

Como instalar o JavaFX?

Faça download da JavaFX SDK aqui:

<https://gluonhq.com/products/javafx>

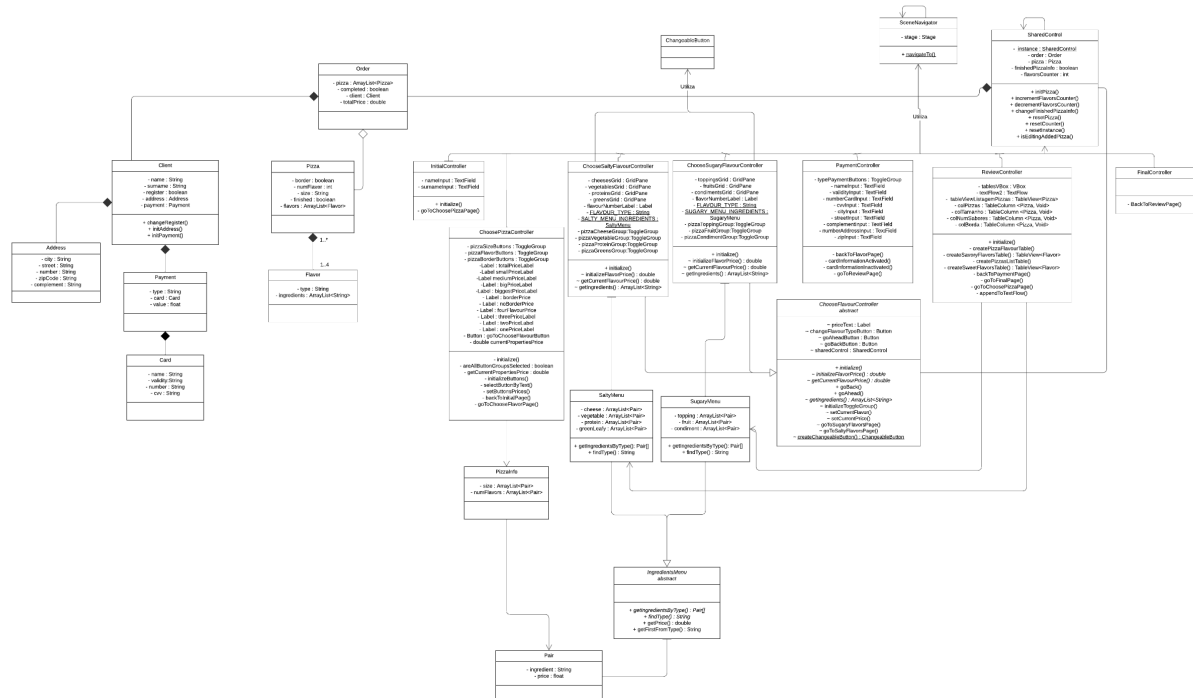
## 1.3 Relatório de Implementação: Sistema de Compra de Pizzas Customizáveis

### Introdução

Este relatório descreve o desenvolvimento de um sistema de compra de pizzas customizáveis utilizando JavaFX. O sistema permite que os usuários selecionem pizzas, sabores, métodos de pagamento e revisem seus pedidos por meio de uma interface gráfica interativa.

### 1. Estrutura de Classes

As mudanças que foram feitas no [diagrama de classes](#) que seguem abaixo:



Primeiramente, alteramos todo o conteúdo para o inglês, com o objetivo de padronizar o código e o diagrama de classes, garantindo maior consistência e alinhamento com boas práticas de desenvolvimento.

Além disso, mantivemos a ideia de compartilhar o pedido entre as classes por meio de uma classe principal. No entanto, decidimos remover o esquema de estados. Assim, onde antes utilizávamos um estado, agora passamos a usar diretamente a interface do front-end. Por exemplo, a classe InitialState foi substituída por InitialController, simplificando a estrutura e facilitando a interação com a interface.

