

Ano Letivo 2023/2024 | 1º Semestre

JavaEats

Aplicação de gestão para restaurantes

Projeto Temático em Desenvolvimento de Aplicações

Licenciatura em Tecnologias da Informação

Autores:

Bruno Migueis | 102238

Daniel Silva | 113900

Gabriel Cravo | 87689

Lucas Duarte | 109190

Miguel Pirré | 114017

Grupo 2

Águeda | 19 de janeiro de 2024



Ano Letivo 2023/2024 | 1º Semestre

JavaEats

Aplicação de gestão para restaurantes

Projeto Temático em Desenvolvimento de Aplicações

Licenciatura em Tecnologias da Informação

Autores:

Bruno Migueis | 102238 | brunomigueis@ua.pt

Daniel Silva | 113900 | d.m.silva@ua.pt

Gabriel Cravo | 87689 | gcravo@ua.pt

Lucas Duarte | 109190 | lucasdurante2@ua.pt

Miguel Pirré | 114017 | pirre@ua.pt

Professor Orientador:

Gonçalo Paiva Dias

Grupo 2

Águeda | 19 de janeiro de 2024

Agradecimentos

Serve este espaço para agradecer a quem merece, a quem acredita em nós, e a quem luta incondicionalmente pelo nosso sucesso e bem-estar profissional. Deixamos os seguintes agradecimentos: Em primeiro lugar, ao nosso professor orientador do projeto temático, o docente Gonçalo Paiva Dias que acompanhou todos os nossos passos ao longo de todo o trabalho, orientando-nos sempre da melhor forma possível. De seguida, ao docente Joaquim Ferreira, um dos professores responsáveis pelo Módulo Temático em Desenvolvimento de Aplicações. Por fim, mas não menos importante, agradecemos a todos os nossos colegas, que de uma forma ou de outra, contribuíram para que conseguíssemos levar a bom porto esta tarefa, com conselhos e incentivos variados. À ESTGA e a todas estas pessoas e outras que cruzaram o nosso caminho nesta jornada o nosso muito obrigado. Um grande bem-haja para todos vós!

Resumo

O presente relatório foi elaborado como parte integrante do Projeto Temático em Desenvolvimento de Aplicações, associado às unidades curriculares Engenharia de Software e Sistemas de Bases de Dados, constituindo assim o Módulo Temático em Desenvolvimento de Aplicações, orientado no âmbito da Licenciatura em Tecnologias da Informação da Escola Superior de Tecnologia e Gestão de Águeda. O projeto foi encaminhado pelo docente Gonçalo Paiva Dias durante o primeiro semestre do segundo ano da licenciatura.

Pretendia-se com este projeto analisar os requisitos do sistema e construir o seu modelo conceptual, desenvolver um modelo de desenho correspondente, incluindo o esquema relacional da base de dados e o modelo de desenho orientado a objetos da aplicação; implementar bases de dados relacionais num sistema de gestão de bases de dados, utilizando a linguagem SQL; implementar uma aplicação a instalar em cada cliente com recurso a uma linguagem de programação orientada a objetos (Java), incluindo acesso à bases de dados; e, por fim, adotar uma estratégia de teste e especificar os respetivos casos de teste.

Numa fase inicial descreveu-se o âmbito do trabalho, baseado no tema escolhido, fez-se o planeamento do projeto, identificou-se e caracterizou-se os requisitos necessários, funcionais, não funcionais e de hardware a ser implementados, bem como os utilizadores do sistema. Posteriormente seguiu-se o desenho do sistema, onde se identificou e descreveu os casos de utilização, identificou e caracterizou todas as classes/interfaces, as suas associações, atributos e métodos, e onde se produziu o modelo de dados persistente. Por fim, na fase de implementação e testes, implementou-se a solução.

O presente trabalho culmina com uma reflexão final acerca deste percurso ao longo do desenvolvimento do sistema de gestão para estabelecimentos na área da restauração, bem como algumas sugestões a colmatar lacunas do mesmo.

Índice geral

1.	Introdução	1
1.1	Enquadramento.....	1
1.2	Objetivos do Projeto	1
1.3	Público-alvo.....	2
2.	Estado da arte.....	3
2.1	Aplicações Relevantes.....	3
3.	Planeamento e Execução.....	6
3.1	Plano do projeto.....	6
3.2	Metodologia adotada	6
3.3	Calendarização das tarefas prevista	7
3.4	Calendarização das tarefas executada.....	8
3.5	Divisão de tarefas	9
4.	Modelo de requisitos	10
4.1	Requisitos funcionais.....	10
4.2	Restrições e requisitos não funcionais.....	12
4.3	Requisitos de interface e facilidade de uso.....	14
4.4	Requisitos de desempenho.....	15
4.5	Requisitos de segurança e integridade dos dados	15
4.6	Requisitos de interface com sistemas externos e ambientes de execução	16
5.	Normas e regulamentação específicas aplicáveis	17
5.1	Requisitos de hardware.....	18
6.	Casos de Utilização	19
6.1	Atores	19
6.2	Casos de utilização	20
6.2.1	Criar perfil.....	21
6.2.2	Editar perfil	21
6.2.3	Remover perfil	22
6.2.4	Criar mesa	22
6.2.5	Remover mesa.....	23

6.2.6	Editar menu	23
6.2.7	Criar pedidos	24
6.2.8	Remover pedido	24
6.2.9	Visualizar status da mesa.....	25
6.2.10	Fechar conta.....	25
6.2.11	Receber pedido	26
6.2.12	Atualizar status do pedido.....	26
6.2.13	Editar itens mesa.....	27
6.2.14	Consultar relatório de vendas.....	27
6.2.15	Emitir relatório de vendas.....	28
6.2.16	Criar categorias	28
6.2.17	Editar categorias	29
6.2.18	Apagar categorias	29
6.2.19	Criar zonas	30
6.2.20	Editar zonas.....	30
6.2.21	Apagar zonas.....	31
6.2.22	Criar Cargos.....	31
6.2.23	Editar Cargos	32
6.2.24	Apagar Cargos	32
6.3	Cobertura de requisitos	33
7.	Prototipagem baixa-fidelidade	34
8.	Análise de tecnologias	36
9.	Desenvolvimento da Aplicação.....	41
9.1	Diagrama de Classes.....	41
9.2	Modelo de Dados Persistente	43
9.3	Website	44
10.	Interface	45
10.1	Dashboard Principal	45
10.2	Dashboard Gerente	46
10.3	Dashboard Gerente (Ver Balcão).....	47
10.4	Dashboard Gerente (Gerir Cargos).....	48

10.5	Dashboard Gerente (Gerir Categorias)	49
10.6	Dashboard Gerente (Gerir Funcionários)	50
10.7	Dashboard Gerente (Gerir Mesas)	51
10.8	Dashboard Gerente (Gerir Zonas)	52
10.9	Dashboard Gerente (Gerir Itens)	53
10.10	Dashboard Gerente (Relatórios)	54
10.11	Dashboard Funcionário.....	56
10.12	Dashboard Reservar Mesas	57
10.13	Dashboard Pedido.....	58
10.13.1	Dashboard Pedido (Transferir Mesa)	60
10.13.2	Dashboard Pedido (Consulta de Mesa).....	61
10.13.3	Dashboard pedido (Imprimir 2 ^a via do recibo)	63
10.14	Dashboard Fechar Conta	65
10.15	Dashboard Balcão.....	67
11.	Manual de Instalação	68
11.1	Requisitos obrigatórios do sistema	68
11.2	Passos de instalação.....	68
11.2.1	Java	68
11.2.2	JDK.....	70
11.2.3	Ficheiro executável JavaEats	72
11.3	Resoluções de problemas.....	75
11.4	Tags RFID	78
12.	Testes e validação.....	81
12.1	Testes unitários	81
12.1.1	CargoTest	82
12.1.2	CategoriaTest	84
12.1.3	FuncionarioTest.....	85
12.1.4	ItemPedidoTest	88
12.1.5	ItemTest	92
12.1.6	MesaTest	95
12.1.7	PagamentoAtendimentoTest	97

12.1.8	PedidoTest.....	103
12.1.9	SerialCommunicationServiceTest.....	106
12.1.10	SessaoUtilizadorTest.....	108
12.1.11	ZonaTest.....	112
12.2	Testes de integração.....	115
12.3	Testes de sistema e de validação.....	115
12.4	Testes de usabilidade	116
13.	Análise de Resultados	120
14.	Reflexão Crítica e Conclusões	122
14.1	Atividades Desenvolvidas	122
14.2	Estratégias de Trabalho Adotadas.....	122
14.3	Planeamento Previsto e Cronograma Executado.....	123
14.4	Sugestões para o Futuro.....	123
14.5	Síntese das Experiências.....	124
14.6	Conclusão	125
15.	Bibliografia	126
ANEXO A	127
ANEXO B	132
ANEXO C	135
ANEXO D	137
ANEXO E	140

Índice de tabelas

Tabela 1 - Requisitos funcionais	11
Tabela 2 - Restrições	12
Tabela 3 - Requisitos não funcionais	13
Tabela 4 - Requisitos de interface e facilidade de uso	14
Tabela 5 - Requisitos de desempenho	15
Tabela 6 - Requisitos de segurança e integridade dos dados.....	15
Tabela 7 - Requisitos de interface com sistemas externos e ambientes de execução.....	16
Tabela 8 - Requisitos de hardware computador	18
Tabela 9 - Requisitos de hardware servidor	18
Tabela 10 - Atores do sistema	19
Tabela 11 - Caso de utilização criar perfil.....	21
Tabela 12 - Caso de utilização editar perfil.....	21
Tabela 13 - Caso de utilização remover perfil	22
Tabela 14 - Caso de utilização criar mesa.....	22
Tabela 15 - Caso de utilização remover mesa	23
Tabela 16 - Caso de utilização editar menu	23
Tabela 17 - Caso de utilização criar pedidos.....	24
Tabela 18 - Caso de utilização remover pedido	24
Tabela 19 - Caso de utilização visualizar status da mesa	25
Tabela 20 - Caso de utilização fechar conta.....	25
Tabela 21 - Caso de utilização receber pedido.....	26
Tabela 22 - Caso de utilização atualizar status do pedido.....	26
Tabela 23 - Caso de utilização editar itens da mesa.....	27
Tabela 24 - Caso de utilização consultar relatório de vendas	27
Tabela 25 - Caso de utilização emitir relatório de vendas.....	28
Tabela 26 - Caso de utilização criar categorias	28
Tabela 27 - Caso de utilização editar categorias	29
Tabela 28 - Caso de utilização apagar categoria	29
Tabela 29 - Caso de utilização criar zonas	30

Tabela 30 - Caso de utilização editar zonas	30
Tabela 31 - Caso de utilização apagar zonas.....	31
Tabela 32 - Caso de utilização criar cargo	31
Tabela 33 - Caso de Utilização editar cargo.....	32
Tabela 34 - Caso de utilização apagar cargo	32
Tabela 35 - Cobertura de requisitos	33
Tabela 36 - Testes de usabilidade funcionário	116
Tabela 37 - Testes de usabilidade gerente	117
Tabela 38 - Análise de resultados.....	121

Índice de figuras

Figura 1 - TheFork Manager	4
Figura 2 - Vendus POS System	4
Figura 3 - Posmovi.....	5
Figura 4 - Calendarização prevista.....	7
Figura 5 - Calendarização executada	8
Figura 6 - Divisão das tarefas	9
Figura 7 - Diagrama de casos de utilização	20
Figura 8 - Dashboard principal	34
Figura 9 - Dashboard mesas.....	35
Figura 10 - Dashboard pedido.....	35
Figura 11 - Mysql Workbench	36
Figura 12 - Github página do projeto javaeats	37
Figura 13 - IntelliJ projeto javaeats	38
Figura 14 - Scene builder (dashboard fechar conta)	39
Figura 15 - Discord	39
Figura 16 - Zoom	40
Figura 17 - Diagrama de classes	42
Figura 18 - Modelo de Base de dados Conceptual.....	43
Figura 19 - Modelo da base de dados relacional.....	43
Figura 20 - Website JavaEats	44
Figura 21 - Dashboard principal	45
Figura 22 - Dashboard principal login	46
Figura 23 - Dashboard gerente.....	46
Figura 24 - Dashboard balcão	47
Figura 25 - Dashboard gerente cargos	48
Figura 26 - Dashboard gerente categorias.....	49
Figura 27 - Dashboard gerente funcionários.....	50
Figura 28 - Dashboard gerente mesas.....	51
Figura 29 - Dashboard gerente zonas.....	52

Figura 30 – Dashboard gerente itens.....	53
Figura 31 - Dashboard gerente relatórios.....	54
Figura 32 - Relatório PDF.....	55
Figura 33 - Dashboard principal funcionário	56
Figura 34 - Dashboard funcionário mesas	56
Figura 35 - Dashboard funcionário reservar	57
Figura 36 - Dashboard funcionário próxima reserva	57
Figura 37 - Dashboard funcionário pedido	58
Figura 38 - Dashboard funcionário adicionar itens.....	59
Figura 39 - Dashboard funcionário pedido criado	59
Figura 40 - Dashboard funcionário transferir mesa	60
Figura 41 - Dashboard funcionário mesa transferida.....	60
Figura 42 - Dashboard funcionário consulta de mesa.....	61
Figura 43 - Recibo pedido.....	62
Figura 44 - Imprimir 2 ^a via do recibo	63
Figura 45 - Recibo 2º via atendimento.....	64
Figura 46 - Dashboard fechar conta	65
Figura 47 - Dashboard fechar conta pagamento	65
Figura 48 - Recibo atendimento.....	66
Figura 49 - Dashboard principal balcão	67
Figura 50 - Dashboard balcão	67
Figura 51 - Download java	68
Figura 52 - Confirmar download	69
Figura 53 - Executar o ficheiro	69
Figura 54 - Instalar o java	70
Figura 55 - Download jkd development kit	70
Figura 56 - Instalar o ficheiro jdk	71
Figura 57 - Clicar no continuar	71
Figura 58 - Confirmar a pasta de instalação	72
Figura 59 - Download ficheiro executável JavaEats.....	72
Figura 60 - Descompactar ficheiro JavaEats.zip.....	73

Figura 61 - Escolha da pasta para descompactação	73
Figura 62 - Pasta JavaEats	74
Figura 63 - Ficheiro JavaEats na pasta.....	74
Figura 64 - Aplicação JavaEats ao abrir.....	74
Figura 65 - Erro ao executar a aplicação.....	75
Figura 66 - Pesquisar variáveis	75
Figura 67 - Variáveis de ambiente.....	76
Figura 68 - Escolher path e depois editar.....	76
Figura 69 - Variáveis de ambiente necessárias.....	77
Figura 70 - Código do arduino (RFID)	79
Figura 71 - Pulseira RFID	80
Figura 72 - Arduino rfid esquema	80
Figura 73 - Cargotest (testToString()).....	82
Figura 74 - Cargotest (getDescricao())	82
Figura 75 - Cargotest (getIdCargo())	82
Figura 76 - Cargotest (setDescricao()).....	83
Figura 77 - Cargotest (setIdCargo()).....	83
Figura 78 - Cargotest (exibirInformacoesCargo())	83
Figura 79 - Categoriatest (testToString())	84
Figura 80 - Categoriatest (getNome()).....	84
Figura 81 - Categoriatest (setNome())	84
Figura 82 - Categoriatest (getId())	85
Figura 83 - Categoriatest (setId()).....	85
Figura 84 - Funcionariotest (getNome())	85
Figura 85 - Funcionariotest (setNome())	86
Figura 86 - Funcionariotest (getPassword())	86
Figura 87 - Funcionariotest (setPassword())	86
Figura 88 - Funcionariotest (getId())	86
Figura 89 - Funcionariotest (setId()).....	87
Figura 90 - Funcionariotest (getIdCargo()).....	87
Figura 91 - Funcionariotest (setIdCargo())	87

Figura 92 - Funcionariotest (getIdCartao()).....	87
Figura 93 - Funcionariotest (setIdCartao())	88
Figura 94 - Itempeditotest (getProduto())	88
Figura 95 - Itempeditotest (setProduto()).....	88
Figura 96 - Itempeditotest (produtoProperty()).....	89
Figura 97 - Itempeditotest (getQuantidade())	89
Figura 98 - Itempeditotest (setQuantidadadade())	89
Figura 99 - Itempeditotest (quantidadeProperty()).....	90
Figura 100 - Itempeditotest (getQuantidadeOriginal())	90
Figura 101 - Itempeditotest (setQuantidadeOriginal()).....	90
Figura 102 - Itempeditotest (getPreco()).....	91
Figura 103 - Itempeditotest (setPreco())	91
Figura 104 - Itempeditotest (precoProperty())	91
Figura 105 - Itempeditotest (getIdItem())	92
Figura 106 - Itempeditotest (setIdItem()).....	92
Figura 107 - Itemtest (getPreco()).....	92
Figura 108 - Itemtest (setPreco())	93
Figura 109 - Itemtest (getNome())	93
Figura 110 - Itemtest (setNome()).....	93
Figura 111 - Itemtest (getId()).....	94
Figura 112 - Itemtest (setId())	94
Figura 113 - Itemtest (getIdCategoria())	94
Figura 114 - Itemtest (setIdCategoria())	95
Figura 115 - Mesatest (editarStatus()).....	95
Figura 116 - Mesatest (getStatus()).....	95
Figura 117 - Mesatest (getId())	96
Figura 118 - Mesatest (setId())	96
Figura 119 - Mesatest (getId_zona()).....	96
Figura 120 - Mesatest (setId_zona())	97
Figura 121 - Pagamentoatendimentotest (getId())	97
Figura 122 - Pagamentoatendimentotest (setId())	98

Figura 123 - Pagamentoatendimentotest (getDataHoraInicio())	98
Figura 124 - Pagamentoatendimentotest (setDataHoraInicio())	99
Figura 125 - Pagamentoatendimentotest (getDataHoraFim())	99
Figura 126 - Pagamentoatendimentotest (setDataHoraFim())	100
Figura 127 - Pagamentoatendimentotest (getPrecoTotal())	100
Figura 128 - Pagamentoatendimentotest (setPrecoTotal())	101
Figura 129 - Pagamentoatendimentotest (getTipoPagamento())	101
Figura 130 - Pagamentoatendimentotest (setTipoPagamento())	102
Figura 131 - Pagamentoatendimentotest (getIdMesa())	102
Figura 132 - Pagamentoatendimentotest (setIdMesa())	103
Figura 133 - Pidotest (pedido())	103
Figura 134 - Pidotest (setDescricao())	104
Figura 135 - Pidotest (getDescricao())	104
Figura 136 - Pidotest (getId())	105
Figura 137 - Pidotest (setId())	105
Figura 138 - Serialcommunicationservicetest (getPortaSerial())	106
Figura 139 - Serialcommunicationservicetest (setPortaSerial())	106
Figura 140 - Serialcommunicationservicetest (verificarFuncionarioPorUID())	107
Figura 141 - Serialcommunicationservicetest (getUltimoUID())	107
Figura 142 - Serialcommunicationservicetest (setUltimoUID())	108
Figura 143 - Sessaoutilizadortest (getInstance())	108
Figura 144 - Sessaoutilizadortest (testGetInstance())	109
Figura 145 - Sessaoutilizadortest (getIdUtilizador())	109
Figura 146 - Sessaoutilizadortest (getCargo())	110
Figura 147 - Sessaoutilizadortest (getNomeFuncionario())	110
Figura 148 - Sessaoutilizadortest (setInstance())	111
Figura 149 - Sessaoutilizadortest (terminarSessaoUtilizador())	111
Figura 150 - Sessaoutilizadortest (testToString())	112
Figura 151 - Zonatest (testToString())	112
Figura 152 - Zonatest (getNome())	113
Figura 153 - Zonatest (setNome())	113

Figura 154 - Zonatest (getId()).....	114
Figura 155 - Zonatest (setId()).....	114
Figura 156 - Documento de consentimento de participação no teste.....	118
Figura 157 - Questionário de satisfação.....	119

Listas de siglas e abreviaturas

ESTGA	Escola Superior de Tecnologia e Gestão de Águeda
POS	<i>Point of Sale</i> (Ponto de Venda)
RF	Requisitos funcionais
Ref	Referência
RSeg	Requisito de segurança e integridade
RDes	Requisito de desempenho
RInt	Requisito de interfaces e facilidades de uso
RNF	Requisito não funcional
RI	Requisito de interface com sistemas externos e ambientes de execução
PBF	Protótipo de baixa-fidelidade
RFID	Radio Frequency Identification
NFC	Near Field Communication
PDV	Ponto de venda
IDE	Integrated Development Environment
JDK	Java Development Kit

1. Introdução

No âmbito do Módulo Temático em Desenvolvimento de Aplicações, do segundo semestre do segundo ano da Licenciatura em Tecnologias da Informação da Escola Superior de Tecnologias e Gestão de Águeda, o qual engloba, para além do Projeto Temático em Desenvolvimento de Aplicações, as unidades curriculares associadas: Sistemas de Bases de Dados e Engenharia de Software, foi proposto elaborar uma aplicação, em Java, que tivesse conexão e manipulação com uma base de dados. Decidimos então, em grupo, dar origem ao **JavaEats**, uma aplicação de gestão para estabelecimentos na área da restauração. Neste relatório serão debatidos e abordados os seguintes capítulos para além da própria Introdução: Planeamento e Execução do trabalho; Levantamento do Estado de Arte; Esquema Funcional e modelo de Requisitos; Casos de Utilização; Análise de Tecnologias; Modelo de dados Persistente e Modelo estrutural; Desenvolvimento da aplicação; Análise de Resultados; Testes e validação; Reflexão Crítica e Conclusão.

É ainda composto por uma Bibliografia e Anexos. Estes pretendem complementar a informação apresentada nos capítulos anteriores.

1.1 Enquadramento

Existe uma forte procura por parte de gestores de estabelecimentos tais como restaurantes, cafés, bares, entre outros, por otimizar e reduzir os erros de gestão de pedidos, em prol da eficiência e satisfação do cliente. O que abre uma janela para uma solução tecnológica que atenda às dores desse nicho. Um sistema como o nosso, simples e prático de utilizar no dia a dia, pode ser bastante útil para responder a esse problema, permitindo a automatização de processos como a gestão de pedidos, reservas, pagamentos, faturação, entre outros. Com a implementação do **JavaEats**, os nossos utilizadores podem melhorar a eficiência dos seus estabelecimentos, reduzir o tempo de espera dos clientes e aumentar a satisfação dos mesmos.

1.2 Objetivos do Projeto

O projeto visa complementar com uma interface em Java e JavaFX, o sistema que será desenvolvido. O sistema resultante permitirá uma grande melhoria face à não utilização deste, nos pontos referentes a:

Eficiência – A aplicação permitirá aos funcionários do restaurante agilizar o processo de pedidos, isto é, estes poderão inserir os pedidos por mesa de forma rápida e precisa, devido à organização do menu, por categorias, conseguindo assim um atendimento mais fácil e eficiente, reduzindo o risco de erros de comunicação entre funcionários. Para além disso, editar, remover ou adicionar itens, categorias, mesas, entre outros, será um processo muito mais rápido e eficiente para o gerente.

Gestão de mesas – A aplicação possibilitará uma visão clara do estado de cada mesa, esclarecendo rapidamente ao empregado se está disponível, ocupada ou reservada. Isto proporcionará uma melhor gestão de tempo, monitorização e capacidade do estabelecimento.

Pagamentos – A aplicação permitirá ver o estado de cada mesa (pago ou não pago), os itens pedidos por cliente e o valor total do aglomerado de pedidos, calculado automaticamente pelo sistema, bem como o troco (caso o pagamento seja em numerário) o que irá reduzir significativamente erros de cálculo humano.

Estatísticas – A aplicação possibilitará, ao gerente, ter acesso a diversas estatísticas relevantes, tais como gráficos de faturação por dia, por mesa, quantidade e valor de pagamentos feitos em cartão e em numerário, itens e categorias mais populares, entre outros. O que não só facilitará a gestão do estabelecimento, como também levará a uma visão mais realista do que os clientes pretendem e gostam.

1.3 Público-alvo

O JavaEats é direcionado a estabelecimentos da área da restauração, que desejam aperfeiçoar o processo de gestão dos mesmos, através de uma solução prática e completa que permita otimizar as operações diárias. O nosso propósito é exatamente fornecer soluções versáteis para atender às múltiplas necessidades dos restaurantes e gestores, através duma interface intuitiva e a capacidade do gerente ter uma *dashboard* única onde terá controlo total do seu negócio. Promovendo assim uma melhor experiência dos clientes e um aumento significativo na eficiência operacional.

2. Estado da arte

Um pouco por todo o mundo existem notícias da implementação de sistemas e tecnologias com o objetivo de auxiliar e automatizar a gestão e monitoramento de restaurantes, tipicamente com algum problema de mobilidade.

2.1 Aplicações Relevantes

Numa fase inicial do projeto teve-se como objetivo analisar as soluções já existentes no mercado e estudar como são implementadas. Foram extraídas ideias e funcionalidades úteis dessas soluções para melhorar o desenvolvimento do trabalho.

Identificaram-se aplicações que embora não respondam completamente às necessidades do projeto a desenvolver, considerou-se importante a sua identificação. Essas aplicações mostraram-nos as funcionalidades que ao implementarmos toda a gestão da nossa aplicação irá melhorar. O *TheFork Manager* mostrou-nos a necessidade de criação de eventos como reservas de mesa e de criar a possibilidade de gestão dos funcionários, mesas, menu etc... A aplicação *Vendus POS System* tornou-se um ponto de inspiração nossa no que toca ao design por nos mostrar um design simples, intuitivo e de fácil utilização, mostrou-nos também a importância de permitir gerar relatórios de vendas para uma gestão e avaliação de resultados mais concisa e simples. Por fim a aplicação *POSMOVI* mostrou-nos a importância da gestão das mesas e da sua organização no restaurante, implementar a possibilidade de separar as mesas por piso ou área ajuda na criação de reservas, entrega de pratos

TheFork Manager

Conforme as informações recolhidas na nossa pesquisa, a empresa "TheFork Manager" disponibiliza dois serviços - software e hardware. O serviço de software consiste num painel de controlo que permite a monitorização em tempo real e a apresentação de dados relativos aos restaurantes, tais como estatísticas de reservas, registo de pedidos e eventos. No serviço de hardware, é disponibilizado um dispositivo a ser conectado aos sistemas do restaurante, permitindo a gestão e monitorização dos dados a serem enviados para a plataforma digital.

Esta solução foi uma das escolhidas por possuir características que o grupo considerou fundamentais para o desenvolvimento do projeto. Uma característica desejada para integrar na solução proposta pelo grupo é o conceito de reservas, permitindo assim uma gestão eficiente do espaço e das mesas do restaurante, bem como uma melhor experiência para os clientes.

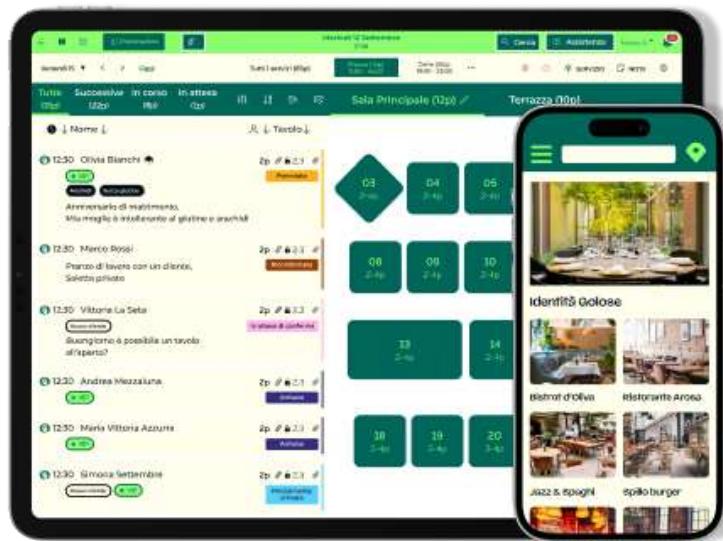


FIGURA 1 - THEFORK MANAGER

Vendus POS System

Este serviço de software consiste num painel de controlo que permite a monitorização em tempo real e a apresentação de dados relacionados com as transações de vendas, o stock e os relatórios de gestão. Esta solução foi selecionada por conter características consideradas essenciais para o desenvolvimento do projeto. Uma delas é a interface gráfica, pois oferece cores simples e claras; a outra é a criação de relatórios detalhados relativamente às vendas. Optamos por não incluir o stock na aplicação, pois consideramos que não seria relevante para os objetivos do projeto.



FIGURA 2 - VENDUS POS SYSTEM

POSMOVI

Este Sistema POS (Ponto de Venda) consiste num sistema de vendas dedicado a restaurantes, proporcionando uma interface gráfica de mesas personalizadas que permite criar áreas específicas para as mesas, sendo este um dos pontos destacados. Esta aplicação foi escolhida pela sua simplicidade na adição de produtos, gestão de categorias e visualização dos itens adicionados ao pedido. Além disso, a possibilidade de ter várias áreas no restaurante é uma mais-valia, especialmente se houver esplanada ou outro piso disponível.



FIGURA 3 - POSMOVI

3. Planeamento e Execução

No início do trabalho foi necessário utilizar os conhecimentos adquiridos em Engenharia de Software relacionados com a modelação de um projeto. Este ponto demonstra o trabalho necessário antes de se passar para a implementação da aplicação em si.

3.1 Plano do projeto

Antes de se abordar o escalonamento das tarefas propriamente dito coligiram-se as tarefas, que se consideraram macro, a realizar. Com isto verificou-se que será necessário o seguinte:

- ◆ Documentação do projeto por meio de um relatório entregue em duas fases;
- ◆ Descrição do tema e dos utilizadores do sistema
- ◆ Identificação dos requisitos e funcionalidades do sistema;
- ◆ Levantamento do estado da arte;
- ◆ Descrição dos casos de utilização que fundamentam o que o sistema irá permitir fazer;
- ◆ Identificação e caracterização de todas as classes/interfaces, associações, atributos e métodos
- ◆ Produção do modelo de dados persistente;
- ◆ Implementação da aplicação;
- ◆ Realização de testes de software;

De realçar também que as reuniões semanais, redação das atas, criação e gestão do grupo, bem como estrutura do relatório também se incluem como tarefas.

Todas estas atas podem ser vistas no **ANEXO E** de forma a se verificar o que foi discutido sobre o projeto ao longo do tempo.

3.2 Metodologia adotada

Neste projeto, adotou-se a “metodologia em cascata” como metodologia de desenvolvimento de software. Este modelo consiste em cinco fases sequenciais e lineares: Análise de Requisitos, Planeamento, Implementação, Testes e Manutenção. Cada fase tem como pré-requisito a conclusão da fase anterior e produz um resultado que serve de base para a próxima fase. A principal vantagem deste modelo é a sua simplicidade e facilidade de aplicação, que facilitam a documentação e o controlo da qualidade. A principal desvantagem é a sua rigidez e inflexibilidade, que dificultam a adaptação a mudanças nos requisitos ou no design ao longo do processo, podendo causar problemas de comunicação, atrasos e desperdícios. No entanto, estes problemas não se verificaram neste projeto e este modelo resultou muito bem.

3.3 Calendarização das tarefas prevista

No âmbito da fase de planeamento, a elaboração de um cronograma é uma forma de organizar e gerir as tarefas durante a execução de um projeto. Consiste na determinação da melhor forma de posicionar as tarefas ao longo do tempo de acordo com a duração das mesmas, das relações de precedência entre elas e dos prazos a cumprir.

Com base nisto, foi elaborado um cronograma correspondente a este projeto no qual estão presentes as atividades previstas bem como o tempo previsto de execução de cada uma delas.

#	Tarefa	Duração	Início	Conclusão
1	Fase I (Planeamento)	15 dias	Qui, 12/10/2023	Qui, 26/10/2023
2	Criação do repositório para o projeto	30 minutos	Qui, 12/10/2023	Qui, 12/10/2023
3	Criação dos meios de comunicação para o grupo	10 minutos	Qui, 12/10/2023	Qui, 12/10/2023
4	Estruturação do relatório	3 horas	Qui, 12/10/2023	Qui, 12/10/2023
5	Identificação dos requisitos do sistema	4 dias	Sex, 13/10/2023	Seg, 16/10/2023
6	Identificação das funcionalidades do sistema	4 dias	Ter, 17/10/2023	Sex, 20/10/2023
7	Identificação dos utilizadores do sistema	1 dia	Sab, 21/10/2023	Sab, 21/10/2023
8	Levantamento do estado da arte	5 dias	Dom, 22/10/2023	Qui, 26/10/2023
9	Fase II (Desenho do sistema)	22 dias	Sex, 27/10/2023	Sex, 17/11/2023
10	Identificação dos casos de utilização	3 dias	Sex, 27/10/2023	Dom, 29/10/2023
11	Criação do diagrama de casos de utilização	2 dias	Seg, 30/10/2023	Ter, 31/10/2023
12	Descrição dos casos de utilização	3 dias	Qui, 02/11/2023	Sab, 04/11/2023
13	Identificação de todas as classes/interfaces	4 dias	Dom, 05/11/2023	Qua, 08/11/2023
14	Produção do modelo estrutural	3 dias	Qui, 09/11/2023	Sab, 11/11/2023
15	Produção do modelo de dados persistente	4 dias	Dom, 12/11/2023	Qua, 15/11/2023
16	Protótipo de baixa fidelidade	2 dias	Qui, 16/11/2023	Sex, 17/11/2023
17	Entregas das Fases I e II	15 minutos	Sex, 17/11/2023	Sex, 17/11/2023
18	Fase III (Implementação)	63 dias	Sab, 18/11/2023	Sex, 19/01/2024
19	Implementação da base de dados	4 dias	Sab, 18/11/2023	Ter, 21/11/2023
20	Desenvolvimento da aplicação	40 dias	Qua, 22/11/2023	Dom, 31/12/2023
21	Testes	5 dias	Ter, 02/01/2024	Sab, 06/01/2024
22	Análise de resultados e documentação	2 dias	Dom, 07/01/2024	Seg, 08/01/2024
23	Conclusão da escrita do relatório final	11 dias	Ter, 09/01/2024	Sex, 19/01/2024
24	Entrega final	15 minutos	Sex, 19/01/2024	Sex, 19/01/2024
25	Reuniões e Atas	92 dias	Seg, 16/10/2023	Seg, 15/01/2024

FIGURA 4 - CALENDARIZAÇÃO PREVISTA

3.4 Calendarização das tarefas executada

Apresenta-se, em seguida, a execução real do trabalho, isto é, como realmente as tarefas se sucederam ao longo do trabalho. À medida que se iam realizando e implementando as tarefas, observou-se que eram necessários ajustes relativamente às mesmas. Veja-se a figura refere a designação de cada tarefa, duração, data de início e conclusão.

#	Tarefa	Duração	Início	Conclusão
1	Fase I (Planeamento)	11 dias	Qui, 12/10/2023	Qui, 22/10/2023
2	Criação do repositório para o projeto	30 minutos	Qui, 12/10/2023	Qui, 12/10/2023
3	Criação dos meios de comunicação para o grupo	10 minutos	Qui, 12/10/2023	Qui, 12/10/2023
4	Estruturação do relatório	3 horas	Qui, 12/10/2023	Qui, 12/10/2023
5	Levantamento do estado da arte	3 dias	Sex, 13/10/2023	Dom, 15/10/2023
6	Identificação das funcionalidades do sistema	3 dias	Seg, 16/10/2023	Qua, 18/10/2023
7	Identificação dos utilizadores do sistema	1 dia	Qui, 19/10/2023	Qui, 19/10/2023
8	Identificação dos requisitos do sistema	3 dias	Sex, 20/10/2023	Dom, 22/10/2023
9	Fase II (Desenho do sistema)	26 dias	Seg, 23/10/2023	Sex, 17/11/2023
10	Identificação dos casos de utilização	1 dia	Seg, 23/10/2023	Seg, 23/10/2023
11	Criação do diagrama de casos de utilização	1 dia	Ter, 24/10/2023	Ter, 24/10/2023
12	Protótipo de baixa fidelidade	2 dias	Qua, 25/10/2023	Qui, 26/10/2023
13	Produção do modelo de dados persistente	8 dias	Sex, 27/10/2023	Sab, 04/11/2023
14	Descrição dos casos de utilização	4 dias	Dom, 05/11/2023	Qua, 08/11/2023
15	Identificação de todas as classes/interfaces	4 dias	Qui, 09/11/2023	Dom, 12/11/2023
16	Produção do modelo estrutural	5 dias	Seg, 13/11/2023	Sex, 17/11/2023
17	Entregas das Fases I e II	15 minutos	Sex, 17/11/2023	Sex, 17/11/2023
18	Fase III (Implementação)	63 dias	Sab, 18/11/2023	Sex, 19/01/2024
19	Correção de erros da Fase II	12 dias	Sab, 18/11/2023	Qua, 29/11/2023
20	Implementação da base de dados	5 dias	Qui, 30/11/2023	Seg, 04/12/2023
21	Desenvolvimento da aplicação	37 dias	Ter, 05/12/2023	Qua, 10/01/2024
22	Testes	3 dias	Qui, 11/01/2024	Sab, 13/01/2024
23	Análise de resultados e documentação	1 dia	Dom, 14/01/2024	Seg, 14/01/2024
24	Conclusão da escrita do relatório final	11 dias	Ter, 09/01/2024	Sex, 19/01/2024
25	Reuniões e Atas	95 dias	Seg, 16/10/2023	Qui, 18/01/2024

FIGURA 5 - CALENDARIZAÇÃO EXECUTADA



3.5 Divisão de tarefas

Apresenta-se, em seguida, como as tarefas foram divididas pelos membros integrantes do grupo e como estas se sucederam ao longo do desenvolvimento do projeto.

#	Tarefa	Duração	Percentagem (%)					
1	Fase I (Planeamento)	11 dias						
2	Criação do repositório para o projeto	30 mins	0	0	0	100	0	0
3	Criação dos meios de comunicação para o grupo	10 mins	50	0	50	0	0	0
4	Estruturação do relatório	3 horas	15	40	15	15	15	15
5	Levantamento do estado da arte	3 dias	15	15	15	40	15	15
6	Identificação das funcionalidades do sistema	3 dias	15	40	15	15	15	15
7	Identificação dos utilizadores do sistema	1 dia	20	20	20	20	20	20
8	Identificação dos requisitos do sistema	3 dias	20	20	20	20	20	20
9	Fase II (Desenho do sistema)	26 dias						
10	Identificação dos casos de utilização	1 dia	20	20	20	20	20	20
11	Criação do diagrama de casos de utilização	1 dia	15	15	15	40	15	15
12	Protótipo de baixa fidelidade	2 dias	0	100	0	0	0	0
13	Produção do modelo de dados persistente	8 dias	15	15	40	15	15	15
14	Descrição dos casos de utilização	4 dias	15	15	15	15	40	40
15	Identificação de todas as classes/interfaces	4 dias	20	20	20	20	20	20
16	Produção do modelo estrutural	5 dias	15	15	40	15	15	15
18	Fase III (Implementação)	63 dias						
19	Correção de erros da Fase II	12 dias	20	20	20	20	20	20
20	Implementação da base de dados	5 dias	20	20	20	20	20	20
21	Desenvolvimento da aplicação	37 dias						
22	Dashboard Principal	5 dias	0	20	10	70	0	0
23	Dashboard Balcão	5 dias	0	0	45	45	10	0
24	Dashboard Gerente	3 dias	0	100	0	0	0	0
25	Dashboard Mesa	9 dias	15	55	15	15	0	0
26	Dashboard GerirCargo	3 dias	0	0	20	20	60	0
27	Dashboard GerirFuncionario	6 dias	0	20	20	60	0	0
28	Dashboard GerirZonas	3 dias	0	90	10	0	0	0
29	Dashboard GerirMesas	3 dias	0	90	10	0	0	0
30	Dashboard GerirCategorias	3 dias	0	0	10	90	0	0
31	Dashboard GerirItem	3 dias	0	65	0	35	0	0
32	Dashboard Pedido	32 dias	60	0	30	10	0	0
33	Dashboard FecharConta	8 dias	80	0	10	10	0	0
34	Dashboard ReservarMesa	7 dias	0	45	20	35	0	0
35	Dashboard GerirRelatorios	9 dias	0	90	10	0	0	0
36	Conexão e Gestão da Base de Dados	4 dias	20	20	20	20	20	20
37	Design das Interfaces	5 dias	20	20	20	20	20	20
38	Tratamento de erros	4 dias	20	20	20	20	20	20
39	Sessão do Utilizador	3 dias	0	0	100	0	0	0
40	Testes	3 dias	0	0	50	50	0	0
41	Análise de resultados e documentação	1 dia	40	20	0	40	0	0
42	Conclusão da escrita do relatório final	11 dias	20	20	20	20	20	20
			Bruno Migueis	Daniel Silva	Gabriel Cravo	Lucas Duarte	Miguel Pirré	

FIGURA 6 - DIVISÃO DAS TAREFAS

4. Modelo de requisitos

No ponto que se segue são descritas as funcionalidades do sistema, isto é, especifica-se o que o sistema deve fazer, obrigatoriamente, e como o sistema se deve comportar, tendo em conta, por exemplo, a usabilidade e desempenho. Os subtópicos seguintes descrevem então os requisitos funcionais e não funcionais do sistema.

