

Projeto
VendingMachine

Francisca Baptista
(a87967)

João Macedo
(a87946)

Hélder Lopes
(a87951)

Marco Martins
(a90072)

26 de junho de 2022

Resumo

No âmbito da unidade curricular Projeto foi-nos proposta a realização de um projeto, no qual o nosso grupo optou pela elaboração de uma proposta pensada por nós, sendo a mesma a criação de um simulador de uma *vending machine* (VM).

Para atingir os objetivos da realização do projeto, numa primeira fase, começámos por criar uma base de dados (BD) e ,posteriormente, uma interface interativa a fim de estabelecer uma comunicação entre os mesmos, com o objetivo de obter uma experiência *frontend* e *backend*.

Para a criação da base de dados foi necessário:

- Realizar um prévio levantamento de requisitos para a BD;
- Proceder à sua análise de requisitos;
- Criar o seu modelo conceptual;
- Criar o seu modelo lógico;
- Obtenção do seu modelo físico com base nas tabelas e atributos previamente definidos.

Para a interface HTML foi necessário:

- Ter em mente um design simples e cativante para os diferentes menus;
- Fazer a divisão adequada dos diferentes tipos de produtos para posteriormente os organizar;
- Escolher a melhor arquitetura de código para facilitar a edição/alteração de informação;
- Saber diferenciar os diferentes componentes da interface, como por exemplo, botões, imagens, texto, etc, a fim de obtermos uma interface compacta e com todas as funcionalidades pretendidas.

Posteriormente, para estabelecer a relação entre a BD e a interface foi necessário a implementação de um programa em *javascript* para gerir a base de dados em função da interface e vice-versa.

Conteúdo

1	Introdução	3
2	Base de Dados	4
2.1	Levantamento e Análise de Requisitos	4
2.1.1	Requisitos de Descrição	4
2.1.2	Requisitos de Exploração	4
2.1.3	Requisitos de Controlo	5
2.2	Modelo Conceptual	5
2.2.1	Identificação e caracterização das entidades	5
2.2.2	Identificação e caracterização dos relacionamentos	6
2.2.3	Identificação e caracterização da associação dos atributos com as entidades e relacionamentos	6
2.3	Modelo Lógico	8
2.3.1	Desenho do modelo lógico	8
2.3.2	Validação do modelo através da normalização	8
2.4	Implementação Física	9
2.4.1	Tradução do esquema lógico para o sistema de gestão de bases de dados escolhido em SQL	9
3	Frontend	12
3.1	Página Inicial	12
3.2	Menu Inicial	12
3.3	Menu Categorias	13
3.4	NavBar	14
3.5	Menu Carrinho	14
4	BackEnd	17
4.1	Ficheiros Controller e Ficheiro Rota	17
4.2	Memória Cache	18
4.2.1	Funcionamento do código	19
4.3	Adicionar/Remover produtos do carrinho de compras	19
4.4	Enviar Email	19
4.5	Alterações na Base de Dados	21

Capítulo 1

Introdução

Após várias discussões de grupo chegámos à conclusão que a simulação de uma *vending machine* era uma escolha viável pois consideramos que a sua evolução ao longo dos anos estagnou/foi pouca, sendo que este tipo de máquinas são bastante comuns na sociedade, estando presentes em variados locais, como por exemplo escolas, hospitais, centros comerciais, etc, sendo por isso relativamente útil fazer um *update* no seu modo de operação.

O objetivo deste trabalho foi aplicar conhecimentos que fomos adquirindo ao longo do curso para implementar em algo do dia a dia que achamos que podia ser melhorado, bem como o desenvolvimento de novas capacidades e conhecimentos aos quais não estávamos familiarizados, como por exemplo o conceito de *frontend*, *backend* e comunicação entre uma BD e uma interface HTML.

Depois de uma análise ao funcionamento de uma *vending machine* reparámos que o número de máquinas deste tipo digitais é reduzido, estando presentes em muito raros locais. Com a implementação de uma máquina digital pensámos em aplicar um carrinho de compras (*shopping cart*) que permite selecionar mais que um produto por vez antes de finalizar a compra, sendo este método eficaz para evitar a utilização repetida da máquina pelo cliente. Em relação ao administrador, pensámos que seria interessante a notificação automática através de email da realização de uma venda feita pela *vending machine* com a informação detalhada da mesma.

Já com a ideia que pretendíamos estruturada foi necessário escolher as melhores ferramentas e programas para a realização da mesma. Optámos por utilizar o MySQL para a criação da base de dados, e a utilização do node.js para fazer a conexão com o a interface HTML relativa ao simulador da *vending machine*.

Capítulo 2

Base de Dados

2.1 Levantamento e Análise de Requisitos

2.1.1 Requisitos de Descrição

1. Guardar o nome dos produtos;
2. Guardar a quantidade de cada produto em stock;
3. Guardar a validade de cada produto;
4. Guardar a que categoria cada produto pertence;
5. Guardar a informação de cada produto;
6. Guardar a imagem do produto;
7. Guardar o preço de venda de cada produto;
6. Guardar o preço total a pagar em cada compra;
8. Guardar o saldo do utilizador;
9. Alterar os produtos existentes;
10. Alterar as imagens dos produtos;
11. Alterar quantidade de cada produto;
12. Alterar o preço dos produtos.

