# DOCUMENTAÇÃO

## Como foi feito

### Armazenamento de Informações

O código foi feito à base de um dicionário que absorvia as informações de um arquivo.csv o qual contém todas as informações referentes às transações.

```python
from Controllers.Constants import DEFAULT_COMMA_CODE
class TransactionController:
  transactions = []

  def get_all_from_db(self):
    with open('transactions.csv') as data:
      db_data = data.read()

      db_data_array = db_data.split(',')
      transactions = []

      for db_transaction in db_data_array:
        transaction = {}
        # Normalizando as víruglas para evitar que conflitem com csv
        normalized_string = db_transaction.replace(DEFAULT_COMMA_CODE, ',')

        # Removendo as divisões entre chaves e valores
        splitted_values = normalized_string.split(';')

        for without_pipes_string in splitted_values:
          # unpacking em dois valores do array 'chave:valor'
          key, value = without_pipes_string.split(':')

          transaction[key] = value

        transactions.append(transaction)

      self.transactions = transactions
      return transactions
```

```python
def list_all_transactions():
    transactions = TransactionController.get_all_from_db(
        TransactionController())
    os.system('cls')

    print("Lista de transações: \n")

    transactionsValues = []
    for i in range(len(transactions)):
        transactionsValues.append([transactions[i]['id'], transactions[i]
                                   ['category'], transactions[i]['name'], transactions[i]['value']])

    print("{: ^5} {: ^20} {: ^20} {: ^20}".format(
        "ID", "Categoria", "Nome", "Valor"))
    print("-" * 65)
    for row in transactionsValues:
        print("{: ^5} {: ^20} {: ^20} {: ^20}".format(*row))

    MainController.return_main_menu()
```

## Funções relacionadas à busca de transações por ID

Funções como “Buscar uma transação por id”, “Deletar uma transação por id”, “Editar uma transação por id” funcionavam em base de uma “comparação” feita pelo programa, do ID indicado pelo usuário e ao ID existente nas transações do dicionário, quando era encontrada uma transação que continha um ID comum ao informado pelo usuário, a mesma

podia ser editada, quando o foco era atualizar informações referentes à ela, diretamente exibida no terminal quando o foco era apenas a visualização da transação em questão e Deletada, quando o foco era retirar tal transação do dicionário.

```
    def search_transaction_by_id():
        os.system('cls')
        print("Buscar transação por id: \n")
        id = input("Id: ")

        transaction = TransactionController.get_transaction_by_id(
            TransactionController(), id)

        if transaction == None:
            print("\nTransação não encontrada!")
            MainController.return_main_menu()
        else:
            print("\nTransação encontrada:")
            print("{: ^5} {: ^20} {: ^20} {: ^20}".format(
                "ID", "Categoria", "Nome", "Valor"))
            print("-" * 65)
            print("{: ^5} {: ^20} {: ^20} {: ^20}".format(
                transaction['id'], transaction['category'], transaction['name'], transaction['value']))

            MainController.return_main_menu()
```

```
            MainController.return_main_menu()

    def delete_transaction_by_id():
        os.system('cls')
        print("Deletar transação por id: \n")
        id = input("Id: ")

        transaction = TransactionController.get_transaction_by_id(
            TransactionController(), id)

        if transaction == None:
            print("\nTransação não encontrada!")
            MainController.return_main_menu()
        else:
            TransactionController.delete_transaction_by_id(TransactionController(), id)
            print("\nTransação deletada com sucesso!")
            MainController.return_main_menu()
```

```
def update_transaction_by_id():
    os.system('cls')
    print("Editar transação por id: \n")
    id = input("Id: ")

    transaction = TransactionController.get_transaction_by_id(
        TransactionController(), id)

    if transaction == None:
        print("\nTransação não encontrada!")
        MainController.return_main_menu()
    else:
        print("\nTransação encontrada:")
        print("{: ^5} {: ^20} {: ^20} {: ^20}".format(
            "ID", "Categoria", "Nome", "Valor"))
        print("-" * 65)
        print("{: ^5} {: ^20} {: ^20} {: ^20}".format(
            transaction['id'], transaction['category'], transaction['name'], transaction['value']))

        print("\nDigite os novos dados da transação: \n")
        category = input("Categoria: ")
        name = input("Nome: ")
        value = input("Valor: ")

        newTransaction = {
            'category': category,
            'name': name,
            'value': value,
            'id': id
        }
```

