Fernanda Silva Bucheri RA: 135529
Marcos Ferreira Villar Coelho RA: 135534

Implementação de um Banco de Dados no Cloud Firestore

São José dos Campos - Brasil

Agosto de 2021

Fernanda Silva Bucheri RA: 135529
Marcos Ferreira Villar Coelho RA: 135534

Implementação de um Banco de Dados no Cloud Firestore

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Aspectos de Implementação de Banco de Dados.

Docente: Profª. Drª Daniela Leal Musa.

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Agosto de 2021

Resumo

Este trabalho tem como objetivo implementar um banco de dados referente a uma loja utilizando o *Cloud Firestore*, banco de dados da plataforma *Firebase*.

Palavras-chaves: Firebase. Firestore. banco de dados.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Os SGBDs (sistema de gerenciamento de banco de dados) ou simplesmente banco de dados são de suma importância já que são os responsáveis por armazenar inúmeros dados maneira consistente, organizada, protegida e acessível.

Existem duas categorias, os SQL e os NoSQL. Neste trabalho será utilizado o banco de dados *Cloud Firestore*, sendo este NoSQL e orientado a documentos.

2 Objetivos

2.1 Geral

Temos como principal objetivo realizar um estudo sobre o banco de dados NoSQL *Cloud Firestore*.

2.2 Específico

- Criar um banco de dados na plataforma *Firebase*;
- Escrever os códigos responsáveis por criar as coleções utilizadas, o gerenciador das solicitações e o código principal (main);
- Realizar testes como cadastrar, modificar, ativar, desativar e mostrar clientes, produtos e pedidos;

3 Fundamentação Teórica

Um banco de dados pode ser SQL (relacional) ou NoSQL (não relacional). Por muito tempo utilizava-se predominantemente os SQL, mas em meados dos anos 2000 outros modelos foram ganhando espaço, sendo que para diferenciar e categorizar essa nova classe de banco de dados foi criado o termo "NoSQL". (1) (2)

3.0.1 NoSQL

Os bancos de dados NoSQL são muito úteis para aplicativos modernos, como dispositivos móveis, Web e jogos, pois eles possuem alta performance, flexibilidade, escalabilidade e são altamente funcionais. (1) São flexíveis porque não possuem estrutura fixa. Sendo assim, quanto mais dados, mais aumenta-se o número de servidores, podendo estes ser de alta performance ou não, o que otimiza o armazenamento e barateia também. Outro ponto é que possuem um alto grau de distribuição de dados, garantindo assim um número maior de solicitações. Entretanto, o NoSQL não garante a consistência da informação. (3)

Muitas empresas utilizam esse tipo de banco de dados, como por exemplo *Twitter*, *Facebook*, *Amazon* e *Google*.

3.0.2 Firebase

Plataforma adquirida pela empresa Google em 2014 (4), o *Firebase* é um Baas *(Backend as a Service)* para aplicações *Web* e *Mobile* (5), ou seja, torna o desenvolvimento de aplicativos mais rápido e simples.

Ele possui diversos serviços, separados em quatro grandes categorias: *Analytics*, *Develop*, *Grow* e *Earn.*(5)

Dentre esses serviços há duas soluções de banco de dados baseadas em nuvem e acessíveis ao cliente que oferecem suporte à sincronização de dados em tempo real: o *Cloud Firestore* e o *Realtime Database* (6)

3.0.3 Realtime Database

É o BD original do *Firebase*. Uma solução bastante eficiente para aplicativos móveis que requerem dados sincronizados em tempo real. (6)

3.0.4 Cloud Firestore

O *Cloud Firestore* é o foco deste trabalho.

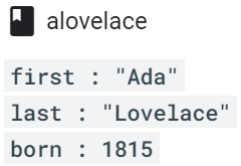
Ele é o mais novo BD da plataforma. Baseia-se nos resultados do *Realtime Database* entretanto seu modelo de dados é mais intuitivo, além de possuir consultas mais rápidas e avançadas e melhor escalabilidade.

Ele é um banco de dados orientado a documentos, o que significa que os dados são armazenados em documentos que são organizados em coleções. Os documentos podem contem subcoleções e objetos aninhados, que podem incluir strings, listas, entre outros. Cada um dos documentos possui um conjunto de pares chave-valor, sendo que o *Firestore* é otimizado para guardar grandes coleções de documentos pequenos. (7)

3.0.4.1 Documentos

Sempre identificados por um nome, os documentos são registros que possuem campos mapeados para valores. A Figura 1 mostra um documento representando o usuário "alovelace".

Figura 1 – Representação de um documento.



Fonte: Firebase (7)

Ainda é possível estruturar o nome do usuário no exemplo da Figura 1 como um mapa (objeto complexo e aninhado em um documento) como mostra a Figura 2.

Figura 2 – Representação de um mapa.

Fonte: Firebase (7)

3.0.4.2 Coleções

Coleções armazenam documentos. Por exemplo, é possível criar uma coleção "users" para armazenar documentos que representam usuários como mostrado na Figura 3.

Figura 3 – Representação de uma coleção.

Fonte: Firebase (7)

Documentos dentro de uma mesma coleção podem contem campos diferentes ou armazenar diferentes tipos de dados em campos iguais.

4 Desenvolvimento

Aqui será detalhado o desenvolvimento do projeto.

4.1 Descrição do Banco de Dados

Foi escolhida como base de dados uma loja virtual. Há três coleções: Clientes, Pedidos e Produtos.

Na coleção Cliente existem os seguintes dados:

- enabled;
- nome;
- idade;
- sexo;
- endereço;
  - rua;
  - número;
  - bairro;
  - cidade;
  - estado;
  - cep;
  - referencia;

Já na coleção Produtos há os seguintes dados:

- enabled;
- tipo;
- nome;
- cor;
- preco;

- emEstoque;
- material;

Por fim, na coleção Pedidos existem os seguintes campos:

- enabled;
- idCliente;
- produtosPedidos;
  - idProduto;
  - qntdProduto;

  [...]
- tipoPagamento;
- statusPagamentoço;
- statusEntrega;
- material;

Vale ressaltar que pode existir um ou mais produtos pedidos.

A variável ”enabled” presente em todas as coleções informa se o documento, ou seja, o cadastro de determinado cliente, produto ou pedido está ativo ou não.

Em todas as coleções há, ainda, um documento com ID ”-1” responsável por armazenar o ultimoID, fazendo com que seja possível obter um controle de quantos documentos foram criados. Isso auxilia na criação de novos ID’s.

5 Resultados Obtidos e Discussões

5.0.1 Códigos da implementação realizada no SGDB

Para realizar a implementação no SGBD foram criados os cinco códigos apresentados abaixo, escritos utilizando a linguagem de programação *Python*.

Inicialmente estão os três códigos (5.1, 5.2 e 5.3) referentes as coleções ”Cliente”, ”Produto” e ”Pedido” , respectivamente.

Posteriormente está o código 5.4 responsável por gerenciar as solicitações feitas ao banco como inclusão, modificação, desativação ou ativação de um documento bem como impressão de todos os dados presentes no banco.