Outra mudança importante foi a exclusão do relacionamento das classes anteriormente conectadas à classe “Sabor”. Antes, essa conexão era utilizada para verificar se a escolha do usuário, enviada pelo front-end, correspondia a algo disponível nas classes “Salgado” ou “Doce”. Agora, essas classes foram renomeadas para SugaryMenu e SaltyMenu e são usadas para organizar dados de forma mais clara e acessível, permitindo que as informações sejam facilmente puxadas e exibidas no front-end. Essa modificação visa facilitar futuras manutenções e adaptações.

Por fim, criamos outras classes adicionais com o mesmo objetivo de organização e modularidade, como a classe `PizzaInfo`, que contribui para uma estrutura mais limpa e eficiente.

## 2.1 Classes de Modelo

As classes de modelo representam as entidades principais do sistema e contêm a lógica relacionada aos dados e operações do pedido de pizza. Elas se mantiveram basicamente as mesmas da etapa anterior.

### Order

- Responsável por armazenar informações sobre o pedido, incluindo pizzas, as informações do cliente, endereço, pagamento e se todas as informações do pedido já foram preenchidas.

### Client

- Guarda informações sobre o cliente que está realizando o pedido: seu nome, sobrenome, endereço, a forma de pagamento que está sendo utilizada e se suas informações já estão registradas.

### Flavour

- Representa um sabor de pizza. Ele pode ser doce ou salgado, e contém uma `ArrayList` de `Pair` que são os ingredientes que o compõem.

### Address

- Representa o endereço de entrega que o cliente fornece.
  - o Atributos: `street`, `number`, `city`, `zipcode`, `complement`.

### Payment

- Representa a informação de pagamento do pedido.
  - o Atributos: `type`, `value`, `card` (opcional).

### Card

- Representa os detalhes de um cartão de crédito.

Atributos: `number`, `validity`, `cvv`, `name`.

## 2.2 Classes de armazenamento

Essas classes servem para fornecer as informações constantes que serão utilizadas: os nomes dos sabores e os preços das opções. Os métodos dentro delas servem para selecionar esses valores.

### Pair

- É a base das classes de armazenamento. Ela contém uma opção (`String`), e um preço (`double`) associado a essa opção.

### IngredientsMenu

- É uma classe abstrata que implementa algumas das funções utilizadas em `SaltyMenu` e `SugaryMenu` para selecionar valores.
- Métodos importantes:
  - o **`double getPrice()`**: dado o tipo de um ingrediente (fruta, cobertura, condimento, proteína, queijo, folhas verdes ou vegetal) e o nome de um ingrediente específico desse tipo, retorna o preço dele.

- **double getFirstFromType():** dado um tipo de ingrediente e uma array list de ingredientes, retorna o primeiro ingrediente daquela array list que é do tipo informado.
- **abstract Pair[] getIngredientsByType();**
- **abstract String findType();**

### **SaltyMenu**

- Representa todo o menu de sabores salgados. Os sabores salgados estão divididos em 4 categorias: queijos, proteínas, vegetais e folhas verdes. Cada uma dessas categorias é uma lista constante de valores do tipo Pair.
- Métodos importantes:
  - **Pair[] getIngredientsByType():** dado um tipo (proteína, queijo, folhas verdes ou vegetal), retorna todos os ingredientes desse tipo existentes, associados com seus preços.
  - **abstract String findType():** dado um ingrediente, busca pelo seu tipo e o retorna (proteína, folhas verdes, queijo ou vegetal).

### **SugaryMenu**

- Representa todo o menu de sabores doces. Os sabores doces estão divididos em 3 categorias: coberturas, condimentos e fruta. Cada uma dessas categorias é uma lista constante de valores do tipo Pair.
- Métodos importantes:
  - **Pair[] getIngredientsByType():** dado um tipo (condimento, cobertura ou fruta), retorna todos os ingredientes desse tipo existentes, associados com seus preços
  - **abstract String findType():** dado um ingrediente, busca pelo seu tipo e o retorna (condimento, cobertura ou vegetal)

