UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS ARARANGUÁ

Lucas Eduardo Schlenert de Oliveira
Manoella Rockembach
Rodrigo Ferraz Souza

Trabalho Final da Disciplina de Banco de Dados I:
Site de loja online que atua sobre um Banco de Dados

2021

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 O BANCO DE DADOS

O objetivo geral do sistema desenvolvido como trabalho final para a disciplina de Banco de Dados é criar uma loja virtual de compra e venda de produtos diversos. A aplicação desenvolvida consiste em um site, onde o backend utiliza NodeJS, Knex como conexão com o banco de dados e express para as conexões http da API. Já o frontend é feito com ReactJS, e o banco de dados é feito com PostgreSQL através do SQLite, para trabalhar junto a API em um servidor.

1.1 DESCRIÇÃO DETALHADA DO SISTEMA

O usuário pode registrar-se como cliente ou como fornecedor, ambos tem como dados necessários para cadastro nome, e-mail e senha para login. Informações de endereço, CEP, cidade, estado e imagem de perfil podem ser preenchidos posteriormente em seu perfil. O fornecedor pode também colocar em sua página de perfil redes sociais e informações para contato.

O cliente possui um saldo vinculado a sua conta que é usado para comprar produtos no site. O fornecedor também possui um saldo vinculado a sua conta que é gerado pela venda de seus produtos. Tanto o cliente como o fornecedor possuem um extrato com os valores movimentados do seu saldo e a data da movimentação.

Os produtos possuem nome, descrição, valor, quantidade e imagem. Estes podem ser separados em categorias. Os clientes podem deixar comentários com avaliações na página do produto, cada comentário possui data, descrição e nota. Os clientes podem criar várias listas de desejos com os seus produtos favoritos de acordo com a ocasião.

O cliente pode realizar a compra de vários produtos de uma só vez. Em cada compra fica registrada a data e o valor final que será debitada do saldo do cliente.

1.2 MODELO CONCEITUAL

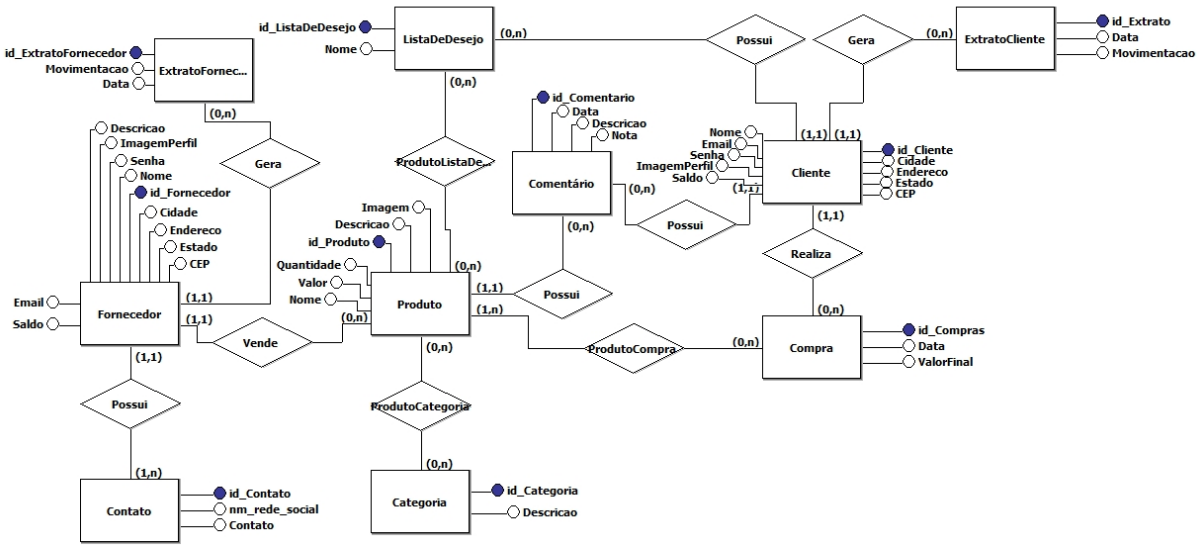


Figura 1 – Modelo Conceitual do Banco de Dados.