Por fim está o código 5.5 da função ”main” , ou seja, da função principal.

```
1  class Cliente(object):
2      def __init__(self, nome='', idade=0, sexo='', rua='', numero='', bairro='', cidade=''
       ,
3                   estado='', cep='', referencia='', enabled=True):
4          self.nome = nome
5          self.idade = idade
6          self.sexo = sexo
7          self.endereco = self.asdict(rua, numero, bairro, cidade, estado, cep, referencia)
8          self.enabled = enabled
9
10     @staticmethod
11     def asdict(rua, numero, bairro, cidade, estado, cep, referencia):
12         return {'rua': rua, 'numero': numero, 'bairro': bairro,
13                 'cidade': cidade, 'estado': estado, 'cep': cep,
14                 'referencia': referencia}
15
16     @staticmethod
17     def from_dict(source):
18         return Cliente(source['nome'], source['idade'], source['sexo'],
19                        source['endereco']['rua'], source['endereco']['numero'],
20                        source['endereco']['bairro'], source['endereco']['cidade'],
21                        source['endereco']['estado'], source['endereco']['cep'],
22                        source['endereco']['referencia'], source['enabled'])
23
24     def __str__(self):
25         return '{\n' \
26                f'  nome: {self.nome},\n' \
27                f'  idade: {self.idade},\n' \
28                f'  sexo: {self.sexo},\n' \
29                f'  enabled: {self.enabled},\n' \
30                '  endereco: {\n' \
31                f'      rua: {self.endereco["rua"]},\n' \
32                f'      numero: {self.endereco["numero"]},\n' \
33                f'      bairro: {self.endereco["bairro"]},\n' \
34                f'      cidade: {self.endereco["cidade"]},\n' \
35                f'      estado: {self.endereco["estado"]},\n' \
36                f'      cep: {self.endereco["cep"]},\n' \
37                f'      referencia: {self.endereco["referencia"]},\n' \
```

```
38                '   },\n' \
39                '}\n'
```

Code Listing 5.1 – Coleção "Cliente".

```
1  class Produto(object):
2      def __init__(self, tipo='', nome='', cor='', preco=0, emEstoque=0, material='',
           enabled=True):
3          self.tipo = tipo
4          self.nome = nome
5          self.cor = cor
6          self.preco = preco
7          self.emEstoque = emEstoque
8          self.material = material
9          self.enabled = enabled
10
11     @staticmethod
12     def from_dict(source):
13         return Produto(source['tipo'], source['nome'], source['cor'], source['preco'],
14                        source['emEstoque'], source['material'], source['enabled'])
15
16     def __str__(self):
17         return '{\n' \
18                f'  tipo: {self.tipo},\n' \
19                f'  nome: {self.nome},\n' \
20                f'  enabled: {self.enabled},\n' \
21                f'  cor: {self.cor},\n' \
22                f'  preco: {self.preco},\n' \
23                f'  emEstoque: {self.emEstoque},\n' \
24                f'  material: {self.material}\n' \
25                '}'
```

Code Listing 5.2 – Coleção "Produto".

```
1  class Pedido(object):
2      def __init__(self, idCliente=0, idProduto=0, qntdProduto=0, tipoPagamento='',
3                   statusPagamento='', statusEntrega='', enabled=True):
4          self.idCliente = idCliente
5          self.produtosPedidos = [{'idProduto': idProduto, 'qntdProduto': qntdProduto}]
6          self.tipoPagamento = tipoPagamento
7          self.statusPagamento = statusPagamento
8          self.statusEntrega = statusEntrega
9          self.enabled = enabled
10
11     def add_pedido(self, idProduto, qntdProduto):
12         self.produtosPedidos.append({'idProduto': idProduto, 'qntdProduto': qntdProduto})
13
14     @staticmethod
15     def from_dict(source):
16         lista_produtos_pedidos = source.get('produtosPedidos')
17
18         # Define os atributos dos produtosPedidos direto no dict do documento
19         source['idProduto'] = lista_produtos_pedidos[0].get('idProduto')
20         source['qntdProduto'] = lista_produtos_pedidos[0].get('qntdProduto')
21
22         # Cria uma classe passando os atributos na ordem correta
23         pedido = Pedido(source['idCliente'], source['idProduto'], source['qntdProduto'],
24                         source['tipoPagamento'], source['statusPagamento'],
25                         source['statusEntrega'], source['enabled'])
26
```

```
27          # Remove o primeiro elemento da lista que já foi adicionado à classe
28          lista_produtos_pedidos.pop(0)
29
30          # Passa por todos os dicts na lista para adicionar na classe criada
31          for produtoPedido in lista_produtos_pedidos:
32              pedido.add_pedido(produtoPedido.get('idProduto'), produtoPedido.get('
                    qntdProduto'))
33          return pedido
34
35      def __str__(self):
36          produtos_pedidos = '   produtosPedidos: [ \n'
37          for _ in self.produtosPedidos:
38              produtos_pedidos += '       {\n' \
39                                  f'          idProduto: {_.get("idProduto")}, \n' \
40                                  f'          qntdProduto: {_.get("qntdProduto")} \n' \
41                                  '       },\n'
42          produtos_pedidos += '   ],'
43          return '{\n' \
44                 f'   idCliente: {self.idCliente}, \n' \
45                 f'   enabled: {self.enabled}, \n' \
46                 f'{produtos_pedidos} \n' \
47                 f'   tipoPagamento: {self.tipoPagamento}, \n' \
48                 f'   statusPagamento: {self.statusPagamento}, \n' \
49                 f'   statusEntrega: {self.statusEntrega},\n' \
50                 '}\n'
```