2.1.2 Requisitos de Exploração

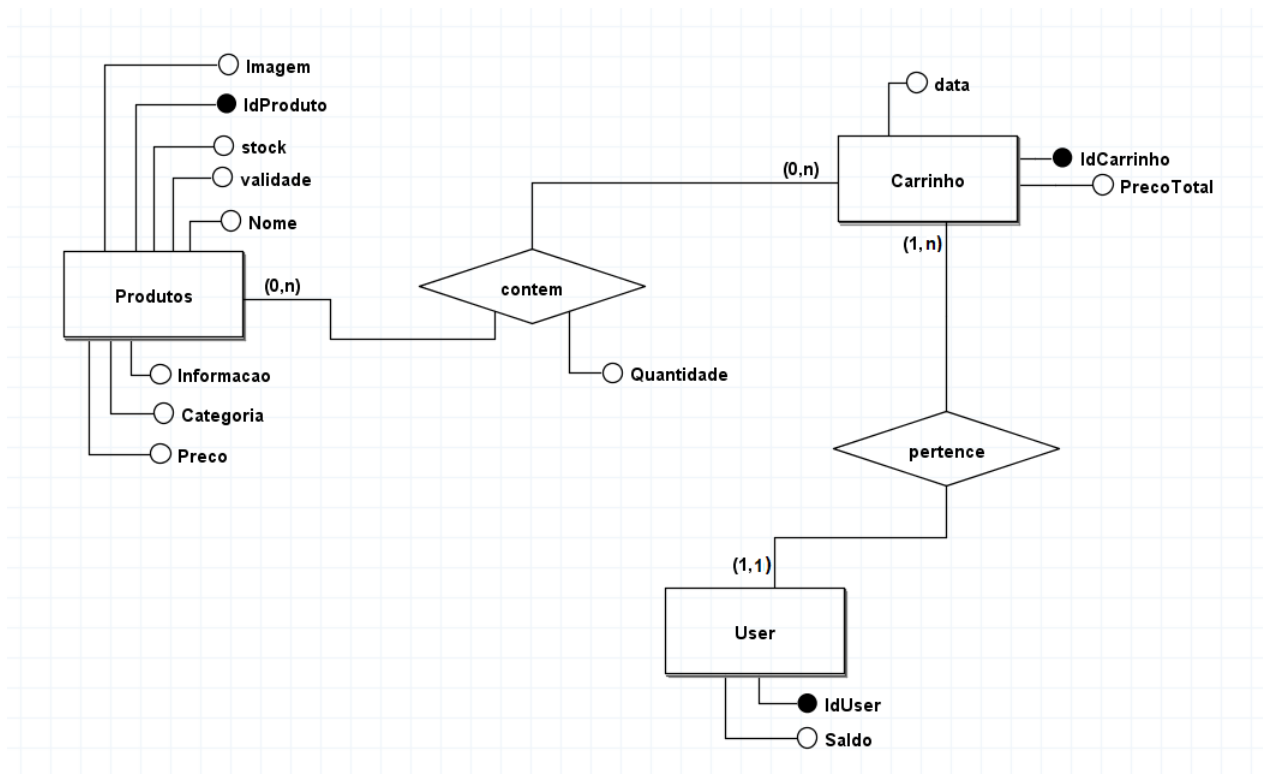
1. Ver os produtos existentes;
2. Ver a quantidade em stock de cada produto;
3. Ver a validade de cada produto;
4. Ver a informação de cada produto;
5. Ver a categoria de cada produto;
6. Ver os produtos que pertencem a cada categoria;
7. Ver o nome do produto;
8. Ver o saldo do utilizador;
9. Ver a imagem de cada produto;

2.1.3 Requisitos de Controlo

O administrador da vending machine terá acesso a todas as funções da base de dados. Apenas este poderá alterar diretamente os atributos dos produtos, como a quantidade em stock quando existe reposição do mesmo, bem como adicionar novos produtos às categorias, realizar a alteração dos preços dos produtos, etc. O cliente apenas terá acesso à realização de uma compra, fazendo assim uma inserção de uma nova componente na tabela "carrinho" assim como as componentes necessárias na tabela "produtosCarrinho" e dar o update do saldo do cliente e do stock dos produtos.

2.2 Modelo Conceptual

Após realizarmos o levantamento de requisitos e a sua análise, procedemos à criação do modelo conceptual da base de dados.



No modelo conceptual conseguimos perceber o tipo de ligações que existem entre entidades e os atributos necessários a cada uma delas, para conseguirmos realizar todos os requisitos.

2.2.1 Identificação e caracterização das entidades

Atendendo aos requisitos levantados, foram concebidas as seguintes entidades:

1. **Produtos:** contém as informações do conjunto de todos os produtos da vending machine;
2. **Carrinho:** Contém a data da compra, o preço final a pagar e os produtos da compra;
3. **User:** Contém apenas o saldo disponível pelo utilizador quando este vai realizar uma compra.

2.2.2 Identificação e caracterização dos relacionamentos

A base de dados possui dois relacionamentos: contém e pertence.

1. Contém: tem um atributo "quantidade" referente ao número de vezes que determinado produto aparece no carrinho.
2. Pertence: associa um carrinho a um cliente.

2.2.3 Identificação e caracterização da associação dos atributos com as entidades e relacionamentos

Após a criação do modelo conceptual procedemos para o tratamento de dados, juntando os mesmos em tabelas de relacionamentos e entidades.

1. Nesta primeira tabela vemos os diferentes relacionamentos entre entidades, com a sua respetiva cardinalidade e descrição.

Entidade	Relacionamento	Entidade	Cardinalidade	Descrição
Produtos	Contem	Carrinho	0:N	Um produto pode pertencer a 0 ou mais carrinhos
Carrinho	Pertence	User	N:1	Um carrinho pertence apenas a um user
Carrinho	Contem	Produtos	0:N	Um carrinho pode conter zero ou mais produtos
User	Pertence	Carrinho	1:N	Cada user tem zero ou mais carrinhos

2. Na segunda tabela é nos apresentada a informação da entidade "Carrinho" nomeadamente os seus atributos, a descrição dos mesmos, o tipo de dados e a sua nulidade.

Atributo	Descrição	Tipo de Dados e Tamanho	Nulo
IdCarrinho	Número único que identifica unicamente Produto	Número inteiro positivo (incrementado)	Não
PrecoTotal	Preço total a pagar	Número decimal positivo	Não
data	Data da realização da compra	date	Não

3. Na terceira tabela vemos, à semelhança do que acontece na segunda tabela, a informação da entidade "Produtos", referente aos seus atributos, a descrição dos mesmos, o tipo de dados e a sua nulidade.

Atributo	Descrição	Tipo de Dados e Tamanho	Nulo
IdProduto	Número único que identifica unicamente Produto	Número inteiro positivo (incrementado)	Não
Stock	Quantidade de cada produto em stock	Número inteiro positivo	Sim
Validade	Validade de cada produto	Data	Sim
Informação	Descrição sobre cada produto	Sequência de Caracteres Tamanho Variável (Max: 45 caracteres)	Não
Categoria	Nome de que categoria cada produto pertence	Sequência de Caracteres Tamanho Variável (Max: 45 caracteres)	Não
Preco	Valor a pagar por cada produto	Número decimal positivo	Não
Nome	Nome do produto	Sequência de Caracteres Tamanho Variável (Max: 45 caracteres)	Não
Imagem	Endereço da imagem de cada produto	Sequência de Caracteres Tamanho Variável (Max: 500 caracteres)	Não

4. Por fim, na quarta tabela, vemos a informação da entidade "User", sendo apresentados os seus atributos, a sua descrição, o tipo de dados e a sua nulidade.