## Função relacionada à busca de transações por Categoria

A função “Listar transações por categoria” foi feita de forma similar, a principal diferença era que, a início era criada uma lista de transações que iria armazenar todas as transações que tinham uma categoria em comum à indicada pelo usuário, após isso seria então, exibida na tela todas aquelas transações que haviam sido clonadas à lista.

```
    def list_transactions_by_category():
        os.system('cls')
        print("Listar transações por categoria: \n")
        category = input("Categoria: ")

        transactions = TransactionController.get_transactions_by_category(TransactionController(), category)

        if transactions == []:
            print("\nNão há transações nessa categoria!")
            MainController.return_main_menu()
        else:
            print("\nTransações encontradas:")
            print("{: ^5} {: ^20} {: ^20} {: ^20}".format(
                "ID", "Categoria", "Nome", "Valor"))
            print("-" * 65)
            for transaction in transactions:
                print("{: ^5} {: ^20} {: ^20} {: ^20}".format(
                transaction['id'], transaction['category'], transaction['name'], transaction['value']))

            MainController.return_main_menu()
```

## Função relacionada a exibição de transações sem critério

Já a função “Listar todas as transações” é a que possui o funcionamento mais simples dentre todas, já que a mesma foi feita com o foco de apenas exibir no terminal todas as transações que estavam no dicionário, sem nenhum tipo de critério.

```
def list_all_transactions():
    transactions = TransactionController.get_all_from_db(
        TransactionController())
    os.system('cls')

    print("Lista de transações: \n")

    transactionsValues = []
    for i in range(len(transactions)):
        transactionsValues.append([transactions[i]['id'], transactions[i]
                                   ['category'], transactions[i]['name'], transactions[i]['value']])

    print("{: ^5} {: ^20} {: ^20} {: ^20}".format(
        "ID", "Categoria", "Nome", "Valor"))
    print("-" * 65)
    for row in transactionsValues:
        print("{: ^5} {: ^20} {: ^20} {: ^20}".format(*row))

    MainController.return_main_menu()
```

## Função relacionada ao envio de dados por email

Essa função possui um funcionamento um pouco mais complexo devido ao fato de utilizar o módulo ‘smtplib’ do Python para efetuar o envio de todas as transações para o email selecionado pelo usuário, acessando para isso, o servidor do endereço de email informado..

```python
def send_data_by_email(self, email):
  try:
    transactions = self.get_all_from_db()
    transactions_string = '\nSeu resumo de transações: \n\nTotal de transações: ' + str(len(transactions)) + '\n\n'

    for transaction in transactions:
      transactions_string += f'Id da transação: {transaction["id"]}, Nome: {transaction["name"]}, Categoria: {transaction["category"]}, Valor: {transaction["value"]} \n'

    server = smtplib.SMTP('smtp-mail.outlook.com: 587')
    server.starttls()
    server.login("projetoipcesar@outlook.com", "projetoip10")
    server.sendmail("projetoipcesar@outlook.com", email, transactions_string.encode('utf-8'))
    server.quit()
  except:
    print('Erro ao enviar email.')
```

```python
def send_data_by_email():
  os.system('cls')
  print("Enviar dados por e-mail: \n")
  email = input("E-mail: ")

  TransactionController.send_data_by_email(transactionController, email)

  print("\nDados enviados com sucesso!")
  MainController.return_main_menu()
```

# Definições e Escolhas do Grupo

## 1: Arquitetura do Código:

A equipe optou por utilizar a arquitetura MVC (Model-view-controller) para o projeto. Essa arquitetura separa a lógica dos negócios(Model), a exposição dos dados(View) e o controle da interação entre ambos (Controller), o que por sua vez acaba tornando o código em algo mais organizado e de fácil entendimento e manutenção.

## 2: Interface do Usuário:

A equipe decidiu utilizar a biblioteca ‘os’ para limpar a tela do terminal tornando-o mais entendível. Além disso, foram utilizadas as funções input() e print() respectivamente para receber e enviar informações ao usuário.

## 3: Formatação dos Dados Exibidos:

A equipe escolheu exibir os dados em formato de tabela, usando de um alinhamento e espaçamento para cada coluna, o que por sua vez torna a exibição muito mais “limpa” e de fácil entendimento por parte do usuário.

## 4: Banco de Dados:

O banco de dados utilizado pela equipe foi o arquivo.csv que continha todas as informações referentes às transações, de lá que são retiradas tudo aquilo que é exibido ao usuário durante o funcionamento da aplicação.

## 5: Linguagem de Programação:

A linguagem de programação utilizada foi o Python em sua versão “Python 3.9”, tal linguagem encaixou perfeitamente com o projeto graças à sua facilidade de uso, grande abundância de recursos e de extrema adaptação, que por sua vez a torna bastante utilizada atualmente em aplicativos e scripts do gênero.

## 6: Validação e Entrada de Dados

Foram amplamente utilizadas verificações para validar as entradas de dados vindos do usuário, como uma análise do tipo de dado, a existência de valor obrigatórios e a verificação de específicos formatos.