Code Listing 5.3 – Coleção "Pedido".

```
1   from Classes.Cliente import Cliente
2   from Classes.Pedido import Pedido
3   from Classes.Produto import Produto
4
5
6   def ultimo_id(collection, db):
7       # Abre uma conexão com essa classe no firestore
8       connection = db.collection(collection)
9
10      # Recebe qual foi o último id adicionado nessa coleção
11      return connection.document('-1').get().to_dict().get('ultimoId')
12
13
14  def inicializa_conexao(entrada, db):
15      # Pega o nome da collection a partir do nome da classe
16      collection_name = entrada.__class__.__name__
17
18      # Abre uma conexão com essa classe no firestore
19      connection = db.collection(collection_name)
20
21      # Recebe qual foi o último id adicionado nessa coleção
22      document_id = connection.document('-1').get().to_dict().get('ultimoId')
23
24      return connection, document_id
25
26
27  def adiciona_entrada(nova_entrada, db):
28      connection, document_id = inicializa_conexao(nova_entrada, db)
29
30      # Adiciona um à esse id e transforma pra string novamente
31      document_id = str(document_id + 1)
32
```

```
33      # Adiciona uma nova entrada no id definido
34      connection.document(document_id).set(vars(nova_entrada))
35
36      print(f"Entrada adicionada!")
37
38      # Aumenta em 1 o último id adicionado
39      connection.document('-1').update({'ultimoId': int(document_id)})
40
41
42  def encontra_id(id, collection, db):
43      try:
44          # Abre uma conexão com essa collection no firestore
45          connection = db.collection(collection)
46
47          # Acessa o documento com o id especificado
48          doc = connection.document(str(id))
49          if doc.get().exists:
50              return connection, doc
51          return None
52      except Exception as e:
53          print(e)
54
55
56  def modifica_documento(id, changes_dict, collection, db):
57      connection, doc = encontra_id(id, collection, db)
58
59      if doc is None:
60          print('Id não encontrado!')
61          return 1
62
63      print('O documento a seguir vai ser modificado:')
64      print(doc.get().to_dict())
65
66      # Pega todas os atributos da coleção que o usuário quer alterar
67      possible_names = globals()[collection]().__dict__.keys()
68
69      lista_endereco = ['rua', 'numero', 'bairro', 'cidade', 'estado', 'cep', 'referencia']
70
71      if collection == 'Cliente':
72          endereco_atual = doc.get().to_dict().get('endereco')
73
74      # Passa por todas as entradas das mudanças no changes_dict
75      for key in changes_dict.keys():
76          # Checa se essa chave da entrada é válida para essa coleção
77          if key in possible_names:
78              # Nos casos do endereco e produtosPedidos, verifica se os atributos
                    correspondem ao \
79              # esperado e atualizam o request do usuário
80              if key == 'endereco' and collection == 'Cliente':
81                  for atributo in changes_dict.get(key):
82                      if atributo in lista_endereco:
83                          endereco_atual[atributo] = changes_dict.get(key).get(atributo)
84                      else:
85                          print(f'{atributo} não é um atributo do endereco e foi
                                desconsiderado')
86
87                  # Atualiza a mudança do endereço no endereço atual do documento
88                  changes_dict[key] = endereco_atual
89
90              # Atualiza esse atributo
```

```
91              doc.update({key: changes_dict.get(key)})
92
93      print('O documento foi alterado:')
94      print(doc.get().to_dict())
95      return 0
96
97
98  def deleta_id(id, collection, db):
99      connection, doc = encontra_id(id, collection, db)
100
101     if doc is None:
102         return 'Id não encontrado!'
103
104     deleted = doc.get().to_dict()
105     doc.delete()
106
107     print('O documento a seguir foi deletado:')
108     print(deleted)
109
110     print('Documento deletado')
111
112
113 def deleta_nome(nome, collection, db):
114     if collection == 'Pedido':
115         return 'A collection Pedido não possui o atributo nome'
116
117     # Abre uma conexão com essa classe no firestore
118     connection = db.collection(collection)
119
120     try:
121         # Recebe todos os documentos com o filtro passado
122         docs = connection.where("nome", "==", nome).get()
123         for doc in docs:
124             # Verifica se o documento existe antes de tentar deletar
125             if doc.exists:
126                 deleted = doc.to_dict()
127                 connection.document(doc.id).delete()
128                 print('O documento a seguir foi deletado:')
129                 print(deleted)
130             else:
131                 print('Nome não encontrado!')
132                 return
133         print('Todas as entradas foram deletadas')
134     except Exception as e:
135         print(e)
136
137
138 def desativa_id(id, collection, db):
139     connection, doc = encontra_id(id, collection, db)
140
141     if doc is None:
142         return 'Id não encontrado!'
143
144     deleted = doc.get().to_dict()
145     doc.update({'enabled': False})
146
147     print('O documento a seguir foi desativado (enabled setado para false):')
148     print(deleted)
149
150     print('Documento desativado')
```

```
151
152
153 def desativa_nome(nome, collection, db):
154     if collection == 'Pedido':
155         return 'A collection Pedido não possui o atributo nome'
156
157     # Abre uma conexão com essa classe no firestore
158     connection = db.collection(collection)
159
160     try:
161         # Recebe todos os documentos com o filtro passado
162         docs = connection.where("nome", "==", nome).get()
163         for doc in docs:
164             # Verifica se o documento existe antes de tentar deletar
165             if doc.exists:
166                 deleted = doc.to_dict()
167                 connection.document(doc.id).update({'enabled': False})
168                 print('O documento a seguir foi desativado (enabled setado para false):')
169                 print(deleted)
170             else:
171                 print('Nome não encontrado!')
172                 return
173         print('Todas as entradas foram desativadas')
174     except Exception as e:
175         print(e)
176
177
178 def ativa_id(id, collection, db):
179     connection, doc = encontra_id(id, collection, db)
180
181     if doc is None:
182         return 'Id não encontrado!'
183
184     deleted = doc.get().to_dict()
185     doc.update({'enabled': True})
186
187     print('O documento a seguir foi ativado (enabled setado para true):')
188     print(deleted)
189
190     print('Documento ativado')
191
192
193 def ativa_nome(nome, collection, db):
194     if collection == 'Pedido':
195         return 'A collection Pedido não possui o atributo nome'
196
197     # Abre uma conexão com essa classe no firestore
198     connection = db.collection(collection)
199
200     try:
201         # Recebe todos os documentos com o filtro passado
202         docs = connection.where("nome", "==", nome).get()
203         for doc in docs:
204             # Verifica se o documento existe antes de tentar deletar
205             if doc.exists:
206                 deleted = doc.to_dict()
207                 connection.document(doc.id).update({'enabled': True})
208                 print('O documento a seguir foi ativado (enabled setado para true):')
209                 print(deleted)
210             else:
```

```
211                 print('Nome não encontrado!')
212                 return
213         print('Todas as entradas foram ativadas')
214     except Exception as e:
215         print(e)
216
217
218 def imprime_todos_pedidos_completo(db):
219     try:
220         # Abre uma conexão com essa collection no firestore
221         connection = db.collection('Pedido')
222
223         # Acessa todos os documentos de pedidos existentes
224         docs = connection.get()
225
226         for doc in docs:
227             if doc.exists and doc.id != '-1':
228                 # Transforma o documento para dicionário
229                 doc_dict = doc.to_dict()
230
231                 # Cria uma classe a partir desse dicionário
232                 pedido = Pedido.from_dict(doc_dict)
233
234                 # Procura pelo cliente correspondente ao pedido e passa pra uma classe
235                 matching_cliente = db.collection('Cliente').document(
236                     str(doc_dict['idCliente'])).get().to_dict()
237                 classe_cliente = Cliente.from_dict(matching_cliente)
238
239                 # Cria uma lista de classe dos produtos pedidos nesse pedido
240                 matching_produtos = []
241                 for produto in pedido.produtosPedidos:
242                     matching_produto = db.collection('Produto').document(
243                         str(produto['idProduto'])).get().to_dict()
244                     produto_classe = Produto.from_dict(matching_produto)
245                     matching_produtos.append(produto_classe)
246                 print('------------------------------\n')
247                 print("Pedido: ")
248                 print(str(pedido))
249                 print('Cliente:')
250                 print(str(classe_cliente))
251                 print('Produtos pedidos:')
252                 for produto in matching_produtos:
253                     print(str(produto))
254
255     except Exception as e:
256         print(e)
```