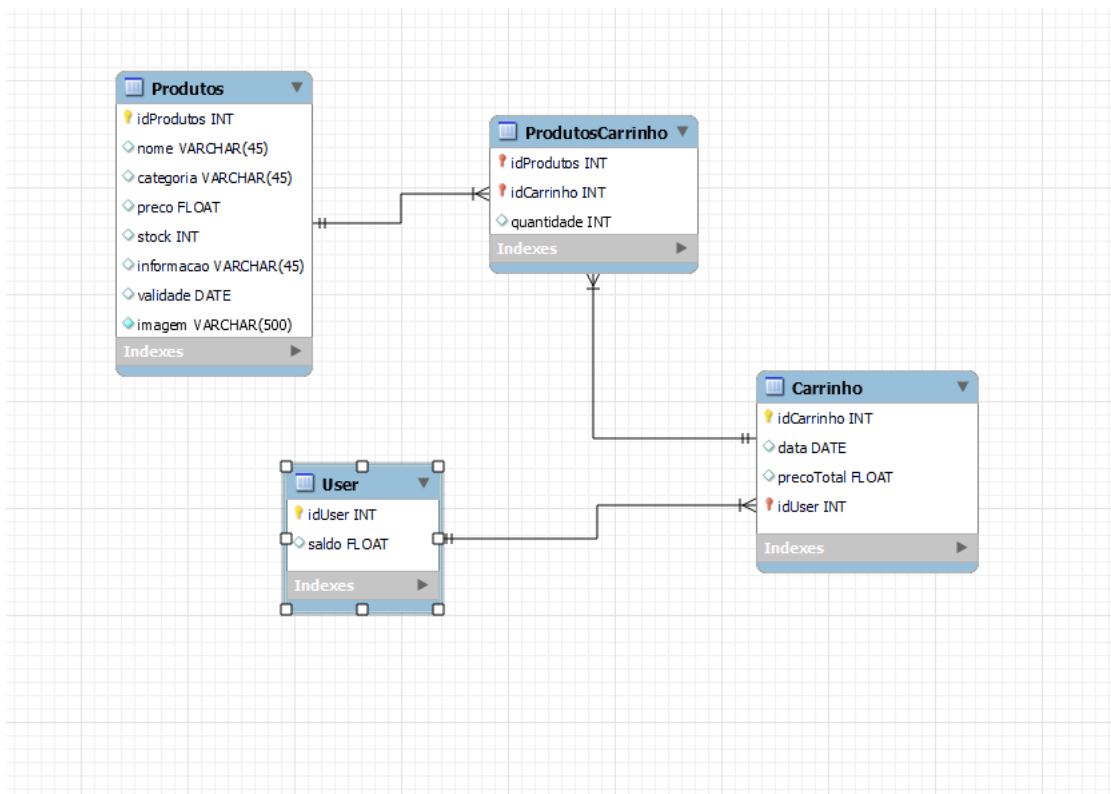
Atributo	Descrição	Tipo de Dados e Tamanho	Nulo
IdUser	Número único que identifica unicamente Produto	Número inteiro positivo	Não
Saldo	Número que indica o saldo do utilizador	Número inteiro positivo	Sim

2.3 Modelo Lógico

Após a construção do modelo conceptual procedemos à transformação do mesmo para o respetivo modelo lógico, respeitando o tipo de dados e a normalização da base de dados, com vista a evitar possíveis erros e problemas de eficiência.

2.3.1 Desenho do modelo lógico

Na imagem que se segue, podemos ver o modelo lógico obtido, como referido anteriormente. As diferentes tabelas presentes neste modelo referem-se às entidades do modelo conceptual e aos relacionamentos.



2.3.2 Validação do modelo através da normalização

A fim de obtermos uma base de dados funcional, privada de erros, sem problemas de eficiência, com garantia de integridade de dados, etc, foi necessário certificarmo-nos de que a mesma se encontrava normalizada. Primeiramente, tivemos de verificar em todas as tabelas a não existência de atributos multivalorados, ou seja, apenas podem estar presentes atributos atómicos. Seguidamente para as relações existentes foi necessário, conforme a sua cardinalidade, verificar se era necessário criar tabelas para as representar, isto é, nas relações de cardinalidade 1 para n (relação "pertence"), por exemplo, não era adequado. Em casos que as relações não fossem deste tipo de cardinalidade, como por exemplo a relação "contém", que é n para n , foi preciso criar uma tabela no modelo lógico para representar essa mesma relação, tabela essa que contém a chave primária de cada uma das entidades envolvidas ("Produtos" e "Carrinho"), para além dos atributos pertencentes a esse relacionamento. Apesar de na relação "pertence" não ser necessário criar uma tabela como anteriormente dito, foi necessário implementar uma chave estrangeira (*foreign key*) na tabela

”carrinho” para identificar o cliente.

2.4 Implementação Física

2.4.1 Tradução do esquema lógico para o sistema de gestão de bases de dados escolhido em SQL

Foi realizada inserção de dados em produtos, como valores para cada atributo da tabela:

	idProdutos	nome	categoria	preco	stock	informacao	validade	imagem
▶	1	coca cola	bebidas frias	0.9	50	Lata de 33cl	2023-02-05	https://i.postimg.cc/R02gspsr/coca.png
	2	7up	bebidas frias	0.9	50	Lata de 33cl	2023-02-05	https://i.postimg.cc/MZtq4ySP/7up.png
	3	Ice tea Limao	bebidas frias	0.9	50	Lata de 33cl	2023-02-05	https://i.postimg.cc/xdBf1YCX/limao.png
	4	Ice tea Pessego	bebidas frias	0.9	50	Lata de 33cl	2023-02-05	https://i.postimg.cc/bNf7ggdS/pessego.png
	5	Pepsi	bebidas frias	0.9	50	Lata de 33cl	2023-02-05	https://i.postimg.cc/Yqpfk8kK/pepsi.png
	6	Agua	bebidas frias	0.6	50	Garrafa de 0.5l	2023-02-05	https://i.postimg.cc/qMB9WVRp/agua.png
	7	Doritos	Snacks	1.2	50	Pacote Doritos 150g	2023-02-05	https://i.postimg.cc/HkDbVJZg/doritos.png
	8	Lays Original	Snacks	1.2	50	Pacote Batatas 150g	2023-02-05	https://i.postimg.cc/tg11kpm5/lays.png
	9	Lays Camponesa	Snacks	1.2	50	Pacote Batatas sabor camponesas 150g	2023-02-05	https://i.postimg.cc/xTT1g9Xz/lays1.png
	10	Pringles Paprika	Snacks	1.2	50	Embalagem Batatas sabor Paprika	2023-02-05	https://i.postimg.cc/J7K1sSp4/pringlespa.png
	11	Kinder Chocolate	Snacks	0.8	50	Kinder Chocolate	2023-02-05	https://i.postimg.cc/c1VNtZQW/kinder.png
	12	Kinder Bueno	Snacks	0.8	50	Kinder Bueno	2023-02-05	https://i.postimg.cc/G2WStL6S/kinderbueno....
	13	Caneta	Material Esc...	0.45	50	Caneta Bic tinta azul	NULL	https://i.postimg.cc/MKPvMppz/caneta.png
	14	Lapis	Material Esc...	0.3	50	Lapis N2	NULL	https://i.postimg.cc/pXwXvz0D/lapis.png
	15	Embalagem de ...	Material Esc...	1.2	50	Embalagem de 6 canetas Bic tinta Azul	NULL	https://i.postimg.cc/6Q39LcTx/packca.webp
	16	Embalagem de L...	Material Es...	1.2	50	Embalagem de 6 lapis N2	NULL	https://i.postimg.cc/3NhV9r90/packla.png
	17	Canetas de Cor	Material Esc...	2	50	Embagem de 6 Canetas de Cor	NULL	https://i.postimg.cc/Y2gmVw3f/marcadores....
	18	Afia	Material Esc...	0.5	50	Um Afia sem deposito	NULL	https://i.postimg.cc/65W5pBwQ/afia.png
	19	Mascara Cirurgica	COVID	0.45	50	Uma Mascara Cirurgica	NULL	https://i.postimg.cc/kXzZhyjd/mascara.png
	20	Mascara FFP2	COVID	0.5	50	Uma Mascara FFP2	NULL	https://i.postimg.cc/fy073Yyx/mask.png
	21	Par de Luvas	COVID	0.6	50	Um par de luvas Latex	NULL	https://i.postimg.cc/6ppYjyZn/kisspng-medic...
	22	Pacote de Masc...	COVID	5	50	Embalagem de 15 Mascaras	NULL	https://i.postimg.cc/DyvL3VMr/pack.png
	23	Alcool Gel	COVID	1	20	Embalagem de 200ml de Alcool gel	NULL	https://i.postimg.cc/C1JmJz0s/des.png
	24	PowerBank	Produtos	10	20	PowerBank de 10000mAh universal	NULL	https://i.postimg.cc/RF0xzZwr/bateria.png
	25	Cabo Iphone	Produtos	4	30	Cabo para carregador Iphone	NULL	https://i.postimg.cc/XJTWc0BT/caboi.png
	26	Carregor tipo C	Produtos	3	30	Cabo para carregador Android	NULL	https://i.postimg.cc/5NyXTsq4/download.png
	27	Capa Iphone 11...	Produtos	7	30	Capa para Iphone 11/12/13	NULL	https://i.postimg.cc/4xz1Xv1t/capai.png
	28	Capa Samsun G...	Produtos	7	30	Capa para Android Galaxy 13	NULL	https://i.postimg.cc/13SKph1C/download.jpg
	29	Fones ouvido wi...	Produtos	6	30	Fones Bluetooth	NULL	https://i.postimg.cc/vHJ5cP8T/fone.webp
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Bem como outra inserção em user, para haver a simulação de uma compra:

	idUser	saldo
▶	1	100
*	NULL	NULL

Por fim, obtemos o modelo físico da nossa base de dados, em código SQL.

```

-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-----
-- Schema mydb
-----

-----
-- Schema mydb
-----

CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-----
-- Table `mydb`.`Produtos`
-----
} CREATE TABLE IF NOT EXISTS `mydb`.`Produtos` (
  `idProdutos` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NULL,
  `categoria` VARCHAR(45) NULL,
  `preco` FLOAT NULL,
  `stock` INT NULL,
  `informacao` VARCHAR(45) NULL,
  `validade` DATE NULL,
  PRIMARY KEY (`idProdutos`))
ENGINE = InnoDB;

-----
-- Table `mydb`.`User`
-----
} CREATE TABLE IF NOT EXISTS `mydb`.`User` (
  `idUser` INT NOT NULL AUTO_INCREMENT,
  `saldo` FLOAT NULL,
  PRIMARY KEY (`idUser`))
ENGINE = InnoDB;

-----
-- Table `mydb`.`Carrinho`
-----
} CREATE TABLE IF NOT EXISTS `mydb`.`Carrinho` (
  `idCarrinho` INT NOT NULL AUTO_INCREMENT,
  `data` DATE NULL,
  `precoTotal` FLOAT NULL,
  `idUser` INT NOT NULL,
  PRIMARY KEY (`idCarrinho`, `idUser`),
  INDEX `idUser_idx` (`idUser` ASC) VISIBLE,
  CONSTRAINT `idUser`
    FOREIGN KEY (`idUser`)
      REFERENCES `mydb`.`User` (`idUser`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

```
-----  
-- Table `mydb`.`ProdutosCarrinho`  
-----
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`ProdutosCarrinho` (  
  `idProdutos` INT NOT NULL,  
  `idCarrinho` INT NOT NULL,  
  `quantidade` INT NULL,  
  PRIMARY KEY (`idProdutos`, `idCarrinho`),  
  INDEX `idCarrinho_idx` (`idCarrinho` ASC) VISIBLE,  
  CONSTRAINT `idProdutos`  
    FOREIGN KEY (`idProdutos`)  
      REFERENCES `mydb`.`Produtos` (`idProdutos`)  
      ON DELETE NO ACTION  
      ON UPDATE NO ACTION,  
  CONSTRAINT `idCarrinho`  
    FOREIGN KEY (`idCarrinho`)  
      REFERENCES `mydb`.`Carrinho` (`idCarrinho`)  
      ON DELETE NO ACTION  
      ON UPDATE NO ACTION)  
ENGINE = InnoDB;
```