### **PizzaInfo**

- Guarda as opções e preços de características das pizzas além dos ingredientes disponíveis para formarem seus sabores. As 3 categorias que estão inclusas incluem os tamanhos disponíveis, o preço extra pago pela inserção de mais sabores na pizza e o custo de colocar borda recheada nela.
- Métodos importantes:
  - **double getPrice():** dado uma propriedade (tamanho, borda, número de sabores) e uma opção existente para essa propriedade, retorna o preço dessa opção ou 0 se ela não existir
  - **Pair[] getProperties():** dado uma propriedade (tamanho, borda, número de sabores), retorna todas as opções possíveis para ela, e os preços associados a essas opções

## **2.3 Classes de Controle**

As classes de controle gerenciam a interação entre a interface gráfica e as classes de modelo. Todas as controllers compartilham a classe **SharedControl** para acesso centralizado ao pedido.

### **SharedControl**

- Responsável por manter uma referência única ao pedido em andamento e compartilhá-lo entre diferentes controladores. Para garantir isso, a classe

SharedControl possui uma instância de si mesma como atributo de classe, e todos os controladores acessam essa mesma instância.

### **SceneNavigator**

- Responsável por administrar a navegação entre telas.

### **InitialController**

- Controla a tela inicial.
  - o Ações: adicionar nome e sobrenome do cliente, iniciar um novo pedido.

### **ChoosePizzaController**

- Controla a tela de escolha do tamanho, do número de sabores e da borda da pizza.
  - o Ações: selecionar o tamanho da pizza, sua borda e seu número de sabores. Atualizar dinamicamente o preço total do pedido conforme as opções vão sendo selecionadas. Guardar os valores selecionados na pizza sendo montada ao avançar para a escolha de sabores.

### **ChooseFlavorController**

- Classe abstrata que serve como base para a implementação dos controladores da tela de escolha de sabores doce e da tela de escolha de sabores salgados, implementando funções comuns a elas.

### **ChooseSaltyFlavourController**

- Controla a tela de escolha de ingredientes para a confecção de um sabor salgado.
  - o Ações: possibilitar a troca para a escolha de um sabor doce. Escolher os ingredientes do sabor salgado. Exibir dinamicamente o preço total do pedido conforme as opções vão sendo selecionadas. Possibilitar a ida para um próximo sabor bem como a volta para uma tela anterior e edição de escolhas. Ao sair da tela com algum ingrediente selecionado, atualizar a pizza que está sendo confeccionada colocando o sabor montado nela e atualizar o preço do pedido.

### **ChooseSugaryFlavourController**

- Controla a tela de escolha de ingredientes para a confecção de um sabor doce.
  - o Ações: possibilitar a troca para a escolha de um sabor doce. Escolher os ingredientes do sabor salgado. Atualizar dinamicamente o preço total do pedido conforme as opções vão sendo selecionadas. Possibilitar a ida para um próximo sabor bem como a volta para uma tela anterior e edição de escolhas. Ao sair da tela com algum ingrediente selecionado, atualizar a pizza que está sendo confeccionada colocando o sabor montado nela e atualizar o preço do pedido.

### **PaymentController**

- Controla a tela de escolha de pagamento.
  - o Ações: selecionar o método de pagamento, inserir dados do endereço, inserir dados do cartão (se aplicável), navegar para a revisão do pedido ou voltar para a edição dos sabores. Ao sair da tela, atualizar os valores do endereço e do pagamento do cliente conforme as informações fornecidas.

### **ReviewController**

- Controla a tela de revisão do pedido.
  - o Ações: exibir o pedido completo: todas as pizzas com seus sabores e preços além dos dados do cliente. Permitir a exclusão de pizzas do pedido. Permitir a edição de pizzas já confeccionadas. Finalizar o pedido ou voltar para a tela de escolha de pagamento.

### **FinalController**

- Controla a tela final.
  - o Ações: exibir mensagem de confirmação, navegar de volta à tela inicial e resetar o pedido, excluindo todas as informações existentes sobre o cliente.

## **2.4 Extra: classe de implementação de componente de interface**

### **ChangeableButton**

- Implementa um botão que pode ser selecionado e “desselecionado” a partir da classe RadioButton do JavaFX.