4.1 Requisitos funcionais

Os requisitos funcionais são definidos como as funcionalidades ou atividades que um sistema/software deve realizar. Representa o que o software faz, em termos de tarefas e serviços, através de cálculos, detalhes técnicos, manipulação de dados e de processamento e outras funcionalidades específicas que definem o que um sistema, idealmente, será capaz de realizar.

Em seguida está apresentada uma tabela (**Tabela 1**) que contém os requisitos funcionais alusivos ao desenvolvimento do sistema.

Funcionalidade do Utilizador: Indica que o requisito está relacionado a uma ação ou interação específica do utilizador.

Funcionalidade do Sistema: Refere-se a operações internas do sistema que não são diretamente perceptíveis pelo utilizador, mas que são necessárias para o funcionamento adequado da aplicação.

Integração: Diz respeito a requisitos que envolvem a interação com outros sistemas, módulos ou componentes.

Refª	Requisito funcional	Tipo
RF.1	Sistema de seleção de perfis	Funcionalidade do Utilizador
RF.2	Gestão de perfil	Funcionalidade do Utilizador
RF.3	Criar pedido	Funcionalidade do Utilizador
RF.4	Remover pedido	Funcionalidade do Utilizador

RF.5	Criar mesa	Funcionalidade do Utilizador
RF.6	Editar itens mesa	Funcionalidade do Utilizador
RF.7	Remover mesa	Funcionalidade do Utilizador
RF.8	Editar menu	Funcionalidade do Utilizador
RF.9	Calcular total da despesa mesa	Funcionalidade do Sistema
RF.10	Dividir a despesa da mesa	Funcionalidade do Sistema
RF.11	Aceitar diferentes métodos de pagamento	Funcionalidade do Sistema
RF.12	Imprimir recibo	Integração
RF.13	Imprimir a consulta de mesa	Funcionalidade do Sistema
RF.14	Tempo dos pedidos no balcão	Funcionalidade do Sistema
RF.15	Enviar pedido de comida para o balcão	Funcionalidade do Sistema
RF.16	Visualizar status mesa	Funcionalidade do Sistema
RF.17	Pesquisar item menu	Funcionalidade do Sistema
RF.18	<i>Checklist</i> itens menu	Funcionalidade do Sistema
RF.19	Consultar/emittir relatório de vendas	Funcionalidade do Sistema / Integração
RF.20	Receber pedido balcão	Funcionalidade do Utilizador
RF.21	Atualizar status pedido	Funcionalidade do Utilizador
RF.22	Criar itens	Funcionalidade do Utilizador
RF.23	Editar itens	Funcionalidade do Utilizador
RF.24	Apagar itens	Funcionalidade do Utilizador
RF.25	Gerir categorias	Funcionalidade do Utilizador
RF.26	Gerir zonas	Funcionalidade do Utilizador
RF.27	Gerir cargo	Funcionalidade do Utilizador

TABELA 1 - REQUISITOS FUNCIONAIS

4.2 Restrições e requisitos não funcionais

Os requisitos não funcionais devem conter elementos específicos, tais como a descrição da tarefa a ser executada pelo software, a origem do requisito e o seu utilizador, a relação da troca de informação entre o software e o utilizador e, se existirem, algumas restrições lógicas associadas à tarefa. Dentro dos requisitos não funcionais estão incluídos os requisitos de interface e de facilidade de uso, de desempenho, de segurança e integridade de dados, de interface com sistemas externos e ambientes de execução, entre outros.

Os requisitos não funcionais estão relacionados com os requisitos funcionais e indicam como o sistema/software deve ser feito e como deve funcionar, ou seja, são os critérios que qualificam os requisitos funcionais.

Foi-se elaborada uma tabela para as restrições (**Tabela 2**) e outra para os requisitos não funcionais (**Tabela 3**).

Restrição	Descrição
Ambiente de Implantação	O sistema deve ser implantado em servidores que atendam aos padrões de segurança exigidos para a gestão de vendas de um restaurante
Segurança de Dados	O sistema deve proteger os dados, especialmente em relação às informações de clientes e transações financeiras
Compatibilidade	O sistema deve ser compatível com os dispositivos utilizados no restaurante, como terminais de ponto de venda (PDV)
Idioma	O sistema deve suportar o idioma português, pois o foco são os restaurantes portugueses
Disponibilidade	O sistema deve estar disponível 24/7, com tempo de inatividade planeado mínimo para manutenção

TABELA 2 - RESTRIÇÕES

Ref ^a	Requisito não funcional	Descrição
RNF.1	Desempenho	O sistema deve ser capaz de lidar com um grande número de pedidos simultâneos durante as horas de pico de pedidos
RNF.2	Escalabilidade	O sistema deve ser escalável para acomodar um aumento na procura durante eventos especiais ou épocas movimentadas
RNF.3	Usabilidade	A interface do utilizador deve ser intuitiva e projetada para funcionários de restaurante, permitindo que estes a utilizem facilmente, mesmo em momentos de movimento intenso
RNF.4	Segurança	Medidas de segurança devem proteger os dados do sistema contra acesso não autorizado e garantir a integridade das transações financeiras
RNF.5	Backup e Recuperação	Deve existir um plano de <i>backup</i> eficaz e um processo de recuperação de dados em caso de falhas
RNF.6	Integração	O sistema deve ser capaz de se integrar com sistemas de pagamento para aceitar diferentes métodos de pagamento
RNF.7	Manutenção e Suporte	Um plano de manutenção regular e suporte técnico deve estar disponível para garantir a estabilidade contínua do sistema
RNF.8	Desempenho de Impressão	O sistema de impressão de recibos e pedidos deve ser rápido e confiável para garantir a eficiência no atendimento ao cliente
RNF.9	Tempo de Resposta	O sistema deve fornecer tempos de resposta rápidos, garantindo um atendimento eficiente aos clientes

TABELA 3 - REQUISITOS NÃO FUNCIONAIS

4.3 Requisitos de interface e facilidade de uso

Os requisitos de interface e facilidade de uso correspondem às expectativas e especificações, desenhados para assegurar que um produto, serviço, processo ou ambiente, seja fácil de utilizar.

Em seguida está apresentada uma tabela (**Tabela 4**) que contém os requisitos de interface e usabilidade, e respetivos requisitos funcionais, alusivos ao desenvolvimento do sistema.

Ref ^a	Requisito de interface e usabilidade	Req. funcionais relacionados
RInt.1	Aplicação simples e intuitiva	
RInt.2	Implementar um design intuitivo para a interface do utilizador	Ref ^a : RF.1 - Sistema de Seleção de Perfis Ref ^a : RF.16 - Visualizar status mesa Ref ^a : RF.17 - Pesquisar itens menu Ref ^a : RF.19 - Consultar/Emitir relatório de vendas Ref ^a : RF.25 - Gerir categorias Ref ^a : RF.26 - Gerir zonas Ref ^a : RF.26 - Gerir cargo
RInt.3	Garantir a consistência do design	Ref ^a : RF.1 - Sistema de Seleção de Perfis
RInt.4	Dispor as informações do menu de forma lógica e simples	Ref ^a : RF.3 - Criar Pedido Ref ^a : RF.8 - Editar Menu
RInt.5	Facilitar a movimentação dentro da aplicação	

TABELA 4 - REQUISITOS DE INTERFACE E FACILIDADE DE USO

4.4 Requisitos de desempenho

Os requisitos de desempenho são requisitos criados à volta das necessidades da infraestrutura para garantir que funcione sem lentidão ou outros problemas.

Ref ^a	Requisito de desempenho	Req. funcionais relacionados
RDes.1	Aplicação funcional só com acesso local à internet	
RDes.2	Ligaçāo com a impressora	Ref ^a : RF.19 – Consultar/Emitir relatório de vendas

TABELA 5 - REQUISITOS DE DESEMPENHOS

4.5 Requisitos de segurança e integridade dos dados

Os requisitos de segurança e integridade de dados são requisitos que preservam todos os dados, seja por meio de criptografia, hashing certificados digitais, autenticações e controlo de acesso.

Ref ^a	Requisito de segurança, privacidade e integridade de dados	Req. funcionais relacionados
RSeg.1	Sistema de Backup	
RSeg.2	Validação de dados dos perfis	Ref ^a : RF.1 - Sistema de Seleção de
RSeg.3	O sistema não deve permitir que o funcionário insira uma nova lista sem estar autenticado.	

TABELA 6 - REQUISITOS DE SEGURANÇA E INTEGRIDADE DOS DADOS

4.6 Requisitos de interface com sistemas externos e ambientes de execução

Os requisitos de interface com sistemas externos e com ambientes de execução podem ser colocados quer no produto, quer no processo, e são derivados do ambiente onde o sistema está a ser desenvolvido, baseando-se em informação do domínio de aplicação, na necessidade do sistema em interagir com outros sistemas, etc.

Em seguida está apresentada uma tabela que contém os requisitos de interface com sistemas externos e com ambientes de execução, e respetivos requisitos funcionais alusivos ao desenvolvimento do sistema.

Requisito	Descrição
R.I.1 – Tags RFID	É necessária conexão com um arduíno que possui um leitor de tags RFID
R.I.2 – Impressora Térmica	É necessária conexão com uma impressora térmica para realizar a impressão de recibos.

TABELA 7 - REQUISITOS DE INTERFACE COM SISTEMAS EXTERNOS E AMBIENTES DE EXECUÇÃO

5. Normas e regulamentação específicas aplicáveis

Realizou-se uma pesquisa sobre todas as normas e regulamentações necessárias e aplicáveis, este passo é sempre importante para garantir a conformidade e qualidade da aplicação:

Faturação Eletrónica:

Norma: EN 16931: Esta norma estabelece o formato comum da fatura eletrónica na União Europeia (Norma Europeia EN).

Lei n.º 123/2019: Regula a utilização da fatura eletrónica, transpondo a Diretiva 2014/55/EU para a legislação portuguesa.

Proteção de Dados Pessoais:

Regulamento Geral de Proteção de Dados (RGPD): A Lei n.º 58/2019 assegura a aplicação do RGPD em Portugal. Os sistemas de gestão devem garantir a conformidade com a privacidade e a proteção de dados pessoais dos clientes.

Normas Fiscais e Contabilidade:

Código do Imposto sobre o Valor Acrescentado (CIVA): Estabelece as regras fiscais relacionadas com o IVA.

Código dos Regimes Contributivos do Sistema Previdencial de Segurança Social: Define as obrigações contributivas das empresas.

Normas para Software:

Regulamento (UE) 2016/679: Relativo à livre circulação de dados pessoais e à proteção da privacidade no setor das comunicações eletrónicas (Regulamento ePrivacy).

ISO/IEC 25010: Define critérios de qualidade para software, incluindo funcionalidade, desempenho, segurança e manutenibilidade.

Emissão de Documentos Eletrónicos:

Decreto-Lei n.º 28/2019: Regula a obrigação de utilização de programas informáticos de faturação e a emissão de documentos de transporte por via eletrónica.

5.1 Requisitos de hardware

Os requisitos de hardware necessários para conseguir executar a aplicação **JavaEats** incluem: um servidor (para hospedar a aplicação), estações de trabalho (computadores nos pontos de atendimento para funcionários), impressoras térmicas (para imprimir os recibos dos clientes, pedidos do balcão e relatórios de venda) e um router (para existir conexão à rede e ser provedor de uma rede local). Para além dos requisitos necessários também existem requisitos opcionais, tais como: *tags* RFID e leitor de *tags* RFID, que neste caso servem para iniciar sessão.

Computador: 2 unidades (gerente e funcionário).

Componente	Requisitos recomendados
Processador	Intel core i5 8 ^a geração ou superior
Memória (RAM)	8 GB
Armazenamento	SSD de 256 Gb
Placa Gráfica	Integrada
Conectividade	Placa de Rede Ethernet (porta Ethernet), Bluetooth e WiFi(opcional)
Sistema Operativo	Windows 10 ou superior
Portas e Conectores	Portas USB para periféricos

TABELA 8 - REQUISITOS DE HARDWARE COMPUTADOR

Computador (servidor): 1 unidade.

Componente	Requisitos recomendados
Processador	Intel core i5 de 8 ^a geração ou superior
Memória (RAM)	16 GB
Armazenamento	SSD de 500 Gb
Rede	Ethernet Gigabit (conexão de rede estável e rápida)

TABELA 9 - REQUISITOS DE HARDWARE SERVIDOR

6. Casos de Utilização

Os diagramas de casos de utilização tipicamente contêm uma lista de ações ou passos que definem as interações entre um ator e o sistema a fim de atingir um dado objetivo. Em relação à aplicação em questão, há um conjunto de atores que interagirão com esta, que neste caso, consistirá no gerente, empregado de mesa e balcão. Todos terão ao dispor funcionalidades que a plataforma fornece, tais como: criar/editar/remover funcionários, criar/remover zonas, criar/remover mesas, criar/editar/remover itens, entre outras, onde cada uma dessas ações se podem, tecnicamente, transformar em casos de utilização. Os subtópicos seguintes descrevem os atores e os casos de utilização que integram o sistema, assim como o diagrama de casos de uso. Detalham-se os casos de utilização referindo o seguinte: atores, requisitos funcionais e grau de prioridade associados; pré-condições (o estado do sistema antes do caso de uso), pós-condições (o estado do sistema depois de concluído o caso de uso), fluxo básico (série de afirmações declarativas que listam os passos do caso de uso), fluxo alternativo (alternativas ao caminho básico) e ainda a sua finalidade (descrição sumária)

6.1 Atores

O ator especifica um papel executado por um utilizador ou outro sistema que interage com o sistema. O ator deve ser externo ao sistema e deve ter associações exclusivamente para casos de uso, componentes ou classes, podendo herdar o papel de outro. Na tabela abaixo apresentada (**Tabela 10**) é possível descrever os atores que interagem com o sistema.

Actor	Descrição
Funcionário	Pessoa capaz de receber e fazer pedidos no restaurante
Gerente	Pessoa capaz de fazer a gestão de tudo na aplicação, zonas, categorias, itens, cargos, perfis.
Empregado de balcão	Pessoa que recebe os pedidos que necessitam de ser preparados do funcionário. Também é a pessoa que os marca como prontos, indicando que podem seguir para a mesa.

TABELA 10 - ATORES DO SISTEMA

6.2 Casos de utilização

No seguinte diagrama evidenciam-se os casos de uso (contém uma lista de ações ou passos que definem as interações entre um ator e o sistema).

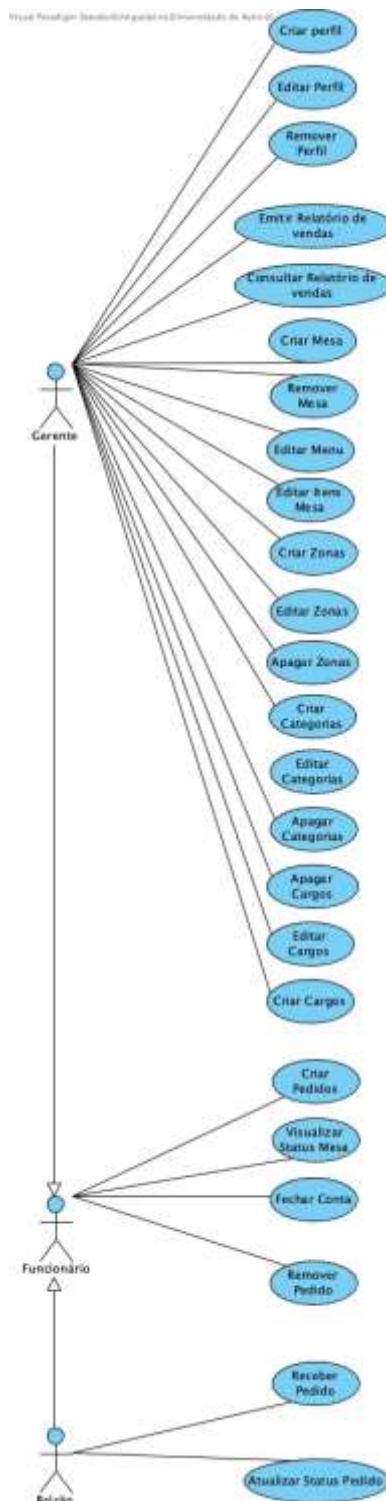


FIGURA 7 - DIAGRAMA DE CASOS DE UTILIZAÇÃO

6.2.1 Criar perfil

Nome:	Criar perfil
Atores:	Gerente (inicia)
Prioridade (1/3):	3
Finalidade:	Criação de perfis para os Funcionários
Requisitos funcionais:	R.F.1 – Sistema de seleção de perfis, R.F.2 - Gestão de perfil
Pré-condições:	O programa já vem com uma conta de Gerente inicializada.
Sumário:	O gerente faz a criação de perfis para os seus Funcionários
Sequência típica dos eventos	
Ações dos atores	
Respostas do sistema	
O ator Gerente cria um perfil.	Sistema guarda os perfis dos Funcionários com os devidos cargos, nome e senha.

TABELA 11 - CASO DE UTILIZAÇÃO CRIAR PERFIL

6.2.2 Editar perfil

Nome:	Editar perfil
Atores:	Gerente (inicia)
Prioridade (1/3):	2
Finalidade:	Editar perfis dos Funcionários
Requisitos funcionais:	R.F.2 - Gestão de perfil
Pré-condições:	O programa necessita de perfis criados.
Sumário:	O Gerente faz a edição de perfis para os seus Funcionários. Por exemplo, alterar o nome, senha ou cargo do Funcionário.
Sequência típica dos eventos	
Ações dos atores	
Respostas do sistema	
O ator Gerente edita um perfil.	Sistema guarda os perfis dos Funcionários com as devidas alterações.

TABELA 12 - CASO DE UTILIZAÇÃO EDITAR PERFIL

6.2.3 Remover perfil

Nome:	Remover perfil
Atores:	Gerente (inicia)
Prioridade (1/3):	3
Finalidade:	Remover perfis dos Funcionários
Requisitos funcionais:	R.F.2 - Gestão de perfil
Pré-condições:	O programa necessita de perfis criados.
Sumário:	O Gerente faz a remoção de perfis dos seus Funcionários.
Sequência típica dos eventos	
Ações dos atores Respostas do sistema	
O ator Gerente remove um perfil.	Sistema remove o perfil do Funcionário.

TABELA 13 - CASO DE UTILIZAÇÃO REMOVER PERFIL

6.2.4 Criar mesa

Nome:	Criar mesa
Atores:	Gerente (inicia)
Prioridade (1/3):	3
Finalidade:	Criação de mesas
Requisitos funcionais:	R.F.5- Criar mesa
Pré-condições:	Necessário estar no perfil de Gerente.
Sumário:	O Gerente adiciona uma ou mais nova(s) mesa(s) ao restaurante.
Sequência típica dos eventos	
Ações dos atores Respostas do sistema	
O ator Gerente cria uma ou mais mesas.	Sistema guarda a(s) mesa(s) criadas com o respetivo número.

TABELA 14 - CASO DE UTILIZAÇÃO CRIAR MESA

6.2.5 Remover mesa

Nome:	Remover mesa
Atores:	Gerente (inicia)
Prioridade (1/3):	3
Finalidade:	Remover mesas.
Requisitos funcionais:	R.F.7 - Remover mesa.
Pré-condições:	Necessário estar no perfil de Gerente.
Sumário:	O Gerente remove uma ou mais nova(s) mesa(s) ao restaurante.

Sequência típica dos eventos

Ações dos Atores	Respostas do sistema
O ator Gerente remove uma ou mais mesas.	Sistema remove a(s) mesa(s).

TABELA 15 - CASO DE UTILIZAÇÃO REMOVER MESA

6.2.6 Editar menu

Nome:	Editar menu
Atores:	Gerente (inicia)
Prioridade (1/3):	3
Finalidade:	Editar menu
Requisitos funcionais:	R.F.8 - Editar menu, R.F.22 – Criar itens, R.F.23 – Editar itens, R.F.24 – Apagar itens
Pré-condições:	O programa já vem com um menu, vazio, criado.
Sumário:	O menu vem criado vazio para que o Gerente o possa personalizar.

Sequência típica dos eventos

Ações dos Atores	Respostas do sistema
O ator Gerente edita o menu, as categorias e os produtos do menu.	Sistema guarda as alterações feitas pelo Gerente ao menu.

TABELA 16 - CASO DE UTILIZAÇÃO EDITAR MENU

6.2.7 Criar pedidos

Nome:	Criar pedidos
Atores:	Funcionário (inicia)
Prioridade (1/3):	3
Finalidade:	Criar os pedidos.
Requisitos funcionais:	R.F.3 - Criar Pedido, R.F.17 - Enviar pedido para o balcão, R.F.17 - Pesquisar item menu, R.F.18 - Checklist itens menu
Pré-condições:	O programa necessita de pelo menos uma mesa criada.
Sumário:	O Funcionário faz a criação do(s) pedido(s) com os itens para a(s) mesa(s).
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Funcionário cria o(s) pedido(s) para as respetivas mesas.	Guarda na(s) mesa(s) o(s) itens do(s) pedido(s) que o Funcionário selecionou.

TABELA 17 - CASO DE UTILIZAÇÃO CRIAR PEDIDOS

6.2.8 Remover pedido

Nome:	Remover pedido
Atores:	Funcionário (inicia)
Prioridade (1/3):	3
Finalidade:	Remover os pedidos cancelados.
Requisitos funcionais:	R.F.4 - Remover pedido do balcão.
Pré-condições:	O programa necessita de pelo menos um pedido criado.
Sumário:	O empregado de balcão remove o(s) pedido(s) cancelado(s)
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Empregado de balcão remove o(s) pedido(s) que acabaram por ser cancelado(s).	Sistema remove do balcão o(s) pedido(s) que havia(m) sido criado(s) anteriormente.

TABELA 18 - CASO DE UTILIZAÇÃO REMOVER PEDIDO

6.2.9 Visualizar status da mesa

Nome:	Visualizar status da mesa
Atores:	Funcionário (inicia)
Prioridade (1/3):	3
Finalidade:	Visualizar status da mesa
Requisitos funcionais:	R.F.16 - Visualizar status da mesa
Pré-condições:	O programa necessita de pelo menos uma mesa criada.
Sumário:	O Funcionário visualiza o status da mesa, ou seja, este verifica se a mesa está ocupada (com pedidos), à espera de pagamento ou se a mesa está livre.
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Funcionário entra no sistema e verifica o status de uma determinada mesa.	Sistema mostra ao ator Funcionário uma cor diferente para cada estado da mesa.

TABELA 19 - CASO DE UTILIZAÇÃO VISUALIZAR STATUS DA MESA

6.2.10 Fechar conta

Nome:	Fechar conta
Atores:	Funcionário (inicia)
Prioridade (1/3):	3
Finalidade:	Fechar a conta de uma determinada mesa.
Requisitos funcionais:	R.F.9 - Calcular total da despesa mesa, R.F.10 - Dividir a despesa da mesa, R.F.11 – Aceitar diferentes métodos de pagamento, R.F.12 - Imprimir recibo, R.F.13 – Imprimir consulta de mesa
Pré-condições:	O programa necessita de pelo menos um pedido criado.
Sumário:	Calcular o total da conta na mesa, dividir esse valor caso solicitado, fechar a conta da mesa passando o status da mesa para livre e imprime o recibo.
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Funcionário fecha a conta de uma determinada mesa.	Sistema calcula o total da conta da mesa, faz a divisão do valor, altera o status da mesa para livre e imprime o recibo.

TABELA 20 - CASO DE UTILIZAÇÃO FECHAR CONTA

6.2.11 Receber pedido

Nome:	Receber pedido
Atores:	Funcionário (inicia), Balcão
Prioridade (1/3):	3
Finalidade:	O ator Balcão recebe os itens do pedido que necessitam ser confeccionados.
Requisitos funcionais:	R.F.20 - Receber pedido balcão, R.F.14 - Tempo dos pedidos balcão
Pré-condições:	Existir itens no pedido que necessitam ser confeccionados.
Sumário:	O Balcão recebe do Funcionário os itens do pedido para preparar.
Sequência típica dos eventos	
Ações dos Atores	Respostas do sistema
O ator Funcionário cria um pedido no sistema. O ator Balcão recebe os itens do pedido que precisam de ser preparados.	O sistema faz a seleção e envia para o ator Cozinheiro os itens do pedido que são necessários confeccionar na cozinha.

TABELA 21 - CASO DE UTILIZAÇÃO RECEBER PEDIDO

6.2.12 Atualizar status do pedido

Nome:	Atualizar status do pedido
Atores:	Balcão (inicia), Funcionário
Prioridade (1/3):	3
Finalidade:	O ator Funcionário recebe do ator Balcão informação acerca do estado do pedido.
Requisitos funcionais:	R.F.21 - Atualizar status pedido
Pré-condições:	É necessário haver pelo menos um pedido para a cozinha.
Sumário:	O ator Funcionário recebe a atualização do status do pedido da cozinha.
Sequência típica dos eventos	
Ações dos Atores	Respostas do sistema
O ator Balcão atualiza o status do pedido que foi para a cozinha.	O sistema envia uma notificação do status ao ator Funcionário

TABELA 22 - CASO DE UTILIZAÇÃO ATUALIZAR STATUS DO PEDIDO

6.2.13 Editar itens mesa

Nome:	Editar itens mesa
Atores:	Funcionário (inicia)
Prioridade (1/3):	2
Finalidade:	Editar os itens da mesa caso engano ou erro.
Requisitos funcionais:	R.F.6 - Editar itens mesa
Pré-condições:	É necessário que haja um pedido na determinada mesa
Sumário:	O ator Funcionário pode fazer as alterações necessária para corrigir algum erro que possa acontecer.
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Funcionário edita os itens associados a determinada mesa.	O sistema permite que seja editado os itens associados a determinada mesa.

TABELA 23 - CASO DE UTILIZAÇÃO EDITAR ITENS DA MESA

6.2.14 Consultar relatório de vendas

Nome:	Consultar relatório de vendas
Atores:	Gerente (inicia)
Prioridade (1/3):	2
Finalidade:	Permitir que seja analisado as vendas realizadas ao longo do dia.
Requisitos funcionais:	R.F.19 - Consultar/Emitir relatório de vendas
Pré-condições:	Iniciar sessão como Gerente e também que existam vendas.
Sumário:	Permite que o Gerente obtenha uma análise detalhado sobre as vendas realizadas no dia e as emita, sendo assim possível que o Gerente identifique tendências e tome decisões.
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Gerente solicita ao sistema o relatório de vendas do dia.	O Sistema mostra ao Gerente uma análise detalhada sobre as vendas do dia, como o total de vendas, itens mais vendidos, médias etc.

TABELA 24 - CASO DE UTILIZAÇÃO CONSULTAR RELATÓRIO DE VENDAS

6.2.15 Emitir relatório de vendas

Nome:	Emitir relatório de vendas
Atores:	Gerente (inicia)
Prioridade (1/3):	2
Finalidade:	Permitir que seja impressa a analise das vendas realizadas no dia
Requisitos funcionais:	R.F.19 - Consultar/Emitir relatório de vendas
Pré-condições:	Iniciar sessão como Gerente e também que existam vendas.
Sumário:	Permite que o Gerente obtenha impressa uma análise detalhada sobre as vendas do dia (para fins estatísticos no restaurante).
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Gerente solicita ao sistema uma impressão do relatório de vendas do dia.	O Sistema envia o pedido do Gerente para a impressora.

TABELA 25 - CASO DE UTILIZAÇÃO EMITIR RELATÓRIO DE VENDAS

6.2.16 Criar categorias

Nome:	Criar categorias
Atores:	Gerente (inicia)
Prioridade (1/3):	3
Finalidade:	Permitir que sejam criadas categorias de itens.
Requisitos funcionais:	R.F.25 - Gerir categorias
Pré-condições:	É necessário que se esteja na janela correta para fazer a devida criação das categorias
Sumário:	Permite que o Gerente crie variadas categorias para os seus itens, garantindo assim uma melhor organização dos produtos disponíveis.
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Gerente cria uma categoria.	O Sistema atualiza na base de dados a lista de categorias, e permite que quando um item seja criado essa categoria seja selecionada.

TABELA 26 - CASO DE UTILIZAÇÃO CRIAR CATEGORIAS



6.2.17 Editar categorias

Nome:	Editar categorias
Atores:	Gerente (inicia)
Prioridade (1/3):	2
Finalidade:	Permitir que sejam editadas as categorias dos itens.
Requisitos funcionais:	R.F.25 - Gerir categorias
Pré-condições:	É necessário que haja categorias para que possam ser editadas.
Sumário:	Permite que o Gerente edite as categorias caso tenha acontecido algum erro na sua criação ou caso.
Sequência típica dos eventos	
Ações dos Atores	Respostas do sistema
O ator Gerente realiza a alteração numa das categorias.	O Sistema realiza as alterações necessárias na base de dados.
Sequências alternativas	
Uma sequência alternativa seria apagar a categoria e voltar a criar.	

TABELA 27 - CASO DE UTILIZAÇÃO EDITAR CATEGORIAS

6.2.18 Apagar categorias

Nome:	Apagar categoria
Atores:	Gerente (inicia)
Prioridade (1/3):	3
Finalidade:	Permitir apagar quaisquer categorias que não são desejadas.
Requisitos funcionais:	R.F.25 - Gerir categorias
Pré-condições:	É necessário que haja pelo menos uma categoria criada.
Sumário:	Permitir ao Gerente apagar categorias que já não sejam necessárias.
Sequência típica dos eventos	
Ações dos Atores	Respostas do sistema
O ator Gerente escolhe uma categoria e realiza a sua eliminação.	O Sistema recebe as instruções e procede a alterar na base de dados as categorias, retirando a que foi eliminada.

TABELA 28 - CASO DE UTILIZAÇÃO APAGAR CATEGORIA

6.2.19 Criar zonas

Nome:	Criar zonas
Atores:	Gerente (inicia)
Prioridade (1/3):	2
Finalidade:	Permitir criar as zonas onde as mesas se encontram posicionadas.
Requisitos funcionais:	R.F.26 - Gerir zonas
Pré-condições:	É necessário que esteja na interface correta.
Sumário:	Permite que o Gerente realize a criação das zonas onde as mesas se vão encontrar, permite uma melhor organização da sala.
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Gerente cria uma zona com o nome que deseja.	O Sistema recebe as instruções e procede a criar a zona na base de dados.

TABELA 29 - CASO DE UTILIZAÇÃO CRIAR ZONAS

6.2.20 Editar zonas

Nome:	Editar zonas
Atores:	Gerente (inicia)
Prioridade (1/3):	1
Finalidade:	Permitir que seja alterado qualquer característica de uma zona.
Requisitos funcionais:	R.F.26 - Gerir zonas
Pré-condições:	É necessário que haja pelo menos uma zona criada.
Sumário:	Permite que o Gerente realize alterações nas características de certo item, quem seja alterar o nome, o preço ou mesmo a sua categoria.
Sequência típica dos eventos	
Ações dos Atores Respostas do sistema	
O ator Gerente escolhe uma zona e realiza a alteração do seu nome.	O Sistema recebe as instruções e procede a alterar na base de dados o nome da zona que o ator Gerente escolheu editar.
Sequências alternativas	
Em alternativa a editar a categoria é apagar e voltar a criar com o nome desejado.	

TABELA 30 - CASO DE UTILIZAÇÃO EDITAR ZONAS

6.2.21 Apagar zonas

Nome:	Apagar zonas
Atores:	Gerente (inicia)
Prioridade (1/3):	2
Finalidade:	Permitir que seja apagada uma determinada zona.
Requisitos funcionais:	R.F.26 - Gerir zonas
Pré-condições:	É necessário que haja pelo menos um item criado.
Sumário:	Permite que o Gerente apague uma zona onde por exemplo já não haverá mesas no seu restaurante.
Sequência típica dos eventos	
Ações dos Atores	Respostas do sistema
O ator Gerente escolhe uma zona e apaga-a.	O Sistema recebe as instruções e procede a remover da base de dados a zona impedindo que sejam criadas mais mesas nesse local.

TABELA 31 - CASO DE UTILIZAÇÃO APAGAR ZONAS

6.2.22 Criar Cargos

Nome:	Criar cargo
Atores:	Gerente (inicia)
Prioridade (1/3):	3
Finalidade:	Permitir que seja criado um determinado cargo.
Requisitos funcionais:	R.F.27 – Gerir cargo
Pré-condições:	É necessário estar na <i>dashboard</i> correta.
Sumário:	Permite que o Gerente crie cargos novos quando forem necessários para o sistema do restaurante.
Sequência típica dos eventos	
Ações dos atores	Respostas do sistema
O ator Gerente cria um cargo novo.	O Sistema recebe as instruções e procede a adicionar na base de dados as informações do cargo novo.

TABELA 32 - CASO DE UTILIZAÇÃO CRIAR CARGO

6.2.23 Editar Cargos

Nome:	Editar cargo
Atores:	Gerente (inicia)
Prioridade (1/3):	2
Finalidade:	Permitir editar um determinado cargo que já esteja criado.
Requisitos funcionais:	R.F.27 – Gerir cargo
Pré-condições:	É necessário estar na <i>dashboard</i> correta.
Sumário:	Permitir ao Gerente editar cargos existentes
Sequência típica dos eventos	
Ações dos atores	
O ator Gerente altera o nome de um cargo já existente.	O Sistema recebe as instruções e procede a alterar na base de dados as informações do cargo novo.
Sequências alternativas	
Uma sequência alternativa seria apagar o cargo e criar um novo.	

TABELA 33 - CASO DE UTILIZAÇÃO EDITAR CARGO

6.2.24 Apagar Cargos

Nome:	Apagar cargo
Atores:	Gerente (inicia)
Prioridade (1/3):	3
Finalidade:	Permitir apagar um determinado cargo já criado e que não seja necessário.
Requisitos funcionais:	R.F.27 – Gerir cargo
Pré-condições:	É necessário estar na <i>dashboard</i> correta.
Sumário:	Permite que o Gerente apague cargos existentes quando for necessário alterar um cargo já criado e já não seja necessário no restaurante.
Sequência típica dos eventos	
Ações dos atores	
O ator Gerente altera o nome de um cargo já existente.	O Sistema recebe as instruções e procede a alterar na base de dados as informações do cargo novo.

TABELA 34 - CASO DE UTILIZAÇÃO APAGAR CARGO

6.3 Cobertura de requisitos

Uma tabela de cobertura de requisitos permite-nos avaliar de que forma os requisitos funcionais de um sistema são cobertos pelos casos de uso, garantindo assim que todos os requisitos funcionais são considerados. Na tabela abaixo, esses mesmos requisitos são especificados e interligados com o caso de uso correspondente.

Caso de utilização	RF. 1	RF. 2	RF. 3	RF. 4	RF. 5	RF. 6	RF. 7	RF. 8	RF. 9	RF. 10	RF. 11	RF. 12	RF. 13	RF. 14	RF. 15	RF. 16	RF. 17	RF. 18	RF. 19	RF. 20	RF. 21	RF. 22	RF. 23	RF. 24	RF. 25	RF. 26	RF. 27		
Criar Perfil	X	X																											
Editar Perfil		X																											
Remover Perfil		X																											
Consultar relatório vendas																			X										
Emitir relatório vendas																			X										
Criar Mesa					X																								
Remover Mesa							X																						
Editar Itens Mesa						X																							
Editar Menu								X														X	X	X					
Criar pedidos			X													X		X	X										
Remover pedido				X																									
Visualizar status mesa																		X											
Fehar conta									X	X	X	X	X																
Receber pedido															X					X									
Atualizar status pedido																					X								
Criar Zonas																												X	
Editar Zonas																												X	
Apagar Zonas																												X	
Criar Cargo																													X
Editar Cargo																													X
Apagar Cargo																													X
Criar Categoria																												X	
Editar Categoria																												X	
Apagar Categoria																												X	

TABELA 35 - COBERTURA DE REQUISITOS

7. Prototipagem baixa-fidelidade

Pode-se definir um protótipo, neste contexto, como a expressão de uma intenção de design a fim de obter uma simulação da interação final entre o utilizador com a interface. Já a fidelidade, como o nível de detalhe e realismo do protótipo. Assim, um protótipo de baixa-fidelidade é uma forma fácil e rápida de traduzir conceitos complexos de desenho em algo tangível e passível de ser testado. Estes têm como principal objetivo verificar e testar a funcionalidade ao invés da aparência visual do produto mostrando os maiores pedaços de conteúdo, posicionamento dos elementos e alinhamento dos mesmos. Numa fase inicial do projeto, após o levantamento de funcionalidades e requisitos do sistema, chegou-se a um primeiro protótipo.