Code Listing 5.4 – Código do gerenciador das solicitações ao banco.

```
1  import sys
2
3  import firebase_admin
4  from firebase_admin import credentials, firestore
5
6  from Classes.Cliente import Cliente
7  from Classes.Pedido import Pedido
8  from Classes.Produto import Produto
9  from db_manager import adiciona_entrada, ultimo_id, imprime_todos_pedidos_completo,
       desativa_id, \
10     desativa_nome, ativa_id, ativa_nome, modifica_documento
```

```
11
12 # Inicializa a conexão com o firestore
13 cred = credentials.Certificate('trabalhoaibdunifesp-firebase-adminsdk-vv0n2-aef0f37c31.
       json')
14 firebase_admin.initialize_app(cred)
15 db = firestore.client()
16
17
18 def get_input(lower_limit, higher_limit, prompt=''):
19     value = input(prompt)
20     while not value.isnumeric() or not lower_limit <= int(value) <= higher_limit:
21         print(f"'{value}' não está entre {lower_limit} e {higher_limit}")
22         value = input(f"Entre com um numero entre {lower_limit} a {higher_limit}: ")
23     return int(value)
24
25
26 def get_input_in_list(user_list, prompt=''):
27     value = input(prompt)
28     while value not in user_list:
29         value = input(f"Entre com um valor que esteja em {user_list}: ")
30     return str(value)
31
32
33 def op_adicionar_entrada():
34     def adicionar_cliente():
35         nome = str(input('Entre com um nome: '))
36         idade = get_input(0, 130, 'Entre com a idade: ')
37         sexo = str(input('Entre com o sexo: '))
38         rua = str(input('Entre com a rua: '))
39         numero = str(input('Entre com o numero: '))
40         bairro = str(input('Entre com o bairro: '))
41         cidade = str(input('Entre com a cidade: '))
42         estado = str(input('Entre com o estado: '))
43         cep = str(input('Entre com o cep: '))
44         referencia = str(input('Entre com a referencia: '))
45         novo_cliente = Cliente(nome, idade, sexo, rua, numero, bairro, cidade,
46                                estado, cep, referencia, True)
47         return novo_cliente
48
49     def adicionar_pedido():
50         max_id_cliente = ultimo_id('Pedido', db)
51         max_id_produto = ultimo_id('Produto', db)
52         id_cliente = get_input(0, max_id_cliente, 'Entre com o id do cliente: ')
53         id_produto = get_input(0, max_id_produto, 'Entre com o id do produto comprado: ')
54         qntd_produto = get_input(0, 1_000, 'Entre com a quantidade do produto comprado: '
               )
55         tipo_pagamento = str(input('Entre com o tipo de pagamento do pedido: '))
56         status_pagamento = str(input('Entre com o status do pagamento: '))
57         status_entrega = str(input('Entre com o status da entrega: '))
58
59         novo_pedido = Pedido(id_cliente, id_produto, qntd_produto, tipo_pagamento,
               status_pagamento,
60                              status_entrega, True)
61
62         if get_input(0, 1, 'Deseja adicionar mais algum produto comprado nesse pedido?'):
63             quantos_novos_produtos = get_input(0, 10, 'Adicionar quantos novos produtos?'
                   )
64             for i in range(quantos_novos_produtos):
65                 id_produto = get_input(0, max_id_produto, 'Entre com o id do produto
                       comprado: ')
```

```
66                  qntd_produto = get_input(0, 1_000, 'Entre com a quantidade do produto
                        comprado: ')
67                  novo_pedido.add_pedido(id_produto, qntd_produto)
68
69          return novo_pedido
70
71      def adicionar_produto():
72          tipo = str(input('Entre com o tipo do produto: '))
73          nome = str(input('Entre com o nome: '))
74          cor = str(input('Entre com a cor: '))
75          preco = get_input(10, 100_000, 'Entre com o preco: ')
76          estoque = get_input(0, 10_000, 'Entre a quantidade que existe no estoque: ')
77          material = str(input('Entre com o material: '))
78          novo_produto = Produto(tipo, nome, cor, preco, estoque, material, True)
79          return novo_produto
80
81      def entradas():
82          print('1. Adicionar um Cliente')
83          print('2. Adicionar um Pedido')
84          print('3. Adicionar um Produto')
85          print('0. Voltar')
86
87      entradas()
88      op = get_input(0, 3, 'Selecione uma opção: ')
89      while op != 0:
90          if op == 1:
91              adiciona_entrada(adicionar_cliente(), db)
92          if op == 2:
93              adiciona_entrada(adicionar_pedido(), db)
94          if op == 3:
95              adiciona_entrada(adicionar_produto(), db)
96          if op == 0:
97              return
98          entradas()
99          op = get_input(0, 3, 'Selecione uma opção: ')
100
101
102 def op_modificar_entrada():
103     collection = get_input_in_list(['Cliente', 'Pedido', 'Produto'],
104                                    prompt='Deseja modificar um Cliente, Pedido ou Produto
                                           : ')
105     max_id = ultimo_id(collection, db)
106     qual_id = get_input(0, max_id, 'Entre com um id para modificar: ')
107     changes_dict = {}
108     if collection == 'Cliente':
109         possible_names = globals()[collection]().__dict__.keys()
110         key = get_input_in_list(possible_names, prompt='Qual chave vai alterar: ')
111         if key == 'idade':
112             mod = get_input(0, 130, 'Entre com a idade: ')
113         else:
114             mod = str(input('Entre o valor: '))
115         changes_dict = {key: mod}
116     elif collection == 'Produto':
117         possible_names = globals()[collection]().__dict__.keys()
118         key = get_input_in_list(possible_names, prompt='Qual chave vai alterar: ')
119         if key == 'preco':
120             mod = get_input(10, 100_000, 'Entre com o preco: ')
121         elif key == 'estoque':
122             mod = get_input(0, 10_000, 'Entre a quantidade que existe no estoque: ')
123         else:
```

```
124             mod = str(input('Entre o valor: '))
125         changes_dict = {key: mod}
126     elif collection == 'Pedido':
127         key = get_input_in_list(['statusPagamento', 'statusEntrega'],
128                                 prompt='Qual chave vai alterar: ')
129         mod = str(input('Entre o valor: '))
130         changes_dict = {key: mod}
131     modifica_documento(qual_id, changes_dict, collection, db)
132
133
134 def op_desativar_entrada():
135     def desativar_por_id():
136         collection = get_input_in_list(['Cliente', 'Pedido', 'Produto'],
137                                        prompt='Deseja desativar um Cliente, Pedido ou
                                                   Produto: ')
138         max_id = ultimo_id(collection, db)
139         qual_id = get_input(0, max_id, 'Entre com um id para desativar: ')
140         desativa_id(qual_id, collection, db)
141
142     def desativar_por_nome():
143         collection = get_input_in_list(['Cliente', 'Pedido', 'Produto'],
144                                        prompt='Deseja desativar um Cliente, Pedido ou
                                                   Produto: ')
145         nome = str(input('Entre com um nome para desativar: '))
146         desativa_nome(nome, collection, db)
147
148     def entradas():
149         print('1. Desativar por id')
150         print('2. Desativar por nome (todos que tiverem esse nome)')
151         print('0. Voltar')
152
153     entradas()
154     op = get_input(0, 2, 'Selecione uma opção: ')
155     while op != 0:
156         if op == 1:
157             desativar_por_id()
158         if op == 2:
159             desativar_por_nome()
160         if op == 0:
161             return
162         entradas()
163         op = get_input(0, 2, 'Selecione uma opção: ')
164
165
166 def op_ativar_entrada():
167     def ativar_por_id():
168         collection = get_input_in_list(['Cliente', 'Pedido', 'Produto'],
169                                        prompt='Deseja ativar um Cliente, Pedido ou
                                                   Produto: ')
170         max_id = ultimo_id(collection, db)
171         qual_id = get_input(0, max_id, 'Entre com um id para ativar: ')
172         ativa_id(qual_id, collection, db)
173
174     def ativar_por_nome():
175         collection = get_input_in_list(['Cliente', 'Pedido', 'Produto'],
176                                        prompt='Deseja ativar um Cliente, Pedido ou
                                                   Produto: ')
177         nome = str(input('Entre com um nome para ativar: '))
178         ativa_nome(nome, collection, db)
179
```

```
180     def entradas():
181         print('1. Ativar por id')
182         print('2. Ativar por nome (todos que tiverem esse nome)')
183         print('0. Voltar')
184 
185     entradas()
186     op = get_input(0, 2, 'Selecione uma opção: ')
187     while op != 0:
188         if op == 1:
189             ativar_por_id()
190         if op == 2:
191             ativar_por_nome()
192         if op == 0:
193             return
194         entradas()
195         op = get_input(0, 2, 'Selecione uma opção: ')
196 
197 
198 def menu_usuario():
199     def entradas():
200         print('1. Adicionar uma nova entrada')
201         print('2. Modificar uma entrada')
202         print('3. Ativar uma entrada')
203         print('4. Desativar uma entrada')
204         print('5. Printar todos os pedidos completos (com cliente e produto)')
205         print('0. Sair')
206 
207     entradas()
208     op = get_input(0, 5, 'Selecione uma opção: ')
209     while op != 0:
210         if op == 1:
211             op_adicionar_entrada()
212         if op == 2:
213             op_modificar_entrada()
214         if op == 3:
215             op_ativar_entrada()
216         if op == 4:
217             op_desativar_entrada()
218         if op == 5:
219             imprime_todos_pedidos_completo(db)
220         if op == 0:
221             sys.exit()
222         entradas()
223         op = get_input(0, 5, 'Selecione uma opção: ')
224 
225 
226 def main():
227     menu_usuario()
228 
229 
230 if __name__ == "__main__":
231     main()
```

Code Listing 5.5 – Código principal (Main).

5.0.2 Testes realizados

Para demonstrar o funcionamento do projeto foram realizados alguns testes.