## **3. Comparação com a etapa 1**

### **3.1 Requisitos:**

RF-1: O cliente consegue personalizar até 4 sabores em uma pizza. Para cada sabor, ele pode escolher até 1 ingrediente em cada uma das categorias disponíveis para aquela opção de sabor (para sabores doces, as categorias são condimento, fruta, cobertura; para sabores salgados, as categorias são proteína, vegetal, folhas verdes e queijo)

RF-2: O sistema exibe todas as possibilidades possíveis de ingredientes para a opção de sabor escolhida (doce ou salgado) e permite que o cliente troque entre essas opções a qualquer momento

RF-3: Não só o sistema mostra o preço atualizado conforme o cliente adiciona os ingredientes, como também mostra na escolha dos atributos da pizza (tamanho, número de sabores e se tem borda)

RF-4: O cliente pode escolher entre pagar com dinheiro, cartão ou pix. Se escolher pagar com cartão, deve preencher as informações dele. Se escolher pagar com pix, um link será mostrado a ele na tela de revisão

RF-5: O sistema está limitando o cliente a 5 pizzas, mas essa é apenas uma limitação para cumprir o requisito. Na verdade, o sistema foi desenvolvido para que não haja limitação no número de pizzas em um pedido.

RNF-1: O app foi desenvolvido para isso

RNF-2: O tempo de resposta não excede 1 segundo

RNF-3: Após o pedido ser finalizado, todos os dados do cliente são apagados e eles não são salvos em nenhum lugar

RNF-4: O sistema conta com um esquema de cores e botões para a interação bastante intuitivo

RNF-5: Para adicionar novos ingredientes ou modificar o preço de algum deles, basta acrescentá-los nas listas de SaltyMenu ou SugaryMenu

### 3.2 Classes e suas relações

As classes de modelo acabaram não sofrendo tantas alterações em relação à etapa 1. A maior alteração nas classes de modelo está ligada à mudança nas classes de armazenamento: anteriormente, foi estipulado que uma Pizza teria uma Array List de sabores, e esse sabores seriam uma classe abstrata que seria herdada pela classe Doce e pela classe Salgado, cada uma com suas categorias de ingredientes representadas por Array Lists de Strings. A classe Pizza continua tendo uma Array List de sabores, mas agora a classe Flavour (sabor) é uma classe concreta que contém uma Array List de pares ingrediente-preço (classe Pair). A definição dessa Array List de pares que vai em um sabor é feita através dos métodos das classes IngredientsMenu (classe abstrata), SaltyMenu e SugaryMenu, que também guardam os valores constantes dos ingredientes.

Já nas classes de controle houve muitas mudanças. Inicialmente a ideia era ter uma classe de estado para cada tela, uma classe abstrata que definisse um padrão para esses estados e que a classe Order resgistrasse o estado atual do pedido. O que se fez foi inverter essa lógica: em vez do pedido registrar o estado, existe uma classe de estado compartilhada que registra o pedido (SharedControl). Essa classe faz isso através de um atributo estático que é acessado por todos os controladores. A classe abstrata que define o padrão do estado em cada tela não foi implementada, porque a maioria dos controladores das telas são muito diferentes - exceto pelas telas de escolha de ingredientes, em que existe uma para sabor doce (ChooseSugaryFlavourController) e outra para sabor salgado (ChooseSaltyFlavourController). Para esses dois controladores foi criada uma classe abstrata (ChooseFlavourController), pois seus métodos são muito similares.

Por fim, houve a implementação de uma nova classe que seria impossível prever na fase de planejamento do projeto, pois seu único intuito é criar um botão mutável para a interface - capaz de ser selecionado e então ser “desselecionado” (ChangeableButton).

### 1.4 - Relatório de Teste

No projeto, os testes foram alocados em uma pasta específica localizada em ‘*slice-of-life\SOL\tests\application*’. Lá estão localizadas as classes de teste e cada uma será comentada abaixo com uma descrição de seus respectivos testes:

#### ClientTest.java

Nesta classe foi testado 2 (dois) fatores da classe **Client.java**: o método **initAddress()** e o método **initPayment()**:

*public void initAddress():*

- **void addressShouldHaveComplementTest():** Testa se o método *initAddress()* atribui corretamente o valor ao atributo address (*String street*,

*String number, String city, String zipCode, String complement*) caso o complemento **não seja nulo**.