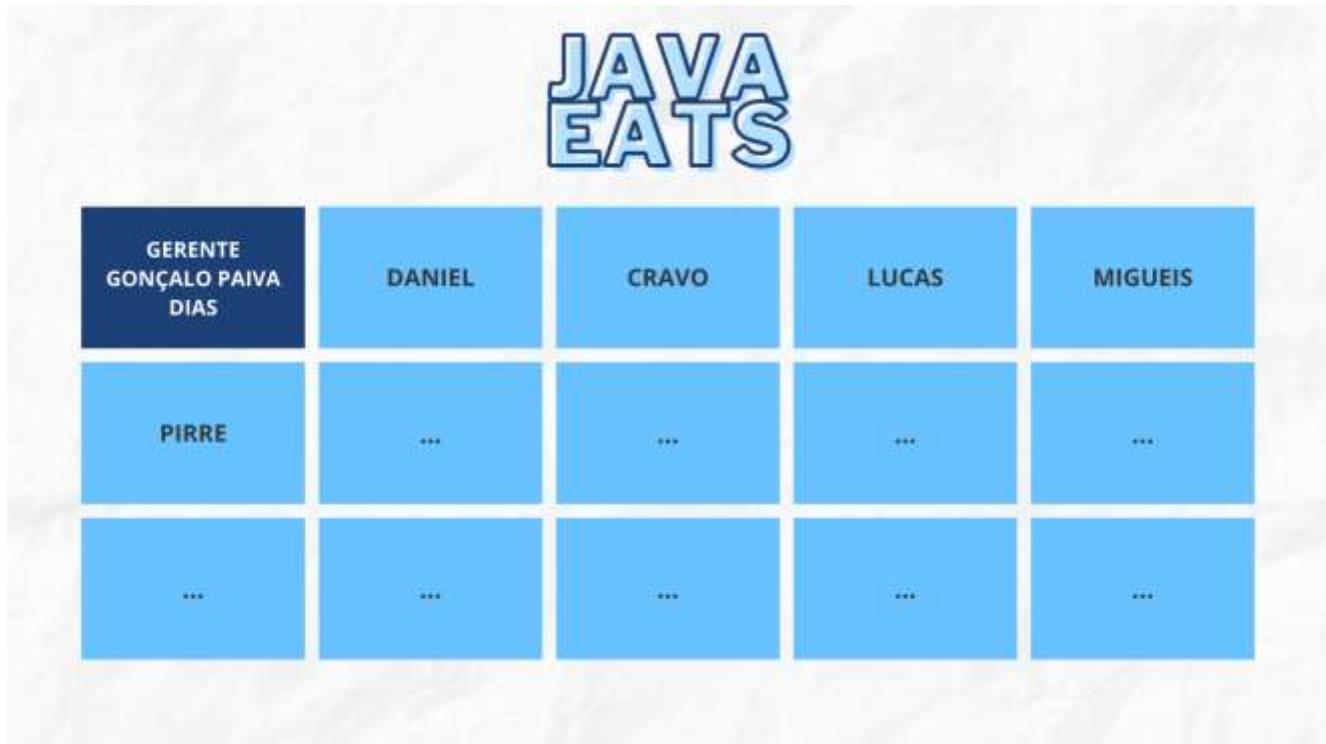


FIGURA 8 - DASHBOARD PRINCIPAL

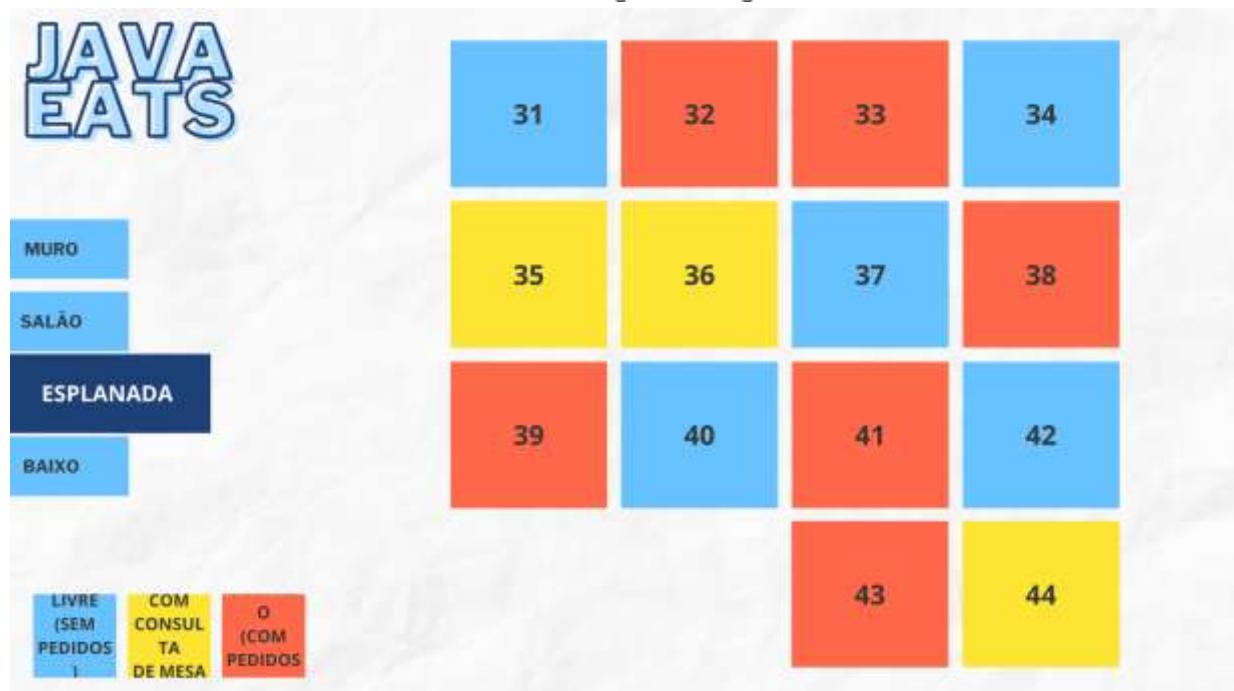


FIGURA 9 - DASHBOARD MESAS



FIGURA 10 - DASHBOARD PEDIDO

8. Análise de tecnologias

Considerando as tarefas definidas e a temática deste trabalho na qual consiste em desenvolver uma aplicação orientada a objetos, em torno do Java e com base de dados, fez-se a análise de um conjunto de ferramentas e soluções para tornar isto possível:

- **MySQL Workbench:** É uma ferramenta visual unificada que permite criar, editar e gerir bancos de dados MySQL. Este software ainda fornece modelação de dados, desenvolvimento SQL, ferramentas de administração, administração de utilizadores, *backups*, etc... Foi aplicada na unidade curricular de Sistemas de base de dados, lecionada pelo professor Gonçalo Dias, então tornou-se prático utilizar este software. <https://www.mysql.com/products/workbench/>

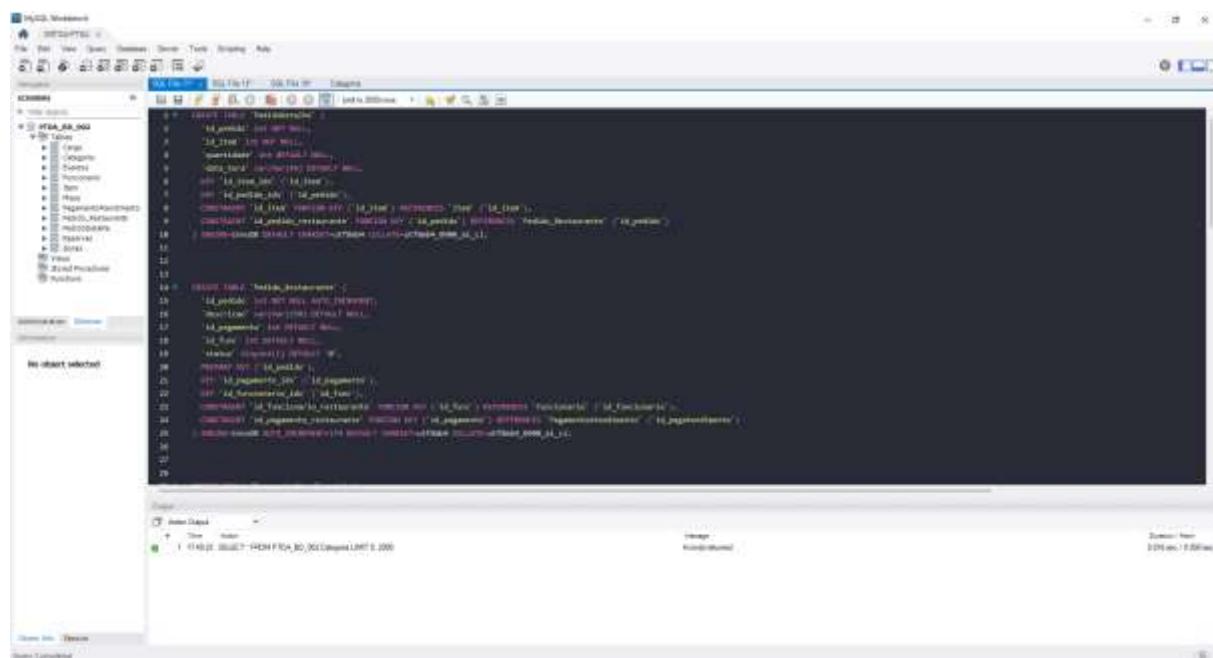


FIGURA 11 - MYSQL WORKBENCH

- **Git (repositório):** Software *open-source* e gratuito concebido para lidar com tudo desde projetos pequenos a muito grandes. Este é amplamente utilizado por desenvolvedores de software para gerir o código fonte dos seus projetos. <https://git-scm.com/>

- ♦ **Git Hub:** O Git Hub é uma plataforma de hospedagem de código fonte que permite aos utilizadores armazenarem e partilharem os seus repositórios Git online. Esta ferramenta fornece uma interface gráfica do utilizador para o Git e adiciona recurso de colaboração, como rastreamento de problemas, solicitações de pull e administração de projetos. Utiliza no nosso projeto, visto que o grupo já está familiarizado com ela devido a projetos anteriores.
<https://www.freecodecamp.org/news/introduction-to-git-and-github/>

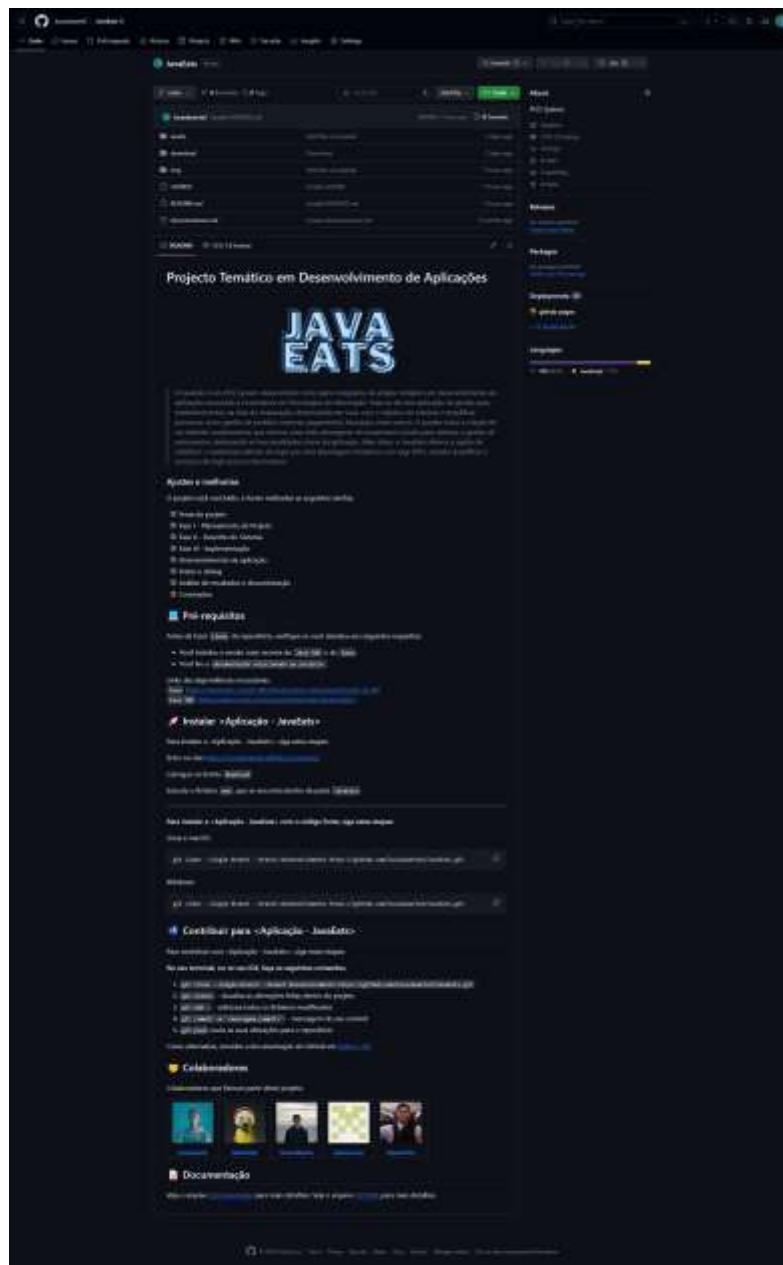


FIGURA 12 - GITHUB PÁGINA DO PROJETO JAVA EATS

- ◆ **IntelliJ:** É um ambiente de desenvolvimento integrado ou *integrated development environment* (IDE) em inglês, para Java e Kotlin. Foi usado no projeto para o desenvolvimento do projeto pois o grupo já está acostumado a trabalhar com ele e continha funcionalidades que facilitavam a utilização do Git. <https://www.jetbrains.com/idea/>

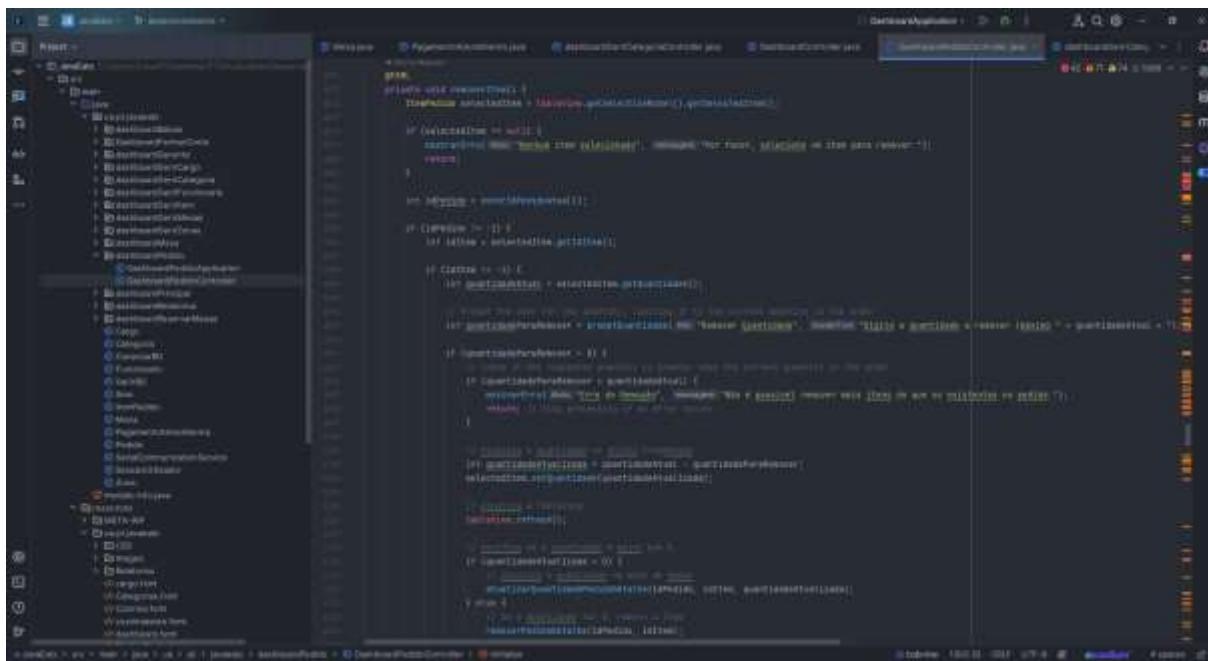


FIGURA 13 - INTELLIJ PROJETO JAVAEATS

- ◆ **JavaFX:** Biblioteca Java com o objetivo a criação de interfaces gráficas do utilizador em java. Esta fornece um conjunto de classes e interfaces em java e APIs que facilitam o desenvolvimento de aplicações. O grupo resolver utilizar esta biblioteca devido á sua fácil implementação e por ser uma biblioteca moderna. <https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784>
- ◆ **Apache PDFBox:** Biblioteca de código aberto em Java que fornece recursos para criar e manipular documentos PDF. Devido a ser uma das bibliotecas mais completa que tinham esta funcionalidade e a necessidade de criar PDFs no projeto. <https://pdfbox.apache.org/>

- **Scene Builder:** É uma ferramenta de apresentação visual que permite aos desenvolvedores conceber interfaces de aplicações JavaFX, sem programação. O resultado é um ficheiro FXML que pode ser depois combinado com o projeto, ao ligar a interface de utilizador com a lógica da aplicação. Pelo fácil manuseamento e aprendizagem o grupo utilizou esta ferramenta.

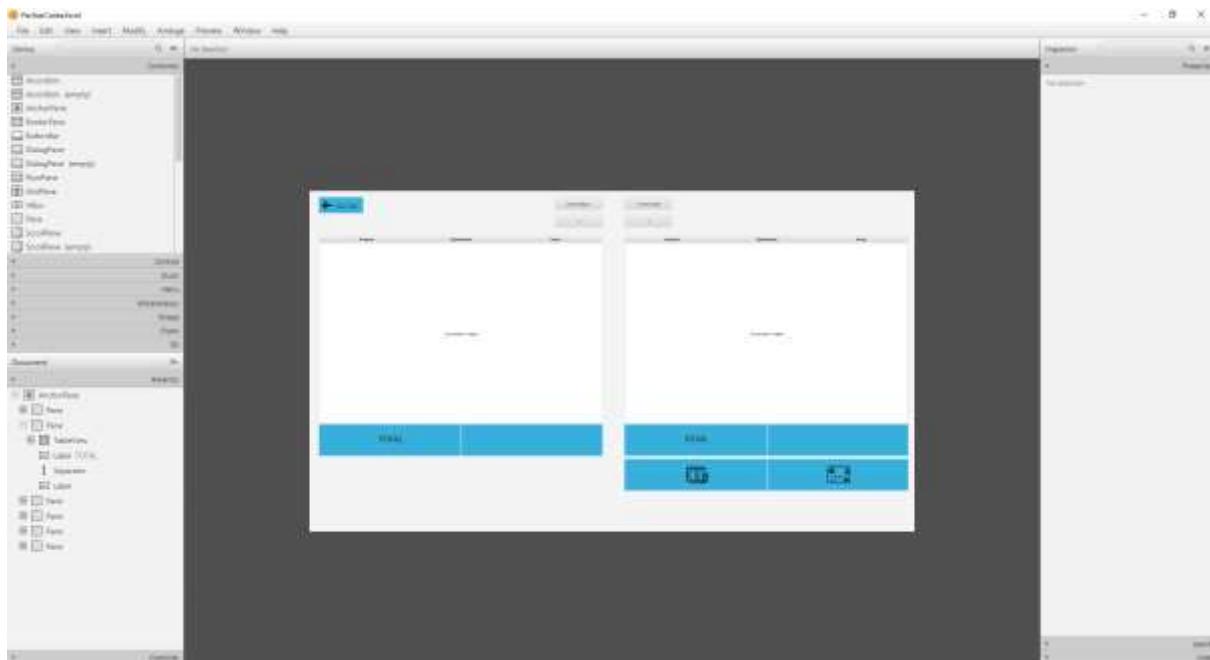


FIGURA 14 - SCENE BUILDER (DASHBOARD FECHAR CONTA)

- **Discord:** É uma plataforma de comunicação gratuita que permite a comunicação entre pessoas por meio de texto, voz e vídeo. Esta aplicação permite ainda a partilha de ficheiros e a criação de servidores personalizáveis. Por ser muito acessível e utilizada diariamente pelo grupo foi escolhida para organizar e resolver problemas sobre o projeto. <https://discord.com/>

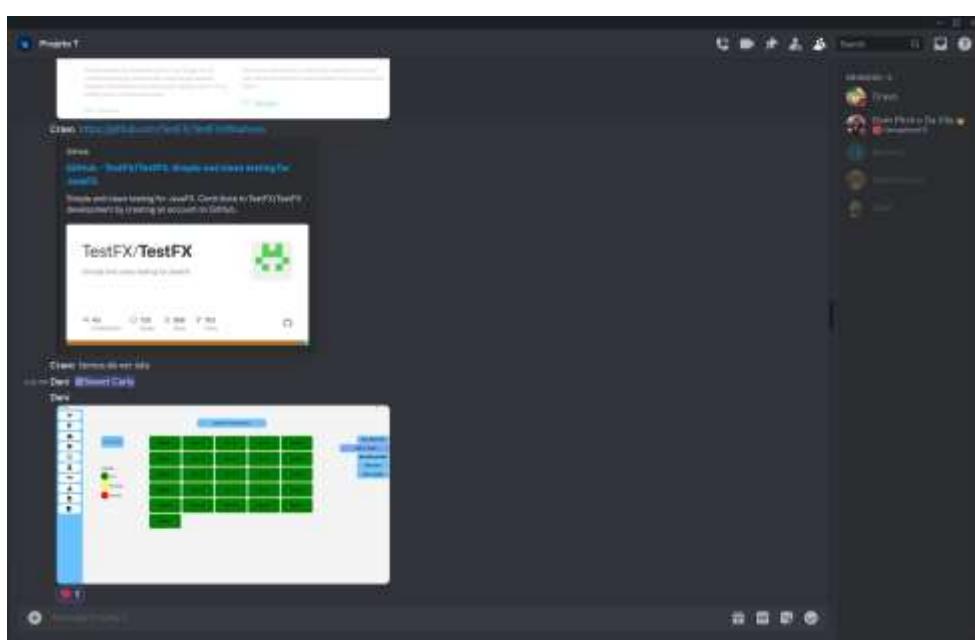


FIGURA 15 - DISCORD

- ◆ **Whatsapp:** Aplicação de comunicação gratuita que tem como principais características a troca de mensagens de texto de forma instantânea, partilha de multimédia e chamada de voz e vídeo. Por ser também muito utilizada por todos os membros do grupo foi utilizada para a organização do projeto. https://www.whatsapp.com/?lang=pt_pt
- ◆ **Zoom:** Plataforma de comunicação por vídeo que proporciona serviços de videoconferência, reuniões online, colaboração em grupo, etc. Esta foi muito útil para as reuniões semanais que tínhamos com o professor orientador Gonçalo Dias. <https://zoom.us/>

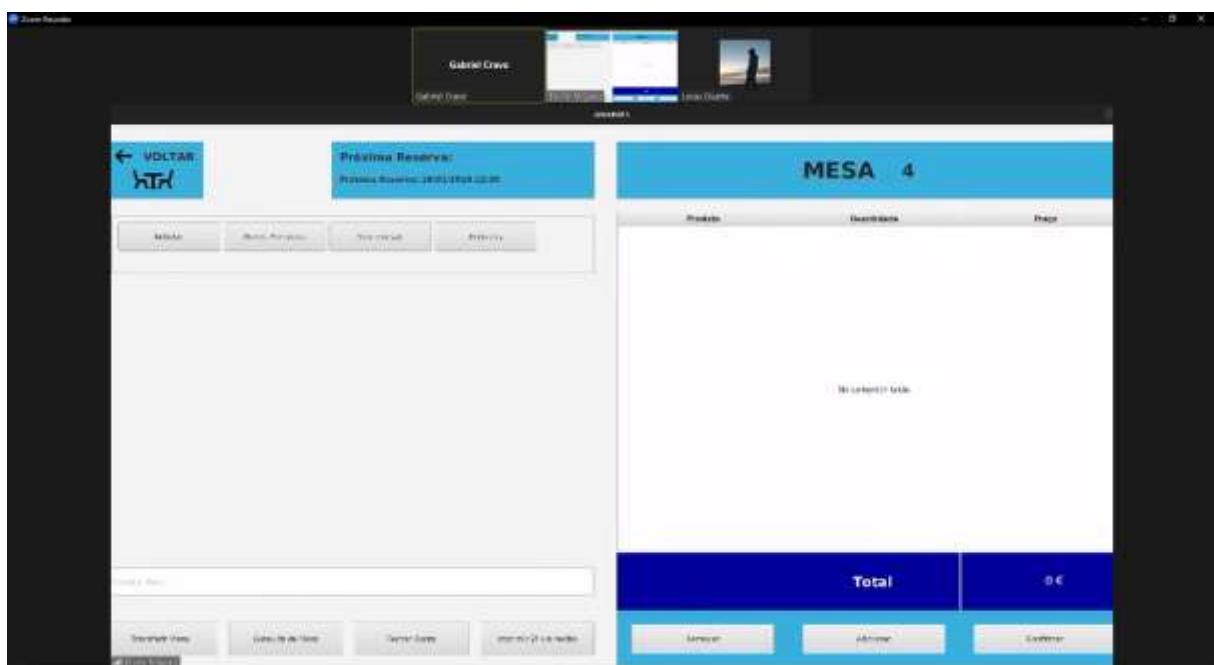


FIGURA 16 - ZOOM

9. Desenvolvimento da Aplicação

9.1 Diagrama de Classes

Um diagrama de classes serve para modelar o vocabulário de um sistema, do ponto de vista do utilizador/problema ou do implementador/solução.

Um diagrama de classes é constituído por itens que podem representar as classes que serão realmente programadas, assim como os principais objetos ou a interação entre classes e objetos do sistema, sendo eles classes, atributos, operações e associações.

Uma classe contém a especificação do objeto, as suas características, atributos e métodos.

Um atributo define as características da classe, como a sua visibilidade, nome, tipo de dados, multiplicidade, valor inicial e propriedade. Quanto à visibilidade, esta pode ser pública (representada pelo símbolo "+") ou privada (representada pelo símbolo "-"). O nome corresponde à identificação do atributo, o tipo de dados indica a especificação dos dados do atributo, o valor inicial e as suas propriedades dependem da linguagem usada na programação (Java) e a multiplicidade indica a possibilidade de fazer relações com outras classes.

Uma operação trata da função requerida a um objeto abstrato e contém características como nome, visibilidade e parâmetros.

Uma associação trata da capacidade das classes se relacionarem. Também pode conter nome, multiplicidade e tipo de navegação, que indica de onde partem as informações da classe e para onde irão.

Na figura abaixo está representado o diagrama de classes correspondente ao sistema em desenvolvimento.

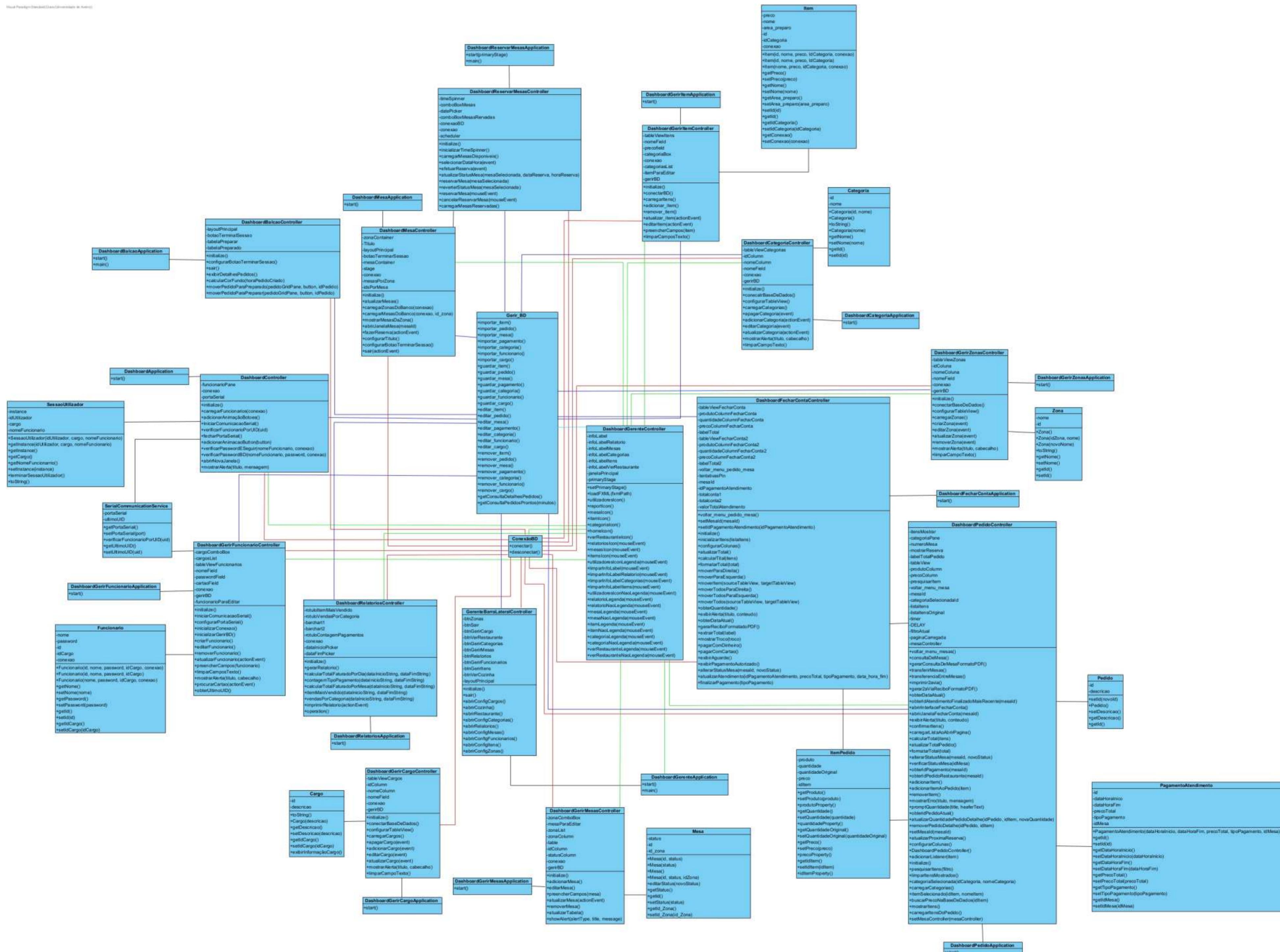


FIGURA 17 - DIAGRAMA DE CLASSES

9.2 Modelo de Dados Persistente

A necessidade de guardar dados de modo persistente fomentou a criação de um modelo de base de dados, seguido da sua implementação.

A base de dados modelada, como se observa nas figuras abaixo, foi criada através da interpretação da lista de requisitos. O modelo foi criado recorrendo à aplicação de modelação Visual Paradigm. Após a aprovação do modelo criado, que representaria as necessidades resultantes dos requisitos, implementou-se a base de dados no Sistema de Gestão de Bases de Dados MySQL Workbench. Na Base de dados ainda foi guardada uma *procedure* para a criação da mesma com o script que está no ANEXO D.

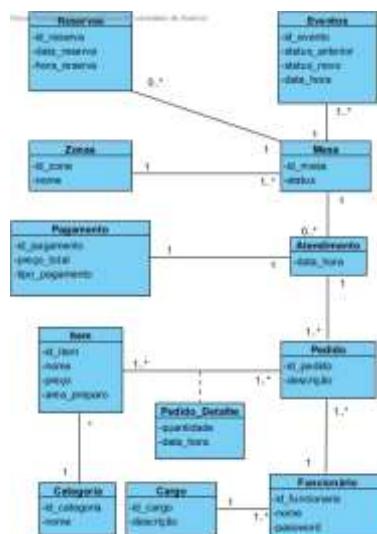


FIGURA 18 - MODELO DE BASE DE DADOS CONCEPCIONAL

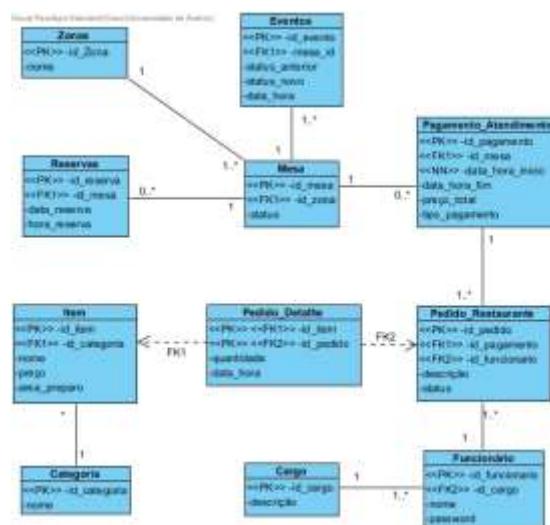


FIGURA 19 - MODELO DA BASE DE DADOS RELACIONAL

9.3 Website

Como forma de complementar a aplicação JavaEats criou-se um website (<https://lucasduarte2.github.io/JavaEats/>), elevando a experiência dos potenciais clientes a outro patamar.

Este ambiente virtual foi cuidadosamente concebido e surge como um portal dinâmico e informativo, proporcionando uma visão abrangente do ecossistema criado para otimizar a gestão de restaurantes. Além de destacar as funcionalidades chave da aplicação, oferece também uma experiência visual atraente, onde os clientes poderão explorar de forma intuitiva os recursos disponíveis. Ao apresentar os membros dedicados do projeto, o website não apenas humaniza a tecnologia, mas também destaca a competência por trás da aplicação, fortalecendo a confiança dos utilizadores e com o intuito de facilitar a experiência dos mesmos. Disponibiliza-se ainda, no website, o download da aplicação **JavaEats** que pode ser feito através do botão “Download”.

Em suma, o website enfatiza a eficácia da aplicação na gestão de restaurantes, proporcionando uma visão abrangente e acessível do que é o **JavaEats**.



FIGURA 20 - WEBSITE JAVAEATS

10. Interface

Uma interface de utilizador é o espaço onde a interação entre humanos e máquinas ocorre. Tem como objetivo a operação e controlo efetivos da máquina no lado do utilizador e o feedback da mesma, que auxilia o operador na tomada de decisões operacionais. Inclui componentes de hardware (físico) e software (lógico) e fornecem um meio de entrada, permitindo ao utilizador manipular o sistema, e de saída, permitindo ao sistema produzir os efeitos (respostas) das ações do utilizador.

10.1 Dashboard Principal

A *dashboard* principal é a porta de entrada para a experiência completa na aplicação JavaEats. Nesta interface existe um botão para cada funcionário criado, no qual o mesmo pode iniciar sessão

no seu perfil. Existem duas formas distintas para iniciar sessão, através da password ou através de uma *tag* NFC.



FIGURA 21 - DASHBOARD PRINCIPAL

10.2 Dashboard Gerente

Caso na *dashboard* principal seja iniciada sessão num perfil com o cargo Gerente, o mesmo irá ter acesso a uma interface na qual estão dispostas as mesas existentes para cada zona correspondente, um botão para fazer reserva de mesa e uma barra lateral posicionada do lado esquerdo onde o Gerente pode ver as mesas do restaurante, ver a cozinha, gerir o restaurante (cargos, categorias, funcionários, mesas, zonas e itens), gerar relatórios de venda com base em estatísticas da aplicação e por fim terminar sessão. Na figura seguinte realiza-se o login numa conta de Gerente.



FIGURA 22 - DASHBOARD PRINCIPAL LOGIN

Pode-se verificar na figura abaixo a aspeto da interface da “Home Page” de um Gerente (interface igual a um funcionário, mas possui a barra lateral à esquerda que permite executar funcionalidades de outro nível hierárquico).



FIGURA 23 - DASHBOARD GERENTE

10.3 Dashboard Gerente (Ver Balcão)

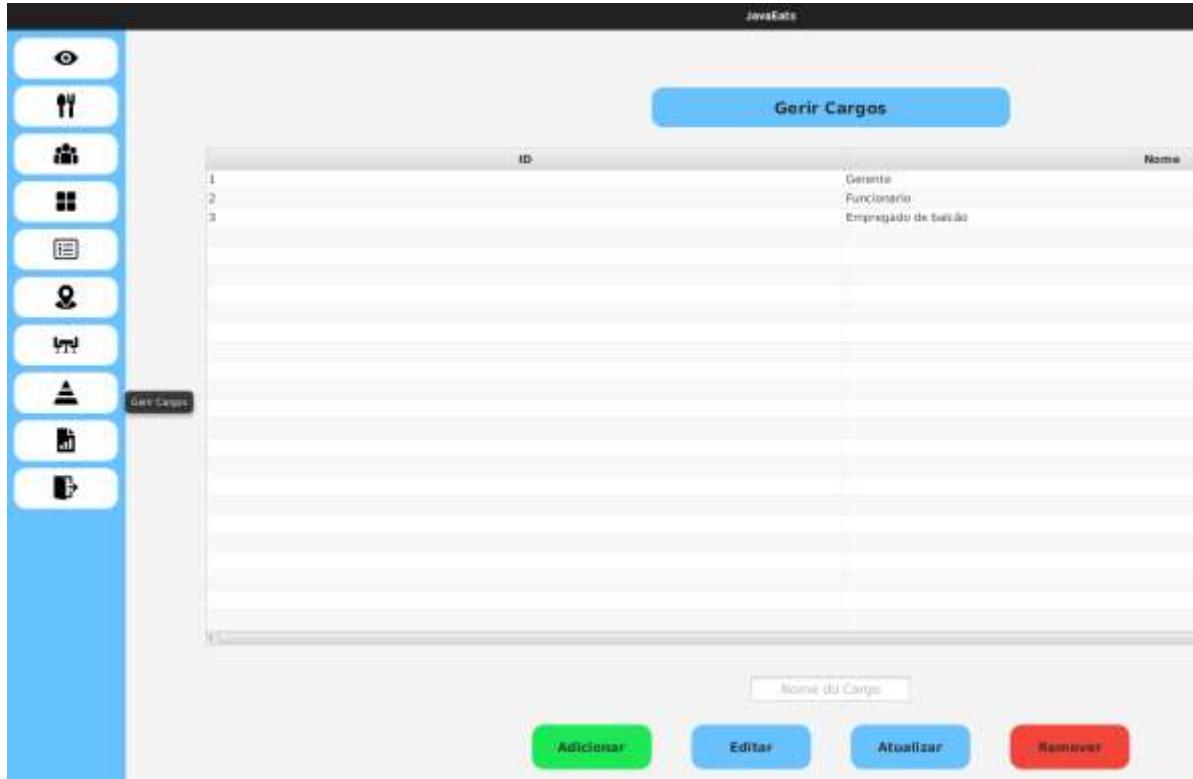
Na barra lateral, logo abaixo do símbolo do olho (ver restaurante/ver mesas) temos o símbolo de um garfo e uma colher. Ao carregar nesse botão (ver cozinha) é possível visualizar os pedidos feitos nas mesas e sinalizá-los como prontos. Esta secção de “Ver Cozinha” é a “Home Page” de um cozinheiro.



FIGURA 24 - DASHBOARD BALCÃO

10.4 Dashboard Gerente (Gerir Cargos)

Ao selecionar o botão “Gerir Cargos” aparecerá uma tabela com o ID e o nome do Cargo correspondente. Logo abaixo da tabela existem quatro botões em que é possível adicionar, editar, atualizar ou apagar um cargo.

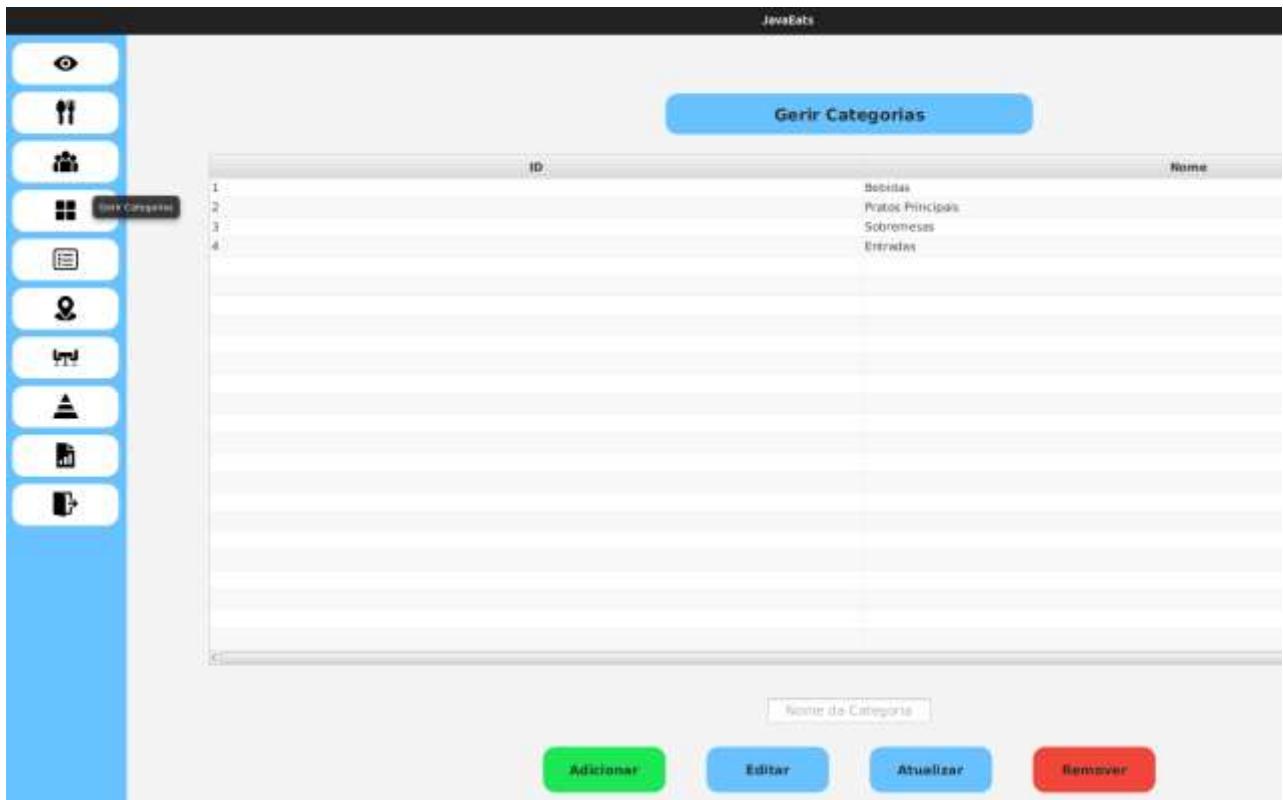


ID	Cargo	Nome
1	Gerente	Gerente
2	Funcionario	Funcionario
3	Empregado de barbearia	Empregado de barbearia

FIGURA 25 - DASHBOARD GERENTE CARGOS

10.5 Dashboard Gerente (Gerir Categorias)

Ao selecionar o botão “Gerir Categorias” aparecerá uma tabela com o ID e o nome da Categoria correspondente. Logo abaixo da tabela existem quatro botões em que é possível adicionar, editar, atualizar ou apagar uma categoria.



The screenshot shows a dashboard titled "Gerir Categorias". On the left, there is a vertical sidebar with various icons: eye, fork/knife, person, computer (highlighted), list, location, printer, triangle, chart, and a document. The main area contains a table with columns "ID" and "Nome". The data in the table is:

ID	Nome
1	Bebidas
2	Pratos Principais
3	Sobremeses
4	Entradas

Below the table, there is a text input field labeled "Nome da Categoria" and four buttons: "Adicionar" (green), "Editar" (blue), "Atualizar" (blue), and "Remover" (red).

FIGURA 26 - DASHBOARD GERENTE CATEGORIAS

10.6 Dashboard Gerente (Gerir Funcionários)

Ao selecionar o botão “Gerir Funcionários” aparecerá uma tabela com o ID, nome, password e cargo de cada funcionário. Logo abaixo da tabela existem quatro botões em que é possível adicionar, editar, atualizar ou remover um funcionário.