Os comandos foram passados através do *Prompt* de comando e as mudanças foram verificadas através do *Prompt* e do *Firebase*.

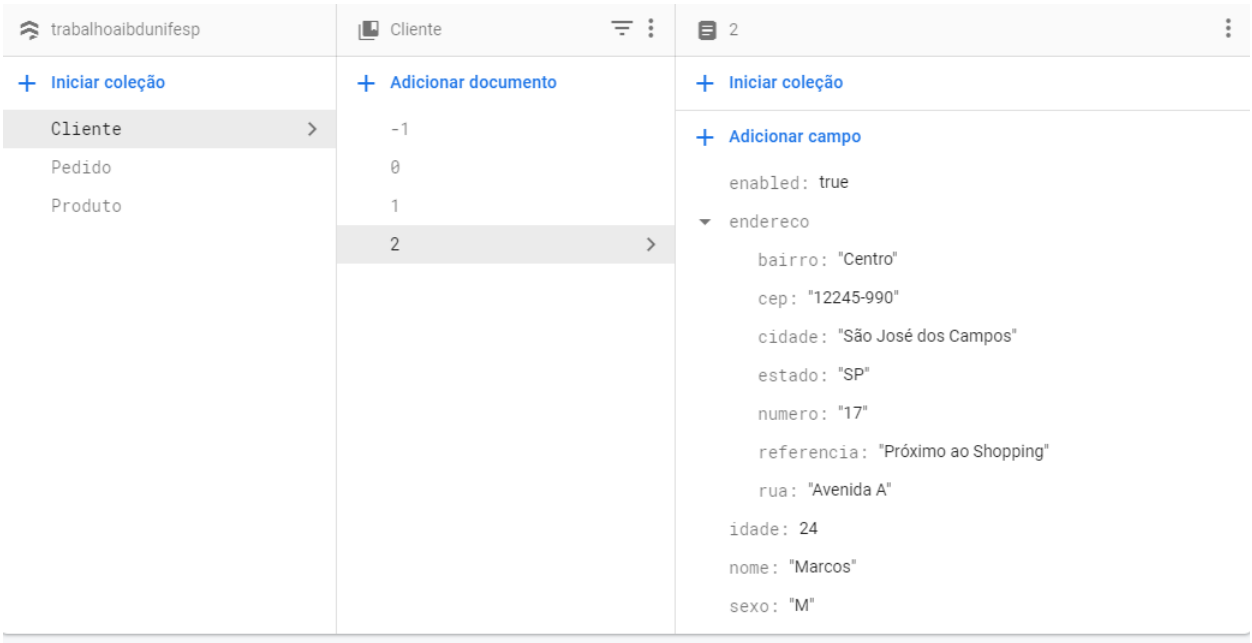
Primeiramente foi cadastrado um novo cliente, como mostrado na Figura 4. Posteriormente foi verificado que de fato o novo cliente estava constando no banco de dados na plataforma do *Firebase* como mostra a Figura 5.

Figura 4 – Cadastro de um novo cliente.

```
1. Adicionar uma nova entrada
2. Modificar uma entrada
3. Ativar uma entrada
4. Desativar uma entrada
5. Printar todos os pedidos completos (com cliente e produto)
0. Sair
Selecione uma opção: 1
1. Adicionar um Cliente
2. Adicionar um Pedido
3. Adicionar um Produto
0. Voltar
Selecione uma opção: 1
Entre com um nome: Marcos
Entre com a idade: 24
Entre com o sexo: M
Entre com a rua: Avenida A
Entre com o numero: 17
Entre com o bairro: Centro
Entre com a cidade: São José dos Campos
Entre com o estado: SP
Entre com o cep: 12245-990
Entre com a referencia: Próximo ao Shopping
Entrada adicionada!
```

Fonte: Os autores.

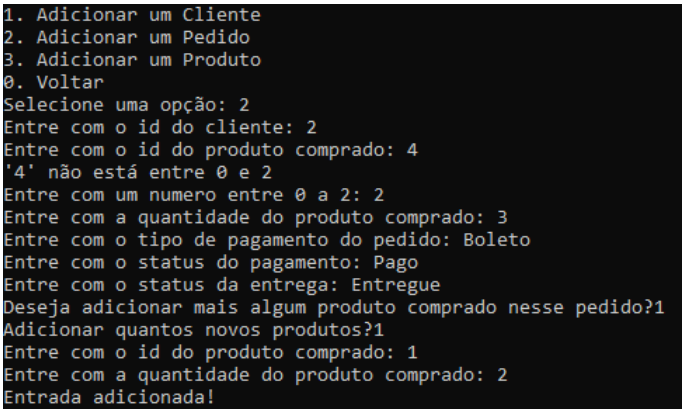
Figura 5 – Verificação do cadastro realizado no banco de dados na plataforma *Firebase*.



Fonte: Os autores.

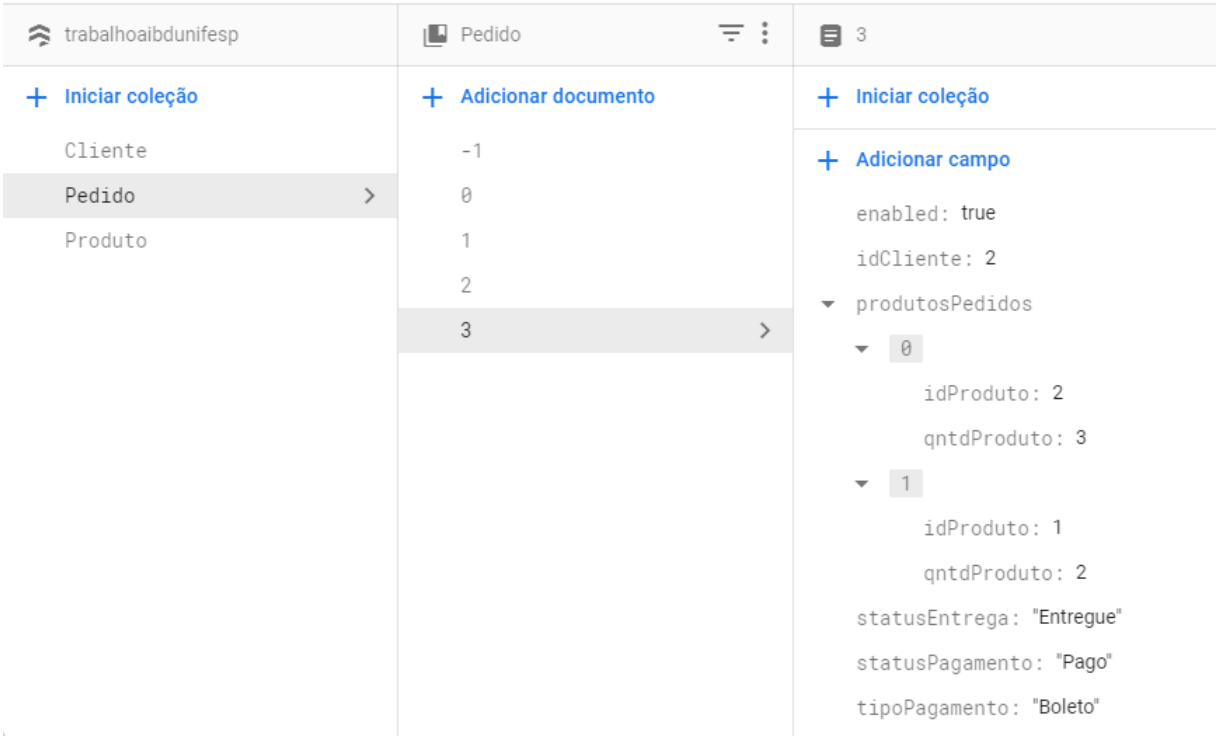
Então foi realizado um novo pedido, como mostrado na Figura 6 e Figura 7.