```
SET SQL_MODE=@OLD_SQL_MODE;
```

```
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
```

```
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Capítulo 3

Frontend

3.1 Página Inicial

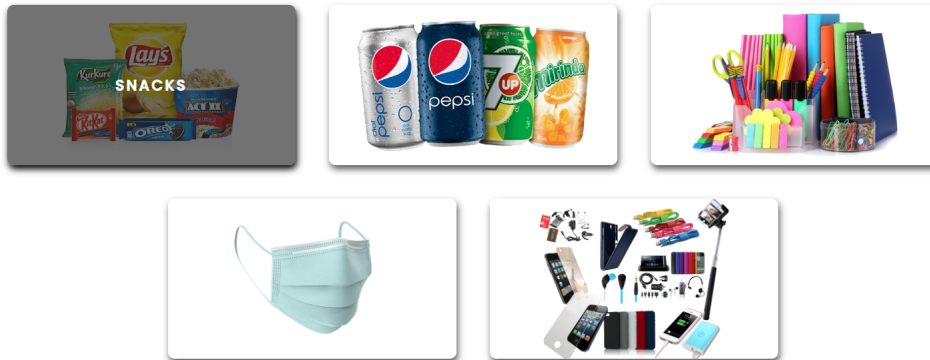
Na página inicial optamos por escrever "Vending Machine" de uma forma diferente, neste caso em loop, e um botão que abrange a tela inteira para entrarmos para o menu inicial.



3.2 Menu Inicial

Entrando no menu inicial temos 5 categorias diferentes à escolha: Snacks, Bebidas Frias, Material Escolar, Artigos Covid e Produtos. Ao passarmos o rato por cima de cada secção podemos ver o nome de cada uma das categorias. As diferentes categorias são estáticas.

MENU



127.0.0.1:5500/chica/Front-End/proj/menusnack/snack.html

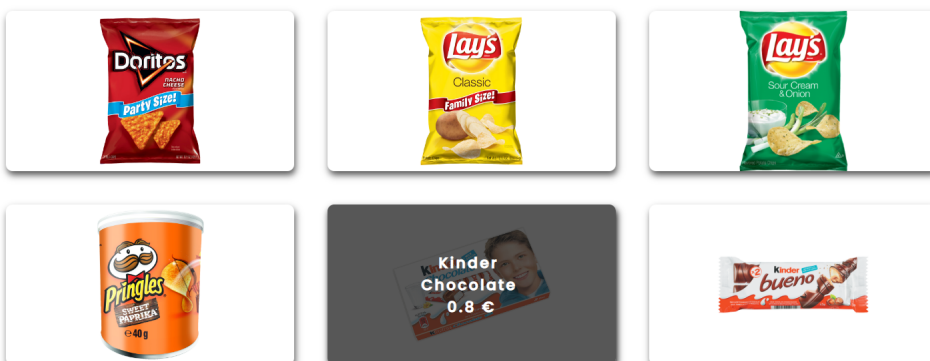
3.3 Menu Categorias

Dentro de cada categoria temos as diferentes opções para cada uma, contendo os produtos associados. Cada produto contém o seu nome e o preço associado ao mesmo.

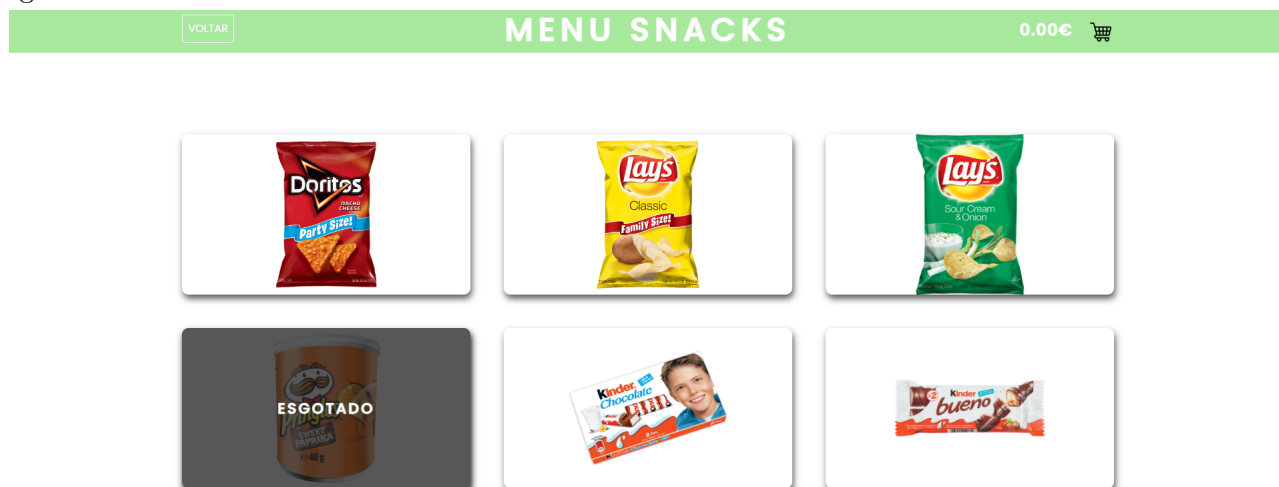
[VOLTAR](#)

MENU SNACKS

0.00€ 



Quando não houver stock do produto desejado, aparece por cima do produto a informação a dizer "esgotado".



3.4 NavBar

Na *navbar* podemos encontrar do lado esquerdo um botão em que ao clicarmos volta para o menu inicial, temos do lado direito um carrinho que nos indica o preço total da compra até ao momento e nos leva para o menu com a lista de produtos no cesto das compras. Temos também a meio da página o nome do menu em que estamos a fazer a compra.



3.5 Menu Carrinho









Ao clicarmos no carrinho, vamos para uma página que vai conter os artigos seleccionados com a sua informação- nome, quantidade, preço- onde é possível alterar a quantidade dos mesmos. Podemos verificar o preço total da compra, tendo depois a opção de cancelar a compra ou finalizar a mesma, podendo também remover produtos livremente do carrinho.


Remove	Imagem	Produto	Preço	Quantidade	Total
		Kinder Chocolate	0.8	<input type="text" value="1"/>	0.80
		Capa Samsun Galaxy 13	7	<input type="text" value="1"/>	7.00
		Cabo Iphone	4	<input type="text" value="1"/>	4.00
		Carregor tipo C	3	<input type="text" value="1"/>	3.00
		Fones ouvido wireless	6	<input type="text" value="1"/>	6.00
Total:		20.80€			<div>Cancelar</div> <div>Finalizar</div>

Se clicarmos na opção finalizar, será analisado se existe ou não saldo no user para realizar a compra. Caso o saldo não seja suficiente então aparecerá uma mensagem "Não possui saldo suficiente!". Um cliente que falhou a sua compra pode alterar o carrinho de compras removendo os produtos escolhidos, como se verificará em duas fotos abaixo em comparação com a que se encontra em seguida.