- **void addressShouldntHaveComplementTest():** Testa se o método *initAddress()* atribui corretamente os valores ao atributo *address* (*String street, String number, String city, String zipCode*) caso o complemento **seja nulo**.

*public void initPayment():*

- **void paymentShouldHaveCardTest():** Testa se o método *initPayment()* atribui corretamente o valor ao atributo *payment* (*int number, Card card*) caso o método de pagamento escolhido pelo usuário seja “Cartão de Crédito/Débito”. Neste teste, é necessário instanciar outro objeto da classe *Card*.
- **void paymentShouldntHaveCardTest():** Testa se o método *initPayment()* atribui corretamente o valor ao atributo *payment* (*int number*) caso o método de pagamento escolhido pelo usuário **não** seja “Cartão de Crédito/Débito”.

### **PizzaInfoTest.java**

Nesta classe foram testado 2 (dois) fatores da classe **PizzaInfo.java**: o método **getProperties()** e o método **getPrice()**:

*public Pair[ ] getProperties(String type):*

- **void getPropertiesForBorderSituationTest():** Testa se ao receber no argumento *type* a String “border”, devolve o objeto da classe *Pair()* referente a tipos de bordas e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.
- **void getPropertiesForSizesSituationTest():** Testa se ao receber no argumento *type* a String “sizes”, devolve o objeto da classe *Pair()* referente a tipos de tamanhos de pizza e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.
- **void getPropertiesForNumFlavoursSituationTest():** Testa se ao receber no argumento *type* a String “number of flavours”, devolve o objeto da classe *Pair()* referente as opções de número de sabores da pizza e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.

*public double getPrice(String type, String item):*

- **void getPriceForBorderTest():** Testa se, ao receber no argumento *type* a String “border” e no argumento *item* uma String cujo o conteúdo é alguma opção de borda do objeto de *Pair* (o qual é retornado pelo método **getProperties(String type)**), devolve o preço de determinado item.
- **void getPriceForSizesTest():** Testa se, ao receber no argumento *type* a String “sizes” e no argumento *item* uma String cujo o conteúdo é alguma opção de tamanho do objeto de *Pair* (o qual é retornado pelo método **getProperties(String type)**), devolve o preço de determinado item.
- **void getPriceForNumFlavoursTest():** Testa se, ao receber no argumento *type* a String “number of flavours” e no argumento *item* uma String cujo o

conteúdo é o número de sabores do objeto de *Pair* (o qual é retornado pelo método ***getProperties(String type)***), devolve o preço de determinado item.

### **PizzaTest.java**

Nesta classe foram testado 2 (dois) fatores da classe **PizzaInfo.java**: o método ***getSugaryFlavours()*** e o método ***getSaltyFlavours()***:

*public List<Flavours> getSugaryFlavours():*

- ***void shouldOnlyReturnSugaryTest()***: Cria uma lista de *String* de sabores, para posteriormente ser adicionada a um objeto de *Flavour*. Após adicionar 4 (quatro) objetos de *Flavour* na pizza, testa se o método filtra os objetos que possuírem “doce” em seu atributo *type*.
- ***void shouldOnlyReturnSaltyTest()***: Cria uma lista de *String* de sabores, para posteriormente ser adicionada a um objeto de *Flavour*. Após adicionar 4 (quatro) objetos de *Flavour* na pizza, testa se o método filtra os objetos que possuírem “salgado” em seu atributo *type*.