ID	Nome	Password	Cargo
1	Lucas	123	1
2	Davi	123	2
3	David	123	2
4	Fernando	111	3
12	Bruno	123	2
14	Daniel	123	3
119	Jônio	111	1
130	Marcos	222	2

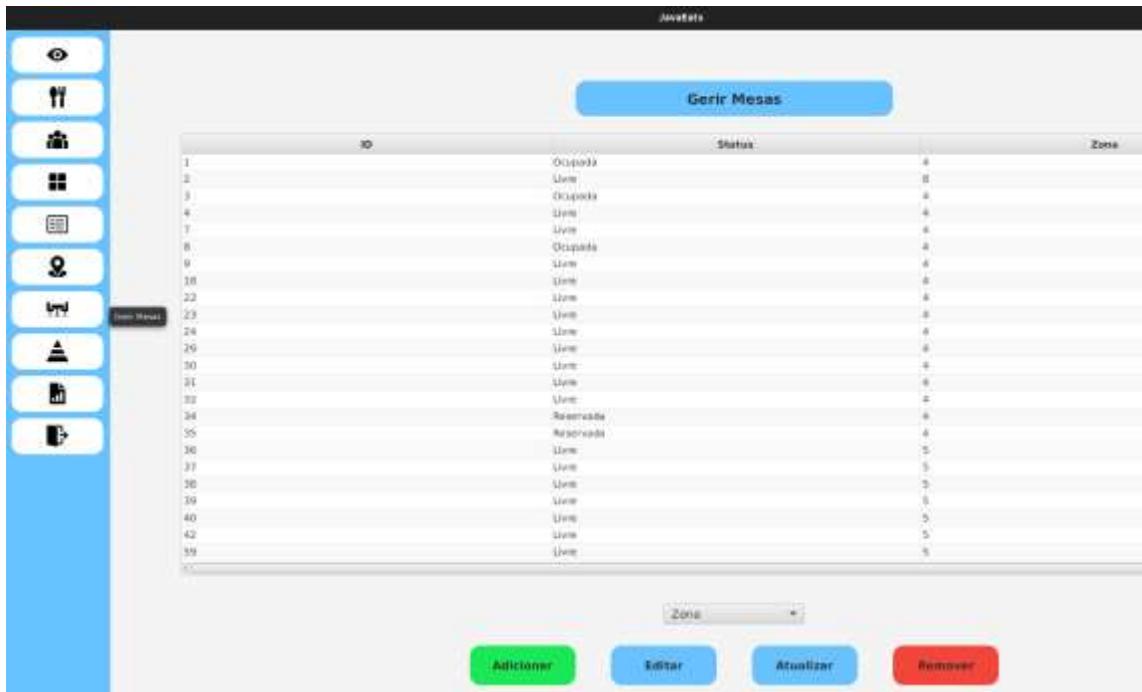
Buttons below the table:

- Adicionar (Add)
- Editar (Edit)
- Atualizar (Update)
- Remover (Remove)
- Procurar Cartão (Search Card)

FIGURA 27 - DASHBOARD GERENTE FUNCIONÁRIOS

10.7 Dashboard Gerente (Gerir Mesas)

Ao selecionar o botão “Gerir Mesas” aparecerá uma tabela com o ID, status e zona de cada mesa. Logo abaixo da tabela existem quatro botões em que é possível adicionar, editar, atualizar ou remover uma mesa.

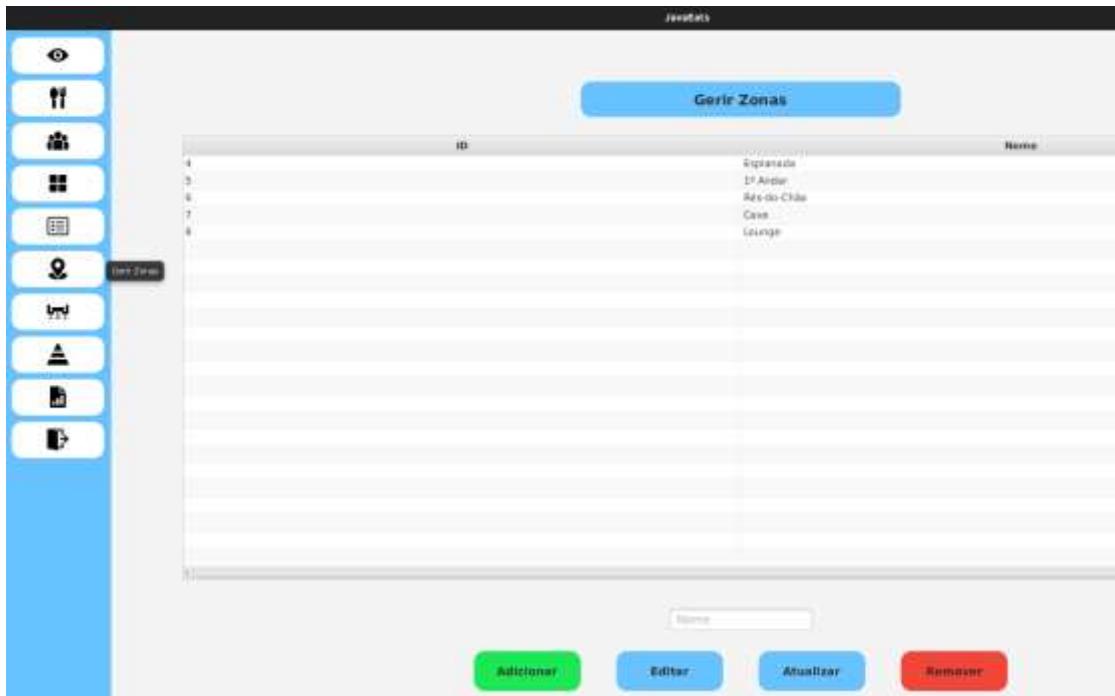


ID	Status	Zona
1	Ocupada	4
2	Livre	5
3	Ocupada	4
4	Livre	4
5	Livre	4
6	Ocupada	4
7	Livre	4
8	Ocupada	4
9	Livre	4
10	Livre	4
11	Livre	4
12	Livre	4
13	Livre	4
14	Livre	4
15	Livre	4
16	Livre	4
17	Livre	4
18	Livre	4
19	Livre	4
20	Livre	4
21	Livre	4
22	Livre	4
23	Livre	4
24	Livre	4
25	Livre	4
26	Livre	4
27	Livre	4
28	Livre	4
29	Livre	4
30	Livre	4
31	Livre	4
32	Livre	4
33	Livre	4
34	Livre	4
35	Livre	4
36	Livre	4
37	Livre	4
38	Livre	4
39	Livre	4
40	Livre	4
41	Livre	4
42	Livre	4
43	Livre	4

FIGURA 28 - DASHBOARD GERENTE MESA

10.8 Dashboard Gerente (Gerir Zonas)

Ao selecionar o botão “Gerir Zonas” aparecerá uma tabela com o ID e o nome da Zona correspondente. Logo abaixo da tabela existem quatro botões em que é possível adicionar, editar, atualizar ou remover uma zona.



The screenshot shows a dashboard titled "Gerir Zonas". On the left, there is a vertical sidebar with various icons. The "Gerir Zonas" icon is highlighted with a black border. The main area contains a table with columns "ID" and "Nome". The data in the table is:

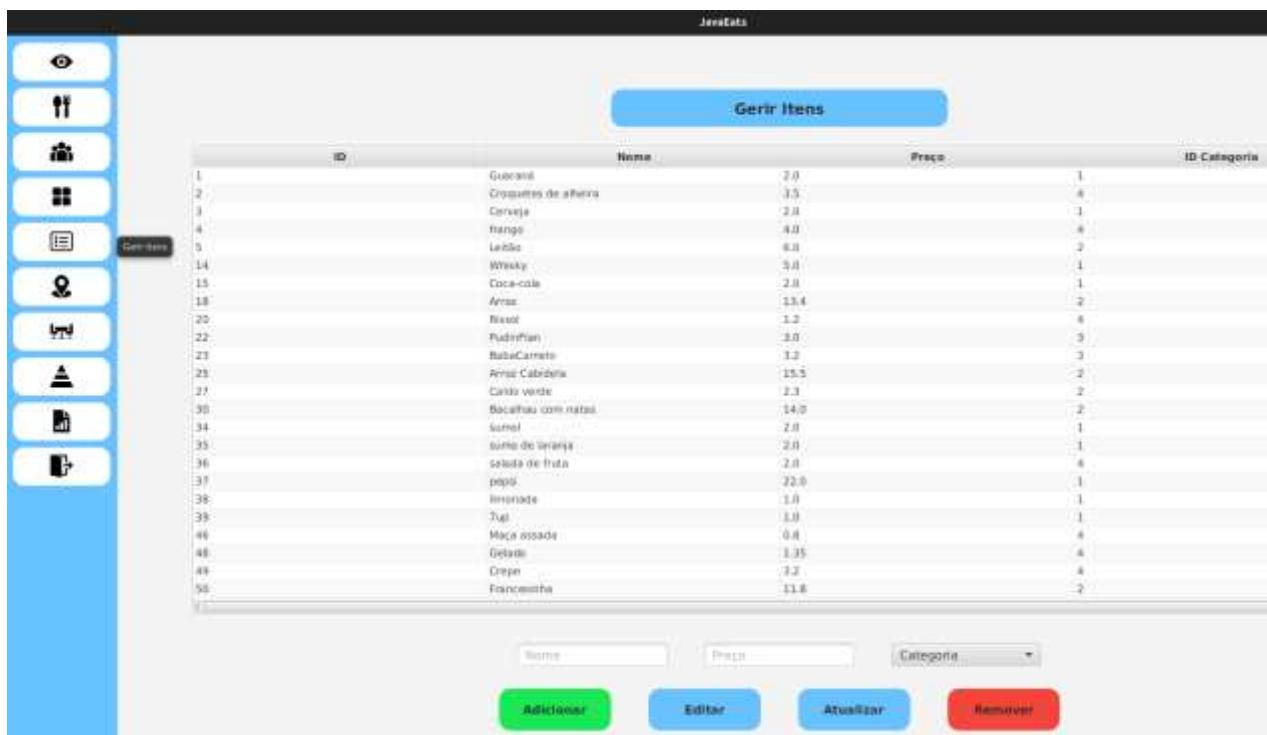
ID	Nome
1	Espraiado
2	2º Andar
3	Ribeira-Chão
4	Carril
5	Laranjeira

Below the table, there are four buttons: "Adicionar" (green), "Editar" (light blue), "Atualizar" (light blue), and "Remover" (red).

FIGURA 29 - DASHBOARD GERENTE ZONAS

10.9 Dashboard Gerente (Gerir Itens)

Ao selecionar o botão “Gerir Itens” aparecerá uma tabela com o ID, nome, preço e ID de categoria de cada item. Logo abaixo da tabela existem quatro botões em que é possível adicionar, editar, atualizar ou remover um item.



The screenshot shows a dashboard titled "Gerir Itens" (Manage Items). On the left, there is a vertical sidebar with various icons. The "Gerir Itens" icon is highlighted with a black border. The main area displays a table with the following data:

ID	Name	Preço	ID Categoria
1	Guerand	7.0	1
2	Croquettes de atum	3.5	4
3	Durante	2.0	1
4	frango	4.0	6
5	Lata	0.0	2
14	Whisky	3.0	1
15	Coca-cola	2.0	1
18	Anas	13.4	2
20	Bisco	1.2	6
22	PudimPan	3.0	3
23	BalaCaramelo	1.2	3
25	Anos Cabral	15.5	2
27	Canto verde	2.3	2
30	Bacalhau com natas	14.0	2
34	sumol	2.0	1
35	tuna do leirão	2.0	1
36	salada de fruta	2.0	6
37	papel	22.0	1
38	limonade	1.0	1
39	Tig	1.0	1
46	Maça assada	0.4	6
48	Gelado	1.35	4
49	Oven	3.2	4
50	Franconha	11.8	2

Below the table are four buttons: "Adicionar" (Add) in green, "Editar" (Edit) in blue, "Atualizar" (Update) in blue, and "Remover" (Remove) in red.

FIGURA 30 – DASHBOARD GERENTE ITENS

10.10 Dashboard Gerente (Relatórios)

Ao selecionar o botão “Relatórios” terá uma interface que contém dois campos para introduzir datas (de “data 1” até “data 2”). Após introduzir as datas pretendidas, é possível gerar o relatório através do botão “Gerar relatório”. Tendo o relatório gerado, surgem dois gráficos de barras (“Total faturado por dia” e “Total faturado por mesa”) e algumas estatísticas dos dados.



FIGURA 31 - DASHBOARD GERENTE RELATÓRIOS

Também é possível imprimir o relatório para um ficheiro PDF ao carregar no botão “Imprimir relatório”. Esta funcionalidade só pode ser realizada após o relatório ter sido gerado. Na figura seguinte está o exemplo do relatório em formato PDF referente ao que foi gerado anteriormente.

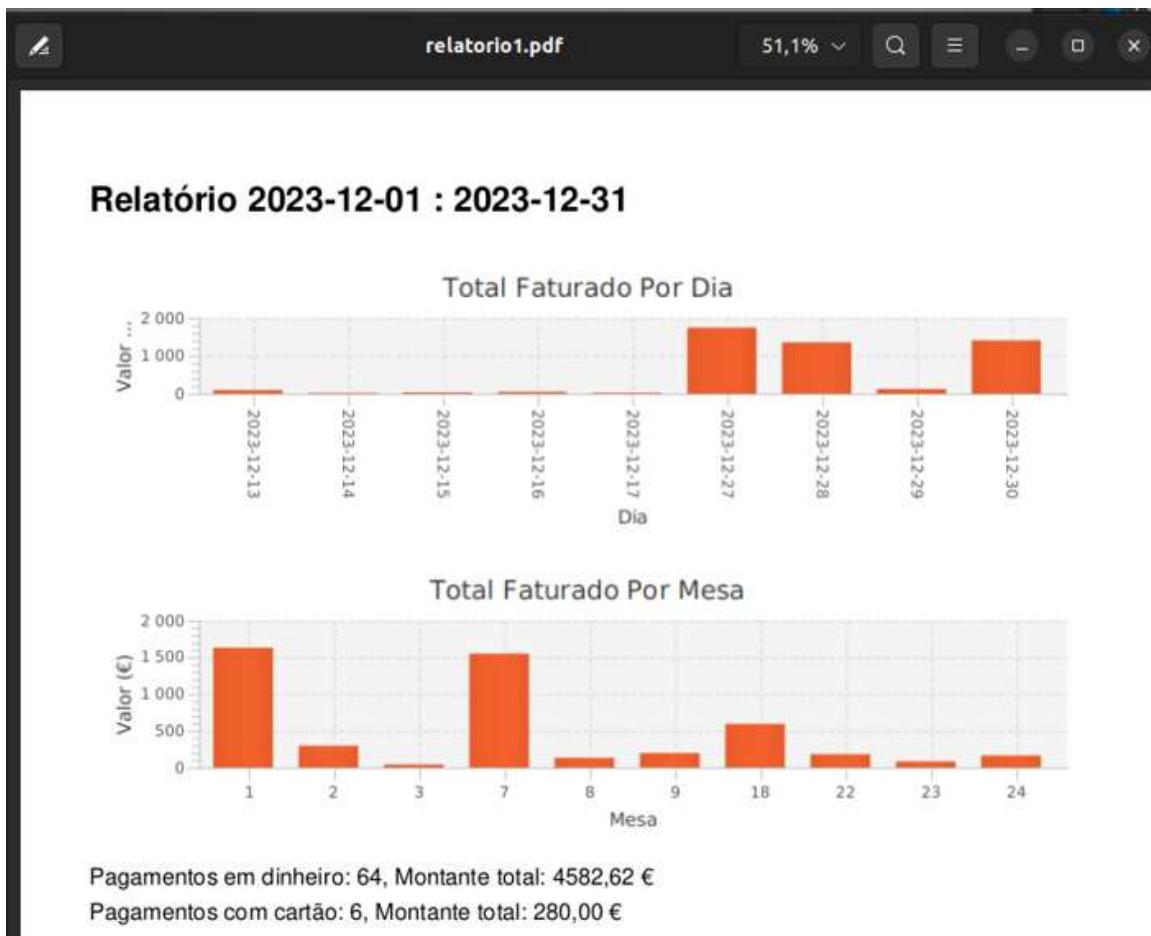


FIGURA 32 - RELATÓRIO PDF

10.11 Dashboard Funcionário

Na *Dashboard Principal* se entramos num perfil de funcionário, o mesmo irá ter acesso a uma interface onde pode fazer reservas de mesa, selecionar as diferentes zonas, as mesas que elas contêm e também fazer pedidos nas mesas. Basicamente um funcionário que não seja gerente não tem acesso à barra lateral localizado do lado esquerdo (funcionalidades especiais do Gerente), mas de resto a interface é toda igual. Na figura seguinte realiza-se o login numa conta de Funcionário.



FIGURA 33 - DASHBOARD PRINCIPAL FUNCIONÁRIO

Pode-se verificar na figura abaixo a aspetto da interface da “Home Page” de um Funcionário (interface igual a um Gerente, mas sem a barra lateral à esquerda).



FIGURA 34 - DASHBOARD FUNCIONÁRIO MESAS

10.12 Dashboard Reservar Mesas

Caso se pretenda realizar uma reserva de mesa, ao carregar no botão “Fazer reserva” vai abrir uma nova janela direcionada às reservas, tal como se pode verificar na figura abaixo. Neste caso em concreto, foi escolhida a mesa 4 no dia 16/01/2024 para as 12h30. Após introduzir os dados mencionados anteriormente, carrega-se no botão “Reservar” e surge uma mensagem de aviso para sabermos que a mesa foi reservada com sucesso.

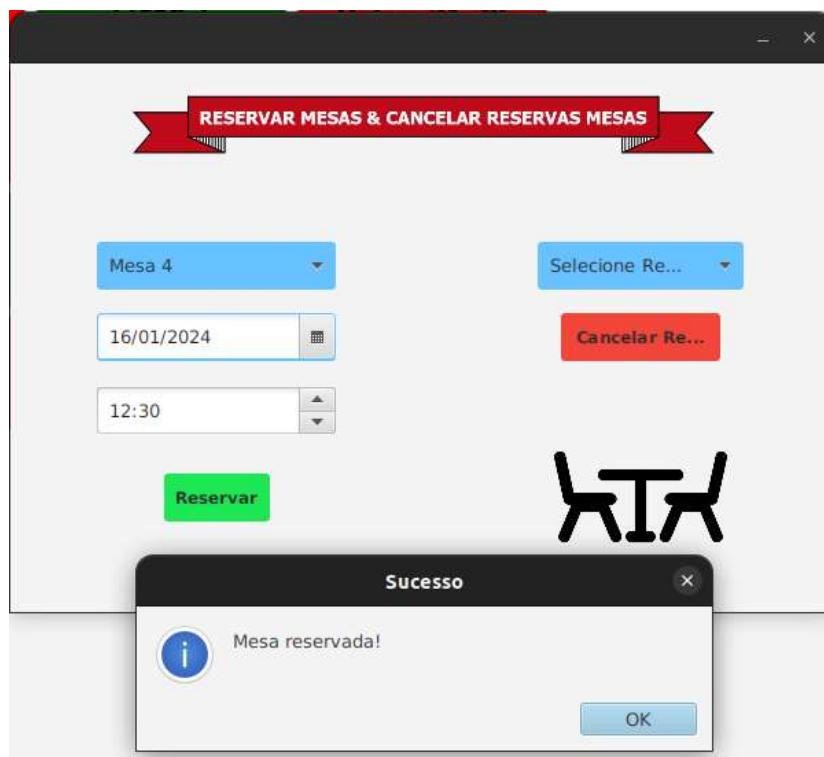


FIGURA 35 - DASHBOARD FUNCIONÁRIO RESERVAR

Na interface da mesa aparecerá a próxima reserva marcada. Caso não exista nenhuma reserva aparece a mensagem correspondente de que não existe nenhuma reserva para a mesa em questão.



FIGURA 36 - DASHBOARD FUNCIONÁRIO PRÓXIMA RESERVA

10.13Dashboard Pedido

Ao clicar numa mesa (o seguinte exemplo foi realizado na mesa 4) a interface passará a ter um aspeto como o da figura abaixo.

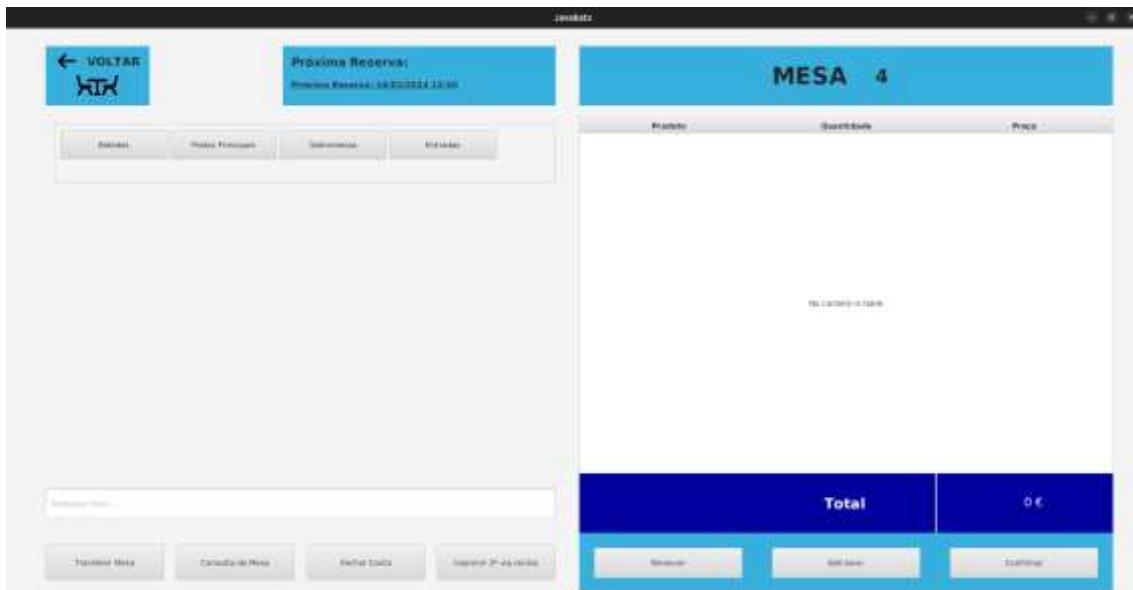


FIGURA 37 - DASHBOARD FUNCIONÁRIO PEDIDO

Dentro da interface da mesa podem-se realizar diversas funcionalidades. A principal é fazer um pedido e para isso temos de adicionar itens para a tabela que se encontra à direita (podemos listar os itens existentes ao carregar num botão de uma categoria tal como bebidas, pratos principais, sobremesas ou entradas ou então podemos pesquisar através da caixa de texto “Pesquisar item”). Depois de introduzido(s) o(s) item(ns) que se pretende(m), carrega-se no botão “Confirmar” para confirmar o pedido e a interface ficará com um aspeto parecido ao da seguinte figura (a tabela está preenchida com os itens escolhidos e surgirá uma mensagem de aviso para confirmar se pretendemos mesmo confirmar ou não o pedido em questão, ao qual também podemos adicionar uma descrição).

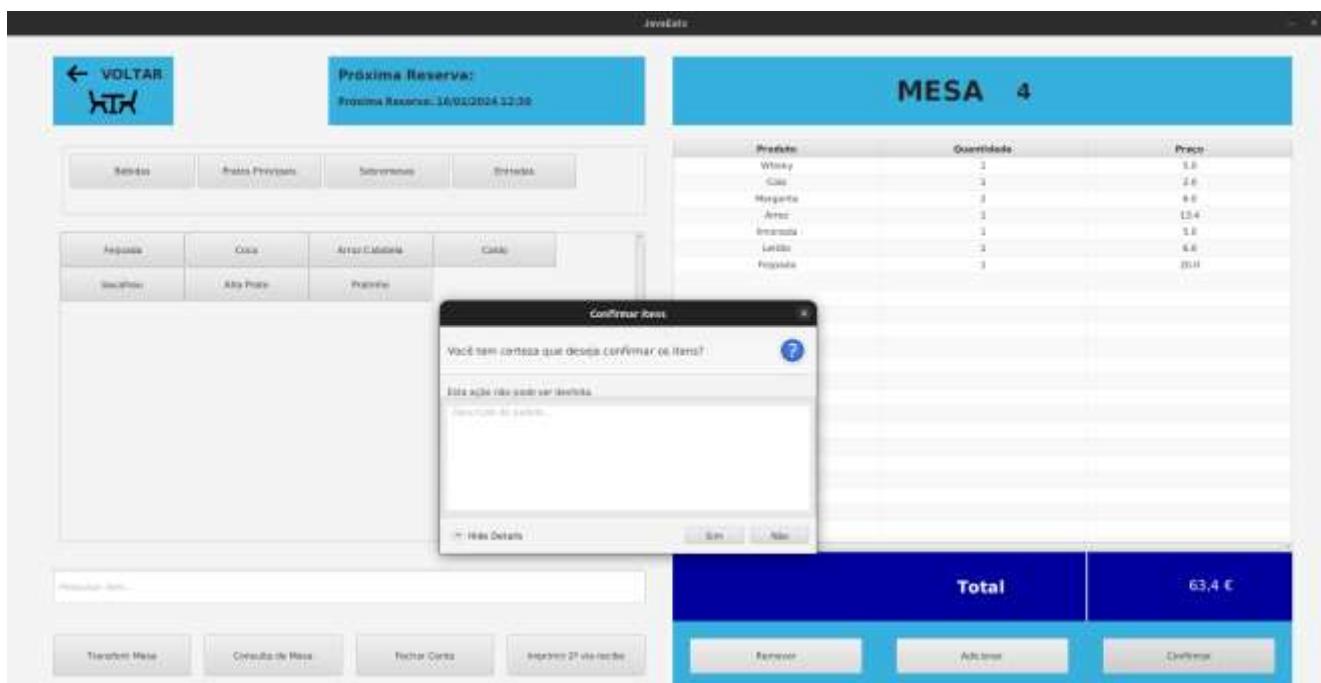


FIGURA 38 - DASHBOARD FUNCIONÁRIO ADICIONAR ITENS

Caso o pedido seja confirmado surgirá uma mensagem de aviso para o funcionário ter conhecimento que o pedido foi realizado com sucesso.

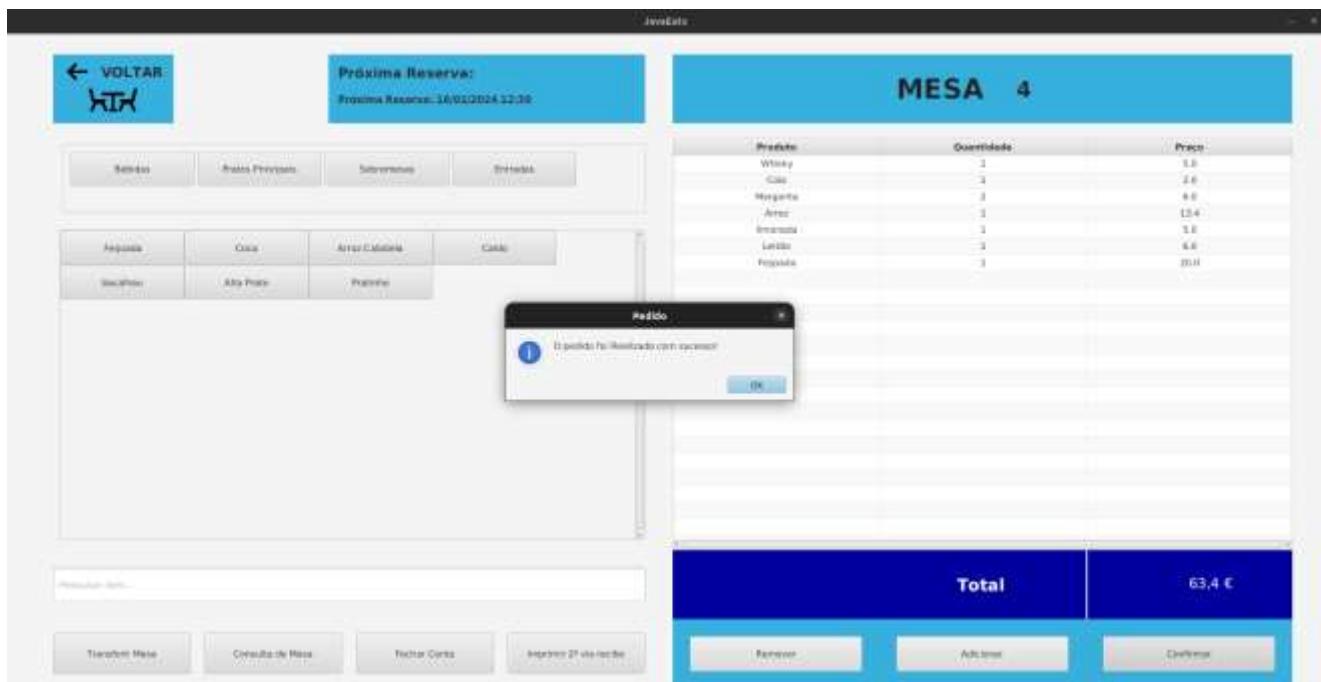


FIGURA 39 - DASHBOARD FUNCIONÁRIO PEDIDO CRIADO

10.13.1 Dashboard Pedido (Transferir Mesa)

Para além de fechar a conta, o funcionário também pode transferir a mesa, se assim for necessário. No processo de transferência de mesa, carrega-se no botão “Transferir Mesa” e de seguida surgirá uma janela na qual se escolhe a mesa destino da transferência. Esta escolha é feita a partir de um menu de *dropdown* que lista as mesas livres do restaurante. Neste caso a mesa escolhida como destino de transferência foi a mesa 29.



FIGURA 40 - DASHBOARD FUNCIONÁRIO TRANSFERIR MESA

Com a mesa selecionada e logo após ter carregado em “OK”, a mesa escolhida passará a ficar com os itens que a mesa origem tinha. Pode-se verificar o sucesso deste processo na figura seguinte, na qual a mesa 29 (mesa destino) fica com os itens que pertenciam à mesa 4.



FIGURA 41 - DASHBOARD FUNCIONÁRIO MESA TRANSFERIDA

10.13.2 Dashboard Pedido (Consulta de Mesa)

Outra funcionalidade que o funcionário pode executar na interface da mesa é consultar a mesa e para o fazer, precisa de carregar no botão “Consultar Mesa”. Esta consulta só é permitida caso a mesa tenha um atendimento em andamento, caso contrário não há itens na mesa e por sua vez não há consulta a fazer. Ao carregar no botão referido, é gerado um ficheiro em formato PDF com o conteúdo da mesa, neste caso os itens todos que ela tem e o valor total deles.



FIGURA 42 - DASHBOARD FUNCIONÁRIO CONSULTA DE MESA

Na figura seguinte verifica-se o que mostra a consulta de mesa em questão (mesa 29). Na consulta de mesa ainda não há método de pagamento pois esta funcionalidade é para simular o processo de quando se pede a conta num restaurante, em que o funcionário nos traz o recibo para que de seguida se proceda ao pagamento da conta.

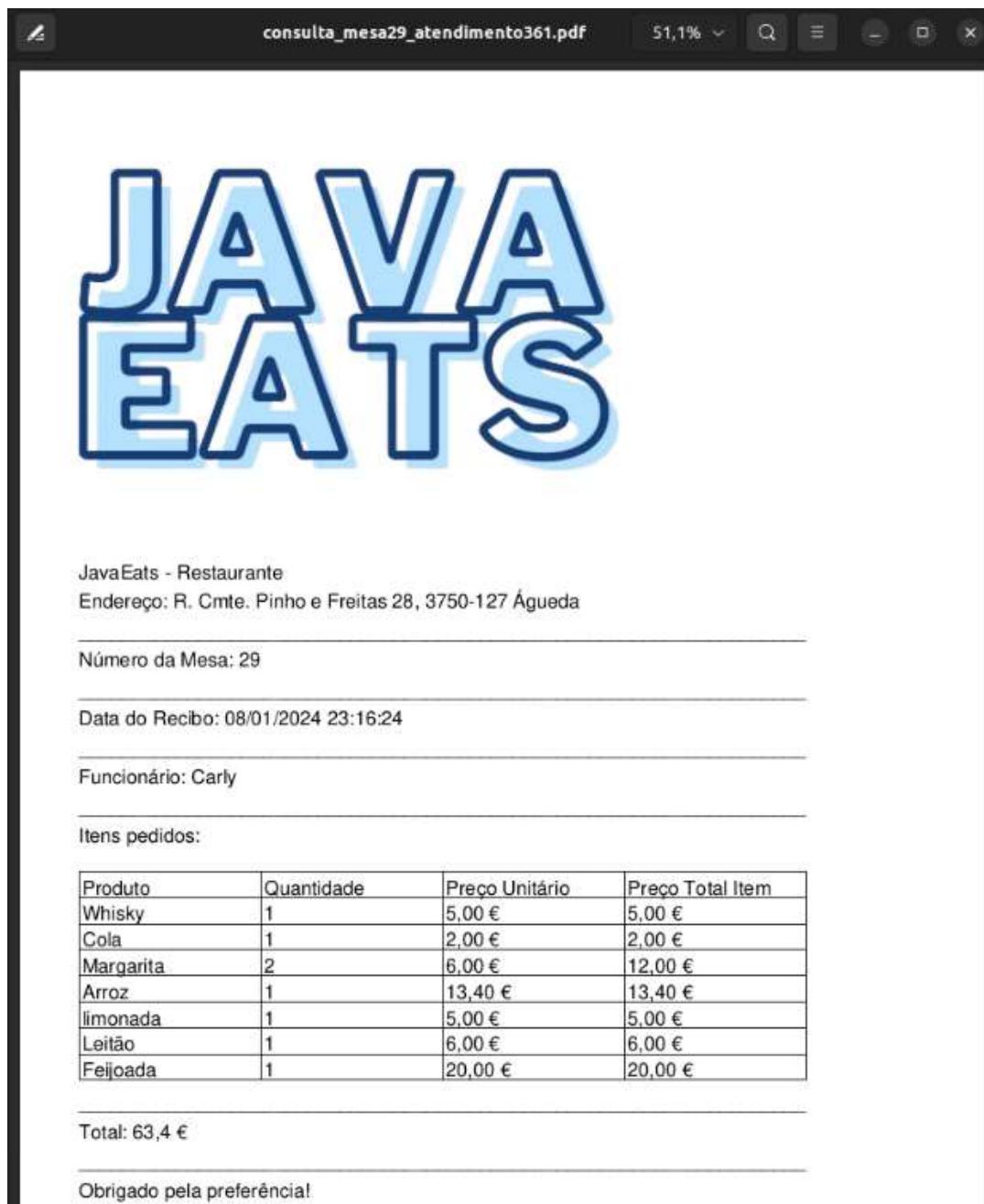


FIGURA 43 - RECIBO PEDIDO

10.13.3 Dashboard pedido (Imprimir 2^a via do recibo)

A última funcionalidade que o funcionário consegue realizar na interface da mesa é imprimir uma segunda via de um recibo. Esta funcionalidade foi implementada no sentido em que surja algum imprevisto ou engano, ou por alguma razão o cliente assim o pretenda. Para imprimir a segunda via carrega-se no botão “Imprimir 2^a via do recibo” e de seguida vai ser gerado um ficheiro em formato PDF com o conteúdo (itens) do atendimento finalizado mais recente para a mesa que queremos imprimir. Caso a mesa não tenha nenhum atendimento finalizado, não será possível imprimir uma segunda via pois também não há nenhum recibo.

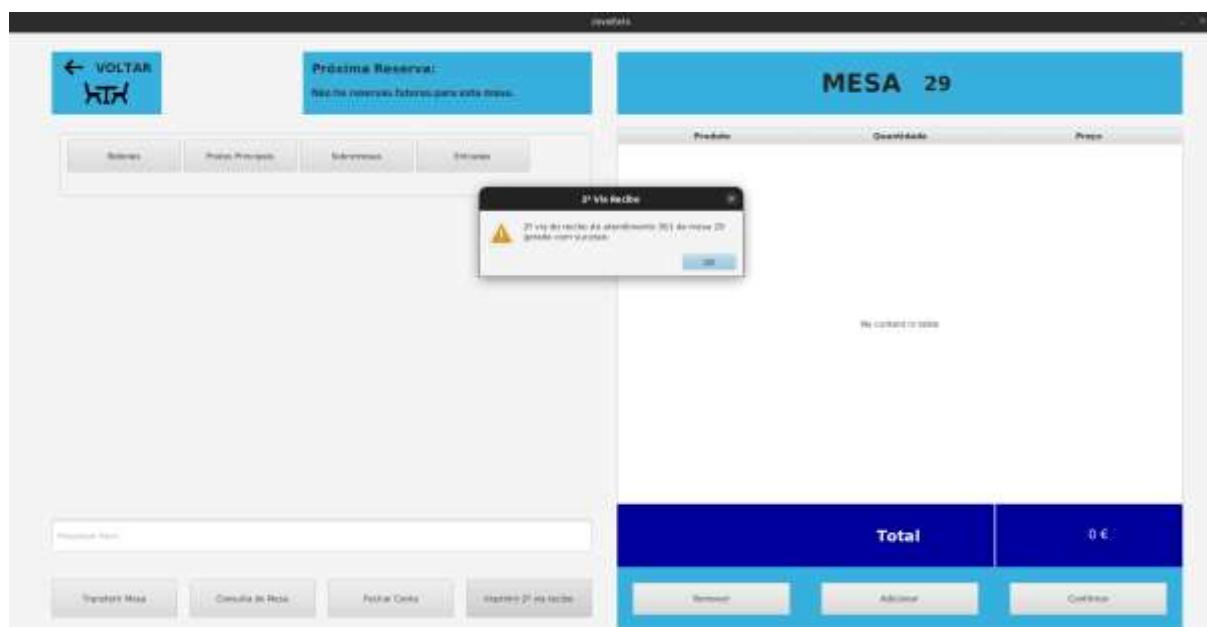


FIGURA 44 - IMPRIMIR 2^a VIA DO RECIBO

Na figura seguinte encontra-se a segunda via do recibo referente ao atendimento finalizado mais recente da mesa 29. Esta segunda via é bastante parecida ao recibo do pagamento, mas inclui mais algumas informações extra, tais como a data e hora de início e de fim do atendimento (ou seja, quando o primeiro pedido foi feito e quando o pagamento foi finalizado, respetivamente) e logo no início do ficheiro uma anotação que refere que estamos perante um recibo “Duplicado/2^a via”.

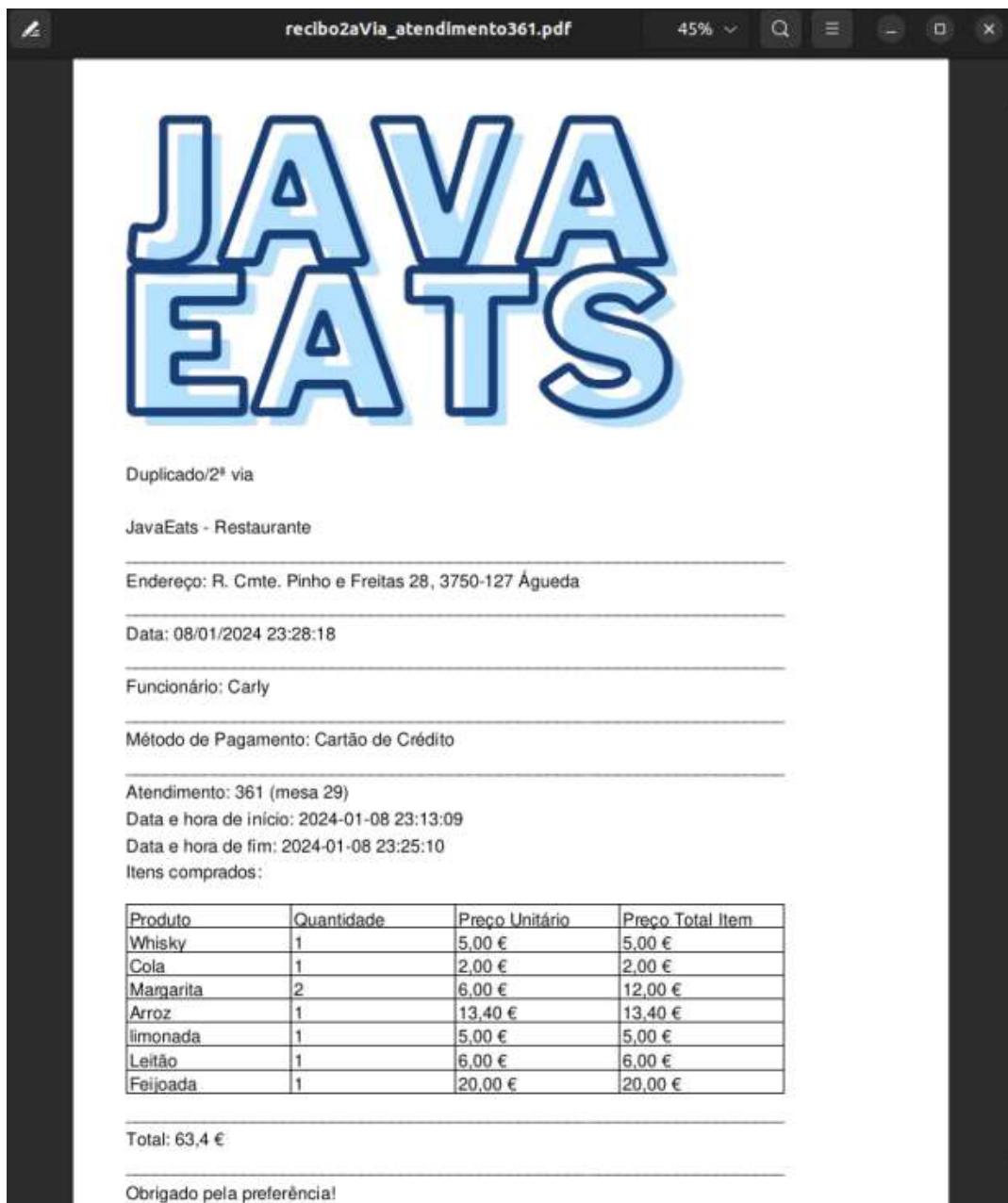


FIGURA 45 - RECIBO 2º VIA ATENDIMENTO

10.14 Dashboard Fechar Conta

Com o pedido realizado já é possível proceder ao pagamento do mesmo (para o fazer carregar-se no botão “Fechar Conta”). Só é permitido fechar a conta caso exista um pedido feito e por sua vez um atendimento por finalizar na mesa (ou seja, um atendimento em curso).