MENU INICIAL

CARRINHO

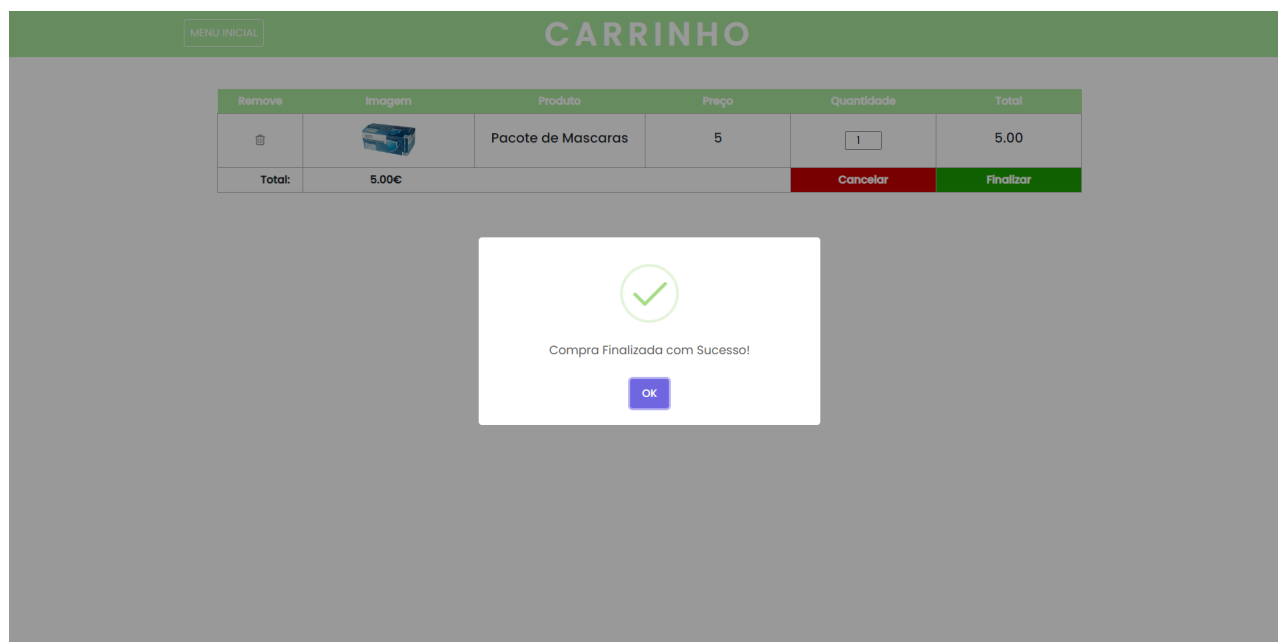
Remove	Imagem	Produto	Preço	Quantidade	Total
		Pacote de Mascaras	5	<input type="text" value="1"/>	5.00
		Alcool Gel	1	<input type="text" value="1"/>	1.00
				<input type="text" value="1"/>	0.60
				<input type="text" value="1"/>	0.50
Total:		7.10€			<div>Cancelar</div> <div>Finalizar</div>



Não Possui Saldo Suficiente!

OK

No caso de uma compra ser bem sucedida irá aparecer "Compra realizada com sucesso".



No caso do user escolher a opção cancelar, então todos os produtos do carrinho são removidos e volta para o menu inicial.

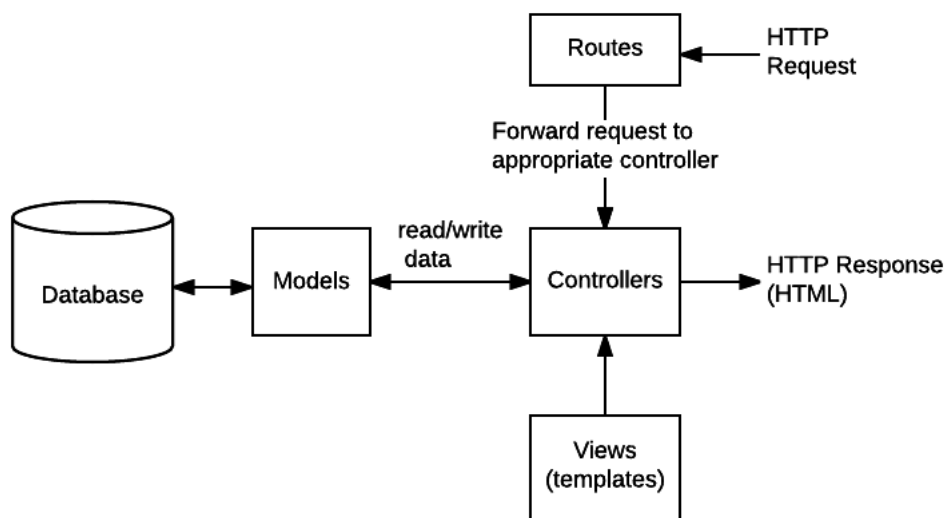
Capítulo 4

BackEnd

4.1 Ficheiros Controller e Ficheiro Rota

Quando um pedido é feito pelo HTTP, esse pedido vai ser recebido pelos ficheiros *routes*, que são ficheiros que tem associado as funções (get, post, put, delete, etc) e um caminho, onde, por fim, apresenta a função que é chamada para realizar o pedido pretendido.

As funções que realizam o pedido encontram-se nos ficheiros Controller. É através das queries do controller que existe consulta e atualização da base de dados. Os ficheiros controllers são aqueles que contém um conjunto de métodos que vão permitir realizar a rota pretendida. Quando é feito o pedido para aceder à rota é realizada a função correspondente à rota que foi associada.



Segue-se um exemplo de um ficheiro Controller referente aos produtos:

...

```

//RETURNS ALL PRODUCTS
function read(req, res) {
  //Create and execute the read query in the database
  connect.con.query('SELECT idProdutos,nome ,categoria,preco,stock ,informacao ,validade, imagem from produtos', function (err, rows, fields) {
    if (!err) {
      //checks the results, if the number of rows is 0 returns data not found, otherwise sends the results (rows).
      if (rows.length == 0) {
        res.status(404).send("Data not found");
      } else {
        res.status(200).send(rows);
      }
    } else {
      console.log('Error while performing Query.', err);
    }
  });
}

```

...