1.3 MODELO LÓGICO

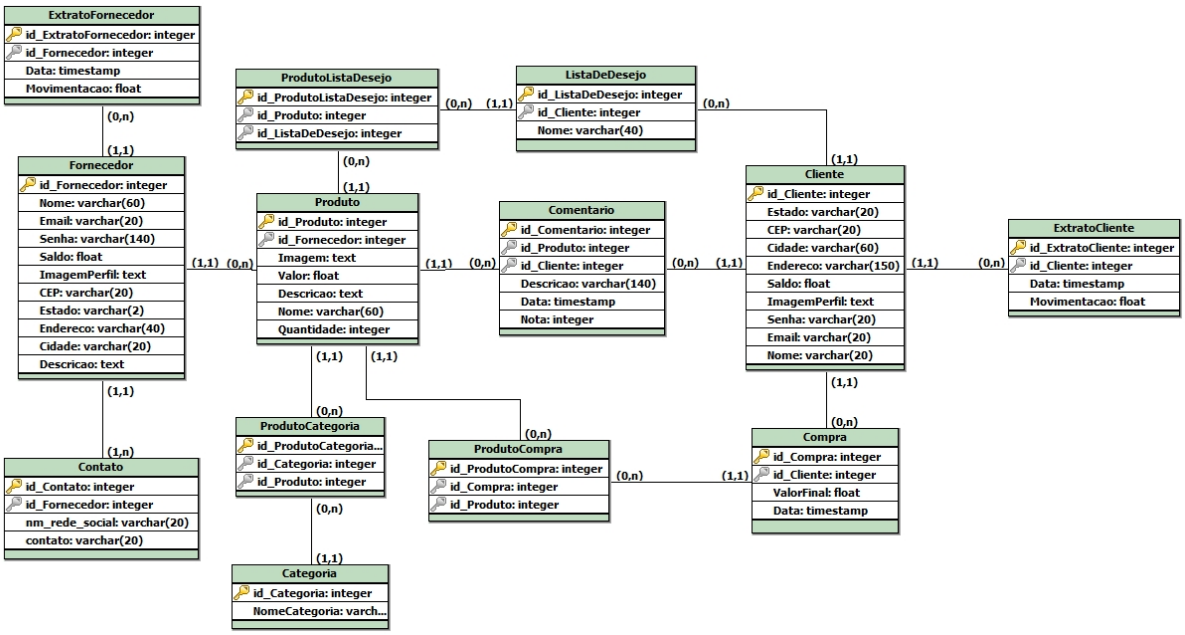


Figura 2 – Modelo Lógico do Banco de Dados.

1.4 SCRIPT DDL

```
01 | -- -----------------------------------------------------
```

```
02 | -- Schema loja_gourmet
03 | -- -----------------------------------------------------
04 | DROP SCHEMA IF EXISTS `loja_gourmet` ;
05 | -- -----------------------------------------------------
06 | -- Schema loja_gourmet
07 | -- -----------------------------------------------------
08 | CREATE SCHEMA IF NOT EXISTS `loja_gourmet` DEFAULT CHARACTER SET
       utf8 ;
09 | USE `loja_gourmet` ;
10 |
11 | DROP TABLE IF EXISTS Comentario;
12 | CREATE TABLE Comentario (
13 | id_Comentario integer PRIMARY KEY,
14 | id_Produto integer,
15 | id_Cliente integer,
16 | Descricao varchar(140),
17 | Data timestamp,
18 | Nota integer
19 | );
20 |
21 | DROP TABLE IF EXISTS Cliente;
22 | CREATE TABLE Cliente (
23 | id_Cliente integer PRIMARY KEY,
24 | Estado varchar(20),
25 | CEP varchar(20),
26 | Cidade varchar(60),
27 | Endereco varchar(150),
28 | Saldo float,
29 | ImagemPerfil text,
30 | Senha varchar(20),
31 | Email varchar(20),
32 | Nome varchar(20)
33 | );
34 |
35 | DROP TABLE IF EXISTS Compra;
36 | CREATE TABLE Compra (
37 | id_Compra integer PRIMARY KEY,
38 | id_Cliente integer,
39 | ValorFinal float,
40 | Data timestamp,
41 | FOREIGN KEY(id_Cliente) REFERENCES Cliente (id_Cliente)
42 | );
43 |
44 | DROP TABLE IF EXISTS ProdutoCompra;
45 | CREATE TABLE ProdutoCompra (
46 | id_ProdutoCompra integer PRIMARY KEY,
47 | id_Compra integer,
```

```
48 |  id_Produto integer,
49 |  FOREIGN KEY(id_Compra) REFERENCES Compra (id_Compra)
50 |  );
51 |
52 |  DROP TABLE IF EXISTS Fornecedor;
53 |  CREATE TABLE Fornecedor (
54 |  id_Fornecedor integer PRIMARY KEY,
55 |  Nome varchar(60),
56 |  Email varchar(20),
57 |  Senha varchar(140),
58 |  Saldo float,
59 |  ImagemPerfil text,
60 |  CEP varchar(20),
61 |  Estado varchar(2),
62 |  Endereco varchar(40),
63 |  Cidade varchar(20),
64 |  Descricao text
65 |  );
66 |
67 |  DROP TABLE IF EXISTS Contato;
68 |  CREATE TABLE Contato (
69 |  id_Contato integer PRIMARY KEY,
70 |  id_Fornecedor integer,
71 |  nm_rede_social varchar(20),
72 |  contato varchar(20),
73 |  FOREIGN KEY(id_Fornecedor) REFERENCES Fornecedor (id_Fornecedor)
74 |  );
75 |
76 |  DROP TABLE IF EXISTS ExtratoCliente;
77 |  CREATE TABLE ExtratoCliente (
78 |  id_ExtratoCliente integer PRIMARY KEY,
79 |  id_Cliente integer,
80 |  Data timestamp,
81 |  Movimentacao float,
82 |  FOREIGN KEY(id_Cliente) REFERENCES Cliente (id_Cliente)
83 |  );
84 |
85 |  DROP TABLE IF EXISTS ExtratoFornecedor;
86 |  CREATE TABLE ExtratoFornecedor (
87 |  id_ExtratoFornecedor integer PRIMARY KEY,
88 |  id_Fornecedor integer,
89 |  Data timestamp,
90 |  Movimentacao float,
91 |  FOREIGN KEY(id_Fornecedor) REFERENCES Fornecedor (id_Fornecedor)
92 |  );
93 |
94 |  DROP TABLE IF EXISTS Produto;
```

```
95 | CREATE TABLE Produto (
96 | id_Produto integer PRIMARY KEY,
97 | id_Fornecedor integer,
98 | Imagem text,
99 | Valor float,
100 | Descricao text,
101 | Nome varchar(60),
102 | Quantidade integer,
103 | FOREIGN KEY(id_Fornecedor) REFERENCES Fornecedor (id_Fornecedor)
104 | );
105 |
106 | DROP TABLE IF EXISTS ProdutoCategoria;
107 | CREATE TABLE ProdutoCategoria (
108 | id_ProdutoCategoria integer PRIMARY KEY,
109 | id_Categoria integer,
110 | id_Produto integer,
111 | FOREIGN KEY(id_Produto) REFERENCES Produto (id_Produto)
112 | );
113 |
114 | DROP TABLE IF EXISTS Categoria;
115 | CREATE TABLE Categoria (
116 | id_Categoria integer PRIMARY KEY,
117 | NomeCategoria varchar(40)
118 | );
119 |
120 | DROP TABLE IF EXISTS ProdutoListaDesejo;
121 | CREATE TABLE ProdutoListaDesejo (
122 | id_ProdutoListaDesejo integer PRIMARY KEY,
123 | id_Produto integer,
124 | id_ListaDeDesejo integer,
125 | FOREIGN KEY(id_Produto) REFERENCES Produto (id_Produto)
126 | );
127 |
128 | DROP TABLE IF EXISTS ListaDeDesejo;
129 | CREATE TABLE ListaDeDesejo (
130 | id_ListaDeDesejo integer PRIMARY KEY,
131 | id_Cliente integer,
132 | Nome varchar(40),
133 | FOREIGN KEY(id_Cliente) REFERENCES Cliente (id_Cliente)
134 | );
135 |
136 | ALTER TABLE Comentario ADD FOREIGN KEY(id_Produto) REFERENCES
        Produto (id_Produto);
137 | ALTER TABLE Comentario ADD FOREIGN KEY(id_Cliente) REFERENCES
        Cliente (id_Cliente);
138 | ALTER TABLE ProdutoCompra ADD FOREIGN KEY(id_Produto) REFERENCES
        Produto (id_Produto);
```

```
139 | ALTER TABLE ProdutoCategoria ADD FOREIGN KEY(id_Categoria)
        REFERENCES Categoria (id_Categoria);
140 | ALTER TABLE ProdutoListaDesejo ADD FOREIGN KEY(id_ListaDeDesejo)
        REFERENCES ListaDeDesejo (id_ListaDeDesejo);
```