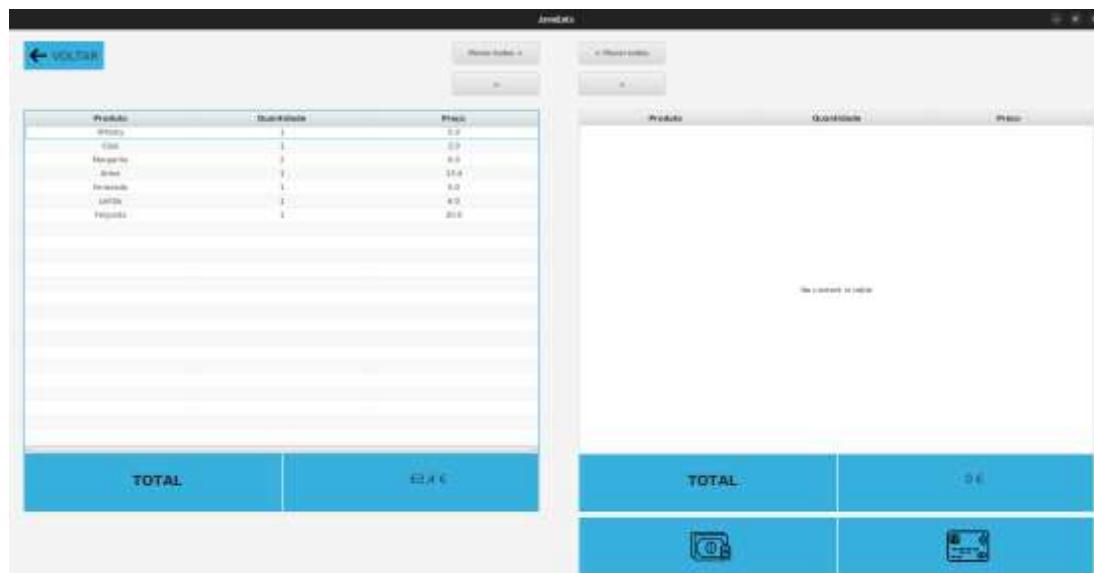


FIGURA 46 - DASHBOARD FECHAR CONTA

Primeiramente movem-se os itens que se pretendem pagar para a tabela da direita. Depois dos itens pretendidos estarem na tabela, pode-se pagar com dinheiro ou com cartão de crédito e o pagamento só acaba quando todos os itens tiverem sido pagos. No caso deste atendimento, o pagamento é feito com cartão como se pode verificar na figura seguinte.

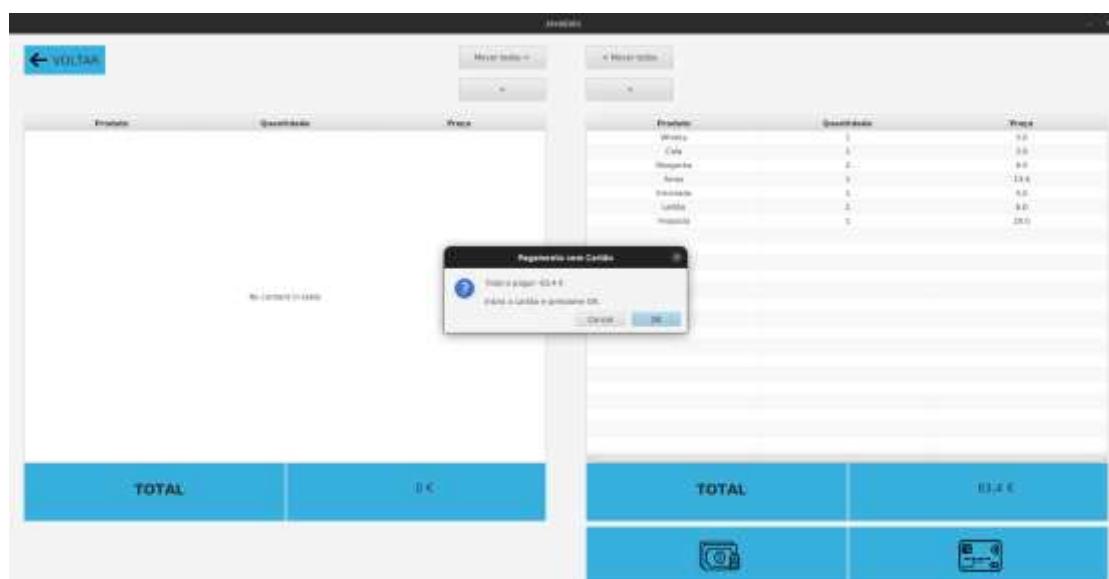


FIGURA 47 - DASHBOARD FECHAR CONTA PAGAMENTO

Depois do pagamento ser realizado com sucesso, é gerado um recibo em formato PDF para consultar o que foi pago. No recibo em questão aparecem algumas informações como a data e hora, o nome do funcionário que atendeu o pedido e fechou a conta, o número da mesa, o método de pagamento utilizado, os itens comprados e o valor total correspondentes aos mesmos. Desta forma fica mais fácil identificar tanto por parte do restaurante como por parte do cliente a que atendimento se refere o recibo.

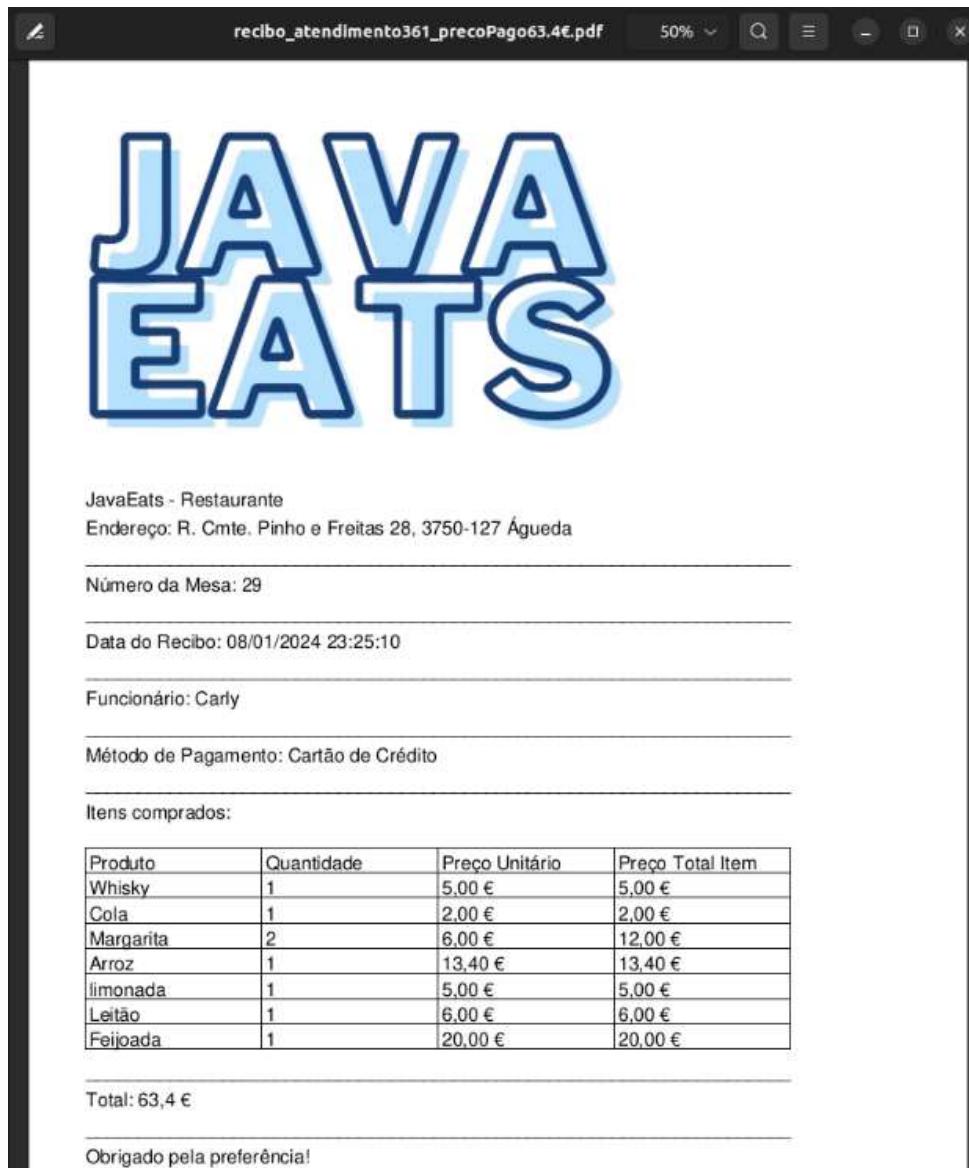


FIGURA 48 - RECIBO ATENDIMENTO

10.15 Dashboard Balcão

Na *Dashboard Principal* se entramos num perfil de empregado de balcão, o mesmo irá ter acesso a uma interface que mostra uma listagem dos pedidos feitos (tabela da esquerda e estão coloridos consoante há quanto tempo foram criados), onde é possível mover pedidos para a tabela da direita (pedidos prontos, na qual ficam a verde).



FIGURA 49 - DASHBOARD PRINCIPAL BALCÃO

Pode-se verificar na figura abaixo a aspeto da interface de um Empregado de Balcão (visualiza todos os pedidos feitos no restaurante).



FIGURA 50 - DASHBOARD BALCÃO

11. Manual de Instalação

Este guia foi elaborado para orientar o cliente de maneira clara e concisa através do processo de instalação da aplicação **JavaEats** no seu sistema. Ao seguir os passos apresentados neste manual, este estará pronto para aproveitar ao máximo os recursos robustos oferecidos pela aplicação.

O **JavaEats** não apenas simplifica a gestão de restaurantes, como também proporciona uma experiência intuitiva tanto para os gerentes quanto para a equipa de funcionários. Antes de prosseguir, o utilizador deve certificar que o seu sistema cumpre os requisitos de hardware mencionados no [tópico 5.1](#) e os requisitos necessários referidos abaixo para que seja possível executar a aplicação.

11.1 Requisitos obrigatórios do sistema

- Ligação à internet local do restaurante (para o funcionamento da base de dados);
- Java: Versão 8 ou superior (<https://www.java.com/en/>);
- JDK: Versão 17 ou superior (<https://www.oracle.com/java/technologies/downloads/>);
- Ficheiro executável **JavaEats** que contém o script para executar a aplicação, as dependências e o ficheiro “.jar” (disponível em <https://lucasduarte2.github.io/JavaEats/>).

11.2 Passos de instalação

11.2.1 Java

Para fazer o download do Java acede-se ao link mencionado anteriormente (<https://www.java.com/en>) e de seguida selecionar o botão “Download Java”.



FIGURA 51 - DOWNLOAD JAVA

Após selecionar o botão anterior, deve pressionar novamente o botão “Download Java” e seguidamente executar o ficheiro “.exe” que é descarregado.



FIGURA 52 - CONFIRMAR DOWNLOAD

Ao executar o ficheiro, deve carregar na opção “Sim” para proceder à instalação do *Java Runtime Environment*.

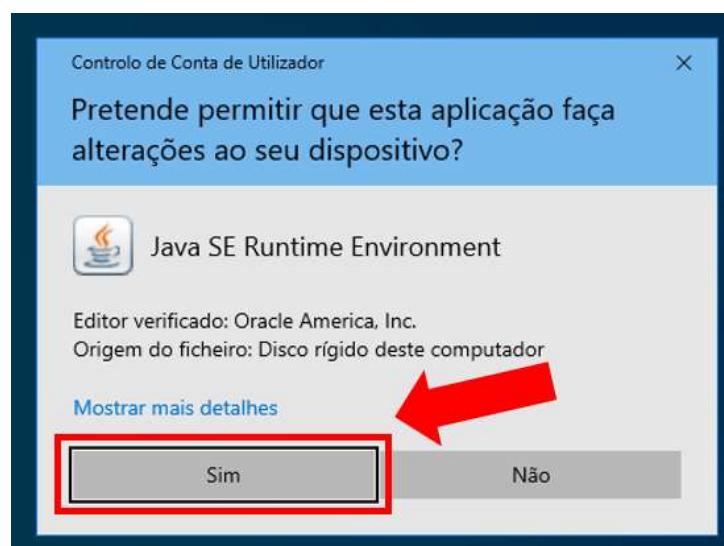


FIGURA 53 - EXECUTAR O FICHEIRO

Quando aparecer a janela de instalação do Java, basta selecionar a opção “Install” e aguardar que acabe o processo de instalação.



FIGURA 54 - INSTALAR O JAVA

11.2.2 JDK

Para fazer o download do JDK acede-se ao link mencionado anteriormente (<https://www.oracle.com/java/technologies/downloads/>), de seguida seleciona-se a coluna “Windows” e por fim escolhe-se o link referente ao “x64 Installer”. Após o ficheiro ser descarregado na totalidade, executa-se o mesmo.

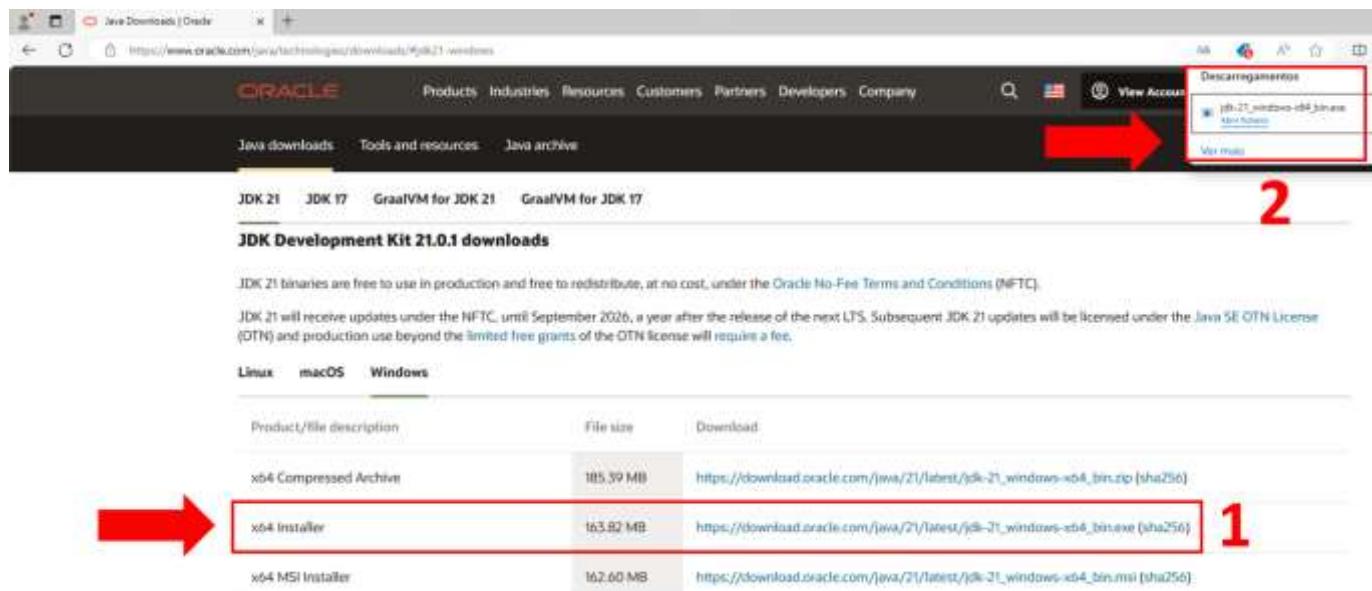


FIGURA 55 - DOWNLOAD JKD DEVELOPMENT KIT

Ao executar o ficheiro, carregar na opção “Sim” para proceder à instalação do *JDK Development Kit*.



FIGURA 56 - INSTALAR O FICHEIRO JDK

Quando aparecer a janela de instalação do JDK, basta selecionar a opção “Next” até que comcece o processo de instalação.



FIGURA 57 - CLICAR NO CONTINUAR

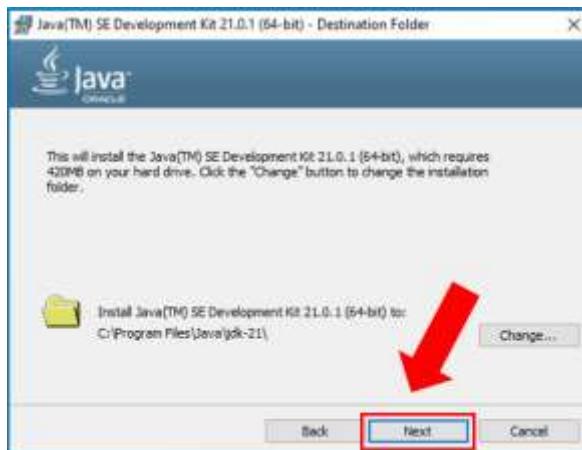


FIGURA 58 - CONFIRMAR A PASTA DE INSTALAÇÃO

11.2.3 Ficheiro executável JavaEats

Para fazer o download do ficheiro executável da aplicação **JavaEats** acede-se ao link mencionado anteriormente (<https://lucasduarte2.github.io/JavaEats/>) e seleciona-se o botão “Download”.

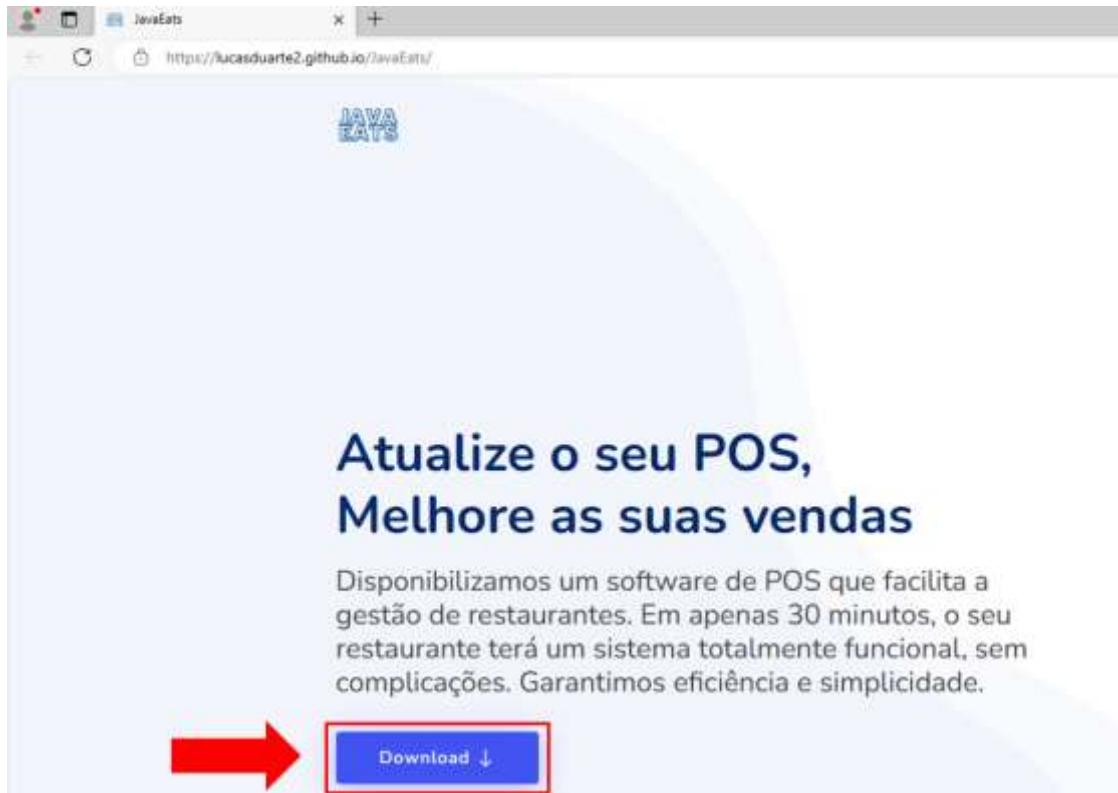


FIGURA 59 - DOWNLOAD FICHEIRO EXECUTÁVEL JAVAEATS

Após fazer o download do ficheiro, acede-se à pasta de transferências do Windows e procede-se à descompactação do ficheiro “JavaEats.zip”, para isso deve-se selecionar a opção “Extrair Todos”.

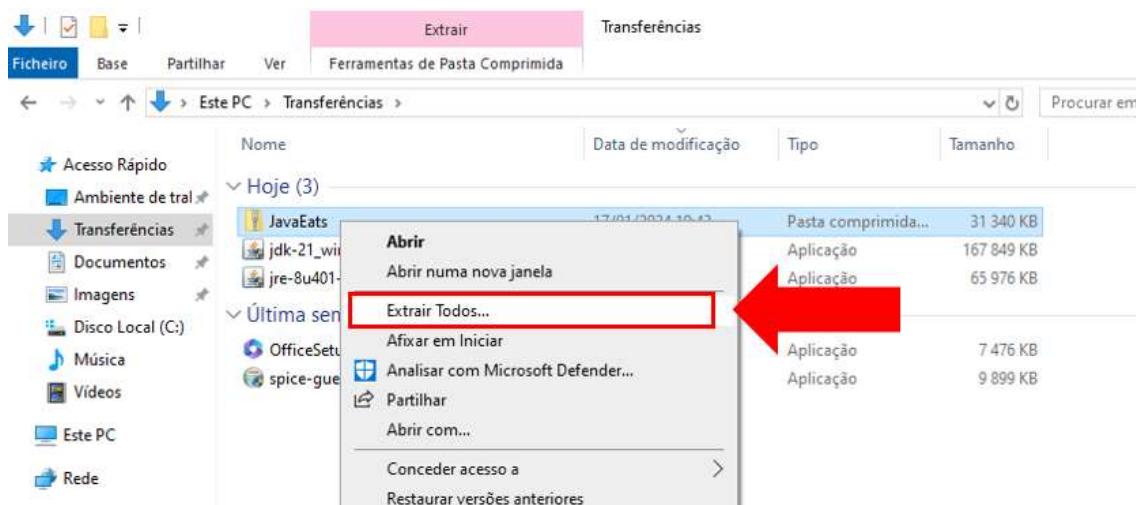


FIGURA 60 - DESCOMPACTAR FICHEIRO JAVAEATS.ZIP

Neste caso a pasta escolhida para guardar o ficheiro descompactado foi o ambiente de trabalho, como se pode ver na figura seguinte (a pasta para guardar fica à liberdade de escolha por parte do utilizador). Com a pasta escolhida, seleciona-se a opção “Extrair” para finalizar a descompactação.

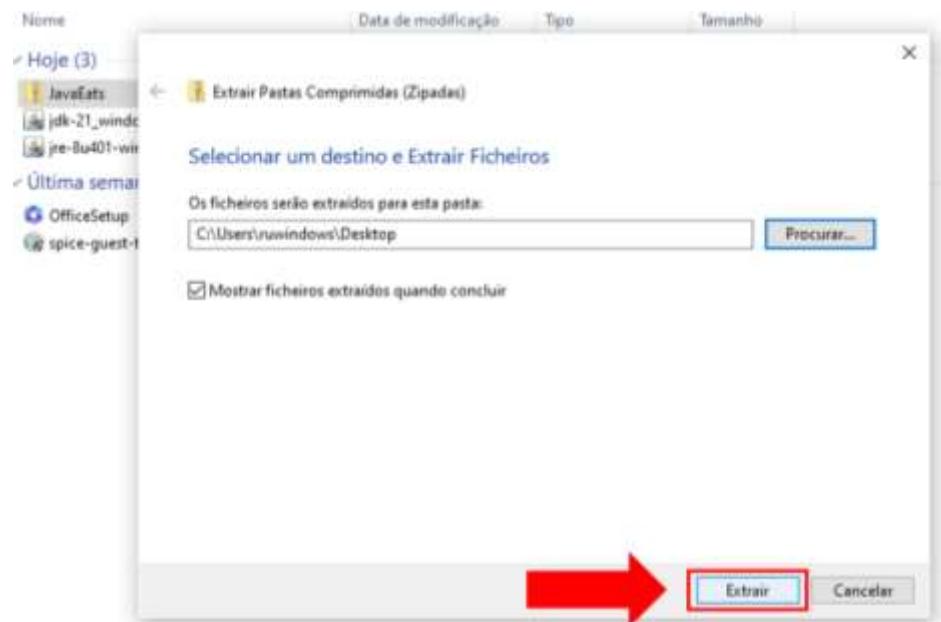


FIGURA 61 - ESCOLHA DA PASTA PARA DESCOMPACTAÇÃO

Estando a pasta descompactada, abre-se a mesma e por fim executa-se o ficheiro “JavaEats” que se encontra lá dentro.



FIGURA 62 - PASTA JAVAEATS

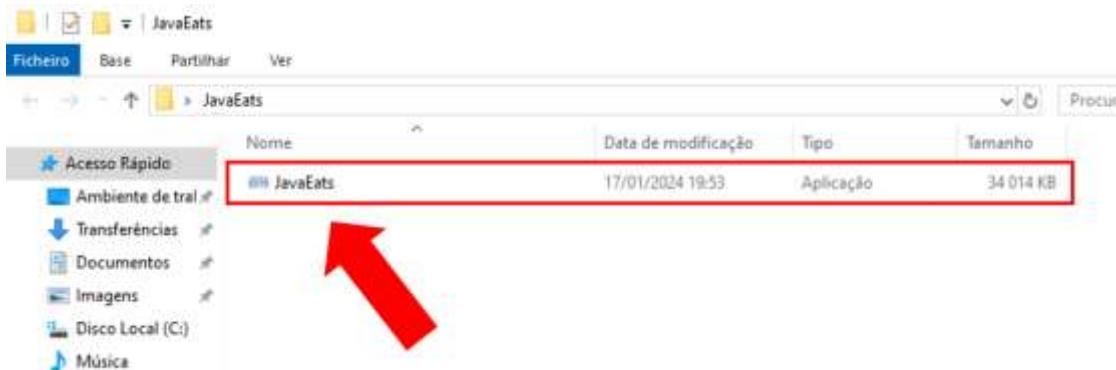


FIGURA 63 - FICHEIRO JAVAEATS NA PASTA

Se tudo até este momento correu como descrito neste manual, o utilizador encontra-se apto para executar a aplicação JavaEats e desfrutar das funcionalidades da mesma na sua normalidade.



FIGURA 64 - APLICAÇÃO JAVAEATS AO ABRIR

11.3 Resoluções de problemas

Caso ao executar a aplicação **JavaEats** o utilizador encontrar o erro descrito na figura abaixo, será necessário editar as variáveis de ambiente do sistema diretamente no Windows pois as mesmas não foram automaticamente definidas.

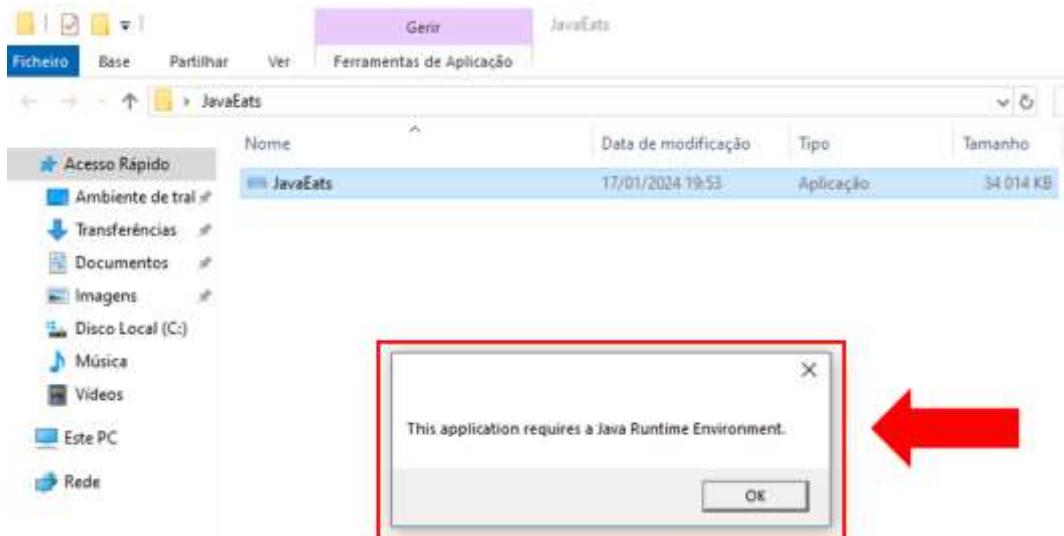


FIGURA 65 - ERRO AO EXECUTAR A APLICAÇÃO

Para configurar essas variáveis, no botão Iniciar do Windows pesquisa-se “variáveis”, tal como se pode verificar na seguinte figura e seleciona-se a opção “Editar as variáveis de ambiente do sistema”.

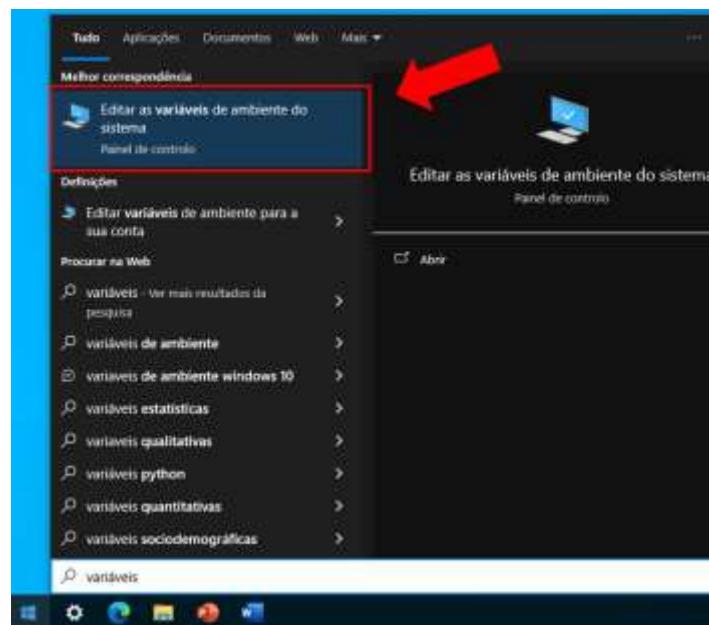


FIGURA 66 - PESQUISAR VARIÁVEIS

Na janela que se abre de seguida, acede-se à opção “Variáveis de ambiente”.

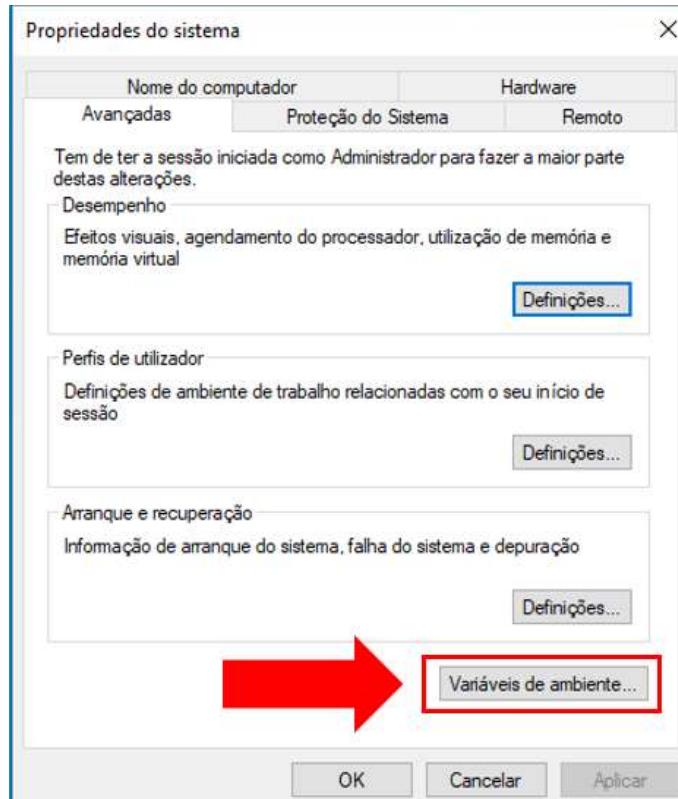


FIGURA 67 - VARIÁVEIS DE AMBIENTE

Após selecionar a opção anterior, carregar na linha “Path” e por fim em “Editar” para que seja possível definir os diretórios referentes às variáveis de ambiente.

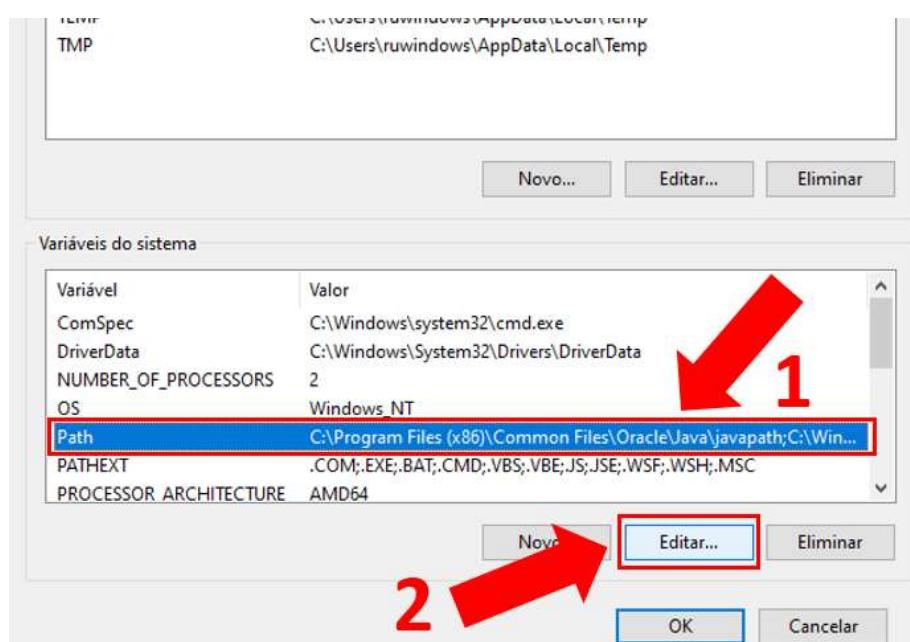


FIGURA 68 - ESCOLHER PATH E DEPOIS EDITAR

Na figura seguinte estão destacados os dois diretórios (“C:\Program Files (x86)\Common Files\Oracle\Java\javapath” e “C:\Program Files\Java\jdk-21\bin”) necessários para adicionar nas variáveis de ambiente do sistema (tenha em conta que os mesmos podem variar consoante a pasta de instalação do Java e do JDK). Com os diretórios devidamente adicionados, basta selecionar a opção “OK” na janela e a aplicação JavaEats está pronta a ser executada.

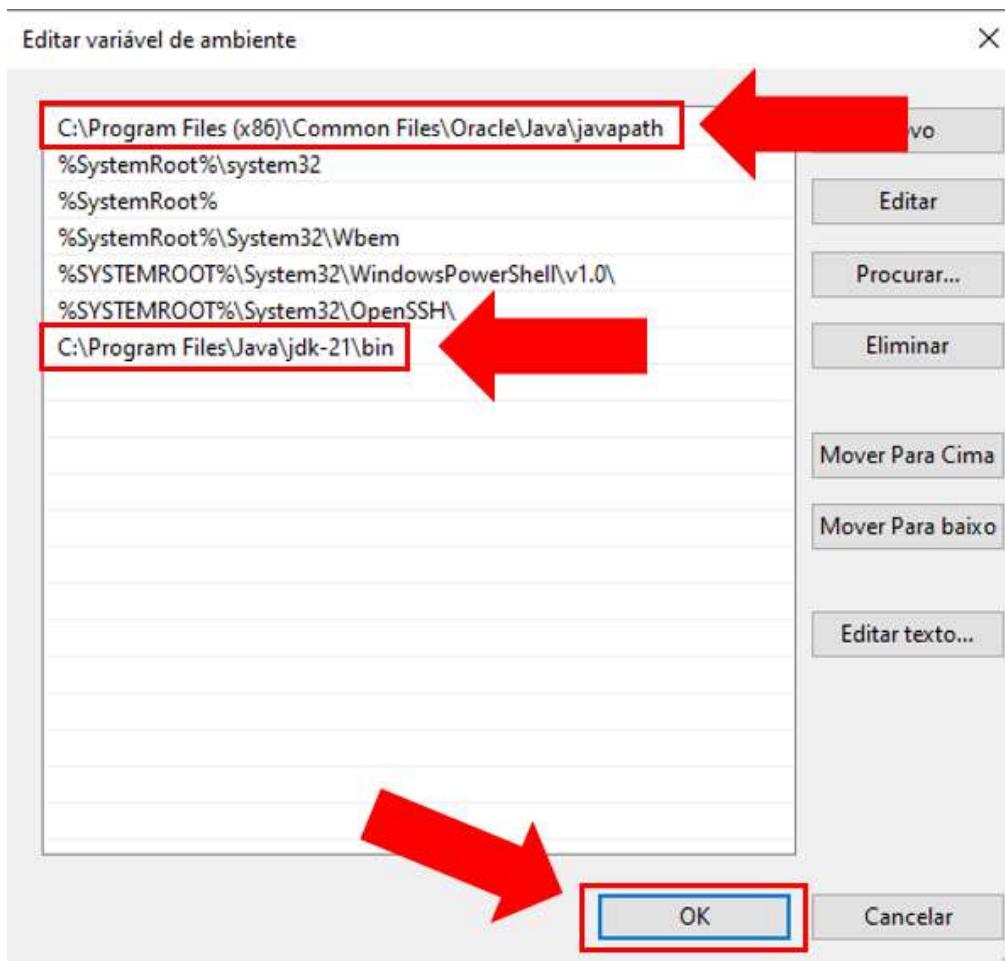


FIGURA 69 - VARIÁVEIS DE AMBIENTE NECESSÁRIAS

11.4 Tags RFID

Caso o gerente do restaurante deseje incorporar as tags RFID como uma opção para iniciar sessão no JavaEats (apesar de ser um *hardware* opcional, visto que a aplicação funciona normalmente sem essa funcionalidade), abaixo encontra-se a descrição para implementar essa inovação.

Num mercado cada vez mais orientado para a tecnologia, a implementação de sistemas inovadores em restaurantes pode ser crucial para melhorar a eficiência operacional e oferecer uma experiência mais fluida tanto para os funcionários quanto para os clientes. O JavaEats, um ponto de venda (POS System) desenvolvido em Java, destaca-se ao utilizar identificação por radiofrequência (RFID) para simplificar o processo de login.

Como funciona:

O JavaEats dá a opção de substituir o tradicional método de login, que envolve inserir manualmente as credenciais, por uma abordagem inovadora com tags RFID. Cada funcionário possui uma tag RFID única associada à sua conta. Ao desejar aceder ao sistema, o funcionário apenas encosta a sua tag RFID num leitor conectado a um Arduino.

O Arduino, responsável pela leitura da tag RFID, transmite os dados para a aplicação Java do JavaEats. Esta aplicação inclui uma classe dedicada que processa as informações provenientes do Arduino, autenticando automaticamente o funcionário correspondente. Este método permite o acesso imediato ao sistema, sem a necessidade de inserir manualmente as credenciais.

Vantagens do uso de Tags RFID no JavaEats:

Eficiência e Rapidez: A utilização de tags RFID no JavaEats elimina a necessidade da introdução manual de credenciais, tornando o processo de login consideravelmente mais rápido. Esta eficiência é particularmente benéfica em ambientes movimentados como os de restaurantes.

Autenticação Segura: A segurança é uma prioridade, e com a implementação das tags RFID, cada tag é única, proporcionando uma camada adicional de proteção. Esta abordagem reduz significativamente os riscos associados com as partilhas de passwords, protegendo contra acessos não autorizados.