Para cada categoria, neste caso "snacks", temos o seguinte código de controlo, sendo semelhante para os diferentes casos:

```

//RETURNS ONLY SNACK
function readSnacks(req, res) {
  //Create and execute the read query in the database
  connect.con.query('SELECT idProdutos,nome ,categoria,preco,stock ,informacao ,validade, imagem from produtos WHERE categoria="snacks"', function (err, rows, fields) {
    if (!err) {
      //checks the results, if the number of rows is 0 returns data not found, otherwise sends the results (rows).
      if (rows.length == 0) {
        res.status(404).send("Data not found");
      } else {
        res.status(200).send(rows);
      }
    } else {
      console.log('Error while performing Query.', err);
    }
  });
}

```

...

Para atualizar o stock de um produto, temos:

```

function updateStock(req, res) {
  const idProdutos = req.sanitize('idProdutos').escape();
  const subtrairStock = req.sanitize('subtrairStock').escape();
  var query="";

  query = connect.con.query('UPDATE produtos SET stock = stock - ? where idProdutos=? ',[subtrairStock, idProdutos], function(err, rows, fields){
    console.log(query.sql);
    if (!err) {
      //checks the results, if the number of rows is 0 returns data not found, otherwise sends the results (rows).
      if (rows.length == 0) {
        res.status(404).send("Data not found");
      } else {
        res.status(200).send(rows);
      }
    } else {
      console.log('Error while performing Query.', err);
    }
  });
}

```

...

4.2 Memória Cache

Quando um user está a adicionar os produtos ao cesto de compras essa informação não vai ser guardada na base de dados, ela vai ser colocada em memória cache permitindo assim não sobrecarregar a base de dados e reduzindo também as consultas a esta. Apenas é feita a alteração na base de dados quando a compra se der por concluída. Esta memória vai ser guardada temporariamente com a informação de cada produto selecionado pelo utilizador.

4.2.1 Funcionamento do código

Temos o `localStorage` e vamos ver se já existe o array "produtos", se existir então continuamos a utilizar esse, pois significa que a compra ainda não foi finalizada e, caso contrário, criamos um array "produtos" vazio. Conforme vão sendo adicionados os produtos são guardadas nesse array as seguintes informações: o id do produto, a sua quantidade, a imagem e o nome do produto. É realizada a condição que compara os "produtoid" que queremos adicionar com os já existentes no array. Caso um produto com esse id já exista no array então vamos adicionar no final do mesmo as informações desse produto com o incremento na quantidade. Depois, através da função `splice` vai ser eliminada a primeira ocorrência desse produto no array, ficando apenas uma ocorrência com o valor de quantidade atualizado no fim do array. No caso de não existir então vai ser adicionado esse produto ao array com quantidade 1.

Ao longo deste código vamos colocando o array "produtos" como string visto que este é o tipo aceite pelo `localStorage`.

```
localStorage.produtos
'[{"id":7,"quantidade":1,"imagem":"https://i.postimg.cc/HkDbVJZg/doritos.png", "nome": "Doritos", "preco":1.2}, {"id":11,"quantidade":1,"imagem":"https://i.postimg.cc/c1VntZQW/kinder.png", "nome": "Kinder Chocolate", "preco":0.8}, {"id":8,"quantidade":1,"imagem":"https://i.postimg.cc/tg11kpm5/lays.png", "nome": "Lays Original", "preco":1.2}, {"id":9,"quantidade":1,"imagem":"https://i.postimg.cc/xT1t9Xz/lays1.png", "nome": "Lays Camponesa", "preco":1.2}]'
```

4.3 Adicionar/Remover produtos do carrinho de compras

Quando clicamos no botão cesto vamos então obter a lista de compras dos produtos que selecionamos. Para fazer essa lista foi necessária a criação de um ciclo que termina quando o array produtos que está em memória cache termina. Dentro deste ciclo vão ser criadas as linhas da tabela onde vão ser colocadas as informações correspondentes a cada componente da tabela.

A primeira coluna da tabela recebe apenas a imagem de um cesto do lixo com a função de apagar da tabela a linha correspondente, apagando assim o item selecionado. Isso vai acontecer através da função `removeItem` que recebe como parâmetro o id do produto que queremos remover no nosso carrinho de compras. Para isso vamos percorrer mais uma vez o array produto até encontrarmos o id pretendido e, depois, aplicamos a função `splice` eliminando-o do array.

A segunda, terceira, quarta e quinta colunas vão receber, respetivamente, a imagem, o nome, o preço e a quantidade do produto selecionado, indo buscar essa informação ao array armazenado em memória cache.

A sexta coluna dá-nos o preço por produto, indo buscar a informação no array do preço desse produto, tendo em conta a quantidade do mesmo.

O preço total vai aparecendo conforme vamos adicionando produtos no carrinho. É calculado consultando mais uma vez o array produtos, multiplicando a quantidade de cada produto com o preço do mesmo e somando os diferentes preços para obter o preço total.

4.4 Enviar Email

Sempre que uma compra é realizada, é enviado um email para administrador com a informação dos produtos comprados bem como o preço total da compra. A função que está encarregue de enviar o email para o administrador encontra-se no `MailController`. Esta função tem uma query que vai nos dar a informação dos componentes que queremos preencher, estando previamente definido por nós o email do administrador