2 CONSULTAS NO BANCO DE DADOS

O código de cada uma das consultas realizadas estão apresentadas na seção3.

2.1 PRIMEIRA CONSULTA

Consulta das categorias dos produtos comprados por usuário:

Esta consulta tem como objetivo mostrar quais categorias pertencem os produtos comprados por um certo usuário, ou seja, quais categorias aquele usuário mais compra.

Para realizarmos a consulta através da API, usamos a rota /user/compras/:id e consultamos as tabelas Compra, ProdutoCompra, Produto e Categoria do banco de dados. As informações, depois de processadas, geram a tabela e o gráfico abaixo:

Categoria	Valor
Eletrônicos	541.99
Camisa Masculina	300.00
Acessório	15.00
Eletrônicos	15.00
Camisa Feminina	60.00
Tênis Feminino	196.00
Tênis Masculino	279.99

Tabela 1 – Tabela resultante da consulta 1.

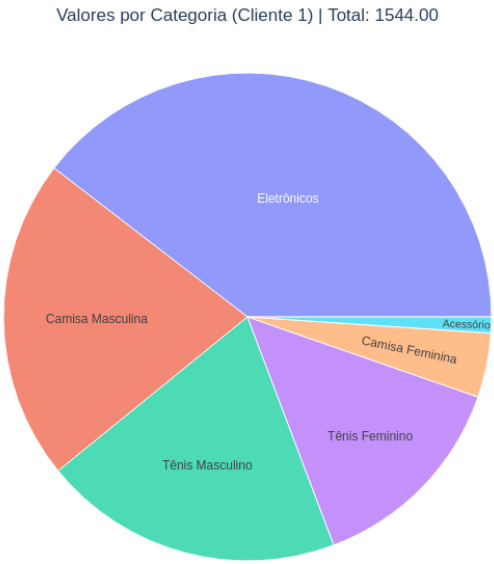


Figura 3 – Gráfico da consulta 1.

2.2 SEGUNDA CONSULTA

Consulta da nota media de todos os produtos de cada fornecedor:

Com esta consulta podemos ver a nota de um fornecedor com base na média das notas dadas a cada um de seus produtos.

Através da rota /fornecedores são obtidos os nomes e ids de todos os fornecedores registrados no sistema (Tabela Fornecedor) e a partir disso, para cada um, é obtido todos os seus produtos através da rota /user/produtos/:id. Depois, é feita a media sobre a nota média de cada um de seus produtos através das tabelas Produto, Comentário.

Fornecedor	**Media**
João	8.0
José	1.0
Pedro	6.5

Tabela 2 – Tabela resultante da consulta 2.

Figura 4 – Gráfico da consulta 2.

2.3 TERCEIRA CONSULTA

Consulta dos fornecedores mais comprados ao longo do tempo por usuário:

Nesta consulta é possível ver quais são os fornecedores mais comprados por um certo usuário, ou seja, os fornecedores favoritos do usuário.

O Endpoint acessado na seguinte requisição é o mesmo que o descrito no item 2.1 e, através dela e das tabelas Compra, ProdutoCompra, Produto e Fornecedor, temos os seguintes resultados:

Mes	Fornecedor	Valor
06	José	120.0
06	Pedro	333.0
08	José	90.0
08	João	684.0
08	Pedro	153.0
09	João	468.0

Tabela 3 – Tabela resultante da consulta 3.