### **SaltyMenuTest.java**

Nesta classe foram testado 2 (dois) fatores da classe **SaltyMenu.java**: o método ***getIngredientsByType()*** e o método ***findType()***:

*public Pair[] getIngredientsByType(String type):*

- ***void getIngredientsByCheeseTest()***: Testa se ao receber no argumento *type* a *String* “cheese”, devolve o objeto da classe *Pair()* referente a tipos de queijos disponíveis e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.
- ***void getIngredientsByVegetableTest()***: Testa se ao receber no argumento *type* a *String* “vegetable”, devolve o objeto da classe *Pair()* referente a tipos de vegetais disponíveis e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.
- ***void getIngredientsByProteinTest()***: Testa se ao receber no argumento *type* a *String* “protein”, devolve o objeto da classe *Pair()* referente a tipos de proteínas disponíveis e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.
- ***void getIngredientsByGreenLeafTest()***: Testa se ao receber no argumento *type* a *String* “green leaf”, devolve o objeto da classe *Pair()* referente a tipos de folhas verdes disponíveis e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.

*public String findType(String ingredient):*

- ***void findCheeseTypeTest()***: Testa se a função ***findType(String ingredient)*** retorna *type* “cheese” caso o argumento *ingredient* receba um dos tipos de queijo disponíveis.



- **void findVegetableTypeTest():** Testa se a função **findType(String ingredient)** retorna *type* “vegetable” caso o argumento *ingredient* receba um dos tipos de vegetais disponíveis.
- **void findProteinTypeTest():** Testa se a função **findType(String ingredient)** retorna *type* “protein” caso o argumento *ingredient* receba um dos tipos de proteínas disponíveis.
- **void findGreenLeafTypeTest():** Testa se a função **findType(String ingredient)** retorna *type* “green leaf” caso o argumento *ingredient* receba um dos tipos de folhas verdes disponíveis.

### **SugaryMenuTest.java**

Nesta classe foram testado 2 (dois) fatores da classe **SugaryMenu.java**: o método **getIngredientsByType()** e o método **findType()**:

*public Pair[ ] getIngredientsByType(String type):*

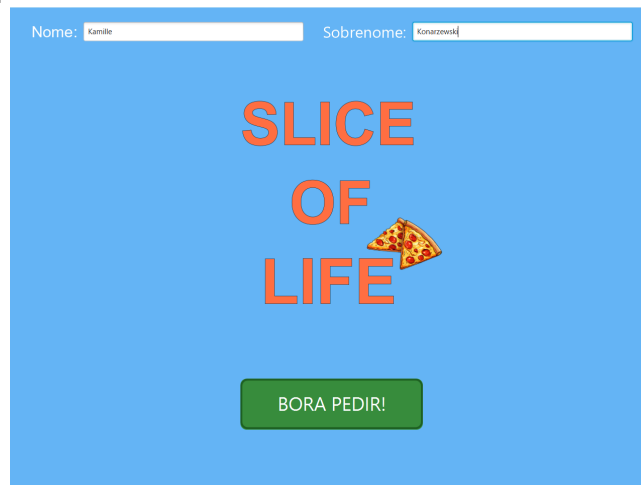
- **void getIngredientsByToppingTest():** Testa se ao receber no argumento *type* a *String* “topping”, devolve o objeto da classe *Pair()* referente a tipos de coberturas disponíveis e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.
- **void getIngredientsByFruitsTest():** Testa se ao receber no argumento *type* a *String* “fruits”, devolve o objeto da classe *Pair()* referente a tipos de frutas disponíveis e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.
- **void getIngredientsByCondimentTest():** Testa se ao receber no argumento *type* a *String* “condiment”, devolve o objeto da classe *Pair()* referente a tipos de proteínas disponíveis e seus respectivos preços. Para realizar tal teste, foi necessário extrair os valores do objeto de *Pair()* e compará-los um a um com a saída esperada.

*public String findType(String ingredient):*

- **void findToppingTypeTest():** Testa se a função **findType(String ingredient)** retorna *type* “topping” caso o argumento *ingredient* receba um dos tipos de cobertura disponíveis.
- **void findFruitsTypeTest():** Testa se a função **findType(String ingredient)** retorna *type* “fruits” caso o argumento *ingredient* receba um dos tipos de vegetais disponíveis.
- **void findCondimentTypeTest():** Testa se a função **findType(String ingredient)** retorna *type* “condiment” caso o argumento *ingredient* receba um dos tipos de proteínas disponíveis.