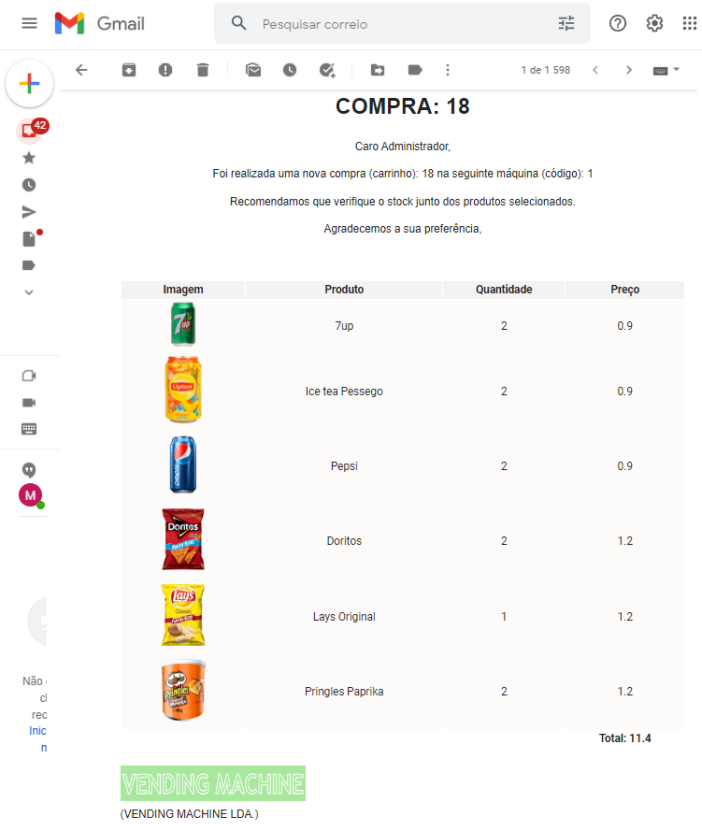
e a estrutura da respetiva mensagem.

```
function sendMail(req, res) {  
  let email= "chicabaptista00@gmail.com";  
  const vendingStamp = "(VENDING MACHINE LDA.)"  
  
  const idCarrinho = req.sanitize('idCarrinho').escape();
```

```
bodycontent += '<br> <br> <h1 style="text-align: center !important"> COMPRA: '+ idCarrinho+'</h1>'  
bodycontent += '<p style="text-align: center !important" > Caro Administrador, </p>';  
bodycontent += '<p style="text-align: center !important" >Foi realizada uma nova compra (carrinho): '+ idCarrinho +' na seguinte máquina (código): 1 </p>';  
bodycontent += '<p style="text-align: center !important" >Recomendamos que verifique o stock junto dos produtos selecionados.</p>';  
bodycontent += '<p style="text-align: center !important" >Agradecemos a sua preferência,</p>';  
bodycontent += '</i></blockquote>';  
bodycontent += '<br> <br>'
```

```
let totalCompra = 0;  
//ADD PRODUCTS TO TABLE  
for(let i=0; i< rows.length; i++){  
  bodycontent += '<tr> <td style="text-align: center !important"><img style="max-width: 100px !important; max-height: 80px !important; text-align: center !important" src=' + rows[i].imagem +  
    ' alt=""></td> <td style="text-align: center !important">'+ rows[i].nome + '</td> <td style="text-align: center !important">'+ rows[i].quantidade.toString() +  
    '</td><td style="text-align: center !important">'+ rows[i].preco + '</td>';  
  
  let temp= parseInt(rows[i].quantidade) * parseFloat(rows[i].preco)  
  totalCompra += temp;  
}
```

Sendo o respetivo email do seguinte formato:



4.5 Alterações na Base de Dados

Quando uma compra é finalizada é necessário que exista redução da quantidade dos produtos em stock, a redução do saldo do user e a inserção da última compra no carrinho. Em baixo, iremos então apresentar o comportamento dos dados da base de dados para cada uma das seguintes situações.

1. Reduzir o stock dos produtos do carrinho (supondo que o stock inicial de cada produto estava fixo em 50 unidades):

1	coca cola	bebidas frias	0.9	50	Lata de 33cl	2023-02-05	https://i.postimg.cc/R02gspsr/coca.png
2	7up	bebidas frias	0.9	48	Lata de 33cl	2023-02-05	https://i.postimg.cc/MZtq4ysP/7up.png
3	Ice tea Limao	bebidas frias	0.9	50	Lata de 33cl	2023-02-05	https://i.postimg.cc/xdBf1YCX/limao.png
4	Ice tea Pessego	bebidas frias	0.9	48	Lata de 33cl	2023-02-05	https://i.postimg.cc/bNf7ggdS/peasego.png
5	Pepsi	bebidas frias	0.9	48	Lata de 33cl	2023-02-05	https://i.postimg.cc/Yqphk8Kk/pepsi.png
6	Agua	bebidas frias	0.6	50	Garrafa de 0.5l	2023-02-05	https://i.postimg.cc/qMB9WRXp/agua.png
7	Doritos	Snacks	1.2	48	Pacote Doritos 150g	2023-02-05	https://i.postimg.cc/HkDbVJZg/doritos.png
8	Lays Original	Snacks	1.2	49	Pacote Batatas 150g	2023-02-05	https://i.postimg.cc/tg11kpm5/lays.png
9	Lays Camponesa	Snacks	1.2	50	Pacote Batatas sabor camponesas 150g	2023-02-05	https://i.postimg.cc/xTT1g9Xz/lays1.png
10	Pringles Paprika	Snacks	1.2	48	Embalagem Batatas sabor Paprika	2023-02-05	https://i.postimg.cc/37K1sSp4/pringlespa.png

Podemos ver que houve uma diminuição do valor do atributo stock.

2. Reduzir saldo do user (partindo de um saldo inicial de 100€):

	idUser	saldo
▶	1	88.6
*	NULL	NULL

Podemos ver que houve uma diminuição do valor do atributo saldo.

3. Inserção da compra realizada na tabela carrinho (guardando o id da compra, a data, o preço total e o id do user que a efetuou):

18	2022-06-20	11.4	1
----	------------	------	---

Capítulo 5

Conclusão

Após a conclusão do projeto, sentimos que mesmo saindo da nossa zona de conforto, explorando mais a linguagem *javascript*, métodos HTML, fizemos um projeto compacto e funcional. Usamos os nossos conhecimentos da unidade curricular de Base de Dados para construir a nossa BD, bem como os nossos conhecimentos na escrita de relatórios através do formato LaTeX usado em vários trabalhos práticos ao longo do curso e, para além disso, conseguimos utilizar o nosso espírito autodidata para ganhar conhecimentos em áreas que não estávamos tão à vontade, conseguindo eventualmente atingir os nossos objetivos para este projeto.

Cremos que as *vending machines* existentes atualmente nos mais diversos sítios da sociedade, apesar de funcionais, não estão completamente otimizadas. Ao investir neste tema, o grupo procurou tornar as mesmas mais otimizadas, adicionando *features* (por exemplo a capacidade de comprar múltiplos produtos numa só compra dependentemente do saldo inserido) e tornando o visual das mesmas mais atual e simples.