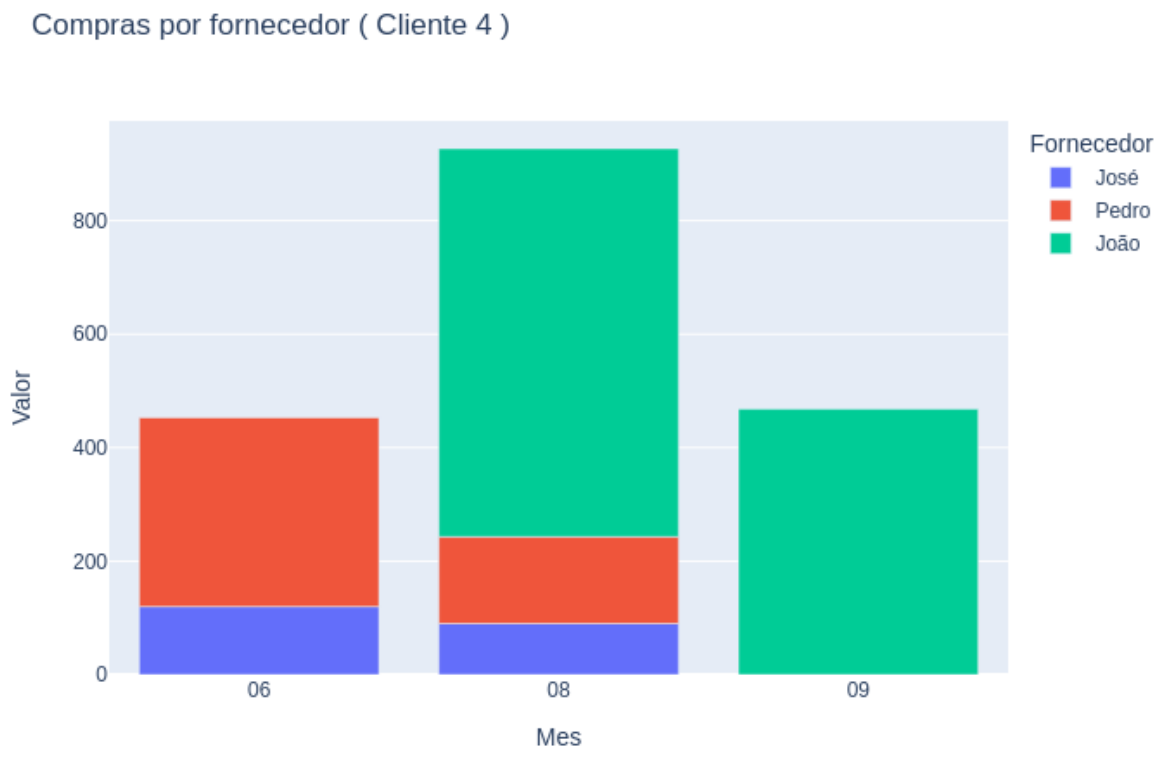


Figura 5 – Gráfico da consulta 3.

3 A APLICAÇÃO DESENVOLVIDA

Como mencionado na seção 1, a plicação desenvolvida é um site de compra e vendas de produtos diversos que atua sobre um banco de dados. As seguintes seções detalham o seu desenvolvimento e conexão com o banco.

3.1 CONEXÃO COM O BANCO DE DADOS

Para que o site se conecte ao banco de dados e consiga realizar operações como inserções, atualizações e exclusões, foi utilizada a biblioteca chamada knex.js. Esta biblioteca de Node.js aceita requisições HTTP na porta 3333 utilizando express.js.

Definição do modulo do express:

```
01 | onst express = require("express");
02 | const bodyParser = require("body-parser");
03 | const routes = require('./routes');
04 | const cors = require("cors");
05 |
06 | const app = express();
07 |
08 | app.use(cors());
09 | app.use(bodyParser.json({ limit: '50mb' }));
10 | app.use(bodyParser.urlencoded({ limit: '50mb', extended: true }));
11 | app.use(express.json());
12 | app.use(routes)
13 | app.listen(3333);
```

Listing 3.1 – modulo do express

Definição do módulo do knex e sua configuração:

```
01 | const knex = require('knex');
02 | const configuration = require("../../knexfile");
03 |
04 | const connection = knex(configuration.development);
05 |
06 | module.exports = connection;
```

Listing 3.2 – Módulo do knex

```
01 |
02 | module.exports = {
03 |
04 |   development: {
05 |     client: 'sqlite3',
06 |     connection: {
07 |       filename: './src/database/db.sqlite3'
08 |     },
```

```
09 |     migrations: {
10 |       directory: './src/database/migrations'
11 |     },
12 |     seeds: {
13 |       directory: './src/database/seeds'
14 |     },
15 |     useNullAsDefault: true,
16 |   },
17 |
18 |   staging: {
19 |     client: 'postgresql',
20 |     connection: {
21 |       database: 'loja_gourmet',
22 |       user: 'username',
23 |       password: 'password'
24 |     },
25 |     pool: {
26 |       min: 2,
27 |       max: 10
28 |     },
29 |     migrations: {
30 |       tableName: 'knex_migrations'
31 |     }
32 |   },
33 |
34 |   production: {
35 |     client: 'postgresql',
36 |     connection: {
37 |       database: 'loja_gourmet',
38 |       user: 'username',
39 |       password: 'password'
40 |     },
41 |     pool: {
42 |       min: 2,
43 |       max: 10
44 |     },
45 |     migrations: {
46 |       tableName: 'knex_migrations'
47 |     }
48 |   }
49 |
50 | };
```

Listing 3.3 – Configuração do knex

3.1.1 Funcionamento da API

API (Application Programming Interface, ou em português Interface de Programação de Aplicativos) é um conjunto de rotinas para acesso a uma aplicação baseada na Web.