Experiência do Funcionário: A simplicidade do processo de login através de RFID, melhora a experiência do funcionário, ao reduzir o tempo que este demora a iniciar o sistema. Esta facilidade além dos benefícios acima citados, também melhora a produtividade.

```

#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 53
#define RST_PIN 49

MFRC522 mfrc522(SS_PIN, RST_PIN);

void setup() {
    //Inicializa a comunicação serial com uma taxa de 9600 bps
    Serial.begin(9600);

    //Inicializa a biblioteca SPI
    SPI.begin();

    //Inicializa o módulo MFRC522
    mfrc522.PCD_Init();
}

void loop() {
    //Verifica se um novo cartão RFID está presente
    if (mfrc522.PICC_IsNewCardPresent()) {
        //Se um novo cartão está presente, tenta ler o número serial do cartão
        if (mfrc522.PICC_ReadCardSerial()) {
            //Imprime o número serial do cartão em formato hexadecimal
            for (byte i = 0; i < mfrc522.uid.size; i++) {
                Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
                Serial.print(mfrc522.uid.uidByte[i], HEX);
            }

            //Imprime uma nova linha após o número serial
            Serial.println();

            //Desativa o cartão RFID para evitar leituras repetidas
            mfrc522.PICC_HaltA();

            //Adiciona um pequeno atraso para evitar leituras repetidas
            delay(100);
        }
    }
}

```

FIGURA 70 - CÓDIGO DO ARDUINO (RFID)

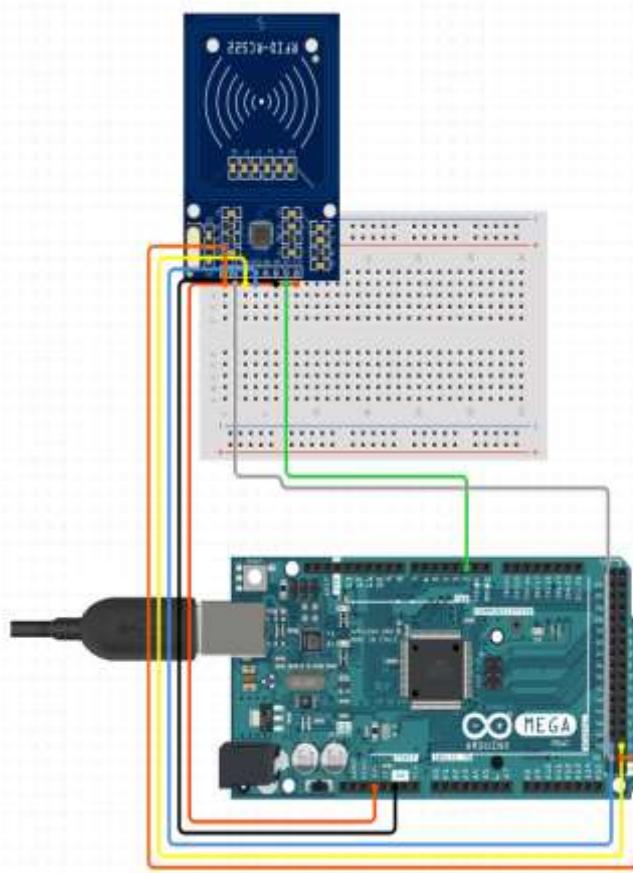


FIGURA 72 - ARDUINO RFID ESQUEMA



FIGURA 71 - PULSEIRA RFID

12. Testes e validação

Os testes de *software* são um processo crítico no desenvolvimento de *software*. São utilizados para avaliar a qualidade do programa e garantir que ele atenda aos requisitos do utilizador. Estes testes podem ser realizados em diferentes estágios do ciclo de vida do *software*, desde a fase de desenvolvimento até à fase de manutenção.

São importantes porque ajudam a identificar defeitos e falhas no programa antes que ele seja finalizado. Isso ajuda a certificar que o software seja confiável, seguro e cumpra com as exigências do utilizador. Vale a pena referir que em situação de trabalho podem reduzir os custos do desenvolvimento, uma vez que os defeitos são identificados e corrigidos mais cedo no ciclo de vida do software.

Existem vários tipos de testes de *software* que podem ser feitos, neste projeto foram feitos testes unitários que são utilizados para avaliar segmentos individuais do programa, testes de integração com a finalidade de confirmar se os vários segmentos de software quando combinados estão a funcionar corretamente, testes de validação para conferir se o *software* atende aos requisitos do utilizador e testes de sistema que são realizados para verificar se todo o sistema funciona corretamente.

Ainda foram feitos testes de usabilidade para avaliar a experiência do utilizador, garantindo assim que o *software* seja intuitivo e que proporcione uma boa interação aos seus utilizadores. Estes testes procuram identificar e solucionar potenciais problemas de navegação, acessibilidade e satisfação do utilizador ao utilizar o programa.

12.1 Testes unitários

Os testes unitários são uma técnica de testes de *software* que se baseia em testar pequenas partes de um programa. O objetivo é garantir que cada parte esteja a funcionar corretamente antes de integrá-la num sistema maior.

Estes são importantes porque ajudam a identificar erros no código de forma rápida e eficiente. Para além disso também economizam tempo e dinheiro, porque os erros são corrigidos antes que se tornem mais difíceis e caros de corrigir. Estes testes ajudam a melhorar a qualidade de *software*, pois garantem que cada segmento do programa esteja a executar corretamente. Isto ajuda a evitar problemas de integração e garante que o software esteja a funcionar conforme o esperado.

Para começar a escrever os testes unitários é preciso identificar os segmentos do programa e criar testes para cada um deles.

Finalmente, estes testes serão escritos em JUnit que foi o que aprendemos na unidade curricular de Engenharia de Software lecionada pelo professor Joaquim Ferreira.

12.1.1 CargoTest

Nesta parte, vão ser verificados os métodos da classe chamada Cargo.

```
@Test  
void testToString() {  
    Cargo cargo = new Cargo( descricao: "Gerente");  
    assertEquals( expected: "Gerente", cargo.toString());  
}
```

FIGURA 73 - CARGOTEST (TESTTOSTRING())

O objetivo do teste é verificar se o método retorna a descrição correta do cargo. Neste caso, o cargo é “Gerente” e o teste passa com sucesso se o método retornar “Gerente”.

```
@Test  
void getDescricao() {  
    Cargo cargo = new Cargo( descricao: "Gerente");  
    assertEquals( expected: "Gerente", cargo.getDescricao());  
}
```

FIGURA 74 - CARGOTEST (GETDESCRICAO())

O código da figura anterior cria um objeto da classe Cargo com o argumento "Gerente" e, em seguida, chama o método getDescricao() dessa instância. A expectativa é que o valor retornado por getDescricao() seja igual a "Gerente".

```
@Test  
void getIdCargo() {  
    Cargo cargo = new Cargo( descricao: "Gerente", idCargo: 1);  
    assertEquals( expected: 1, cargo.getIdCargo());  
}
```

FIGURA 75 - CARGOTEST (GETIDCARGO())

O objetivo do teste é verificar se o método retorna o ID correto do cargo. Neste caso, o cargo é “Gerente” e o ID é 1. O teste passa se o método getIdCargo() retorna 1.

```
@Test
void setDescricao() {
    Cargo cargo = new Cargo( descricao: "Gerente");
    cargo.setDescricao("Gerente do Restaurante");
    assertEquals( expected: "Gerente do Restaurante", cargo.getDescricao());
}
```

FIGURA 76 - CARGOTEST (SETDESCRICAO())

O objetivo do teste é verificar se o método atribui corretamente a descrição do cargo. Neste caso, o cargo é “Gerente” e o método setDescricao() é chamado com o argumento “Gerente do Restaurante”. O teste passa se o método getDescricao() retorna “Gerente do Restaurante”.

```
@Test
void setIdCargo() {
    Cargo cargo = new Cargo( descricao: "Gerente");
    cargo.setIdCargo(2);
    assertEquals( expected: 2, cargo.getIdCargo());
}
```

FIGURA 77 - CARGOTEST (SETIDCARGO())

O objetivo do teste é verificar se o método atribui corretamente o ID do cargo. Neste caso, o cargo é “Gerente” e o ID é definido como 2. O teste passa se o método getIdCargo() retorna 2.

```
@Test
void exibirInformacoesCargo() {
    Cargo cargo = new Cargo( descricao: "Gerente", idCargo: 1);
    //Este teste verifica se a exibição de informações não lança exceções
    assertDoesNotThrow(() -> cargo.exibirInformacoesCargo());
}
```

FIGURA 78 - CARGOTEST (EXIBIRINFORMACOESCARGO())

O objetivo do teste é verificar se o método não lança exceções ao exibir informações do cargo. Neste caso, o cargo é “Gerente” e o ID é 1. O teste passa se o método exibirInformacoesCargo() não lançar exceções.

12.1.2 CategoriaTest

Categoriatest tal como o nome indica vai testar os métodos da classe Categoria.

```
@Test
void testToString() {
    Categoria categoria = new Categoria( id: 1, nome: "Comida");
    assertEquals( expected: "Comida", categoria.toString());
}
```

FIGURA 79 - CATEGORIATEST (TESTTOSTRING())

O objetivo do teste é verificar se o método retorna a descrição correta da categoria. Neste caso, a categoria é “Comida” e o teste passa se o método retornar “Comida”.

```
@Test
void getName() {
    Categoria categoria = new Categoria( id: 1, nome: "Comida");
    assertEquals( expected: "Comida", categoria.getName());
}
```

FIGURA 80 - CATEGORIATEST (GETNAME())

O objetivo do teste é verificar se o método retorna o nome correto da categoria. Neste caso, a categoria tem o ID 1 e o nome “Comida”. O teste passa se o método getName() retorna “Comida”.

```
@Test
void setName() {
    Categoria categoria = new Categoria( id: 1, nome: "Comida");
    categoria.setName("Bebida");
    assertEquals( expected: "Bebida", categoria.getName());
}
```

FIGURA 81 - CATEGORIATEST (SETNAME())

O objetivo do teste é verificar se o método atribui corretamente o nome da categoria. Neste caso, a categoria tem o ID 1 e o nome “Comida”. O método setName() é chamado com o argumento “Bebida”. O teste passa se o método getName() retorna “Bebida”.



```
@Test
void getId() {
    Categoria categoria = new Categoria( id: 1, nome: "Comida");
    assertEquals( expected: 1, categoria.getId());
}
```

FIGURA 82 - CATEGORIATEST (GETID())

O objetivo do teste é verificar se o método retorna o ID correto da categoria. Neste caso, a categoria tem o ID 1 e o nome “Comida”. O teste passa se o método getId() retorna 1.

```
@Test
void setId() {
    Categoria categoria = new Categoria( id: 1, nome: "Comida");
    categoria.setId(2);
    assertEquals( expected: 2, categoria.getId());
}
```

FIGURA 83 - CATEGORIATEST (SETID())

O objetivo do teste é verificar se o método atribui corretamente o ID da categoria. Neste caso, a categoria tem o ID 1 e o nome “Comida”. O método setId() é chamado com o argumento 2. O teste passa se o método getId() retorna 2.

12.1.3 FuncionarioTest

Neste subtópico vão ser descritos os testes para a classe funcionário.

```
@Test
void getNome() {
    Funcionario funcionario = new Funcionario( id: 1, nome: "João", password: "password", idCargo: 2, idCartao: "12345", conexao: null);
    assertEquals( expected: "João", funcionario.getNome());
}
```

FIGURA 84 - FUNCIONARIOTEST (GETNOME())

O objetivo do teste é verificar se o método retorna o nome correto do funcionário. Neste caso, o funcionário tem o nome “João” e o teste passa se o método getNome() retorna “João”.

```

@Test
void setNome() {
    Funcionario funcionario = new Funcionario(1, "João", "password", 2, "12345", null);
    funcionario.setNome("Maria");
    assertEquals("Maria", funcionario.getNome());
}
    
```

FIGURA 85 - FUNCIONARIOTEST (SETNOME())

O objetivo do teste é verificar se o método atribui corretamente o nome do funcionário. Neste caso, o funcionário tem o nome “João” e o método setNome() é chamado com o argumento “Maria”. O teste passa se o método getName() retorna “Maria”.

```

@Test
void getPassword() {
    Funcionario funcionario = new Funcionario(1, "João", "password", 2, "12345", null);
    assertEquals("password", funcionario.getPassword());
}
    
```

FIGURA 86 - FUNCIONARIOTEST (GETPASSWORD())

O objetivo do teste é verificar se o método retorna a senha correta do funcionário. Neste caso, o funcionário tem a senha “password” e o teste passa se o método getPassword() retorna “password”.

```

@Test
void setPassword() {
    Funcionario funcionario = new Funcionario(1, "João", "password", 2, "12345", null);
    funcionario.setPassword("novapassword");
    assertEquals("novapassword", funcionario.getPassword());
}
    
```

FIGURA 87 - FUNCIONARIOTEST (SETPASSWORD())

O objetivo do teste é verificar se o método atribui corretamente a senha do funcionário. Neste caso, o funcionário tem a senha “password” e o método setPassword() é chamado com o argumento “novapassword”. O teste passa se o método getPassword() retorna “novapassword”.

```

@Test
void getId() {
    Funcionario funcionario = new Funcionario(1, "João", "password", 2, "12345", null);
    assertEquals(1, funcionario.getId());
}
    
```

FIGURA 88 - FUNCIONARIOTEST (GETID())

O objetivo do teste é verificar se o método retorna o ID correto do funcionário. Neste caso, o funcionário tem o nome “João” e o ID é 1. O teste passa se o método getId() retorna 1

```
@Test
void setId() {
    Funcionario funcionario = new Funcionario( id: 1, nome: "João", password: "password", idCargo: 2, idCartao: "12345", conexao: null);
    funcionario.setId(3);
    assertEquals( expected: 3, funcionario.getId());
}
```

FIGURA 89 - FUNCIONARIOTEST (SETID())

O objetivo do teste é verificar se o método atribui corretamente o ID do funcionário. Neste caso, o funcionário tem o nome “João” e o ID é definido como 2. O teste passa se o método getId() retorna 2.

```
@Test
void getIdCargo() {
    Funcionario funcionario = new Funcionario( id: 1, nome: "João", password: "password", idCargo: 2, idCartao: "12345", conexao: null);
    assertEquals( expected: 2, funcionario.getIdCargo());
}
```

FIGURA 90 - FUNCIONARIOTEST (GETIDCARGO())

O objetivo do teste é verificar se o método retorna o ID correto do cargo do funcionário. Neste caso, o funcionário é “João” e o ID do cargo é 2. O teste passa se o método getIdCargo() retorna 2.

```
@Test
void setIdCargo() {
    Funcionario funcionario = new Funcionario( id: 1, nome: "João", password: "password", idCargo: 2, idCartao: "12345", conexao: null);
    funcionario.setIdCargo(3);
    assertEquals( expected: 3, funcionario.getIdCargo());
}
```

FIGURA 91 - FUNCIONARIOTEST (SETIDCARGO())

O objetivo do teste é verificar se o método atribui corretamente o ID do cargo do funcionário. Neste caso, o funcionário é “João” e o ID do cargo é definido como 3. O teste passa se o método getIdCargo() retorna 3.

```
@Test
void getIdCartao() {
    Funcionario funcionario = new Funcionario( id: 1, nome: "João", password: "password", idCargo: 2, idCartao: "12345", conexao: null);
    assertEquals( expected: "12345", funcionario.getIdCartao());
}
```

FIGURA 92 - FUNCIONARIOTEST (GETIDCARTAO())

O objetivo do teste é verificar se o método retorna o ID correto do cartão do funcionário. Neste caso, o funcionário é “João” e o ID do cartão é “12345”. O teste passa se o método getIdCartao() retorna “12345”.

```

@Test
void setIdCartao() {
    Funcionario funcionario = new Funcionario(1, "João", "password", 2, "12345", null);
    funcionario.setIdCartao("54321");
    assertEquals("54321", funcionario.getIdCartao());
}
    
```

FIGURA 93 - FUNCIONARIOTEST (SETIDCARTAO())

O objetivo do teste é verificar se o método atribui corretamente o ID do cartão do funcionário. Neste caso, o funcionário é “João” e o ID do cartão é definido como “54321”. O teste passa se o método getIdCartao() retorna “54321”.

12.1.4 ItemPedidoTest

Nesta secção vão se fazer os testes para a classe ItemPedido.

```

@Test
void getProduto() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setProduto("Produto");
    assertEquals("Produto", itemPedido.getProduto());
}
    
```

FIGURA 94 - ITEMPEPIDOTEST (GETPRODUTO())

O objetivo do teste é verificar se o método retorna o produto correto do item de pedido. Neste caso, um novo objeto ItemPedido é criado e o método setProduto() é chamado com o argumento “Produto”. O teste passa se o método getProduto() retorna “Produto”.

```

@Test
void setProduto() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setProduto("Produto");
    assertEquals("Produto", itemPedido.productProperty().get());
}
    
```

FIGURA 95 - ITEMPEPIDOTEST (SETPRODUTO())

O objetivo do teste é verificar se o método atribui corretamente o produto do item de pedido. Neste caso, um novo objeto ItemPedido é criado e o método setProduto() é chamado com o argumento “Produto”. O teste passa se o método produtoProperty().get() retorna “Produto”.



```

@在这
void produtoProperty() {
    ItemPedido itemPedido = new ItemPedido();
    SimpleStringProperty property = itemPedido.produtoProperty();
    assertNotNull(property);
}

```

FIGURA 96 - ITEMPEPIDOTEST (PRODUTOPROPERTY())

O objetivo do teste é verificar se o método retorna uma propriedade de string simples não nula. Neste caso, um novo objeto ItemPedido é criado e o método produtoProperty() é chamado. O teste passa se o método retorna uma propriedade de string simples não nula.

```

@在这
void getQuantidade() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setQuantidade(5);
    assertEquals( expected: 5, itemPedido.getQuantidade());
}

```

FIGURA 97 - ITEMPEPIDOTEST (GETQUANTIDADE())

O objetivo do teste é verificar se o método retorna a quantidade correta do item de pedido. Neste caso, um novo objeto ItemPedido é criado e o método setQuantidade() é chamado com o argumento 5. O teste passa se o método getQuantidade() retorna 5.

```

@在这
void setQuantidade() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setQuantidade(5);
    assertEquals( expected: 5, itemPedido.quantidadeProperty().get());
}

```

FIGURA 98 - ITEMPEPIDOTEST (SETQUANTIDADADE())

Este teste verifica se o método setQuantidade() da classe ItemPedido atribui corretamente a quantidade do item de pedido. Neste caso, um novo objeto ItemPedido é criado e o método setQuantidade() é chamado com o argumento 5. O teste passa se o método quantidadeProperty().get() retorna 5.



```

@在这
void quantidadeProperty() {
    ItemPedido itemPedido = new ItemPedido();
    SimpleIntegerProperty property = itemPedido.quantidadeProperty();
    assertNotNull(property);
}

```

FIGURA 99 - ITEMPEPIDOTEST (QUANTIDADEPROPERTY())

Este teste verifica se o método `quantidadeProperty()` da classe `ItemPedido` retorna uma propriedade de inteiro simples não nula. Neste caso, um novo objeto `ItemPedido` é criado e o método `quantidadeProperty()` é chamado. O teste passa se o método retorna uma propriedade de inteiro simples não nula.

```

@在这
void getQuantidadeOriginal() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setQuantidadeOriginal(3);
    assertEquals(expected: 3, itemPedido.getQuantidadeOriginal());
}

```

FIGURA 100 - ITEMPEPIDOTEST (GETQUANTIDADEORIGINAL())

Este teste verifica se o método `getQuantidadeOriginal()` da classe `ItemPedido` retorna a quantidade original correta do item de pedido. Neste caso, um novo objeto `ItemPedido` é criado e o método `setQuantidadeOriginal()` é chamado com o argumento 3. O teste passa se o método `getQuantidadeOriginal()` retorna 3.

```

@在这
void setQuantidadeOriginal() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setQuantidadeOriginal(3);
    assertEquals(expected: 3, itemPedido.getQuantidadeOriginal());
}

```

FIGURA 101 - ITEMPEPIDOTEST (SETQUANTIDADEORIGINAL())

Este teste verifica se o método `setQuantidadeOriginal()` da classe `ItemPedido` atribui corretamente a quantidade original do item de pedido. Neste caso, um novo objeto `ItemPedido` é criado e o método `setQuantidadeOriginal()` é chamado com o argumento 3. O teste passa se o método `getQuantidadeOriginal()` retorna 3.



```

@Test
void getPreco() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setPreco(10.5);
    assertEquals( expected: 10.5, itemPedido.getPreco());
}

```

FIGURA 102 - ITEMPEPIDOTEST (GETPRECO())

Este teste verifica se o método getPreco() da classe ItemPedido retorna o preço correto do item de pedido. Neste caso, um novo objeto ItemPedido é criado e o método setPreco() é chamado com o argumento 10.5. O teste passa se o método getPreco() retorna 10.5.

```

@Test
void setPreco() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setPreco(10.5);
    assertEquals( expected: 10.5, itemPedido.precoProperty().get());
}

```

FIGURA 103 - ITEMPEPIDOTEST (SETPRECO())

Este teste verifica se o método setPreco() da classe ItemPedido atribui corretamente o preço do item de pedido. Neste caso, um novo objeto ItemPedido é criado e o método setPreco() é chamado com o argumento 10.5. O teste passa se o método precoProperty().get() retorna 10.5.

```

@Test
void precoProperty() {
    ItemPedido itemPedido = new ItemPedido();
    SimpleDoubleProperty property = itemPedido.precoProperty();
    assertNotNull(property);
}

```

FIGURA 104 - ITEMPEPIDOTEST (PRECOPROPERTY())

Este teste verifica se o método precoProperty() da classe ItemPedido retorna uma propriedade de ponto flutuante simples não nula. Neste caso, um novo objeto ItemPedido é criado e o método precoProperty() é chamado. O teste passa se o método retorna uma propriedade simples não nula.



```

@Test
void getIdItem() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setIdItem(1);
    assertEquals( expected: 1, itemPedido.getIdItem());
}

```

FIGURA 105 - ITEMPEPIDOTEST (GETIDITEM())

Este teste verifica se o método getIdItem() da classe ItemPedido retorna o ID correto do item de pedido. Neste caso, um novo objeto ItemPedido é criado e o método setIdItem() é chamado com o argumento 1. O teste passa se o método getIdItem() retorna 1.

```

@Test
void setIdItem() {
    ItemPedido itemPedido = new ItemPedido();
    itemPedido.setIdItem(1);
    assertEquals( expected: 1, itemPedido.idItemProperty().get());
}

```

FIGURA 106 - ITEMPEPIDOTEST (SETIDITEM())

Este teste verifica se o método setIdItem() da classe ItemPedido atribui corretamente o ID do item de pedido. Neste caso, um novo objeto ItemPedido é criado e o método setIdItem() é chamado com o argumento 1. O teste passa se o método idItemProperty().get() retorna 1.

12.1.5 ItemTest

Neste subtópico serão descritos os testes para a classe item da aplicação.

```

@Test
void getPreco() {
    Item item = new Item( id: 1, nome: "lasanha", preco: 10.99, idCategoria: 1, conexao: null);
    assertEquals( expected: 10.99, item.getPreco(), delta: 0.01);
}

```

FIGURA 107 - ITEMTEST (GETPRECO())

Verifica se a função getPreco() da classe Item retorna o preço correto do item. Ele cria um objeto Item com um preço de 10,99 e verifica se a função getPreco() retorna 10,99.

```

@Test
void setPreco() {
    Item item = new Item( id: 1, nome: "lasanha", preco: 10.99, idCategoria: 1, conexao: null);
    item.setPreco(15.99);
    assertEquals( expected: 15.99, item.getPreco(), delta: 0.01);
}
    
```

FIGURA 108 - ITEMTEST (SETPRECO())

Verifica se a função setPreco() da classe Item atualiza corretamente o preço do item. Ele cria um objeto Item com um preço de 10,99, chama a função setPreco() para atualizar o preço para 15,99 e verifica se a função getPreco() retorna 15,99.

```

@Test
void getName() {
    Item item = new Item( id: 1, nome: "lasanha", preco: 10.99, idCategoria: 1, conexao: null);
    assertEquals( expected: "lasanha", item.getName());
}
    
```

FIGURA 109 - ITEMTEST (GETNAME())

Verifica se a função getName() da classe Item retorna o nome correto do item. Ele cria um objeto Item com o nome “lasanha” e verifica se a função getName() retorna “lasanha”.

```

@Test
void setName() {
    Item item = new Item( id: 1, nome: "lasanha", preco: 10.99, idCategoria: 1, conexao: null);
    item.setName("Feijoada");
    assertEquals( expected: "Feijoada", item.getName());
}
    
```

FIGURA 110 - ITEMTEST (SETNAME())

Verifica se a função setName() da classe Item atualiza corretamente o nome do item. Ele cria um objeto Item com o nome “lasanha”, chama a função setName() para atualizar o nome para “Feijoada” e verifica se a função getName() retorna “Feijoada”.

```

@Test
void getId() {
    Item item = new Item( id: 1, nome: "Feijoada", preco: 10.99, idCategoria: 1, conexao: null);
    assertEquals( expected: 1, item.getId());
}
    
```

FIGURA 111 - ITEMTEST (GETID())

Verifica se a função getId() da classe Item retorna o ID correto do item. Ele cria um objeto Item com o ID 1 e verifica se a função getId() retorna 1.

```

@Test
void setId() {
    Item item = new Item( id: 1, nome: "Feijoada", preco: 10.99, idCategoria: 1, conexao: null);
    item.setId(2);
    assertEquals( expected: 2, item.getId());
}
    
```

FIGURA 112 - ITEMTEST (SETID())

Verifica se a função setId() da classe Item atualiza corretamente o ID do item. Ele cria um objeto Item com o ID 1, chama a função setId() para atualizar o ID para 2 e verifica se a função getId() retorna 2.

```

@Test
void getIdCategoria() {
    Item item = new Item( id: 1, nome: "Feijoada", preco: 10.99, idCategoria: 1, conexao: null);
    assertEquals( expected: 1, item.getIdCategoria());
}
    
```

FIGURA 113 - ITEMTEST (GETIDCATEGORIA())

Verifica se a função getIdCategoria() da classe Item retorna o ID da categoria correta do item. Ele cria um objeto Item com o ID da categoria 1 e verifica se a função getIdCategoria() retorna 1.

```
@Test
void setIdCategoria() {
    Item item = new Item( id: 1, nome: "Feijoada", preco: 10.99, idCategoria: 1, conexao: null);
    item.setIdCategoria(2);
    assertEquals( expected: 2, item.getIdCategoria());
}
```

FIGURA 114 - ITEMTEST (SETIDCATEGORIA())

Verifica se a função setIdCategoria() da classe Item atualiza corretamente o ID da categoria do item. Ele cria um objeto Item com o ID da categoria 1, chama a função setIdCategoria() para atualizar o ID da categoria para 2 e verifica se a função getIdCategoria() retorna 2.

12.1.6 MesaTest

Neste subtópico vão ser descritos os testes para a classe Mesa.

```
@Test
void editarStatus() {
    //Cria uma instância de Mesa
    Mesa mesa = new Mesa( id: 1, status: "Disponivel");

    //Chama o método editarStatus para alterar o status
    mesa.editarStatus( novoStatus: "Ocupada");

    //Verifica se o status foi alterado corretamente
    assertEquals( expected: "Ocupada", mesa.getStatus());
}
```

FIGURA 115 - MESATEST (EDITARSTATUS())

Verifica se a função editarStatus() da classe Mesa altera corretamente o status da mesa. Ele cria uma instância de Mesa com o status “Disponível”, chama a função editarStatus() para alterar o status para “Ocupada” e verifica se a função getStatus() retorna “Ocupada”.

```
@Test
void getStatus() {
    //Cria uma instância de Mesa
    Mesa mesa = new Mesa( id: 2, status: "Reservada");

    //Verifica se o método getStatus retorna o status esperado
    assertEquals( expected: "Reservada", mesa.getStatus());
}
```

FIGURA 116 - MESATEST (GETSTATUS())

Verifica se a função getStatus() da classe Mesa retorna o status correto da mesa. Ele cria uma instância de Mesa com o status “Reservada” e verifica se a função getStatus() retorna “Reservada”.



```

@Test
void getId() {
    //Cria uma instância de Mesa
    Mesa mesa = new Mesa(id: 3, status: "Disponível");

    //Verifica se o método getId retorna o id esperado
    assertEquals(expected: 3, mesa.getId());
}

```

FIGURA 117 - MESATEST (GETID())

Verifica se a função getId() da classe Mesa retorna o ID correto da mesa. Ele cria uma instância de Mesa com o ID 3 e verifica se a função getId() retorna 3.

```

@Test
void setId() {
    //Cria uma instância de Mesa
    Mesa mesa = new Mesa();

    //Chama o método setId para definir o id
    mesa.setId(5);

    //Verifica se o método getId retorna o id definido
    assertEquals(expected: 5, mesa.getId());
}

```

FIGURA 118 - MESATEST (SETID())

Verifica se a função setId() da classe Mesa atualiza corretamente o ID da mesa. Ele cria uma instância de Mesa, chama a função setId() para definir o ID como 5 e verifica se a função getId() retorna 5.

```

@Test
void getId_zona() {
    //Cria uma instância de Mesa
    Mesa mesa = new Mesa(id: 6, status: "Disponível", idZona: 1);

    //Verifica se o método getId_zona retorna o id_zona esperado
    assertEquals(expected: 1, mesa.getId_zona());
}

```

FIGURA 119 - MESATEST (GETID_ZONA())

Verifica se a função getId_zona() da classe Mesa retorna o ID da zona correto da mesa. Ele cria uma instância de Mesa com o ID da zona 1 e verifica se a função getId_zona() retorna 1.



```
@Test
void setId_zona() {
    //Cria uma instância de Mesa
    Mesa mesa = new Mesa();

    //Chama o método setId_zona para definir o id_zona
    mesa.setId_zona(2);

    //Verifica se o método getId_zona retorna o id_zona definido
    assertEquals(expected: 2, mesa.getId_zona());
}
```

FIGURA 120 - MESATEST (SETID_ZONA())

Verifica se a função setId_zona() da classe Mesa atualiza corretamente o ID da zona da mesa. Ele cria uma instância de Mesa, chama a função setId_zona() para definir o ID da zona como 2 e verifica se a função getId_zona() retorna 2.

12.1.7 PagamentoAtendimentoTest

```
1  @Test
2  void getId() {
3      PagamentoAtendimento pagamento = new PagamentoAtendimento();
4      pagamento.setId(1L);
5      assertEquals(1L, pagamento.getId());
6  }
```

FIGURA 121 - PAGAMENTOATENDIMENTOTEST (GETID())

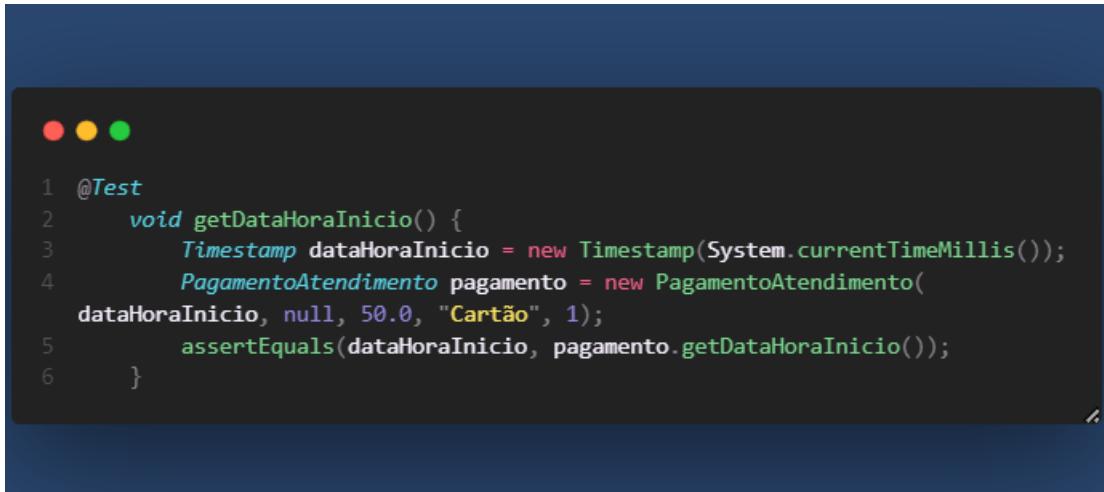
Verifica se a função getId() da classe PagamentoAtendimento retorna o ID correto do pagamento. Ele cria uma instância de PagamentoAtendimento, chama a função setId() para definir o ID como 1L e verifica se a função getId() retorna 1L.



```
1 @Test
2     void setId() {
3         PagamentoAtendimento pagamento = new PagamentoAtendimento();
4         pagamento.setId(1L);
5         assertEquals(1L, pagamento.getId());
6     }
```

FIGURA 122 - PAGAMENTOATENDIMENTOTEST (SETID())

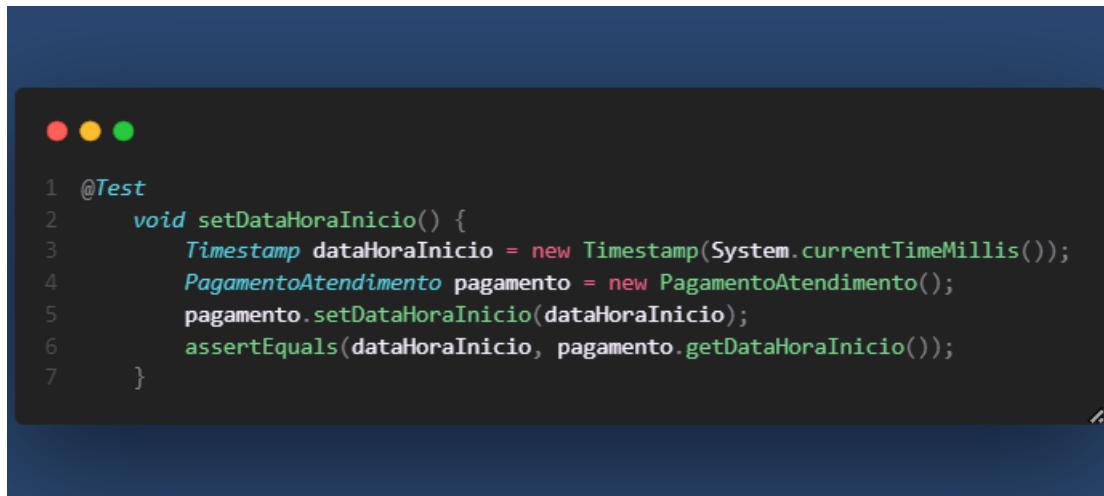
Verifica se a função setId() da classe PagamentoAtendimento atualiza corretamente o ID do pagamento. Ele cria uma instância de PagamentoAtendimento, chama a função setId() para definir o ID como 1L e verifica se a função getId() retorna 1L.



```
1 @Test
2     void getDataHoraInicio() {
3         Timestamp dataHoraInicio = new Timestamp(System.currentTimeMillis());
4         PagamentoAtendimento pagamento = new PagamentoAtendimento(
5             dataHoraInicio, null, 50.0, "Cartão", 1);
6         assertEquals(dataHoraInicio, pagamento.getDataHoraInicio());
7     }
```

FIGURA 123 - PAGAMENTOATENDIMENTOTEST (GETDATAHORAINICIO())

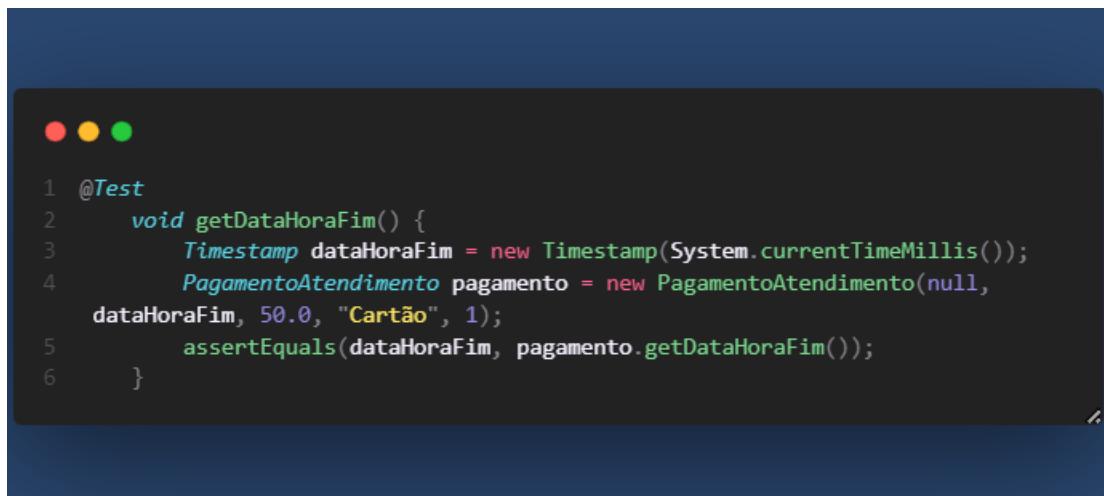
Verifica se a função getDataHoraInicio() da classe PagamentoAtendimento retorna a data e hora de início corretas do pagamento. Ele cria um objeto Timestamp com a data e hora atuais, cria uma instância de PagamentoAtendimento com a data e hora de início definidas como o objeto Timestamp, e verifica se a função getDataHoraInicio() retorna o objeto Timestamp.



```
1 @Test
2     void setDataHoraInicio() {
3         Timestamp dataHoraInicio = new Timestamp(System.currentTimeMillis());
4         PagamentoAtendimento pagamento = new PagamentoAtendimento();
5         pagamento.setDataHoraInicio(dataHoraInicio);
6         assertEquals(dataHoraInicio, pagamento.getDataHoraInicio());
7     }
```

FIGURA 124 - PAGAMENTOATENDIMENTOTEST (SETDATAHORAINICIO())

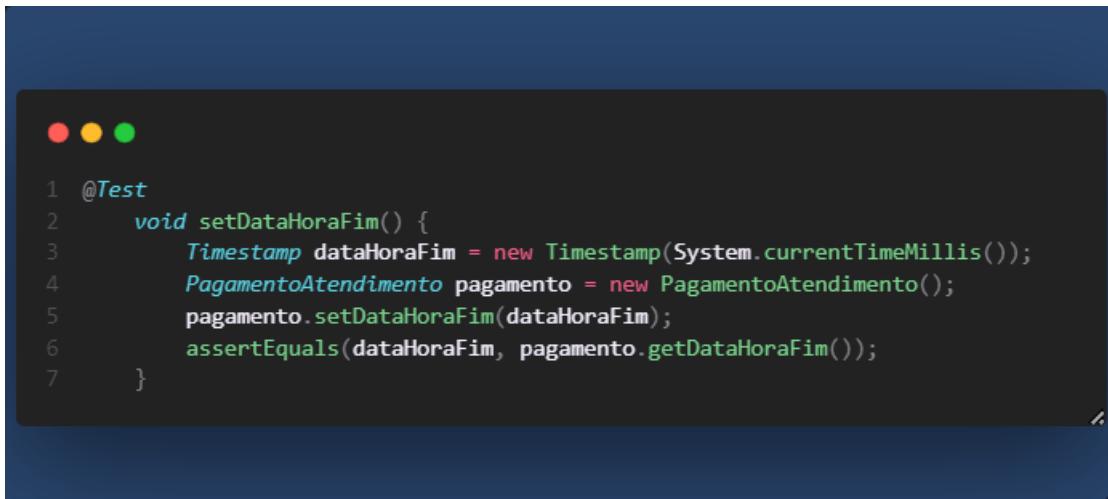
Verifica se a função setDataHoraInicio() da classe PagamentoAtendimento atualiza corretamente a data e hora de início do pagamento. Ele cria um objeto Timestamp com a data e hora atuais, cria uma instância de PagamentoAtendimento, chama a função setDataHoraInicio() para definir a data e hora de início como o objeto Timestamp, e verifica se a função getDataHoraInicio() retorna o objeto Timestamp.



```
1 @Test
2     void getDataHoraFim() {
3         Timestamp dataHoraFim = new Timestamp(System.currentTimeMillis());
4         PagamentoAtendimento pagamento = new PagamentoAtendimento(null,
5             dataHoraFim, 50.0, "Cartão", 1);
6         assertEquals(dataHoraFim, pagamento.getDataHoraFim());
7     }
```

FIGURA 125 - PAGAMENTOATENDIMENTOTEST (GETDATAHORAFIM())

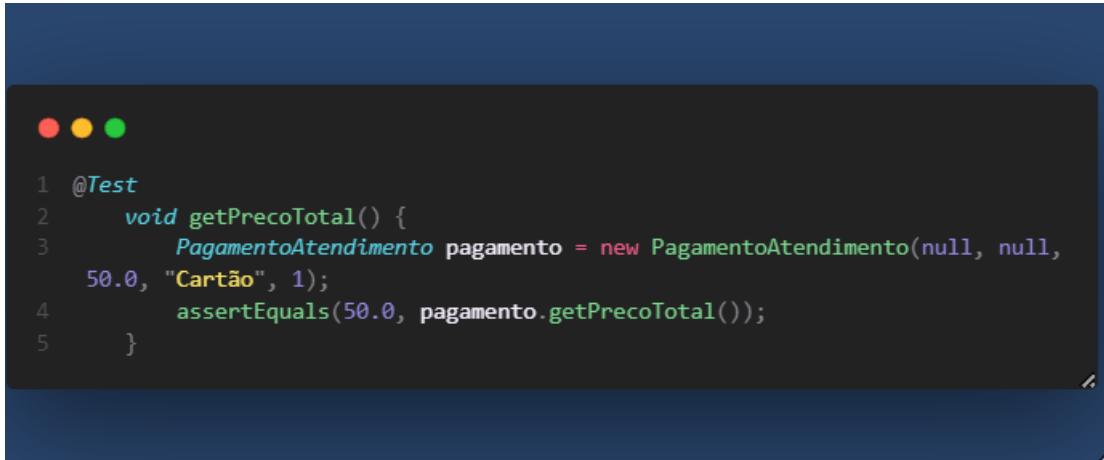
Verifica se a função getDataHoraFim() da classe PagamentoAtendimento retorna a data e hora de término corretas do pagamento. Ele cria um objeto Timestamp com a data e hora atuais, cria uma instância de PagamentoAtendimento com a data e hora de término definidas como o objeto Timestamp, e verifica se a função getDataHoraFim() retorna o objeto Timestamp.



```
1 @Test
2     void setDataHoraFim() {
3         Timestamp dataHoraFim = new Timestamp(System.currentTimeMillis());
4         PagamentoAtendimento pagamento = new PagamentoAtendimento();
5         pagamento.setDataHoraFim(dataHoraFim);
6         assertEquals(dataHoraFim, pagamento.getDataHoraFim());
7     }
```