Figura 6 – Cadastro de um novo pedido.



Fonte: Os autores.

Figura 7 – Verificação do pedido realizado no banco de dados na plataforma *Firebase*.



Fonte: Os autores.

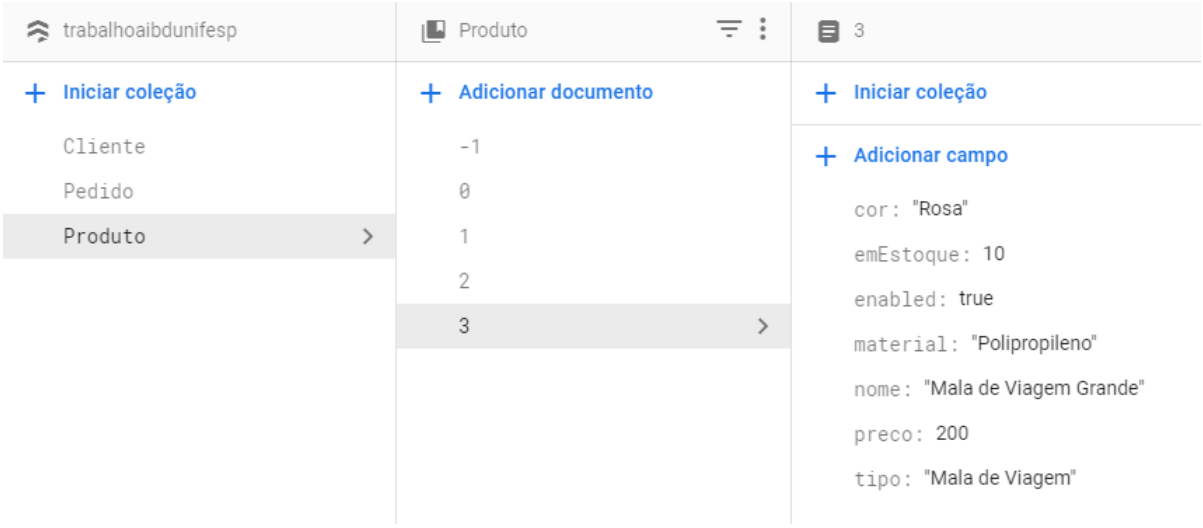
Na sequência foi cadastrado um novo produto, como mostrado na Figura 8 e Figura 9.

Figura 8 – Cadastro de um novo produto.

```
1. Adicionar um Cliente
2. Adicionar um Pedido
3. Adicionar um Produto
0. Voltar
Selecione uma opção: 3
Entre com o tipo do produto: Mala de Viagem
Entre com o nome: Mala de Viagem Grande
Entre com a cor: Rosa
Entre com o preco: 200
Entre a quantidade que existe no estoque: 10
Entre com o material: Polipropileno
Entrada adicionada!
```

Fonte: Os autores.

Figura 9 – Verificação do novo produto no banco de dados na plataforma *Firebase*.



Fonte: Os autores.

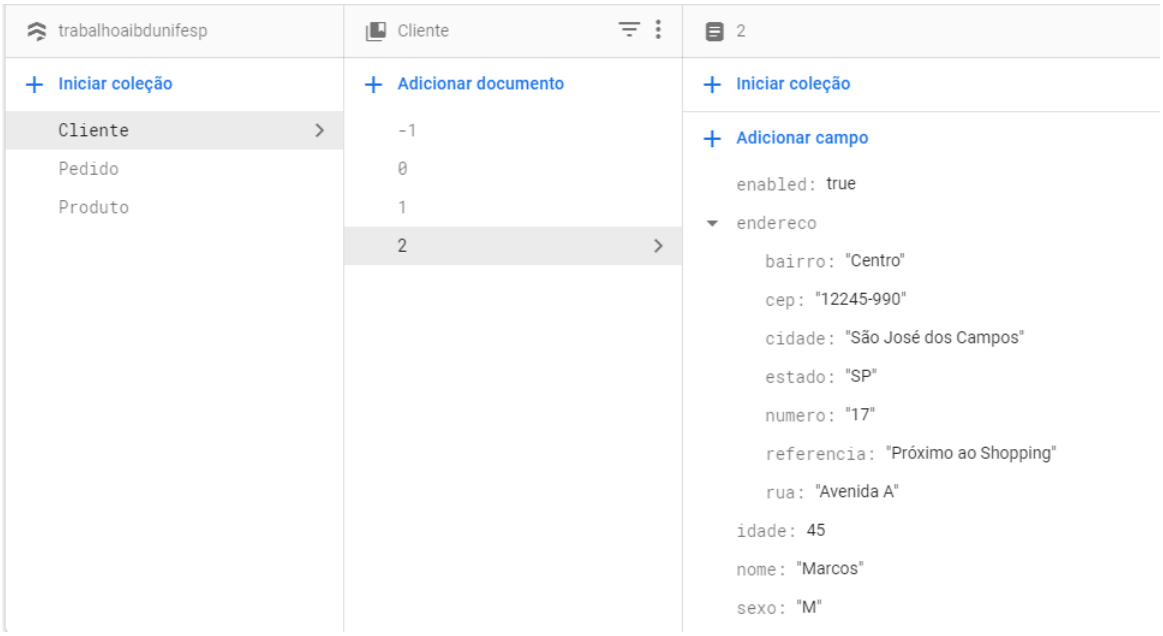
Foi solicitado para modificar o cadastro de um cliente, especificamente a idade, como mostra a Figura 10 e Figura 11.

Figura 10 – Modificação de um cadastro.

```
1. Adicionar um Cliente
2. Adicionar um Pedido
3. Adicionar um Produto
0. Voltar
Selecione uma opção: 0
1. Adicionar uma nova entrada
2. Modificar uma entrada
3. Ativar uma entrada
4. Desativar uma entrada
5. Printar todos os pedidos completos (com cliente e produto)
0. Sair
Selecione uma opção: 2
Deseja modificar um Cliente, Pedido ou Produto: Cliente
Entre com um id para modificar: 2
Qual chave vai alterar: idade
Entre com a idade: 45
O documento a seguir vai ser modificado:
{'idade': 24, 'sexo': 'M', 'nome': 'Marcos', 'endereco': {'cidade': 'São José dos Campos', 'referencia': 'Próximo ao Shopping', 'rua': 'Avenida A', 'numero': '17', 'estado': 'SP', 'cep': '12245-990', 'bairro': 'Centro'}, 'enabled': True}
O documento foi alterado:
{'idade': 45, 'nome': 'Marcos', 'sexo': 'M', 'endereco': {'referencia': 'Próximo ao Shopping', 'cep': '12245-990', 'bairro': 'Centro', 'estado': 'SP', 'cidade': 'São José dos Campos', 'numero': '17', 'rua': 'Avenida A'}, 'enabled': True}
```

Fonte: Os autores.

Figura 11 – Verificação da modificação realizada em um cadastro no banco de dados na plataforma *Firebase*.



Fonte: Os autores.