Uma API nada mais é que um intermediário entre o banco de dados e o frontend do site, ou seja, é aquele que passa o dado já com algum pré-tratamento. Dessa forma é necessário a comunicação entre ambas as partes, que ocorre através do protocolo HTTP com o padrão de corpo de mensagem com o JSON.

Para que a API consiga fazer a conexão entre backend e frontend é necessário planejar os caminhos por onde o frontend percorre na API, ou seja, as suas rotas.

As rotas modeladas para a API deste projeto são as que seguem:

Base Path	**rota**	**Descrção**
Fornecedor	GET-/	Lista dos Fornecedores
Compra	PUT-/	nova compra
User	POST-/fornecedor:id	atualiza os dados de alguem
User	POST-/cliente/:id	atualiza os dados de alguem
User	DELETE-/contato/:id	deleta um contato de um fornecedor
User	PUT-/contato/new	adiciona um contato ao fornecedor
User	DELETE-/lista/:id	Remove ou a lista ou algo dela
User	PUT-/lista/:id	retorna os dados do produto e suas categorias
User	PUT-/lista	Cria uma nova lista
User	GET-/vendas/:id	mostra as vendas de alguem
User	GET-/produtos/:id	retorna as listas de desejos de alguem
User	GET-/extrato/:id	extrato de alguem
User	POST-/addsaldo/:id	faz um deposito
User	GET-/compras/:id	lista de compras de alguem
User	GET-/:id	dados de um cliente ou fornecedor
User	GET-/lista/:id	lista de desejos
Login	POST-/	login normal
Registro	PUT-/cliente	cadastro de cliente
Registro	PUT-/fornecedor	cadastro de fornecedor
Produto	POST-/:id	atualiza os dados de um produto
Produto	PUT-/new	cria novo produto
Produto	DELETE-/:id	exclui um produto
Produto	PUT-/comentario/:id	adiciona um comentario
Produto	GET-/:id	Detalhes de um produto
Produto	GET-/comentario/:id	comentarios de um produto
Produto	GET-/	Listar todos os produtos
Produto	GET-/categorias	Lista de categorias

Tabela 4 – Tabela de rotas.

3.2 MANIPULAÇÕES DO BANCO DE DADOS

Para utilizar a API criada foram feitos três formas de comunicação:

- O site desenvolvido.
- Uma collection no Insomnia.

- Um programa em python.

Todos estão disponíveis no GitHub[1]

3.2.1 Construção do Site

Para que seja possível utilizar o banco projetado em sua melhor forma, idealizamos as seguintes páginas da aplicação em ReactJS:

- Login
- Sign up
- Página Inicial
- Página do produto
- Página do Cliente
- Página do Fornecedor
- Listas de Desejos
- Carrinho
- Pagamento

Cada uma das páginas idealizadas realizam manipulações no banco de dados através das rotas conectadas a API que podem ser vistas na Tabela de rotas.

Como parte inicial do projeto, assim como realizamos a modelagem inicial do banco de dados, realizamos também mockups de todas as páginas listadas acima, como por exemplo, a página inicial:

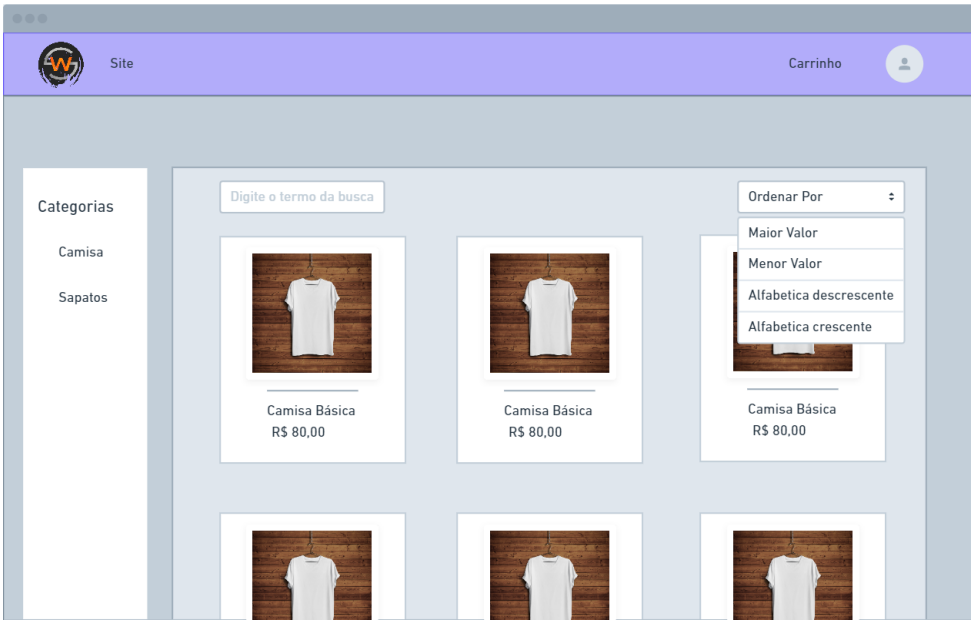
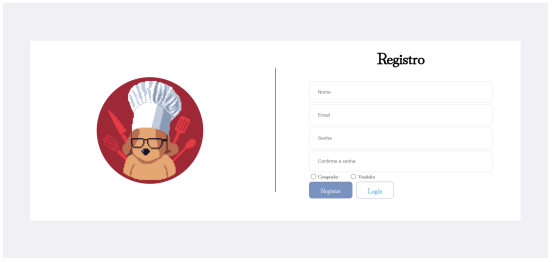


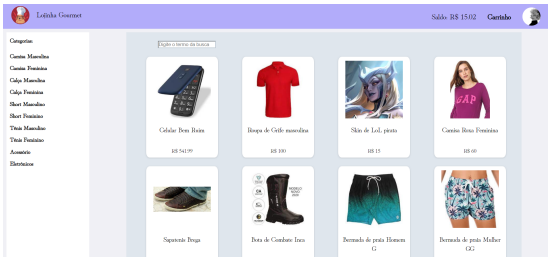
Figura 6 – Mockup da página inicial do site.