## 1.5 - Executável

Essa é a primeira tela que aparece, nela, o cliente precisa digitar seu nome e sobrenome para poder ir para a próxima tela.



A tela inicial do aplicativo Slice of Life. No topo, há dois campos de entrada de texto: "Nome:" com o valor "Kamile" e "Sobrenome:" com o valor "Konarszewski". Abaixo, o logo "SLICE OF LIFE" é exibido em letras maiúsculas, com "SLICE" e "LIFE" em laranja e "OF" em azul. Uma pequena imagem de uma fatia de pizza está ao lado da palavra "LIFE". No centro da tela, há um botão verde com o texto "BORA PEDIR!" em branco.

**Tela Inicial**

Nesta tela o usuário deve clicar em suas opções de preferência, ao escolher, o valor vai sendo somado à “total”.



A tela de escolha do formato da pizza. No topo, há uma barra de navegação com o logo "SLICE OF LIFE" e quatro abas: "Pedido", "Pagamento", "Endereço" e "Finalizar". A aba "Pedido" está selecionada. Abaixo, há três seções de opções:

- TAMANHO DA PIZZA:**
  - ☒ Pequena (25 cm) R\$ 25,00
  - ☐ Média (30 cm) R\$ 30,00
  - ☐ Grande (35 cm) R\$ 35,00
  - ☐ Família (40 cm) R\$ 40,00
- QUANTIDADE DE SABORES:**
  - ☒ 1 sabor R\$ 0,00
  - ☐ 2 sabores R\$ 1,00
  - ☐ 3 sabores R\$ 2,00
  - ☐ 4 sabores R\$ 3,00
- BORDA:**
  - ☒ Com Borda R\$ 1,00
  - ☐ Sem Borda R\$ 0,00

No rodapé, há um botão verde "VOLTAR" à esquerda, o texto "TOTAL DO PEDIDO: R\$ 26,00" no centro e um botão verde "SEGUIR" à direita.

**Tela de Escolha do Formato da Pizza**

Aqui seguem as duas telas para a escolha de sabores. A primeira tela a aparecer mostra as opções salgadas e, clicando no botão “Doces”, troca para a tela de ingredientes doces. Como as outras, o usuário é obrigado a escolher para passar para a tela seguinte: a de revisão de pedidos, caso os dados complementares já tenham sido preenchidos.