FIGURA 126 - PAGAMENTOATENDIMENTOTEST (SETDATAHORAFIM())

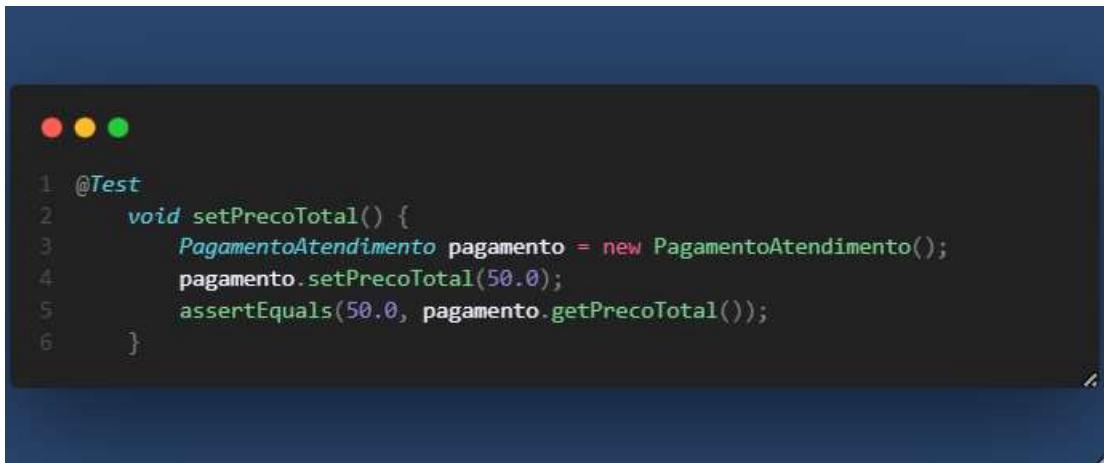
Verifica se a função setDataHoraFim() da classe PagamentoAtendimento atualiza corretamente a data e hora de término do pagamento. Ele cria um objeto Timestamp com a data e hora atuais, cria uma instância de PagamentoAtendimento, chama a função setDataHoraFim() para definir a data e hora de término como o objeto Timestamp, e verifica se a função getDataHoraFim() retorna o objeto Timestamp.



```
1 @Test
2     void getPrecoTotal() {
3         PagamentoAtendimento pagamento = new PagamentoAtendimento(null, null,
4             50.0, "Cartão", 1);
5         assertEquals(50.0, pagamento.getPrecoTotal());
6     }
```

FIGURA 127 - PAGAMENTOATENDIMENTOTEST (GETPRECOTOTAL())

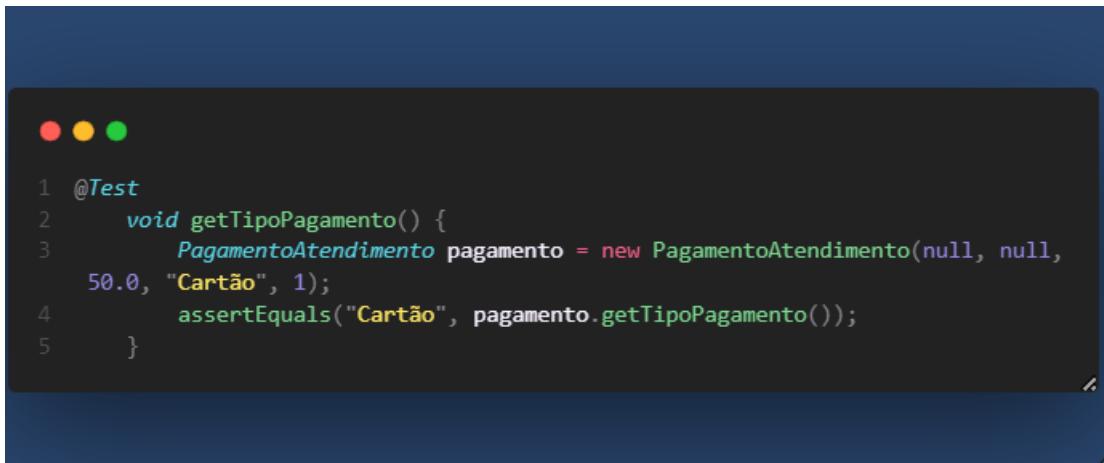
Verifica se a função getPrecoTotal() da classe PagamentoAtendimento retorna o preço total correto do pagamento. Ele cria uma instância de PagamentoAtendimento com o preço total definido como 50.0 e verifica se a função getPrecoTotal() retorna 50.0.



```
1 @Test
2     void setPrecoTotal() {
3         PagamentoAtendimento pagamento = new PagamentoAtendimento();
4         pagamento.setPrecoTotal(50.0);
5         assertEquals(50.0, pagamento.getPrecoTotal());
6     }
```

FIGURA 128 - PAGAMENTOATENDIMENTOTEST (SETPRECOTOTAL())

Verifica se a função setPrecoTotal() da classe PagamentoAtendimento atualiza corretamente o preço total do pagamento. Ele cria uma instância de PagamentoAtendimento, chama a função setPrecoTotal() para definir o preço total como 50.0 e verifica se a função getPrecoTotal() retorna 50.0.



```
1 @Test
2     void getTipoPagamento() {
3         PagamentoAtendimento pagamento = new PagamentoAtendimento(null, null,
4             50.0, "Cartão", 1);
5         assertEquals("Cartão", pagamento.getTipoPagamento());
6     }
```

FIGURA 129 - PAGAMENTOATENDIMENTOTEST (GETTIPOPAGAMENTO())

Verifica se a função getTipoPagamento() da classe PagamentoAtendimento retorna o tipo de pagamento correto. Ele cria uma instância de PagamentoAtendimento com o tipo de pagamento “Cartão” e verifica se a função getTipoPagamento() retorna “Cartão”.



```
1 @Test
2     void setTipoPagamento() {
3         PagamentoAtendimento pagamento = new PagamentoAtendimento();
4         pagamento.setTipoPagamento("Dinheiro");
5         assertEquals("Dinheiro", pagamento.getTipoPagamento());
6     }
```

FIGURA 130 - PAGAMENTOATENDIMENTOTEST (SETTIOPAGAMENTO())

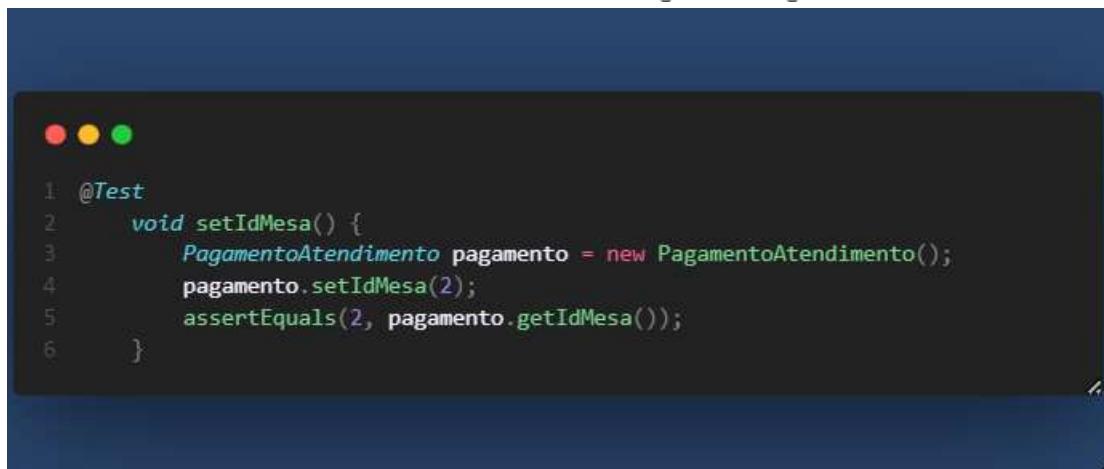
Verifica se a função setTipoPagamento() da classe PagamentoAtendimento atualiza corretamente o tipo de pagamento. Ele cria uma instância de PagamentoAtendimento, chama a função setTipoPagamento() para definir o tipo de pagamento como “Dinheiro” e verifica se a função getTipoPagamento() retorna “Dinheiro”.



```
1 @Test
2     void getIdMesa() {
3         PagamentoAtendimento pagamento = new PagamentoAtendimento(null, null,
4             50.0, "Cartão", 1);
5         assertEquals(1, pagamento.getIdMesa());
6     }
```

FIGURA 131 - PAGAMENTOATENDIMENTOTEST (GETIDMESA())

Verifica se a função getIdMesa() da classe PagamentoAtendimento retorna o ID da mesa correto. Ele cria uma instância de PagamentoAtendimento com o ID da mesa definido como 1 e verifica se a função getIdMesa() retorna 1.



```
1 @Test
2 void setIdMesa() {
3     PagamentoAtendimento pagamento = new PagamentoAtendimento();
4     pagamento.setIdMesa(2);
5     assertEquals(2, pagamento.getIdMesa());
6 }
```

FIGURA 132 - PAGAMENTOATENDIMENTOTEST (SETIDMESA())

Verifica se a função setIdMesa() da classe PagamentoAtendimento atualiza corretamente o ID da mesa. Ele cria uma instância de PagamentoAtendimento, chama a função setIdMesa() para definir o ID da mesa como 2 e verifica se a função getIdMesa() retorna 2.

12.1.8 PedidoTest

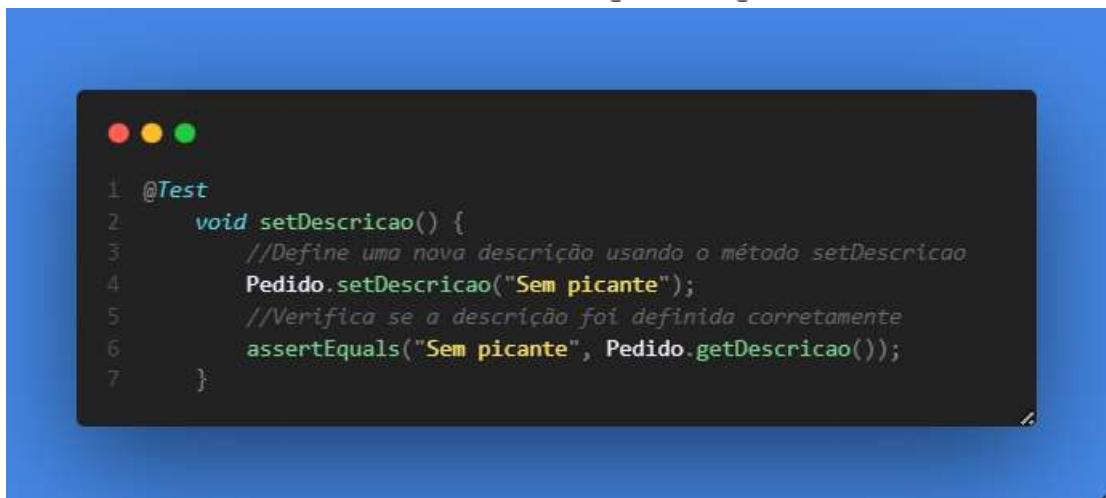
Este tópico tem como propósito apresentar e ilustrar os testes realizados para a classe Pedido.



```
1 @Test
2 void pedido() {
3
4     //Ele apenas chama o método setId e setDescricao após a criação da
5     //instância.
6     Pedido pedido = new Pedido();
7
8     //Define um ID e uma descrição após a criação da instância
9     Pedido.setId(2);
10    Pedido.setDescricao("Teste");
11
12    //Verifica se o ID e a descrição foram configurados corretamente
13    assertEquals(2, Pedido.getId());
14    assertEquals("Teste", Pedido.getDescricao());
15 }
```

FIGURA 133 - PEDIDOTEST (PEDIDO())

Verifica se as funções setId() e setDescricao() da classe Pedido atualizam corretamente o ID e a descrição do pedido. Ele cria uma instância de Pedido, chama a função setId() para definir o ID como 2, chama a função setDescricao() para definir a descrição como “Teste” e verifica se as funções getId() e getDescricao() retornam 2 e “Teste”, respectivamente.



```
1 @Test
2     void setDescricao() {
3         //Define uma nova descrição usando o método setDescricao
4         Pedido.setDescricao("Sem picante");
5         //Verifica se a descrição foi definida corretamente
6         assertEquals("Sem picante", Pedido.getDescricao());
7     }
```

FIGURA 134 - PEDIDOTEST (SETDESCRICAO())

Verifica se a função setDescricao() da classe Pedido atualiza corretamente a descrição do pedido. Ele chama a função setDescricao() para definir a descrição como “Sem picante” e verifica se a função getDescricao() retorna “Sem picante”.



```
1 @Test
2     void getDescricao() {
3         //Define uma nova descrição usando o método setDescricao
4         Pedido.setDescricao("Com picante");
5         //Verifica se o método getDescricao retorna a descrição correta
6         assertEquals("Com picante", Pedido.getDescricao());
7     }
```

FIGURA 135 - PEDIDOTEST (GETDESCRICAO())

Verifica se a função getDescricao() da classe Pedido retorna a descrição correta do pedido. Ele chama a função setDescricao() para definir a descrição como “Com picante” e verifica se a função getDescricao() retorna “Com picante”.



```
1 @Test
2     void getId() {
3         //Define um novo ID usando o método setId
4         Pedido.setId(3);
5         //Verifica se o método getId retorna o ID corretamente
6         assertEquals(3, Pedido.getId());
7     }
```

FIGURA 136 - PEDIDOTEST (GETID())

Verifica se a função getId() da classe Pedido retorna o ID correto do pedido. Ele chama a função setId() para definir o ID como 3 e verifica se a função getId() retorna 3.

```
1 @Test
2     void setId() {
3         //Define um novo ID usando o método setId
4         Pedido.setId(1);
5         //Verifica se o ID foi definido corretamente
6         assertEquals(1, Pedido.getId());
7     }
```

FIGURA 137 - PEDIDOTEST(SETID())

Verifica se a função setId() da classe Pedido atualiza corretamente o ID do pedido. Ele chama a função setId() para definir o ID como 1 e verifica se a função getId() retorna 1.

12.1.9 SerialCommunicationServiceTest

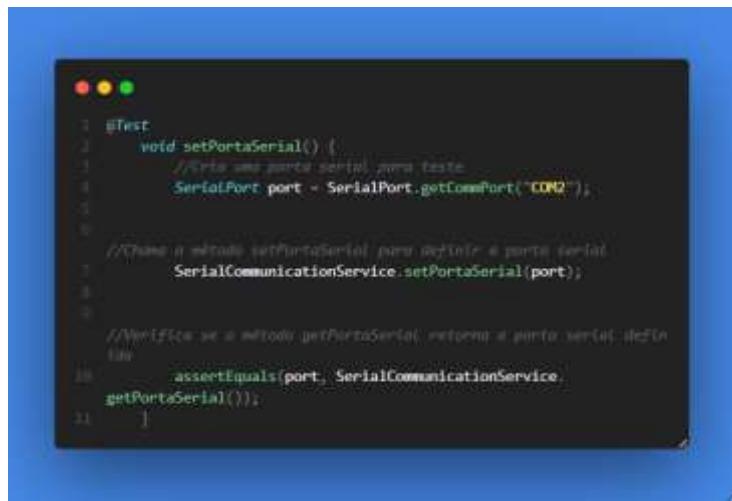
Nesta secção serão descritos os testes para a classe SerialCommunicationService.



```
1 @Test
2     void getPortaSerial() {
3
4         //Verifica se o método getPortaSerial retorna a porta serial correta
5         //criando uma porta serial para teste
6         SerialPort port = SerialCommunicationService.getPortaSerial();
7
8         assertNull(port);
9         //A princípio, a porta serial deve ser nula.
10    }
```

FIGURA 138 - SERIALCOMMUNICATIONSERVICETEST (GETPORTASERIAL())

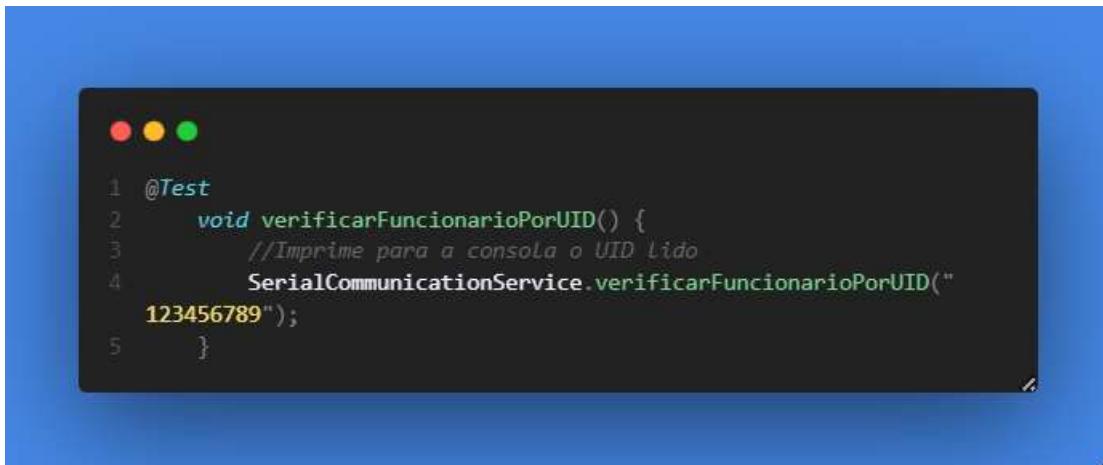
Verifica se a função getPortaSerial() da classe SerialCommunicationService retorna a porta serial correta. Ele chama a função getPortaSerial() e verifica se ela retorna null.



```
1 @Test
2     void setPortaSerial() {
3
4         //Cria uma porta serial para teste
5         SerialPort port = SerialPort.getComPort("COM2");
6
7
8         //Chama o método setPortaSerial para definir a porta serial
9         SerialCommunicationService.setPortaSerial(port);
10
11
12         //Verifica se o método getPortaSerial retorna a porta serial definida
13         assertEquals(port, SerialCommunicationService.getPortaSerial());
14     }
15 }
```

FIGURA 139 - SERIALCOMMUNICATIONSERVICETEST (SETPORTASERIAL())

Verifica se a função setUltimoUID() da classe SerialCommunicationService atualiza corretamente o último UID. Ele chama a função setUltimoUID() para definir o último UID como “987654321” e verifica se a função getUltimoUID() retorna “987654321”.



```
1 @Test
2     void verificarFuncionarioPorUID() {
3         //Imprime para a consola o UID lido
4         SerialCommunicationService.verificarFuncionarioPorUID("123456789");
5     }
```

FIGURA 140 - SERIALCOMMUNICATIONSERVICETEST (VERIFICARFUNCIONARIOPORUID())

Imprime para a consola o UID lido. Esta função não retorna nenhum valor.



```
1 @Test
2     void getUltimoUID() {
3
4         //Verifica se a método getUltimoUID retorna o último UID definido
5         SerialCommunicationService.setUltimoUID("123456789");
6         assertEquals("123456789", SerialCommunicationService.getUltimoUID());
7     }
```

FIGURA 141 - SERIALCOMMUNICATIONSERVICETEST (GETULTIMOUID())

Verifica se a função getUltimoUID() da classe SerialCommunicationService retorna o último UID definido. Ele chama a função setUltimoUID() para definir o último UID como “123456789”, chama a função getUltimoUID() e verifica se ela retorna “123456789”.



```
1 @Test
2     void setUltimoUID() {
3         //Chama o método setUltimoUID para definir o último UID
4         SerialCommunicationService.setUltimoUID("987654321");
5
6         //Verifica se o método getLastimoUID retorna o último UID definido
7         assertEquals("987654321", SerialCommunicationService.
8             getLastimoUID());
```

FIGURA 142 - SERIALCOMMUNICATIONSERVICETEST (SETULTIMOUID())

Verifica se a função setUltimoUID() da classe SerialCommunicationService atualiza corretamente o último UID. Ele chama a função setUltimoUID() para definir o último UID como “987654321” e verifica se a função getLastimoUID() retorna “987654321”.

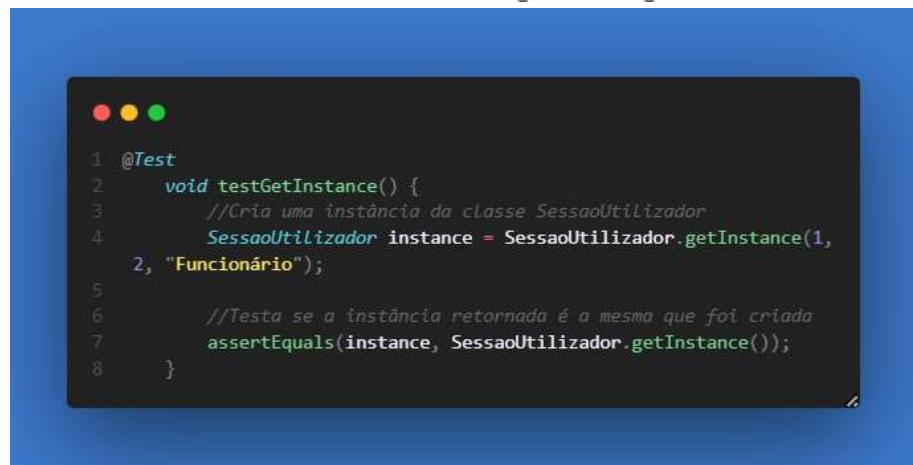
12.1.10 SessaoUtilizadorTest

Este tópico tem como objetivo demonstrar e ilustrar os testes realizados para a classe SessaoUtilizador.

```
1 @Test
2     void getInstance() {
3         //Testa se a instância retornada não é nula
4         assertNotNull(SessaoUtilizador.getInstance());
5     }
```

FIGURA 143 - SESSAOUTILIZADORTEST (GETINSTANCE())

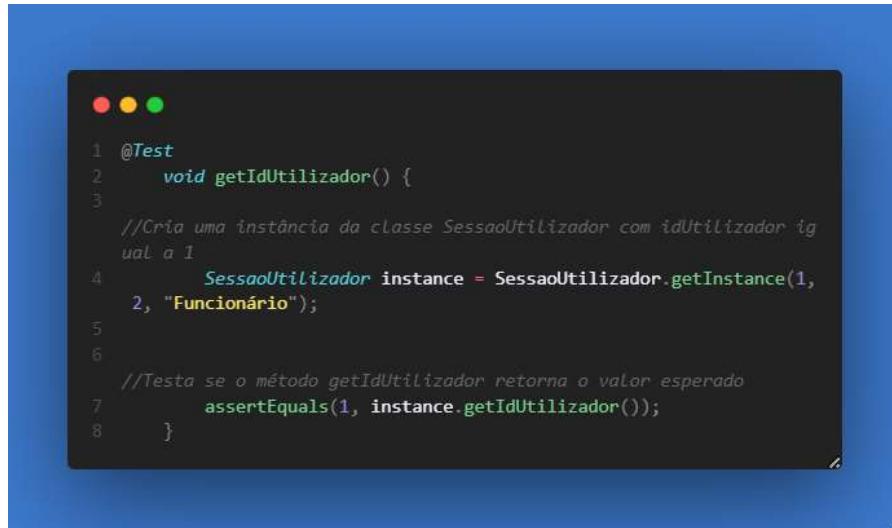
Verifica se a função getInstance() da classe SessaoUtilizador retorna uma instância não nula da classe. Ele chama a função getInstance() e verifica se ela retorna uma instância não nula da classe SessaoUtilizador.



```
1 @Test
2     void testGetInstance() {
3         //Cria uma instância da classe SessaoUtilizador
4         SessaoUtilizador instance = SessaoUtilizador.getInstance(1,
5             2, "Funcionário");
6         //Testa se a instância retornada é a mesma que foi criada
7         assertEquals(instance, SessaoUtilizador.getInstance());
8     }
```

FIGURA 144 - SESSAOUTILIZADORTEST (TESTGETINSTANCE())

Verifica se a função getInstance() da classe SessaoUtilizador retorna a mesma instância criada anteriormente. Ele cria uma instância de SessaoUtilizador com o ID do utilizador igual a 1, o cargo igual a 2 e o nome do funcionário igual a “Funcionário”, chama a função getInstance() duas vezes e verifica se as duas chamadas retornam a mesma instância de SessaoUtilizador.



```
1 @Test
2     void getIdUtilizador() {
3
4         //Cria uma instância da classe SessaoUtilizador com idUtilizador igual a 1
5         SessaoUtilizador instance = SessaoUtilizador.getInstance(1,
6             2, "Funcionário");
7
8         //Testa se o método getIdUtilizador retorna o valor esperado
9         assertEquals(1, instance.getIdUtilizador());
10    }
```

FIGURA 145 - SESSAOUTILIZADORTEST (GETIDUTILIZADOR())

Verifica se a função getIdUtilizador() da classe SessaoUtilizador retorna o ID do utilizador correto. Ele cria uma instância de SessaoUtilizador com o ID do utilizador igual a 1, chama a função getIdUtilizador() e verifica se ela retorna 1.



```
● ● ●  
1 @Test  
2     void getCargo() {  
3  
    //Cria uma instância da classe SessaoUtilizador com cargo igual a 2  
4     SessaoUtilizador instance = SessaoUtilizador.getInstance(1,  
5         2, "Funcionário");  
6  
    //Testa se o método getCargo retorna o valor esperado  
7     assertEquals(2, instance.getCargo());  
8 }
```

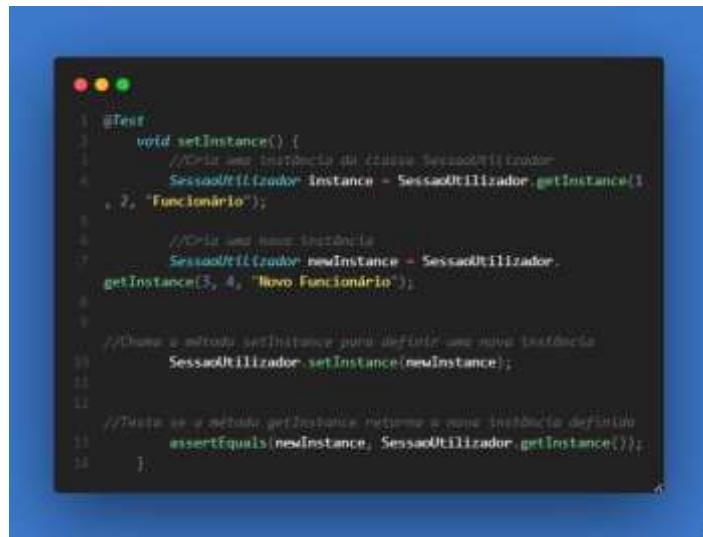
FIGURA 146 - SESSAOUTILIZADORTEST (GETCARGO())

Verifica se a função getCargo() da classe SessaoUtilizador retorna o cargo correto. Ele cria uma instância de SessaoUtilizador com o cargo igual a 2, chama a função getCargo() e verifica se ela retorna 2.

```
● ● ●  
1 @Test  
2     void getNomeFuncionario() {  
3  
    //Cria uma instância da classe SessaoUtilizador com nomeFuncionario  
    //igual a "Funcionário"  
4     SessaoUtilizador instance = SessaoUtilizador.getInstance(1,  
5         2, "Funcionário");  
6  
    //Testa se o método getNomeFuncionario retorna o valor esperado  
7     assertEquals("Funcionário", instance.getNomeFuncionario());  
8 }
```

FIGURA 147 - SESSAOUTILIZADORTEST (GETNOMEFUNCIONARIO())

Verifica se a função getNomeFuncionario() da classe SessaoUtilizador retorna o nome do funcionário correto. Ele cria uma instância de SessaoUtilizador com o nome do funcionário igual a “Funcionário”, chama a função getNomeFuncionario() e verifica se ela retorna “Funcionário”.



```

@Test
void setInstance() {
    //Cria uma instância da classe SessaoUtilizador
    SessaoUtilizador instance = SessaoUtilizador.getInstance(1,
    2, "Funcionário");

    //Cria uma nova instância
    SessaoUtilizador newInstance = SessaoUtilizador.getInstance(3, 4, "Novo Funcionário");

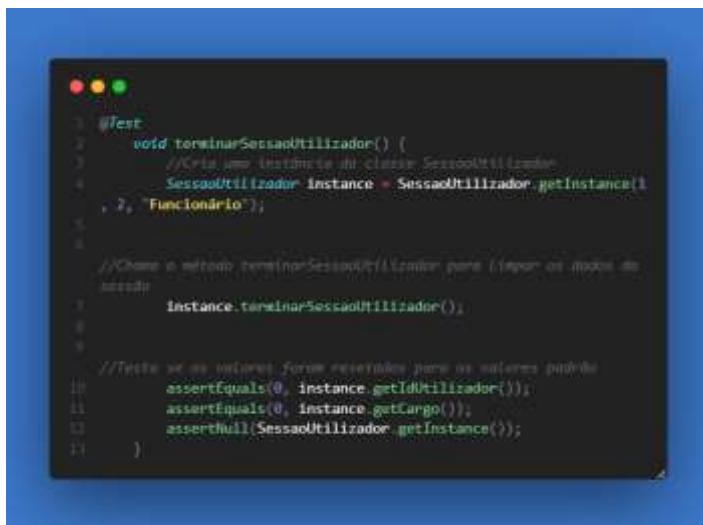
    //Chama o método setInstance para definir uma nova instância
    SessaoUtilizador.setInstance(newInstance);

    //Testa se o método getInstance retorna a nova instância definida
    assertEquals(newInstance, SessaoUtilizador.getInstance());
}

```

FIGURA 148 - SESSAOUTILIZADORTEST (SETINSTANCE())

O teste cria uma instância da classe SessaoUtilizador com o ID do utilizador igual a 1, o cargo igual a 2 e o nome do funcionário igual a “Funcionário”. Em seguida, ele cria uma instância de SessaoUtilizador com o ID do utilizador igual a 3, o cargo igual a 4 e o nome do funcionário igual a “Novo Funcionário”. O teste chama o método setInstance() para definir a nova instância criada como a instância atual da classe SessaoUtilizador. Por fim, ele verifica se o método getInstance() retorna a nova instância definida.



```

@Test
void terminarSessaoUtilizador() {
    //Cria uma instância da classe SessaoUtilizador
    SessaoUtilizador instance = SessaoUtilizador.getInstance(2,
    2, "Funcionário");

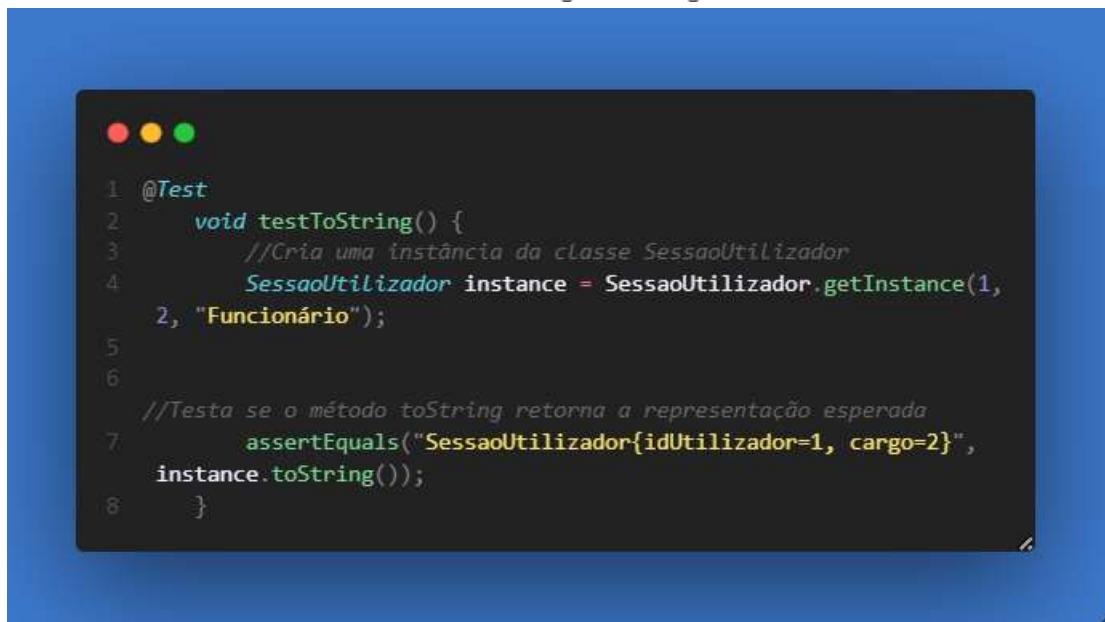
    //Chama o método terminarSessaoUtilizador para limpar os dados da sessão
    instance.terminarSessaoUtilizador();

    //Testa se os valores foram resetados para os valores padrão
    assertEquals(0, instance.getIdUtilizador());
    assertEquals(0, instance.getCargo());
    assertEquals(null, SessaoUtilizador.getInstance());
}

```

FIGURA 149 - SESSAOUTILIZADORTEST (TERMINARSESSAOUTILIZADOR())

Verifica se a função terminarSessaoUtilizador() da classe SessaoUtilizador limpa corretamente os dados da sessão. Ele cria uma instância de SessaoUtilizador, chama a função terminarSessaoUtilizador() para limpar os dados da sessão e verifica se as funções getIdUtilizador() e getCargo() retornam 0 e se a função getInstance() retorna null.



```
1 @Test
2 void testToString() {
3     //Cria uma instância da classe SessaoUtilizador
4     SessaoUtilizador instance = SessaoUtilizador.getInstance(1,
5         2, "Funcionário");
6
7     //Testa se o método toString retorna a representação esperada
8     assertEquals("SessaoUtilizador{idUtilizador=1, cargo=2}",
9         instance.toString());
10 }
```

FIGURA 150 - SESSAOUTILIZADORTEST (TESTTOSTRING())

Verifica se a função `toString()` da classe `SessaoUtilizador` retorna a representação correta da instância. Ele cria uma instância de `SessaoUtilizador` com o ID do utilizador igual a 1, o cargo igual a 2 e o nome do funcionário igual a “Funcionário”, chama a função `toString()` e verifica se ela retorna “`SessaoUtilizador{idUtilizador=1, cargo=2}`”.

12.1.11 ZonaTest

Nesta seção, será abordada uma análise abrangente dos testes realizados para a classe `Zona`.



```
1 @Test
2 void testToString() {
3
4     //Cria uma instância da classe Zona com (id 1 e nome "Esplanada"
5     Zona zona = new Zona(1, "Esplanada");
6
7     //Verifica se o método toString retorna o nome esperado
8     assertEquals("Esplanada", zona.toString());
9 }
```

FIGURA 151 - ZONATEST (TESTTOSTRING())

Verifica se a função `toString()` da classe `Zona` retorna o nome correto da zona. Ele cria uma instância de `Zona` com o ID igual a 1 e o nome “Esplanada”, chama a função `toString()` e verifica se ela retorna “Esplanada”.



```
1  @Test
2      void getName() {
3
4          //Cria uma instância da classe Zona com id 1 e nome "Esplanada"
5          Zona zona = new Zona(1, "Esplanada");
6
7          //Verifica se o método getName retorna o nome esperado
8          assertEquals("Esplanada", zona.getName());
9      }
10 }
```

FIGURA 152 - ZONATEST (GETNOME())

Verifica se a função getName() da classe Zona retorna o nome correto da zona. Ele cria uma instância de Zona com o ID igual a 1 e o nome “Esplanada”, chama a função getName() e verifica se ela retorna “Esplanada”.

```
1  @Test
2      void setName() {
3
4          //Cria uma instância da classe Zona com id 1 e nome "Esplanada"
5          Zona zona = new Zona(1, "Esplanada");
6
7          //Chama o método setName para alterar o nome
8          zona.setName("Salão");
9
10         //Verifica se o método getName retorna o novo nome esperado
11         assertEquals("Salão", zona.getName());
12     }
13 }
```

FIGURA 153 - ZONATEST (SETNOME())

Verifica se a função setName() da classe Zona atualiza corretamente o nome da zona. Ele cria uma instância de Zona com o ID igual a 1 e o nome “Esplanada”, chama a função setName() para definir o nome como “Salão” e verifica se a função getName() retorna “Salão”.



```
1 @Test
2     void getId() {
3
4         //Cria uma instância da classe Zona com id 1 e nome "Esplanada"
5         Zona zona = new Zona(1, "Esplanada");
6
7         //Verifica se o método getId retorna o id esperado
8         assertEquals(1, zona.getId());
9     }
10 }
```

FIGURA 154 - ZONATEST (GETID())

Verifica se a função getId() da classe Zona retorna o ID correto da zona. Ele cria uma instância de Zona com o ID igual a 1 e o nome “Esplanada”, chama a função getId() e verifica se ela retorna 1.



```
1 @Test
2     void setId() {
3         //Cria uma instância da classe Zona
4         Zona zona = new Zona();
5
6         //Chama o método setId para definir o id
7         zona.setId(2);
8
9         //Verifica se o método getId retorna o novo id esperado
10        assertEquals(2, zona.getId());
11    }
12 }
```

FIGURA 155 - ZONATEST (SETID())

Verifica se a função setId() da classe Zona atualiza corretamente o ID da zona. Ele cria uma instância de Zona, chama a função setId() para definir o ID como 2 e verifica se a função getId() retorna 2.

12.2 Testes de integração

Os testes de integração são um tipo de teste que verifica se os segmentos de software trabalham corretamente quando são combinados. Estes são realizados para assegurar a integridade do que foi feito unitariamente e naquilo que esta a ser incorporado.

Identificar interações inesperadas é possível com estes testes. Esta abordagem proativa não apenas contribui para a estabilidade do sistema, mas também permite antecipar e corrigir potenciais problemas de comunicação entre os diversos elementos do *software*.

Outro aspeto dos testes de integração é que estes têm a capacidade de ajudar a prevenir que as funcionalidades do programa, quando implementadas, operem corretamente e que estejam alinhadas com os requisitos do utilizador.

Neste projeto estes foram realizados de forma sistemática ao longo do projeto para construir a estrutura do programa, verificar erros na base de dados e erros na interface.

12.3 Testes de sistema e de validação

Os testes de sistema são um tipo de teste de software que é conduzido em um sistema inteiro. Eles verificam se o sistema cumpre os seus requisitos, sejam quais forem. Os *testers* realizam testes de sistema para avaliar os requisitos funcionais e não funcionais do sistema após a integração de módulos e componentes individuais. O teste do sistema é uma categoria de teste da caixa negra, o que significa que apenas testa características de funcionamento externas do software, em oposição a testar o design interno da aplicação. Os testadores não requerem qualquer conhecimento da programação e estrutura do código do software para avaliar completamente um software construído durante os testes do sistema. Em vez disso, os *testers* estão simplesmente a avaliar o desempenho do software a partir da perspetiva de um utilizador.

Os testes de validação são realizados para garantir que o *software* atenda aos requisitos do utilizador e do negócio. Eles são usados para avaliar se o *software* é adequado para o uso pretendido e se desempenha as necessidades do utilizador. Os testes de validação são usados para verificar se o *software* está pronto para ser lançado. Eles são uma parte importante do processo de desenvolvimento de *software*, pois ajudam a garantir que o *software* atenda às expectativas iniciais.

Neste ponto do projeto foram também feitos testes de desempenho para assegurar que a base de dados não tivesse problemas em manipular os dados quando o sistema estivesse a trabalhar com uma maior carga. Estes testes avaliaram a capacidade da base de dados em lidar com um volume significativo de informações, garantindo que a resposta do sistema permanecesse dentro dos parâmetros aceitáveis de tempo de execução.

Estes testes foram realizados pelos membros do grupo já mais na parte final do projeto de forma a garantir que o software tenha qualidade em que cumpra as necessidades do utilizador e do negócio.