[1] `https://github.com/CodeWracker/TFinal-BD1-Lojinha`

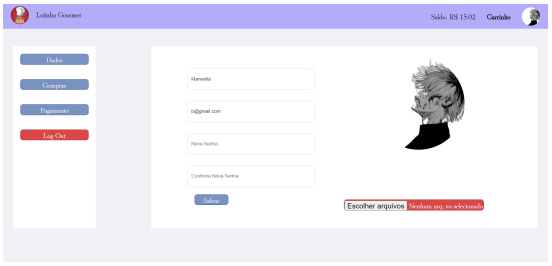
As interface das páginas finais do site desenvolvido ficaram na forma:



(a) Página de Registro



(b) Página inicial



(c) Página de perfil do cliente

(d) Página de um produto

Figura 7 – Páginas do site desenvolvido.

3.2.2 Teste de Requisições com Insomnia

O Insomnia é uma aplicação que nos permite realizar requisições HTTP como forma de testar APIs REST que estão em desenvolvimento. Para cada uma das rotas, foi criada uma requisição no Insomnia que pode interagir com o banco de dados.

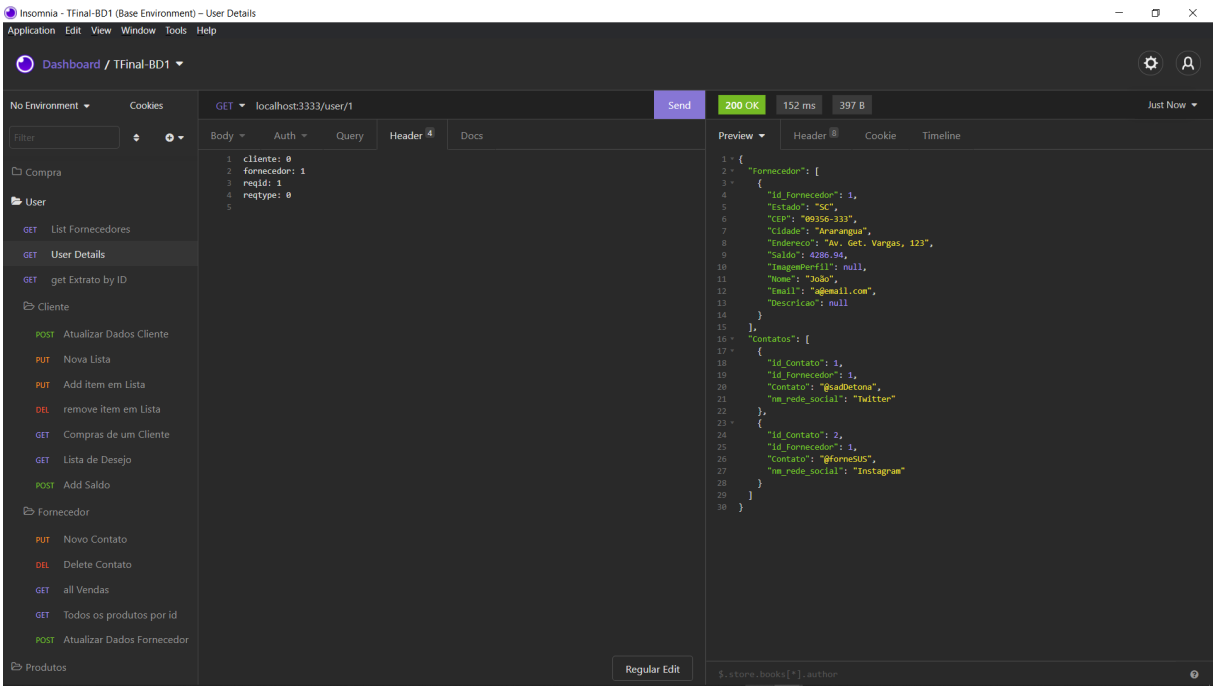


Figura 8 – Base enviroment do Insomnia.

3.2.3 Aplicação em Python

Este item se relaciona com a próxima subsecção e com a seção 2 em sua totalidade, pois foi a partir deste programa que os gráficos apresentados no mesmo foram gerados utilizando a biblioteca plotly.

Esta aplicação feita em um notebook python no Google Colab realiza a conexão com a API através da biblioteca requests.

Para Poder executar este notebook é preciso hostear a aplicação em um serviço que a disponibilize online, portanto estamos utilizando o GitPod.io para tal, através de um caminho disponibilizado pelo GitPod.io.

Figura 9 – Parte do código da aplicação em Python.

3.3 CONSULTAS NO BANCO DE DADOS

A seguir estão os códigos em javascript das consultas apresentadas na seção 2, através da aplicação desenvolvida. Para que estas consultas possam ser feitas, é preciso realizar o encaminhamento das rotas através de endpoints.

Um endpoint é uma extremidade de um canal de comunicação. Quando uma API interage com a aplicação, os pontos de contato dessa comunicação são considerados endpoints. Cada endpoint é o local a partir do qual a API pode acessar os recursos de que precisam para realizar suas funções.