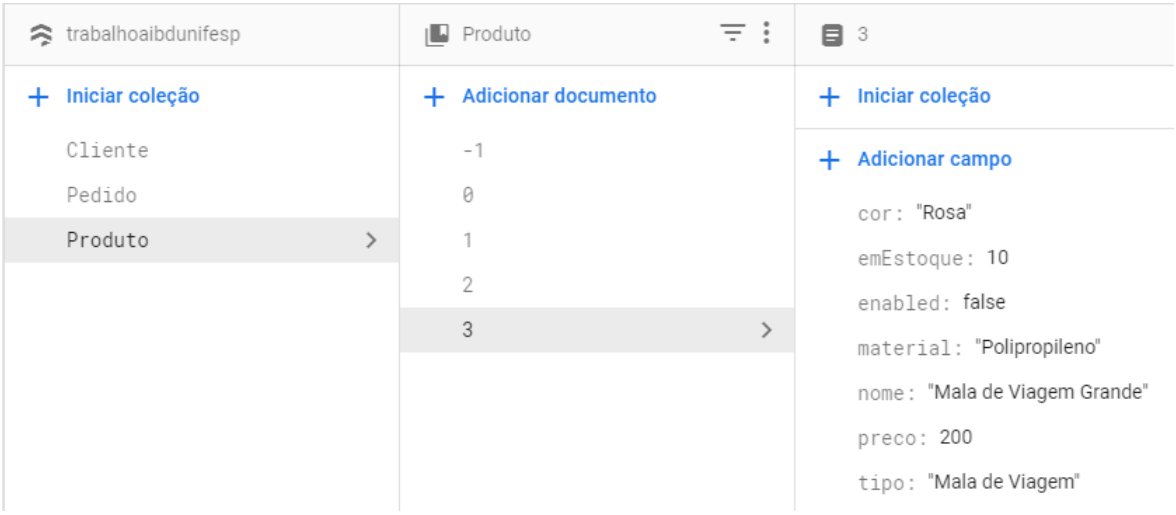
Depois foi desativado um documento através de uma busca por ID como ilustrado na Figura 12 e Figura 13.

Figura 12 – Desativação de um produto.

```
1. Adicionar uma nova entrada
2. Modificar uma entrada
3. Ativar uma entrada
4. Desativar uma entrada
5. Printar todos os pedidos completos (com cliente e produto)
0. Sair
Selecione uma opção: 4
1. Desativar por id
2. Desativar por nome (todos que tiverem esse nome)
0. Voltar
Selecione uma opção: 1
Deseja desativar um Cliente, Pedido ou Produto: Produto
Entre com um id para desativar: 3
O documento a seguir foi desativado (enabled setado para false):
{'emEstoque': 10, 'preco': 200, 'nome': 'Mala de Viagem Grande', 'enabled': True, 'tipo': 'Mala de Viagem', 'cor': 'Rosa', 'material': 'Polipropileno'}
Documento desativado
```

Fonte: Os autores.

Figura 13 – Verificação da desativação de um produto no banco de dados na plataforma *Firebase*.



Fonte: Os autores.

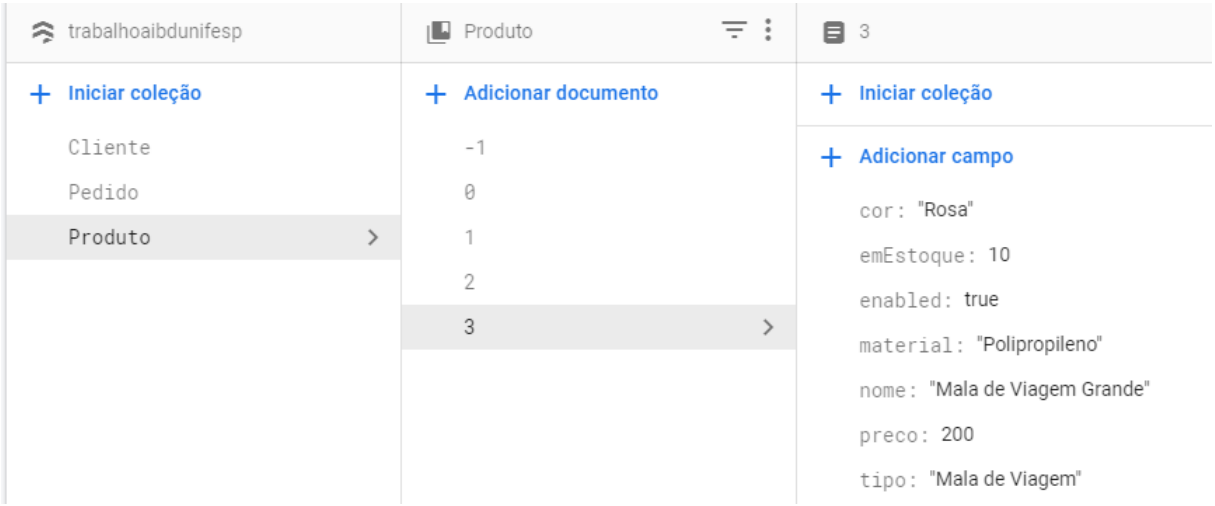
Posteriormente foram ativados todos os documentos com o nome ”Mala de Viagem Grande” como se pode observar na Figura 14 e Figura 15.

Figura 14 – Ativação de produtos.

```
1. Adicionar uma nova entrada
2. Modificar uma entrada
3. Ativar uma entrada
4. Desativar uma entrada
5. Printar todos os pedidos completos (com cliente e produto)
0. Sair
Selecione uma opção: 3
1. Ativar por id
2. Ativar por nome (todos que tiverem esse nome)
0. Voltar
Selecione uma opção: 2
Deseja ativar um Cliente, Pedido ou Produto: Produto
Entre com um nome para ativar: Mala de Viagem Grande
O documento a seguir foi ativado (enabled setado para true):
{'emEstoque': 10, 'preco': 200, 'nome': 'Mala de Viagem Grande', 'enabled': False, 'tipo': 'Mala de Viagem', 'cor': 'Rosa', 'material': 'Polipropileno'}
Todas as entradas foram ativadas
```

Fonte: Os autores.

Figura 15 – Verificação da ativação de produtos realizada no banco de dados na plataforma *Firebase*.



Fonte: Os autores.

Por fim, foi solicitado para imprimir todos os dados no banco de dados. As Figura 16, Figura 17, Figura 18, Figura 19 e Figura 20 mostram os dados impressos.

Figura 16 – Impressão de todos os documentos no banco de dados - parte 1.

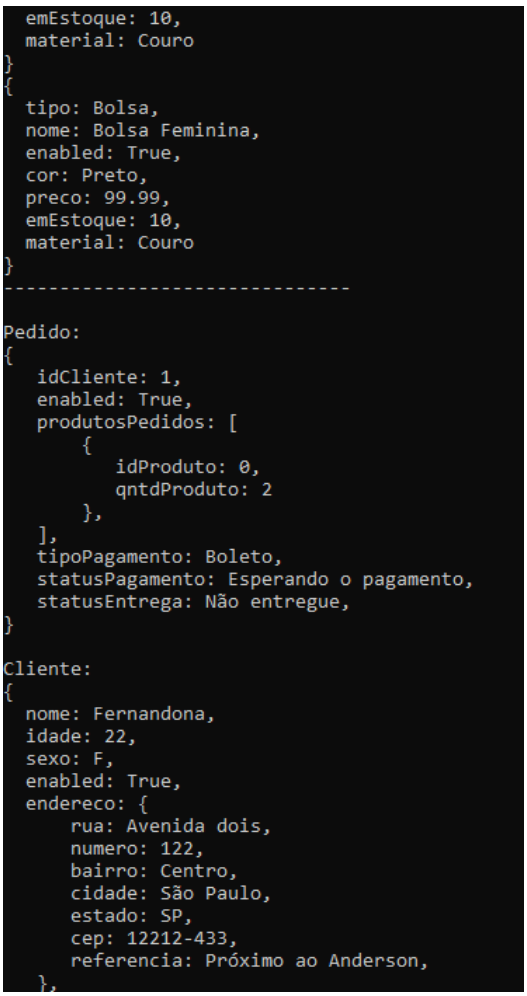
```
Pedido:
{
   idCliente: 0,
   enabled: True,
   produtosPedidos: [
       {
          idProduto: 0,
          qntdProduto: 1
       },
       {
          idProduto: 1,
          qntdProduto: 13
       },
   ],
   tipoPagamento: Boleto,
   statusPagamento: Esperando o pagamento,
   statusEntrega: Não entregue,
}

Cliente:
{
  nome: Maria,
  idade: 22,
  sexo: F,
  enabled: True,
  endereco: {
      rua: Avenida um,
      numero: 122,
      bairro: Centro,
      cidade: São Paulo,
      estado: SP,
      cep: 12212-422,
      referencia: Próximo ao Shopping,
   },
}

Produtos pedidos:
{
  tipo: Necessaire,
  nome: Necessaire Feminina,
  enabled: True,
  cor: Preto,
  preco: 99.99,
  emEstoque: 10,
```