12.4 Testes de usabilidade

Existem vários métodos para avaliar a usabilidade da aplicação. Foi escolhido o teste de usabilidade. O teste de usabilidade é a avaliação de um produto ou serviço ao testá-lo com participantes que representem os utilizadores finais. Durante um teste os participantes tentam completar uma série de tarefas típicas enquanto os responsáveis pelos testes observam, escutam e tiram notas. O objetivo destes testes é identificar problemas relacionados com a usabilidade, reunir dados qualitativos e quantitativos e determinar a satisfação do participante com o produto ou serviço. O grande benefício destes testes é descobrir falhas de usabilidade o mais cedo possível pois quanto mais cedo uma falha é descoberta menos dispendioso é corrigi-la, tendo por isso sendo o método escolhido. Para que um teste de usabilidade seja realizado com sucesso são necessários alguns documentos como:

- ♦ Grelha de tarefas a realizar e observações;
- ♦ Documento de consentimento de participação no teste (Figura 156);
- ♦ Questionário de satisfação global (Figura 157);

Tarefa		Execução			Erros	Dificuldades	Dúvidas	Observações	Duração
#	Descrição	Bastante Dificuldade	Alguma Dificuldade	Sem Dificuldade					
Tarefas como funcionário									
1	Navegar até à zona “Esplanada”								
2	Criar um pedido numa mesa com os itens: “Água” e “Café”								
3	Fechar conta em cartão								
4	Emitir segunda via dessa mesma mesa								
5	Reservar uma mesa para as 23:15								

TABELA 36 - TESTES DE USABILIDADE FUNCIONÁRIO

Tarefa		Execução			Erros	Dificuldades	Dúvidas	Observações	Duração
#	Descrição	Bastante Dificuldade	Alguma Dificuldade	Sem Dificuldade					
Tarefas como gerente									
1	Remover um funcionário								
2	Adicionar categoria								
3	Editar o preço de um item								
4	Adicionar uma nova zona								
5	Adicionar uma nova mesa a essa zona								
6	Editar o nome de um cargo								
7	Gerar e imprimir o relatório de vendas da semana passada								
8	Cancelar uma reserva de mesa								
9	Terminar Sessão								

TABELA 37 - TESTES DE USABILIDADE GERENTE

Participação em teste de usabilidade

No âmbito da unidade curricular do Projeto Temático em Desenvolvimento de Aplicações lecionada na Escola Superior de Tecnologia e Gestão de Águeda, está-se a desenvolver uma aplicação de gestão para estabelecimentos ligados à restauração.

O teste de usabilidade tem como objetivo testar as funcionalidades do protótipo desenvolvido, reforçando desde já que este é um teste à interface e não ao utilizador. Para tal, irá realizar tarefas no mesmo e responder a um breve questionário.

O seu contributo é extremamente relevante para este projeto e garante-se o anonimato de todas as informações de carácter pessoal.

Obrigado pela sua participação.

Consentimento informado

Deste modo, declaro ter compreendido a explicação que me foi dada sobre o projeto a decorrer e que as informações recolhidas são anónimas. Eu entendo que os resultados do projeto podem ser publicados em revistas científicas, apresentados em reuniões/eventos científicos e utilizados em atividades de investigação, sem qualquer violação de confidencialidade/anonimato. Ao participar nesta atividade, autorizo o uso de dados anónimos para a finalidade do projeto que lhe está associada e mencionada acima.

Águeda, de de

Assinatura: _____

FIGURA 156 - DOCUMENTO DE CONSENTIMENTO DE PARTICIPAÇÃO NO TESTE



1. Gostaria de utilizar este sistema com frequência *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

2. O sistema é desnecessariamente complexo *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

3. Achei o sistema fácil de utilizar *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

4. Acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para utilizar o sistema *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

5. Acho que as várias funções do sistema estão bem integradas *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

6. Acho que o sistema apresenta muita inconsistência *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

7. Penso que a maioria das pessoas aprenderia rapidamente como utilizar o sistema *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

8. Achei que o sistema era confuso de utilizar *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

9. Senti-me confiante ao utilizar o sistema *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

10. Precisei de aprender várias coisas novas antes de conseguir utilizar o sistema *

1	2	3	4	5
---	---	---	---	---

Discordo completamente

Concordo completamente

FIGURA 157 - QUESTIONÁRIO DE SATISFAÇÃO

Os ANEXO A e ANEXO B contêm as grelhas de observação e o levantamento do questionário, respetivamente. Estes anexos são essenciais para entender a usabilidade do nosso sistema. Nas grelhas de observação, é possível verificar os erros, dúvidas e dificuldades que os utilizadores sentiram ou detetaram ao utilizar o nosso sistema. Estes problemas foram identificados e corrigidos, melhorando assim a experiência do utilizador e a eficiência do nosso sistema.

13. Análise de Resultados

Ao analisar a lista de requisitos previamente mencionada verificou-se que foram levantados 36 requisitos no total, sem distinção entre funcionais e não funcionais. Em termos quantitativos pode-se afirmar que foram cumpridos (~81) % dos requisitos iniciais, tendo (~8) % ficado parcialmente implementados e (~11) % por implementar.

Requisitos	Cumprido	Não cumprido	Parcialmente cumprido
RF01	X		
RF02	X		
RF03	X		
RF04	X		
RF05	X		
RF06	X		
RF07	X		
RF08	X		
RF09	X		
RF10			X
RF11			X
RF12	X		
RF13	X		
RF14	X		
RF15	X		
RF16	X		
RF17	X		
RF18	X		
RF19	X		
RF20	X		
RF21	X		
RF22	X		
RF23	X		

RF24	X		
RF25	X		
RF26	X		
RF27	X		
RNF.1	X		
RNF.2	X		
RNF.3	X		
RNF.4		X	
RNF.5		X	
RNF.6		X	
RNF.7		X	
RNF.8	X		
RNF.9			X

TABELA 38 - ANÁLISE DE RESULTADOS

14. Reflexão Crítica e Conclusões

Nesta secção faz-se uma breve análise crítica sobre todo o trabalho desenvolvido, subdividido em vários pontos. Em primeiro lugar reflete-se sobre as atividades desenvolvidas e estratégias de trabalho adotadas, em seguida sobre o planeamento executado e por fim termina-se com uma síntese final e sugestões de melhoria.

14.1 Atividades Desenvolvidas

O levantamento inicial de requisitos e funcionalidades foi feito com bastantes incertezas inerentes ao funcionamento do sistema. Devido a isto, considerou-se que existiam requisitos insuficientes e não querendo arriscar em algo demasiado simplista foram adicionados requisitos extra de modo a aumentar o conteúdo e a complexidade da aplicação. Esta situação resultou num trabalho inicial bastante voltado para documentação, tendo sobrado menos tempo de desenvolvimento propriamente dito do que aquele que era desejável.

Apesar disto, conclui-se que este projeto e a aplicação resultante do mesmo foram um sucesso. O número de requisitos implementados de importância elevada, assim como as tarefas macro desenvolvidas até então estão na base dos critérios de sucesso utilizados para se poder qualificar o projeto como bem-sucedido.

14.2 Estratégias de Trabalho Adotadas

A estratégia de trabalho adotada, que acabou por ser modelada, resultou das regras de funcionamento da Unidade Curricular, mais propriamente: reuniões semanais com discussão de conteúdos e trabalho, que resultariam em entregas faseadas do relatório, culminando numa entrega final deste documento e da aplicação. Este sistema foi bastante vantajoso para a dinâmica do grupo, mesmo havendo alguma disparidade de disponibilidade entre os colegas, pois foram estabelecidas metas semanais para cada um dos elementos, de forma a obter uma distribuição de tarefas igualitária que não prejudicasse nem beneficiasse nenhum membro em específico as tarefas referentes ao desenvolvimento da aplicação foram divididas por funcionalidades, o que proporcionou uma maior dedicação de cada um. A comunicação e a entreajuda foram dois pontos muito fortes e importantes, o que se refletiu no resultado final.

14.3 Planeamento Previsto e Cronograma Executado

Houve ligeiras diferenças entre o planeamento e a execução real do trabalho, tendo em conta o levantamento de tarefas, distribuição das tarefas pelos elementos do grupo e respetiva duração. Na execução especificou-se a tarefa “Desenvolvimento da Aplicação” com a devida distribuição pelos elementos do grupo. Foi adicionada a tarefa “Correção de erros da Fase II”, que foi desenvolvida durante a Fase III do projeto. As tarefas “Produção do modelo de dados persistente”, “Descrição dos casos de utilização”, “Produção do modelo estrutural” e “Implementação da base de dados” foram realizadas em mais tempo que o previsto.

14.4 Sugestões para o Futuro

Como trabalho futuro contempla-se a finalização dos requisitos que não foram implementados, total ou parcialmente, bem como a realização de uma fase de testes mais completa o que permitiria desenvolver uma aplicação bastante mais robusta. Para além disto identificaram-se algumas lacunas que terão de ser melhoradas em versões futuras:

- ◆ **Opção de adicionar imagens aos itens/categorias:** Esta funcionalidade iria permitir aos utilizadores, mais propriamente aos funcionários, ver os itens e categorias de forma mais rápida e intuitiva. Isto poderia ser implementado ao adicionar um campo de *upload* de imagens nas interfaces “Gerir Itens” e “Gerir Categorias”. As imagens seriam guardadas nas relações “Item” e “Categoria”, na base de dados, e estariam vinculadas ao id do item ou categoria correspondente.
- ◆ **Tratamento de dados:** O tratamento eficiente de dados é crucial para o desempenho de qualquer aplicação. Neste projeto existe uma grande margem para melhoria nesse aspetto, por exemplo devia-se ter feito uma melhor verificação e validação da inserção de dados, antes de os enviar para a base de dados, bem como a criação de *indexes* para os atributos mais utilizados nas consultas SQL.
- ◆ **Divisão dos pagamentos:** Na aplicação desenvolvida, se um pagamento for dividido entre várias partes (2, 3, 4, etc.), são emitidos múltiplos recibos, a representar a parcela que cada parte pagou. No entanto, na base de dados, o “Pagamento” é registado como um só. Idealmente, cada pagamento dividido deve ser tratado em separado, para isto teria de ser criada uma nova tabela, por exemplo “Pagamento Dividido”, na base de dados.

- ◆ **Orientação a objetos:** A orientação a objetos é uma abordagem de programação que envolve a organização de código em torno de “objetos” - entidades que combinam estados (atributos) e comportamentos (métodos). Isto pode tornar o código mais fácil de entender, manter e reutilizar. O sistema encontra-se parcialmente orientada a objetos, mas longe de estar perfeito.
- ◆ **Sistema de backup:** Um sistema de backup é essencial para prevenir a perda de dados. Devia ser implementada a criação regular de cópias de segurança da base de dados, que poderiam ser restauradas em caso de falha do sistema.
- ◆ **Qualidade dos testes:** Os testes são um ponto crucial para garantir que a aplicação funciona como esperado, bem como para identificar e corrigir bugs. Foram feitos vários testes de software, mas com a consciência que a quantidade e qualidade dos mesmos podia ser mais elevada.

14.5 Síntese das Experiências

A unidade curricular de Projeto Temático em Desenvolvimento de Aplicações tem como objetivo a aplicação dos conhecimentos obtidos juntamente com as restantes subunidades curriculares que compõem o módulo temático, permitindo aos alunos a experiência de trabalho de equipa com vista no cumprimento de um conjunto de objetivos.

O projeto fluiu muito bem e sem conflitos, ainda assim, nos momentos mais atribulados, existiu sempre entreajuda no grupo, permitindo um trabalho de equipa eficiente.

14.6 Conclusão

Com a realização deste projeto, foram postos em prática os conhecimentos adquiridos das unidades curriculares associadas ao Projeto Temático, Sistemas de Bases de Dados e Engenharia de Software, lecionadas pelos docentes Gonçalo Paiva Dias e Joaquim Ferreira.

Durante a realização do projeto houve algumas mudanças nas ideias originais do grupo e foram feitas alterações durante o processo de desenvolvimento para melhorar a qualidade do que foi proposto desenvolver inicialmente.

O grupo superou a maior parte das dificuldades com que se deparou, como por exemplo a atualização dos *status* das mesas em tempo real, o agendamento das reservas, entre outros, conseguindo desenvolver e implementar a esmagadora maioria das tarefas que foram propostas.

15. Bibliografia

- [1] «1 JavaFX Overview (Release 8)». Disponível em: <https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784>.
- [2] «Apache PDFBox | A Java PDF Library». Disponível em: <https://pdfbox.apache.org/>.
- [3] «Discord | Your Place to Talk and Hang Out», *Discord*. Disponível em: <https://discord.com/>.
- [4] «Git». Disponível em: <https://git-scm.com/>.
- [5] «How to Use Git and GitHub – Introduction for Beginners», *freeCodeCamp.org*, 26 de setembro de 2022. Disponível em: <https://www.freecodecamp.org/news/introduction-to-git-and-github/>.
- [6] «IntelliJ IDEA – the Leading Java and Kotlin IDE», *JetBrains*. Disponível em: <https://www.jetbrains.com/idea/>.
- [7] «Java Documentation», *Oracle Help Center*. Disponível em: <https://docs.oracle.com/en/java/>.
- [8] «Java Tutorial». Disponível em: <https://www.w3schools.com/java/default.asp>.
- [9] «Java Tutorial | Learn Java Programming - javatpoint». Disponível em: <https://www.javatpoint.com/java-tutorial>.
- [10] «JavaScript | MDN», 25 de setembro de 2023. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [11] «Learn Java - Online Java Course», *CodeGym*. Disponível em: <https://codegym.cc>.
- [12] «Modelo V de software: verificação e validação | Lucidchart». Disponível em: <https://www.lucidchart.com/blog/pt/v-model-verificacao-e-validacao>.
- [13] «MySQL :: MySQL Workbench». Disponível em: <https://www.mysql.com/products/workbench/>.
- [14] «O que é teste de integração e quais são os tipos de teste?», 8 de outubro de 2021. Disponível em: <https://kenzie.com.br/blog/teste-de-integracao/>.
- [15] «One platform to connect», *Zoom*. Disponível em: <https://zoom.us/>.
- [16] «Sistema POS para Negocios #1 en Latinoamérica - POSMOVI». Disponível em: <https://www.posmovi.com/>.
- [17] «Software Gestão Restaurantes| TheFork Manager». Disponível em: <https://www.theforkmanager.com/pt-pt>.
- [18] «Stack Overflow - Where Developers Learn, Share, & Build Careers», *Stack Overflow*. Disponível em: <https://stackoverflow.com/>.
- [19] «Teste de integração: o que é, tipos com exemplo», 9 de dezembro de 2023. Disponível em: <https://www.guru99.com/pt/integration-testing.html>.
- [20] C. Singureanu, «Testes de Sistema - Tipos, Processo, Ferramentas & Mais!», <https://www.zaptest.com/pt-pt>. Disponível em: <https://www.zaptest.com/pt-pt/o-que-sao-testes-de-sistema-um-mergulho-profundo-nas-abordagens-tipos-ferramentas-dicas-e-truques-e-muito-mais>.
- [21] «Vendus - Software de Faturação e POS», *Vendus*. Disponível em: <https://www.vendus.pt/>.

ANEXO A

Teste de usabilidade -
Levantamento das grelhas de observação

Tarefa		Execução			Erros	Dificuldades	Dúvidas	Observações	Duração
#	Descrição	Bastante Dificuldade	Alguma Dificuldade	Sem Dificuldade					
Tarefas como funcionário									01:12.05
1	Navegar até à zona “Esplanada”			X					00:07.51
2	Criar um pedido numa mesa com os itens: “Água” e “Café”		X				1	Dúvida se o pedido tinha sido criado, devido à falta de feedback do sistema	00:16.35
3	Fechar conta em cartão		X				1	Dúvida se a conta tinha sido fechada, devido à falta de feedback do sistema	00:25.98
4	Emitir segunda via dessa mesma mesa			X					00:09.44
5	Reservar uma mesa para as 23:15			X	1			Foi detetado um erro, pois o utilizador conseguiu reservar uma mesa para um dia anterior ao dia do teste	00:12.77

Tarefa		Execução			Erros	Dificuldades	Dúvidas	Observações	Duração
#	Descrição	Bastante Dificuldade	Alguma Dificuldade	Sem Dificuldade					
Tarefas como funcionário									01:28.16
1	Navegar até à zona “Esplanada”		X				1	O utilizador demonstrou dúvidas em perceber qual era a zona em que se encontrava	00:16.43
2	Criar um pedido numa mesa com os itens: “Água” e “Café”			X					00:14.54
3	Fechar conta em cartão		X			1		Dificuldade ao carregar no ícone do cartão, pois só é possível carregar exatamente nas linhas da imagem e não em qualquer lugar da pane	00:27.91
4	Emitir segunda via dessa mesma mesa			X					00:14.90
5	Reservar uma mesa para as 23:15			X					00:14.38

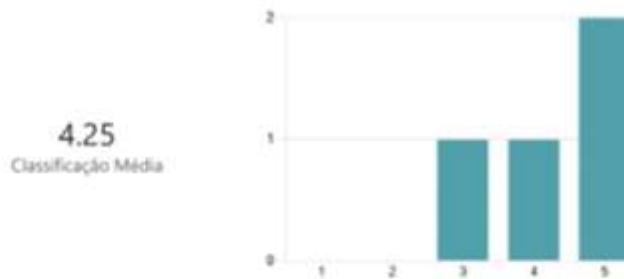
Tarefa		Execução			Erros	Dificuldades	Dúvidas	Observações	Duração
#	Descrição	Bastante Dificuldade	Alguma Dificuldade	Sem Dificuldade					
Tarefas como gerente									02:34.00
1	Remover um funcionário			X					00:14.99
2	Adicionar categoria			X					00:12.68
3	Editar o preço de um item		X			1		O utilizador teve dificuldades ao entender como era feita a edição de um item, devido à falta de um botão “Atualizar”	00:24.35
4	Adicionar uma nova zona			X					00:15.39
5	Adicionar uma nova mesa a essa zona			X					00:13.40
6	Editar o nome de um cargo		X			1		O utilizador teve dificuldades ao entender como era feita a edição de um cargo, devido à falta de um botão “Atualizar”	00:25.39
7	Gerar e imprimir o relatório de vendas da semana passada		X				1	O utilizador teve dúvidas se o relatório já havia sido ou não guardado, dada a falta de feedback, acabando por guardar 4 relatórios iguais	00:20.23
8	Cancelar uma reserva de mesa		X		1			Verificou-se que o tratamento para mostrar apenas as mesas com reserva ativa não estava correto	00:21.43
9	Terminar Sessão			X					00:06.14

Tarefa		Execução			Erros	Dificuldades	Dúvidas	Observações	Duração
#	Descrição	Bastante Dificuldade	Alguma Dificuldade	Sem Dificuldade					
Tarefas como gerente									02:19.30
1	Remover um funcionário			X					00:12.81
2	Adicionar categoria			X					00:14.39
3	Editar o preço de um item		X		1			O utilizador tentou inserir uma <i>String</i> no campo “preço” e o sistema deu erro	00:28.97
4	Adicionar uma nova zona			X					00:11.61
5	Adicionar uma nova mesa a essa zona			X					00:13.54
6	Editar o nome de um cargo			X					00:17.40
7	Gerar e imprimir o relatório de vendas da semana passada		X		1			O utilizador tentou “imprimir” o relatório sem o ter gerado previamente e o programa não deu feedback que isso não é possível	00:20.11
8	Cancelar uma reserva de mesa			X					00:12.18
9	Terminar Sessão			X					00:08.29

ANEXO B

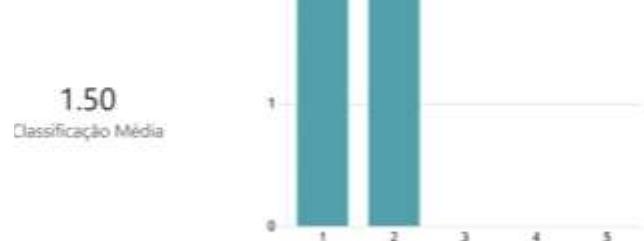
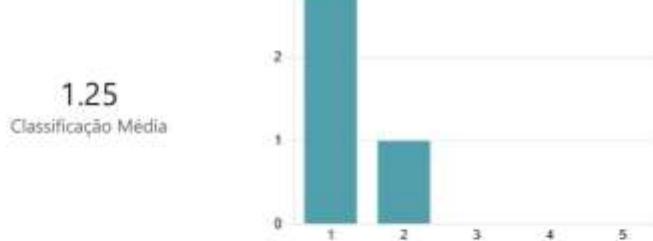
Teste de usabilidade -
Levantamento do questionário de satisfação

1. Gostaria de utilizar este sistema com frequência (0 ponto)



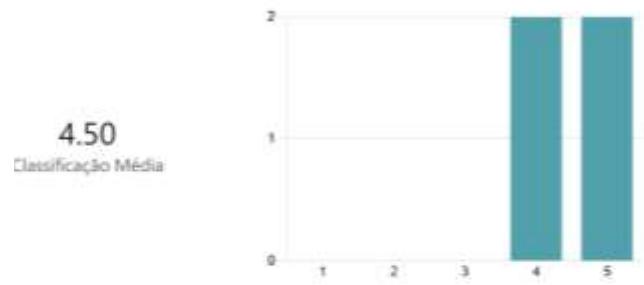
2. O sistema é desnecessariamente complexo (0 ponto)

i que precisaria de ajuda de uma pessoa com conhecimentos técnicos
utilizar o sistema (0 ponto)



3. Achei o sistema fácil de utilizar (0 ponto)

i que as várias funções do sistema estão bem integradas (0 ponto)





6. Acho que o sistema apresenta muita inconsistência (0 ponto)

1. Achei que o sistema era confuso de utilizar (0 ponto)

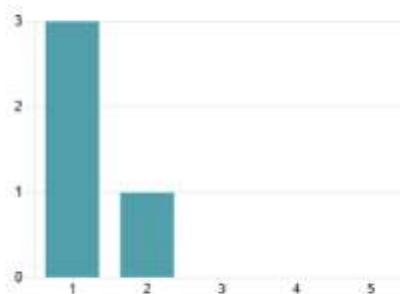


7. Penso que a maioria das pessoas aprenderia rapidamente como utilizar o sistema (0 ponto)

1. Senti-me confiante ao utilizar o sistema (0 ponto)



1. Fui capaz de conseguir utilizar o sistema (0 ponto)



ANEXO C

Java Doc



Hierarchy For All Packages

Package Hierarchies:

ua.pt.javaeats, ua.pt.javaeats.dashboardBalcao, ua.pt.javaeats.DashboardFecharConta,
ua.pt.javaeats.dashboardGerente, ua.pt.javaeats.dashboardGerirCargo, ua.pt.javaeats.dashboardGerirCategoria,
ua.pt.javaeats.dashboardGerirFuncionario, ua.pt.javaeats.dashboardGerirItem, ua.pt.javaeats.dashboardGerirMesas,
ua.pt.javaeats.dashboardGerirZonas, ua.pt.javaeats.dashboardMesa, ua.pt.javaeats.dashboardPedido,
ua.pt.javaeats.dashboardPrincipal, ua.pt.javaeats.dashboardRelatorios, ua.pt.javaeats.dashboardReservarMesas

Class Hierarchy

- java.lang.Object
 - javafx.application.Application
 - ua.pt.javaeats.dashboardPrincipal.DashboardApplication
 - ua.pt.javaeats.dashboardBalcao.dashboardBalcaoApplication
 - ua.pt.javaeats.DashboardFecharConta.DashboardFecharContaApplication
 - ua.pt.javaeats.dashboardGerente.dashboardGerenteApplication
 - ua.pt.javaeats.dashboardGerirCargo.dashboardGerirCargoApplication
 - ua.pt.javaeats.dashboardGerirCategoria.dashboardGerirCategoriaApplication
 - ua.pt.javaeats.dashboardGerirFuncionario.dashboardGerirFuncionarioApplication
 - ua.pt.javaeats.dashboardGerirItem.dashboardGerirItemApplication
 - ua.pt.javaeats.dashboardMesa.DashboardMesaApplication
 - ua.pt.javaeats.dashboardPedido.DashboardPedidoApplication
 - ua.pt.javaeats.dashboardRelatorios.dashboardRelatorioApplication
 - ua.pt.javaeats.dashboardReservarMesas.dashboardReservarMesasApplication
 - ua.pt.javaeats.dashboardGerirMesas.GerirMesasApplication
 - ua.pt.javaeats.dashboardGerirZonas.GerirZonasApplication
 - ua.pt.javaeats.Cargo
 - ua.pt.javaeats.Categoria
 - ua.pt.javaeats.ConectarBD
 - ua.pt.javaeats.dashboardBalcao.dashboardBalcaoController
 - ua.pt.javaeats.dashboardPrincipal.DashboardController
 - ua.pt.javaeats.DashboardFecharConta.DashboardFecharContaController
 - ua.pt.javaeats.dashboardGerente.dashboardGerenteController
 - ua.pt.javaeats.dashboardGerirCargo.dashboardGerirCargoController
 - ua.pt.javaeats.dashboardGerirCategoria.dashboardGerirCategoriaController
 - ua.pt.javaeats.dashboardGerirFuncionario.dashboardGerirFuncionarioController
 - ua.pt.javaeats.dashboardGerirItem.dashboardGerirItemController
 - ua.pt.javaeats.dashboardMesa.DashboardMesaController
 - ua.pt.javaeats.dashboardPedido.DashboardPedidoController
 - ua.pt.javaeats.dashboardRelatorios.dashboardRelatorioController
 - ua.pt.javaeats.dashboardReservarMesas.dashboardReservarMesasController
 - ua.pt.javaeats.Funcionario
 - ua.pt.javaeats.dashboardGerente.GerenteBarraLateralController
 - ua.pt.javaeats.GerirBD
 - ua.pt.javaeats.dashboardGerirMesas.GerirMesasController
 - ua.pt.javaeats.dashboardGerirZonas.GerirZonasController
 - ua.pt.javaeats.Item
 - ua.pt.javaeats.ItemPedido
 - ua.pt.javaeats.Mesa
 - ua.pt.javaeats.PagamentoAtendimento
 - ua.pt.javaeats.Pedido
 - ua.pt.javaeats.SerialCommunicationService
 - ua.pt.javaeats.SessaoUtilizador
 - ua.pt.javaeats.Zona

ANEXO D

Script Base de Dados

```

delimiter \\

CREATE procedure CriarBDPTDA()
BEGIN

CREATE TABLE `Cargo` (
  `id_cargo` int NOT NULL AUTO_INCREMENT,
  `descricao` varchar(250) DEFAULT NULL,
  PRIMARY KEY (`id_cargo`)
) ;

CREATE TABLE `Categoria` (
  `id_categoria` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`id_categoria`)
);

CREATE TABLE `Eventos` (
  `id_evento` INT NOT NULL AUTO_INCREMENT,
  `mesa_id` INT DEFAULT NULL,
  `status_anterior` VARCHAR(255) DEFAULT NULL,
  `status_novo` VARCHAR(255) DEFAULT NULL,
  `data_hora` TIMESTAMP NULL DEFAULT NULL,
  PRIMARY KEY (`id_evento`),
  FOREIGN KEY (`mesa_id`)
    REFERENCES `Mesa` (`id_mesa`)
);

CREATE TABLE `Funcionario` (
  `id_funcionario` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(25) DEFAULT NULL,
  `password` VARCHAR(25) NOT NULL,
  `id_cargo` INT DEFAULT NULL,
  `id_cartao` VARCHAR(100) DEFAULT NULL,
  PRIMARY KEY (`id_funcionario`),
  FOREIGN KEY (`id_cargo`)
    REFERENCES `Cargo` (`id_cargo`)
);

CREATE TABLE `Item` (
  `id_item` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) DEFAULT NULL,
  `preco` DOUBLE DEFAULT NULL,
  `id_categoria` INT DEFAULT NULL,
  PRIMARY KEY (`id_item`),
  FOREIGN KEY (`id_categoria`)
    REFERENCES `Categoria` (`id_categoria`)
);

CREATE TABLE `Mesa` (
  `id_mesa` INT NOT NULL AUTO_INCREMENT,
  `status` VARCHAR(50) NOT NULL,
  `id_zona` INT DEFAULT NULL,
  PRIMARY KEY (`id_mesa`),
  FOREIGN KEY (`id_zona`)
    REFERENCES `Zonas` (`id`)
);

CREATE TABLE `PagamentoAtendimento` (
  `id_pagatendimento` INT NOT NULL AUTO_INCREMENT,
  `data_hora_inicio` DATETIME NOT NULL,
  `preco_total` DOUBLE DEFAULT NULL,
  `tipo_pagamento` VARCHAR(45) DEFAULT NULL,
  `id_mesa` INT DEFAULT NULL,

```



```
'data_hora_fim' DATETIME DEFAULT NULL,  
PRIMARY KEY ('id_pagatendimento'),  
KEY `id_mesa_idx` ('id_mesa'),  
FOREIGN KEY ('id_mesa')  
    REFERENCES `Mesa` ('id_mesa')  
);  
  
CREATE TABLE `Pedido_Restaurante` (  
    `id_pedido` INT NOT NULL AUTO_INCREMENT,  
    `descricao` VARCHAR(250) DEFAULT NULL,  
    `id_pagamento` INT DEFAULT NULL,  
    `id_func` INT DEFAULT NULL,  
    `status` VARCHAR(20) DEFAULT 'Por Fazer',  
    PRIMARY KEY ('id_pedido'),  
    FOREIGN KEY ('id_func')  
        REFERENCES `Funcionario` ('id_funcionario'),  
    FOREIGN KEY ('id_pagamento')  
        REFERENCES `PagamentoAtendimento` ('id_pagatendimento')  
);  
  
CREATE TABLE `PedidoDetalhe` (  
    `id_pedido` INT NOT NULL,  
    `id_item` INT NOT NULL,  
    `quantidade` INT DEFAULT NULL,  
    `data_hora` VARCHAR(45) DEFAULT NULL,  
    KEY `id_item_idx` ('id_item'),  
    KEY `id_pedido_idx` ('id_pedido'),  
    FOREIGN KEY ('id_item')  
        REFERENCES `Item` ('id_item'),  
    FOREIGN KEY ('id_pedido')  
        REFERENCES `Pedido_Restaurante` ('id_pedido')  
);  
  
CREATE TABLE `Reservas` (  
    `id_reserva` INT NOT NULL AUTO_INCREMENT,  
    `id_mesa` INT DEFAULT NULL,  
    `data_reserva` DATE DEFAULT NULL,  
    `hora_reserva` TIME DEFAULT NULL,  
    PRIMARY KEY ('id_reserva'),  
    FOREIGN KEY ('id_mesa')  
        REFERENCES `Mesa` ('id_mesa')  
);  
  
CREATE TABLE `Zonas` (  
    `id` INT NOT NULL AUTO_INCREMENT,  
    `nome` VARCHAR(255) NOT NULL,  
    PRIMARY KEY ('id')  
);  
  
end\\;  
delimiter ;
```

ANEXO E

Atas



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 2	ASS: LUCAS DUARTE
DATA: 23/10/2023	ASS: GABRIEL CRAVO
PIVÔ RESPONSÁVEL: LUCAS DUARTE	ASS: BRUNO MIGUEIS
HORA INÍCIO: 10:00	ASS: DANIEL SILVA
HORA TÉRMINO: 10:35	ASS: MIGUEL PIRRÉ
PRÓXIMA REUNIÃO: 30/10/23	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Pedir opinião sobre os requisitos funcionais do sistema

Interface (esboço) do sistema para a semana?

Próximos passos no planeamento do projeto?

2. ATA DA REUNIÃO

Requisitos funcionais e não funcionais, melhorar a descrição.

Refazer/completar os diagramas de UML.

Preencher o relatório com a informação acima referida.

3. COMENTÁRIOS / NOTAS

A reunião correu conforme esperado, dentro da normalidade.

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 3	Ass: MIGUEL PIRRÉ
DATA: 02/11/2023	Ass: DANIEL SILVA
PIVÔ RESPONSÁVEL: MIGUEL PIRRÉ	Ass: LUCAS DUARTE
HORA INÍCIO: 15:00	Ass: BRUNO MIGUEIS
HORA TÉRMINO: 15:37	ASS: GABRIEL CRAVO
PRÓXIMA REUNIÃO: 09/10/23	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Pedir opiniões e tirar duvidas no relatório.

Análise dos diagramas com as alterações sugeridas da ultima reunião.

Planeamento dos próximos passos.

2. ATA DA REUNIÃO

Melhoramento do relatório

3. COMENTÁRIOS / NOTAS

A reunião como esperado.

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 4	ASS: BRUNO MIGUEIS
DATA: 09/11/2023	ASS: GABRIEL CRAVO
PIVÔ RESPONSÁVEL: BRUNO MIGUEIS	ASS: LUCAS DUARTE
HORA INÍCIO: 15:00	ASS: MIGUEL PIRRÉ
HORA TÉRMINO: 15:35	ASS: DANIEL SILVA
PRÓXIMA REUNIÃO: 16/11/23	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Apresentação do desenho da base de dados

Apresentação das alterações do relatório.

Planeamento dos próximos passos.

2. ATA DA REUNIÃO

As dúvidas foram esclarecidas e foram feitas algumas alterações em alguns tópicos do trabalho.

3. COMENTÁRIOS / NOTAS

A reunião correu como esperado e dentro do tempo marcado.

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 5	ASS: LUCAS DUARTE
DATA: 16/11/2023	ASS: GABRIEL CRAVO
PIVÔ RESPONSÁVEL: DANIEL SILVA	ASS: BRUNO MIGUES
HORA INÍCIO: 15:00	ASS: DANIEL SILVA
HORA TÉRMINO: 15:20	ASS: MIGUEL PIRRÉ
PRÓXIMA REUNIÃO: 24/11/2023	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Agendar a próxima reunião dada a indisponibilidade para quinta-feira dia 23/11;

Apresentar, discutir e pedir opiniões sobre o trabalho elaborado durante a semana;

Tirar dúvidas acerca dos diagramas de classes (base de dados e organização do código),

Obter sugestões de melhoria sobre os relatórios de especificação lógica e de requisitos a entregar;

Planejar os próximos passos.

2. ATA DA REUNIÃO

Foi reagendada a próxima reunião;

Foi discutido o trabalho elaborado;

Foram tiradas as dúvidas sobre os diagramas de classes;

Foram feitas e ouvidas sugestões de melhoria nos relatórios;



3. COMENTÁRIOS / NOTAS

A reunião correu como esperado, foi possível discutir toda a ordem de trabalhos e foram tiradas todas as questões de forma fluída e coordenada.

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 6	ASS: GABRIEL CRAVO
DATA: 24/11/2023	ASS: MIGUEL PIRRÉ
PIVÔ RESPONSÁVEL: GABRIEL CRAVO	ASS: LUCAS DUARTE
HORA INÍCIO: 12:05	ASS: DANIEL SILVA
HORA TÉRMINO: 12:25	ASS: BRUNO MIGUEIS
PRÓXIMA REUNIÃO: 30/11/2023	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Obter feedback do relatório intermédio.

Discussir próximos passos no projeto.

2. ATA DA REUNIÃO

Foram apresentadas propostas para os próximos passos no desenvolvimento do projeto.

Foram discutidos eventuais desafios antecipados.

Ficou acordado que o Lucas Duarte ficará acordado da próxima reunião.

3. COMENTÁRIOS / NOTAS

A reunião foi feita presencialmente e correu como esperado

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 7	ASS: LUCAS DUARTE
DATA: 30/11/2023	ASS: GABRIEL CRAVO
PIVÔ RESPONSÁVEL: LUCAS DUARTE	ASS: BRUNO MIGUEIS
HORA INÍCIO: 15:00	ASS: DANIEL SILVA
HORA TÉRMINO: 15:22	ASS: MIGUEL PIRRÉ
PRÓXIMA REUNIÃO: 07/12/2023	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Alteração dos métodos para gerir a Base de Dados

Revisão do relatório e trabalho feito

Alteração dos diagramas de sequência

Obter feedback do trabalho já feito

2. ATA DA REUNIÃO

Foram apresentadas propostas para os próximos passos no desenvolvimento do projeto.

Foram apresentados os desafios que tivemos durante a semana

Apresentação do sistema do menu login

3. COMENTÁRIOS / NOTAS

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 8	ASS: LUCAS DUARTE
DATA: 07/12/2023	ASS: MIGUEL PIRRE
PIVÔ RESPONSÁVEL: MIGUEL PIRRÉ	ASS: DANIEL SILVA
HORA INÍCIO: 15:00	ASS: GABRIEL CRAVO
HORA TÉRMINO: 15:25	ASS: BRUNO MIIGUEIS
PRÓXIMA REUNIÃO: 14/12/2023	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Mostrar as alterações realizadas.

Apresentar os próximos passos para o desenvolvimento do projeto.

2. ATA DA REUNIÃO

Mostrar as alterações realizadas.

Apresentar os próximos passos para o desenvolvimento do projeto.

3. COMENTÁRIOS / NOTAS

A reunião correu como esperado.

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 9	ASS: BRUNO MIGUEIS
DATA: 14/12/2023	ASS: GABRIEL CRAVO
PIVÔ RESPONSÁVEL: BRUNO MIGUEIS	ASS: MIGUEL PIRRÉ
HORA INÍCIO: 15:00	ASS: LUCAS DUARTE
HORA TÉRMINO: 15:21	ASS: DANIEL SILVA
PRÓXIMA REUNIÃO: 21/12/2023	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Apresentação das seguintes tarefas:

Base de dados e dos scripts da mesma;

Código base (métodos de cada classe);

Código para gerir, conectar e desconectar a base de dados.

2. ATA DA REUNIÃO

Registo das tarefas a fazer para a próxima semana.

Registo do feedback do professor.

Dúvidas pontuais foram esclarecidas.

3. COMENTÁRIOS / NOTAS

Reunião correu dentro da normalidade.

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 10	ASS: DANIEL SILVA
DATA: 21/12/2023	ASS: MIGUEL PIRRÉ
PIVÔ RESPONSÁVEL: DANIEL SILVA	ASS: GABRIEL CRAVO
HORA INÍCIO: 15:00	ASS: BRUNO MIGUEIS
HORA TÉRMINO: 15:15	ASS: LUCAS DUARTE
PRÓXIMA REUNIÃO: 04/01/2024	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Mostrar e pedir opinião sobre problemas mudanças na base de dados;

Mostrar e pedir opiniões acerca do trabalho desenvolvido ao longo da semana;

Próximos passos no planeamento do projeto?

2. ATA DA REUNIÃO

Foram retirados apontamentos sobre o que fazer com as mudanças.

Foram escritos apontamentos para o desenvolvimento do projeto.

3. COMENTÁRIOS / NOTAS

A reunião correu como esperado.

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 11	ASS: GABRIEL CRAVO
DATA: 04/01/2023	ASS: LUCAS DUARTE
PIVÔ RESPONSÁVEL: GABRIEL CRAVO	ASS: MIGUEL PIRRÉ
HORA INÍCIO: 15:00	ASS: BRUNO MIGUEIS
HORA TÉRMINO: 15:20	ASS: DANIEL SILVA
PRÓXIMA REUNIÃO: 11-01-2024	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Mostrar e pedir opinião sobre a aplicação desenvolvida desde a última reunião.

Apresentar duvidas que surgiram.

Próximos passos no projeto.

2. ATA DA REUNIÃO

Foram esclarecidas as dúvidas sobre o que surgiu no tempo em que tivemos sem reunião.

Foram retirados apontamentos sobre o feedback recebido da opinião da reunião.

Foram registados os próximos passos para a semana seguinte.

3. COMENTÁRIOS / NOTAS

A reunião correu de forma esperada e sem problemas.

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 12	ASS: LUCAS DUARTE
DATA: 11/01/2024	ASS: BRUNO MIGUEIS
PIVÔ RESPONSÁVEL: LUCAS DUARTE	ASS: MIGUEL PIRRÉ
HORA INÍCIO: 15:00	ASS: DANIEL SILVA
HORA TÉRMINO: 15:34	ASS: GABRIEL CRAVO
PRÓXIMA REUNIÃO: 18-01-2024	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Mostrar e pedir opinião sobre o relatório;

Realização de testes;

Próximos passos no decorrer da semana?

2. ATA DA REUNIÃO

Dúvidas propostas foram esclarecidas.

Foi registado o feedback sobre o relatório.

3. COMENTÁRIOS / NOTAS

A reunião correu conforme esperado.

(todos os campos apresentados a cinza devem vir preenchidos para a reunião)



PROJECTO TEMÁTICO EM DESENVOLVIMENTO DE APLICAÇÕES

AGENDA E ATA DE REUNIÃO (GRUPO 2)

REUNIÃO	PRESENÇAS
NÚMERO: 13	ASS: DANIEL SILVA
DATA: 18/01/2024	ASS: BRUNO MIGUEIS
PIVÔ RESPONSÁVEL: MIGUEL PIRRÉ	ASS: GABRIEL CRAVO
HORA INÍCIO: 15:00	ASS: LUCAS DUARTE
HORA TÉRMINO: 15:21	ASS: MIGUEL PIRRE
	ASS:

1. ORDEM DE TRABALHOS DA REUNIÃO / QUESTÕES

Mostrar e pedir opinião sobre o trabalho e o relatório (ambos finalizados).

Conclusões finais sobre o relatório.

2. ATA DA REUNIÃO

A reunião correu como esperado, foi feito tudo o que estava na ordem de trabalho.

3. COMENTÁRIOS / NOTAS