3.3.1 Encaminhamento das Rotas

```
01 | ...
02 | const routes = express.Router()
03 |
04 | /*Rotas de registro */
05 | const registerRouter = express.Router({ mergeParams: true })
06 | routes.use('/registro', registerRouter)
07 | registerRouter.put("/cliente", RegisterController.registroCliente)
08 | registerRouter.put("/fornecedor",
       RegisterController.registroFornecedor)
09 |
10 | /*Rotas de Login */
11 | routes.post("/login", LoginController.login)
12 |
13 | /*Rotas de Produto */
14 | const productsRouter = express.Router({ mergeParams: true })
15 | routes.use('/produto', productsRouter)
16 | productsRouter.get('/categoria', ProdutoController.getAllCategorias)
17 | productsRouter.get('/', ProdutoController.getAllProdutos)
18 | productsRouter.put('/comentario/:id',
       ComentarioController.newComentario)
19 | productsRouter.get('/comentario/:id',
       ComentarioController.getAllComentariosById)
20 | productsRouter.get('/:id', ProdutoController.details)
21 | productsRouter.delete('/:id', ProdutoController.deleteProduto)
22 | productsRouter.put('/new', ProdutoController.newProduto)
23 | productsRouter.post('/:id', ProdutoController.updateProduto)
24 |
25 | /*Rotas de Usuario */
26 | const userRouter = express.Router({ mergeParams: true })
27 | routes.use('/user', userRouter)
28 |
29 | routes.get('/fornecedores', UserController.listFornecedores)
30 | userRouter.get('/lista/:id', UserController.getListaDeDesejo)
31 | userRouter.get("/:id", UserController.detailsById)
32 | userRouter.get("/compras/:id", UserController.getComprasByIdCliente)
33 | userRouter.post("/addsaldo/:id", UserController.addSaldo)
34 | userRouter.get('/extrato/:id',
       ExtratoClienteController.getAllExtByID)
35 | userRouter.get('/produtos/:id',
       UserController.listProdutosByIdFornecedor)
36 | userRouter.get('/vendas/:id', VendaController.getVendasById)
37 | userRouter.put('/lista', UserController.newLista)
38 | userRouter.put('/lista/:id', UserController.newListaItem)
39 | userRouter.delete('/lista/:id', UserController.removeListaItem)
40 | userRouter.put('/contato/new', UserController.addContato)
```

```
41 | userRouter.delete('/contato/:id', UserController.removeContato)
42 | userRouter.post('/cliente/:id', UserController.atualizaCliente)
43 | userRouter.post('/fornecedor/:id', UserController.atualizaFornecedor
   )
44 |
45 | /*Rotas de Compra */
46 | const compraRouter = express.Router({ mergeParams: true })
47 | routes.use('/compra', compraRouter)
48 | compraRouter.put('/new', CompraController.newCompra)
49 |
50 | module.exports = routes
```

Listing 3.4 – routes.js

3.3.2 Primeira Consulta

Consulta das categorias dos produtos comprados por usuário, através do Endpoint /user/compras/:id

```
01 | async getComprasByIdCliente(req, res) {
02 |         const { id } = req.params;
03 |         let { reqid, reqtype, quantidade } = req.headers;
04 |         if (!quantidade) {
05 |             quantidade = 10;
06 |         }
07 |         if (parseInt(reqid) != parseInt(id) || reqtype != "0") {
08 |
09 |             return res.status(400).json({ "Erro": "Voc  n o tem
   autoriza  o para ver isso" })
10 |         }
11 |         const comps = await connection("Compra").select("*").where("
   id_Cliente", id)
12 |             .limit(parseInt(quantidade));
13 |
14 |         if (!comps.length) {
15 |
16 |             return res.status(404).json({ "Erro": "Nenhuma compra
   encontrada para este id" })
17 |         }
18 |         let compras = [];
19 |         for (i in comps) {
20 |             const prds = await connection("ProdutoCompra")
21 |                 .select("Produto.*")
22 |                 .select("ProdutoCompra.Quantidade as qtd_compra")
23 |                 .select("Fornecedor.Nome as nome_fornecedor")
24 |                 .leftJoin("Produto", "Produto.id_Produto", "
   ProdutoCompra.id_Produto")
```

```
25 |                 .leftJoin("Fornecedor", "Produto.id_Fornecedor", "
   Fornecedor.id_Fornecedor")
26 |                 .where("ProdutoCompra.id_Compra", comps[i]["
   id_Compra"])
27 |             let produtos = [];
28 |             for (j in prds) {
29 |                 const cats = await connection("ProdutoCategoria")
30 |                     .select("Categoria.*")
31 |                     .leftJoin("Categoria", "
   ProdutoCategoria.id_Categoria", "Categoria.id_Categoria")
32 |                     .where("ProdutoCategoria.id_Produto", prds[j]["
   id_Produto"])
33 |                 const produto = {
34 |                     "Detalhes": prds[j],
35 |                     "Categorias": cats
36 |                 }
37 |                 produtos.push(produto)
38 |             }
39 |             let compra = {
40 |                 "Detalhes": comps[i],
41 |                 "Produtos": produtos
42 |             }
43 |             compras.push(compra)
44 |         }
45 |
46 |         return res.json({ "Compras": compras })
47 |     }
```