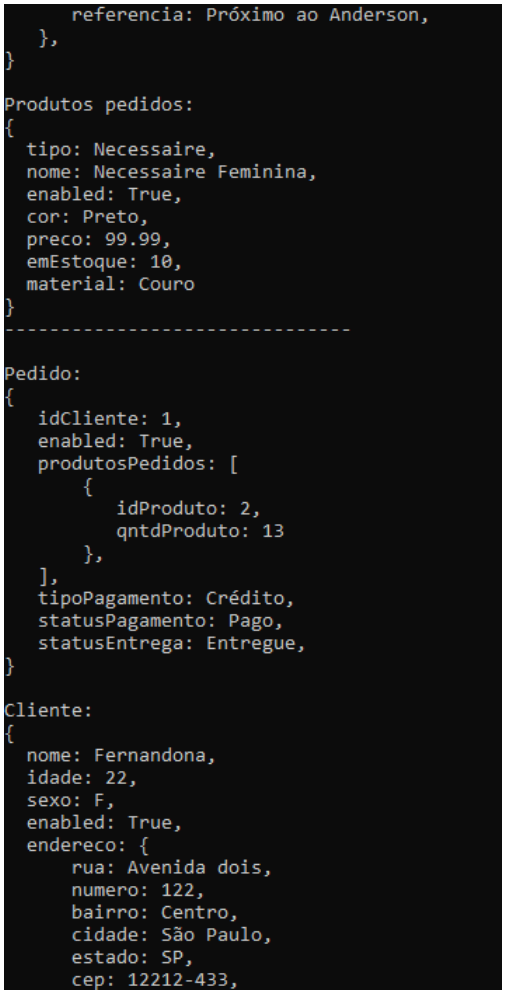
Fonte: Os autores.

Figura 17 – Impressão de todos os documentos no banco de dados - parte 2.



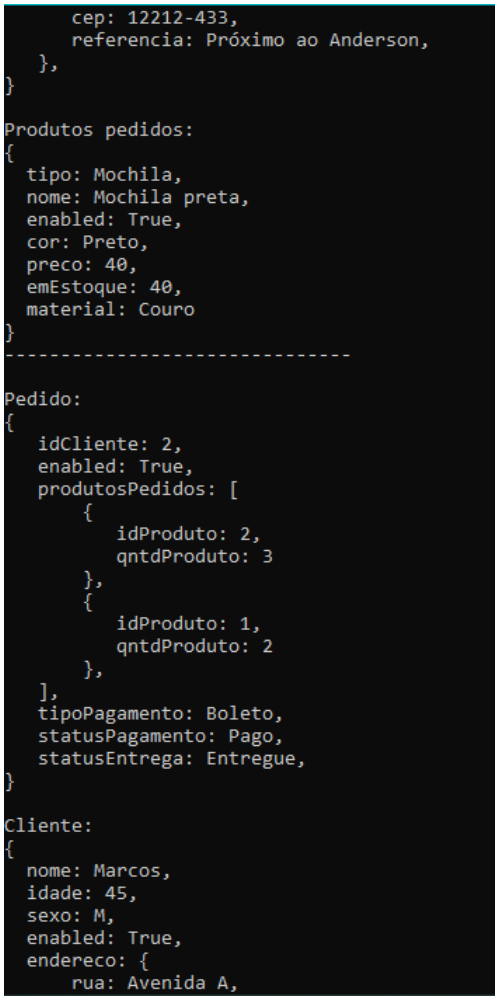
Fonte: Os autores.

Figura 18 – Impressão de todos os documentos no banco de dados - parte 3.



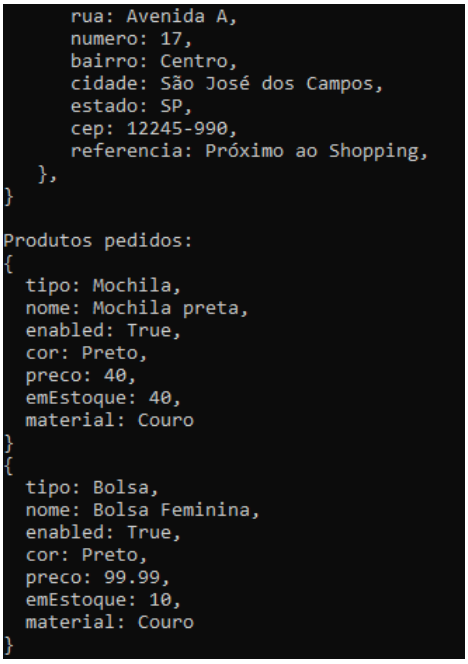
Fonte: Os autores.

Figura 19 – Impressão de todos os documentos no banco de dados - parte 4.



Fonte: Os autores.

Figura 20 – Impressão de todos os documentos no banco de dados - parte 5.



Fonte: Os autores.

Quando os pedidos são impressos, é mostrado também os dados do cliente que o realizou e os dados dos produtos comprados.

6 Considerações Finais

Este projeto foi muito útil para aprofundar os conhecimentos a respeito de bancos de dados NoSQL, em especial o banco de dados orientado a documentos *Cloud Firestore*.

A plataforma *Firebase* tem sido muito utilizada, devido as suas diversas vantagens e ótimos serviços disponíveis, inclusive o *Cloud Firestore*, que permite consultas muito eficientes a dados, além de uma sincronização em tempo real dos dados, o que é extremamente útil em diversas aplicações.

Ademais, os resultados obtidos foram satisfatórios e de acordo com o esperado, sendo assim conclui-se que implementação foi realizada com sucesso.

Referências

1 O que é NoSQL? Disponível em: <https://aws.amazon.com/pt/nosql/>. Acesso em: 10 ago 2021. Citado na página 11.

2 ALECRIM, E. *Bancos de dados são mais importantes nas nossas vidas do que a gente imagina*. 2018. Disponível em: <https://tecnoblog.net/245120/banco-de-dados-importancia/>. Acesso em: 10 ago 2021. Citado na página 11.

3 LEOPOLDINO, E. S. C. B. *Banco de Dados II (Slides)*. 2014. 27 slides. Disponível em: <https://pt.slideshare.net/eric_silva/no-sql-34440686>. Acesso em: 10 ago. 2021. Citado na página 11.

4 GASPERIN, C. A. *Firebase: O Que é e Como Funciona*. Disponível em: <https://micreiros.com/firebase-o-que-e-e-como-funciona/>. Acesso em: 10 ago 2021. Citado na página 11.

5 ORLANDI, C. *Firebase: serviços, vantagens, quando utilizar e integrações*. 2018. Disponível em: <https://blog.rocketseat.com.br/firebase/>. Acesso em: 10 ago 2021. Citado na página 11.

6 ESCOLHER um banco de dados: Cloud Firestore ou Realtime Database. Disponível em: <https://firebase.google.com/docs/firestore/rtdb-vs-firestore?hl=pt-br>. Acesso em: 10 ago 2021. Citado na página 11.

7 MODELO de dados do Cloud Firestore. Disponível em: <https://firebase.google.com/docs/firestore/data-model?hl=pt-br>. Acesso em: 10 ago 2021. Citado 2 vezes nas páginas 12 e 13.