SLICE OF LIFE

Pedido

Pagamento

Endereço

Finalizar

ESCOLHA OS INGREDIENTES DO 1º SABOR

DOÇES

INGREDIENTES SALGADOS

QUEIJS

Gorgonzola  
R\$ 1,00

Ricotta  
R\$ 2,00

Requeijão  
R\$ 3,00

Cheddar  
R\$ 4,00

Brie  
R\$ 5,00

VEGETAL

Brócolis  
R\$ 1,00

Tomate  
R\$ 2,00

Pepradão  
R\$ 3,00

Cebola  
R\$ 4,00

Palmito  
R\$ 5,00

Cogumelo  
R\$ 6,00

PROTEINA

File  
R\$ 1,00

Salmon  
R\$ 2,00

Frango  
R\$ 3,00

Lombo  
R\$ 4,00

Bacon  
R\$ 5,00

Camarão  
R\$ 6,00

FOLHAS VERDES

Margencido  
R\$ 1,00

Salada  
R\$ 2,00

Rúcula  
R\$ 3,00

Agrão  
R\$ 4,00

Coentro  
R\$ 5,00

Espinache  
R\$ 6,00

VOLTAR

TOTAL DO PEDIDO: R\$ 28,00

SEGUIR

SLICE OF LIFE

Pedido

Pagamento

Endereço

Finalizar

ESCOLHA OS INGREDIENTES DO 1º SABOR

SALGADOS

INGREDIENTES DOÇES

COBERTURAS

Açúcar  
R\$ 1,00

Melo Amargo  
R\$ 2,00

Banana  
R\$ 3,00

Amargo  
R\$ 4,00

Pistache  
R\$ 5,00

Doce de Leite  
R\$ 6,00

Brigadeiro  
R\$ 7,00

Paçoca  
R\$ 8,00

Beijinho  
R\$ 9,00

Paçoca de Amendoim  
R\$ 10,00

CONDIMENTOS

Alho  
R\$ 1,00

Alho Poró  
R\$ 2,00

Alho Branco  
R\$ 3,00

Alho Preto  
R\$ 4,00

Alho Verde  
R\$ 5,00

Alho Marinho  
R\$ 6,00

VOLTAR

TOTAL DO PEDIDO: R\$ 31,00

SEGUIR

**Tela de Escolha e Sabor (Salgado ou Doce)**

Nessa tela, caso o usuário escolha pagar com cartão de crédito/débito, deve preencher as informações do cartão, caso contrário, basta preencher o endereço.

SLICE OF LIFE

Pedido

Pagamento

Endereço

Finalizar

TIPO DE PAGAMENTO

☒ Dinheiro

☐ Cartão de crédito/débito

☐ Pix

INFORMAÇÕES DO CARTÃO

Nome:

Validade:

Número:

CVV:

ENDEREÇO

Cidade:

Porto Alegre

Rua/Avenida:

Bento Gonçalves

Número:

1200

Complemento:

CEP:

93400290

VOLTAR

TOTAL DO PEDIDO: R\$ 31,00

SEGUIR

**Tela de Preenchimento de informações complementares**

Essa tela mostra uma revisão completa do pedido, permitindo que o usuário edite as pizzas ou exclua, caso haja mais de uma pizza já registrada. Caso o usuário clique em “Mais uma Pizza?”, ele será direcionado para “Tela de Escolha do Formato da Pizza”. Para finalizar o pedido, basta clicar no botão “Finalizar”.

The screenshot shows the 'Tela de Revisão do Pedido' (Order Review Screen) for 'SLICE OF LIFE'. The top navigation bar includes the logo and tabs for 'Pedido', 'Pagamento', 'Endereço', and 'Finalizar'. The main content area is divided into two columns. The left column, titled 'Listagem do Pedido', shows 'Pizza 1' with details: 'Pequena (25 cm)', '1 sabor', and 'Com borda'. Below this is a table titled 'Listagem das Pizzas' for 'Pizza 1 - R\$ 31,00', with columns for 'Sabor 1', 'Cobertura', 'Fruta', and 'Condimento'. The table shows 'Ao leite' for 'Cobertura', 'Uva' for 'Fruta', and '-' for 'Condimento'. The right column, titled 'Dados Pessoais', shows the name 'Kamille Konarzewski' and an 'Endereço' section with details: 'CEP: 93400290', 'Rua/Avenida Bento Gonçalves, Porto Alegre', 'Número: 1200', and 'Complemento: null'. Below this is a 'Pagamento' section showing 'Total do pedido: R\$ 31,00' and 'Tipo: Dinheiro'. At the bottom, there is a large orange button labeled 'MAIS UMA PIZZA?'. The bottom navigation bar includes a 'VOLTAR' button, the 'TOTAL DO PEDIDO: R\$ 31,00', and a 'FINALIZAR' button.

**Tela de Revisão do Pedido**

Aqui segue a tela final, onde, clicando em “Início”, o usuário reinicia o processo. Podendo fazer outro pedido.

The screenshot shows the 'Tela final' (Final Screen) for 'SLICE OF LIFE'. The top navigation bar includes the logo. The main content area is a large blue rectangle with a white background. In the center, there is an illustration of a delivery person on a red motorcycle, wearing a red helmet and a red backpack. Below the illustration, the text 'ACOMPANHE SEU PEDIDO' is displayed in large, orange, outlined letters. At the bottom, there is a green button labeled 'INÍCIO'.

**Tela final**

### **Experiência do grupo em utilizar testes unitários**

Para o grupo, realizar os testes unitários durante a implementação das classes ajudou a ter um maior conhecimento de algumas dificuldades e sobre o que poderia ser feito para superá-las. Por mais que a grande maioria das barreiras que enfrentamos no projeto tenha sido referente a interface, os testes se provaram úteis para enfrentar alguns empecilhos que surgiram ao tentar estabelecer fluxo de dados entre telas.