Listing 3.5 – Endpoint /user/compras/:id

Estas consultas recebem como retorno informações do banco de dados no formato:

```
01 | {
02 |   "Compras": [
03 |     {
04 |       "Detalhes": {
05 |         "id_Compra": 1,
06 |         "Data": 1628218800,
07 |         "id_Cliente": 1,
08 |         "ValorFinal": 541.99
09 |       },
10 |       "Produtos": [
11 |         {
12 |           "Detalhes": {
13 |             "id_Produto": 1,
14 |             "id_Fornecedor": 1,
15 |             "Imagem":"BASE64IMG",
16 |             "Valor": 541.99,
```

```
            "Quantidade": 10,
            "Descricao": "Um celular fajuto",
            "Nome": "Celular Bem Ruim",
            "qtd_compra": 1,
            "nome_fornecedor": "Jo o"
          },
          "Categorias": [
            {
              "id_Categoria": 10,
              "nome_categoria": "Eletr nicos"
            }
          ]
        }
      ]
    },
    {
      "Detalhes": {
        "id_Compra": 2,
        "Data": 1628046000,
        "id_Cliente": 1,
        "ValorFinal": 300
      },
      "Produtos": [
        {
          "Detalhes": {
            "id_Produto": 2,
            "id_Fornecedor": 2,
            "Imagem":"BASE64IMG",
            "Valor": 100,
            "Quantidade": 10,
            "Descricao": "Camisa Rasgada do Jacar  GG",
            "Nome": "Roupa de Grife masculina",
            "qtd_compra": 3,
            "nome_fornecedor": "Jos "
          },
          "Categorias": [
            {
              "id_Categoria": 1,
              "nome_categoria": "Camisa Masculina"
            }
          ]
        }
      ]
    },...
  ]
}
```

Listing 3.6 – Resposta do Endpoint /user/compras/1

As informações são então processadas pela aplicação em python para gerar a tabela e o gráfico apresentados na seção 2. Essa estratégia foi utilizada em todas as consultas feitas na seção 2.

3.3.3 Segunda Consulta

Consulta da nota media de todos os produtos de cada fornecedor:

```
async listFornecedores(req, res) {
        const forns = await connection("Fornecedor").select("Nome",
   "id_Fornecedor");
        return res.json({ "Fornecedores": forns })
    }
```

Listing 3.7 – Endpoint /fornecedores

Esta primeira consulta recebe o seguinte resultado, que será utilizado no loop para realizar a requisição da Listing 3.9

```
{
  "Fornecedores": [
    {
      "Nome": "Jo o",
      "id_Fornecedor": 1
    },
    {
      "Nome": "Jos ",
      "id_Fornecedor": 2
    },
    {
      "Nome": "Pedro",
      "id_Fornecedor": 3
    }
  ]
}
```

Listing 3.8 – Resposta do Endpoint /fornecedores

```
    async listProdutosByIdFornecedor(req, res) {
        const { id } = req.params;

        let { quantidade } = req.headers;
        if (!quantidade) {
            quantidade = 20;
        }
        const prds = await connection("Produto")
            .select("Produto.*")
            .avg("Comentario.Nota as media")
```

```
11 |             .leftJoin("Comentario", "Comentario.id_Produto", "
   Produto.id_Produto").where("id_Fornecedor", id).limit(parseInt(
   quantidade))
12 |             .groupBy("Produto.id_Produto");
13 |         if (!prds.length) {
14 |             return res.status(404).json({ "Erro": "Nenhum produto
   encontrado com este id de fornecedor" });
15 |
16 |         }
17 |         let produtos = [];
18 |         for (j in prds) {
19 |             const cats = await connection("ProdutoCategoria")
20 |                 .select("Categoria.*")
21 |                 .leftJoin("Categoria", "
   ProdutoCategoria.id_Categoria", "Categoria.id_Categoria")
22 |                 .where("ProdutoCategoria.id_Produto", prds[j]["
   id_Produto"]);
23 |             const produto = {
24 |                 "Detalhes": prds[j],
25 |                 "Categorias": cats
26 |             }
27 |             produtos.push(produto)
28 |         }
29 |         return res.json({ "Produtos": produtos })
30 |     }
```

Listing 3.9 – Endpoint /produtos/1

As respostas deste endpoint são:

```
01 |    {
02 |   "Produtos": [
03 |     {
04 |       "Detalhes": {
05 |         "id_Produto": 1,
06 |         "id_Fornecedor": 1,
07 |         "Imagem":"B64Text",
08 |         "Valor": 541.99,
09 |         "Quantidade": 10,
10 |         "Descricao": "Um celular fajuto",
11 |         "Nome": "Celular Bem Ruim",
12 |         "media": 8
13 |       },
14 |       "Categorias": [
15 |         {
16 |           "id_Categoria": 10,
17 |           "nome_categoria": "Eletr nicos"
18 |         }
19 |       ]
```

```
    },
    {
      "Detalhes": {
        "id_Produto": 8,
        "id_Fornecedor": 1,
        "Imagem":"B64Text",
        "Valor": 150,
        "Quantidade": 19,
        "Descricao": null,
        "Nome": "Bermuda de praia Mulher GG",
        "media": null
      },
      "Categorias": [
        {
          "id_Categoria": 6,
          "nome_categoria": "Short Feminino"
        }
      ]
    },...
  ]
}
```

Listing 3.10 – Respostas do Endpoint /produtos/1

3.3.4 Terceira Consulta

Consulta dos fornecedores mais comprados ao longo do tempo por usuário:

Como mencionado na secção 2.3 as rotas acessadas para esta consulta são as mesmas que a primeira.

O que a difere da primeira consulta são os dados e a forma como estes são processados.

Todos os códigos desenvolvidos neste projeto estão em anexo a este documento e também estão disponíveis no GitHub[2]

[2] `https://github.com/CodeWracker/TFinal-BD1-Lojinha`