT, USER) VALUES('{}', '{}', '{}', '{}', '{}')".format(payload["name"], payload["date"], payload["range"], payload["limit"], self.user[0]))
89         return json.dumps({"status": "Success", "data": None, "message": "Budget Created Successfully"})
90     except ValueError as e:
91         return json.dumps(e.args)
92     except Exception as e:
```