epb
escola profissional de braga

PAP - PROVA DE APTIDÃO PROFISSIONAL

XSYSTEMS

XSYSTEMS

CURSO PROFISSIONAL DE GESTÃO E PROGRAMAÇÃO DE SISTEMAS INFROMÁTICOS

Gino Gabriel Rodrigues Oliveira

Trabalho realizado sob a orientação de:

Prof. Manuel Ferreira

Escola Profissional de Braga

Braga, 10 de maio de 2024

PESSOAS 2030 PROGRAMA DEMOGRAFIA, QUALIFICAÇÕES E INCLUSÃO
PORTUGAL 2030
Cofinanciado pela União Europeia
SELO DE CONFORMIDADE EQAVET
uma escola Rumos education | Knowledge sharing
Os Fundos Europeus mais próximos de si.

Nota Introdutória

A Prova de Aptidão Profissional consiste na apresentação e defesa, perante um júri, de um projeto, consubstanciado num produto, material ou intelectual, numa intervenção ou numa atuação, bem como do respetivo relatório final de realização e apreciação crítica, demonstrativo de saberes e competências profissionais, adquiridos ao longo da formação e estruturante do futuro profissional.

A sua realização constitui-se como um dos imperativos legais para a conclusão do curso profissional de (designação do curso), que confere uma dupla certificação: qualificação profissional de nível IV e o 12º ano como certificação escolar de nível secundário.

A presente prova foi realizada na Escola Profissional de Braga no presente ano letivo de 2023/2024 e a sua defesa realizada, perante um júri final, nas datas estabelecidas no calendário escolar.

Dedicatória

Dedico este projeto a todos aqueles que o tornaram possível, mas com um especial carinho a minha família e amigos mais próximos!

Agradecimentos

Gostaria de agradecer a todos que, de alguma forma, contribuíram para o desenvolvimento bem-sucedido deste projeto.

Aos meus colegas de turma, cuja colaboração e esforço conjunto foram essenciais para superar desafios e alcançar resultados significativos. Agradeço pela troca constante de ideias, pela dedicação compartilhada e pelo ambiente de trabalho colaborativo que me impulsionou.

Aos profissionais da área, que generosamente compartilharam o seu conhecimento e experiência, oferecendo orientações valiosos que enriqueceram o nosso entendimento e aprimoraram a qualidade do projeto.

Aos amigos e colegas envolvidos nos testes, pela vossa paciência e feedback valioso que me permitiram aprimorar e refinar o software para atender às necessidades reais.

Por fim, expresso minha gratidão a todos os que direta ou indiretamente contribuíram para este projeto. Estou profundamente grato por cada um que fez parte desta jornada.

Obrigado por tornarem este projeto uma experiência enriquecedora e gratificante.

Índice

PESSOAS 2030 PORTUGAL 2030 Cofinanciado pela União Europeia SELO DE CONFORMIDADE EQAVET uma escola Rumos education Knowledge sharing
Os Fundos Europeus mais próximos de si.

Índice de Figuras

Os Fundos Europeus mais próximos de si.

Os Fundos Europeus mais próximos de si.

1. Introdução

1.1 Fundamentação

A minha Prova de Aptidão Profissional consiste na criação de um software destinado à gestão de pedidos em estabelecimentos de restauração. Este projeto, que será desenvolvido em colaboração com a minha colega de turma, Maria Ramos, visa otimizar o processo de pedidos e facilitar a operação de restaurantes.

A nossa abordagem para este software envolve uma divisão clara de tarefas. Eu serei responsável pelo desenvolvimento da aplicação que permitirá à equipa do restaurante realizar consultas, edições, efetuar pagamentos e gerir todas as atividades após a realização dos pedidos nas mesas. A aplicação Android criada pela minha colega será utilizada pelos funcionários para registar os pedidos nas mesas.

No âmbito do software que estou encarregue de desenvolver, os pedidos serão encaminhados para a cozinha através de uma interface onde a equipa poderá consultar informações relevantes sobre cada mesa, como o número da mesa, alergénios e observações. Assim que os pedidos estiverem concluídos, poderão ser fechados e marcados como terminados.

Além disso, vou implementar um sistema de pagamento integrado na caixa registadora, que permitirá consultar os pedidos das mesas, realizar pagamentos, gerir dados dos clientes e aceder as configurações essenciais do sistema, como configurações gerais, encerramento da caixa, funções administrativas, entre outras.

A inspiração para este projeto surgiu quando decidimos almoçar num restaurante muito frequentado, onde observamos uma falta de organização por parte da equipa do estabelecimento. Este software visa melhorar a eficiência e a gestão de pedidos em estabelecimentos semelhantes, contribuindo para uma experiência mais agradável para os clientes e uma operação mais eficaz por parte da equipa do restaurante.

PESSOAS 2030 PROGRAMA EUROPEU DE QUALIFICAÇÃO E INCLUSÃO
PORTUGAL 2030
Cofinanciado pela União Europeia
SELO DE CONFORMIDADE EQAVET
uma escola Rumos education | Knowledge sharing
Os Fundos Europeus mais próximos de si.

1.2 Inspiração

A inspiração deste projeto foi impulsionada pela nossa decisão de almoçar em um restaurante de grande movimento, onde identificamos uma lacuna evidente, a ausência de organização por parte da equipa de estabelecimento!

1.3 Objetivos

Nosso objetivo é aprimorar a eficiência e a gestão de pedidos em estabelecimentos similares, proporcionando uma experiência mais agradável aos clientes e garantindo uma operação mais eficaz por parte da equipa do restaurante.

- **Aumento da Eficiência Operacional:** Redução de erros de comunicação, maior rapidez no serviço e menor tempo de espera para os clientes.
- **Aumento da Fidelização de Clientes:** O acesso rápido a pedidos anteriores e a capacidade de oferecer um serviço personalizado podem aumentar a fidelização dos clientes.
- **Redução do Desperdício de Alimentos:** Com uma comunicação mais precisa, o XSYSTEMS pode ajudar a reduzir o desperdício de alimentos ao evitar pedidos duplicados ou incorretos.
- **Personalização e Flexibilidade:** O XSYSTEMS pode ser personalizado de acordo com as necessidades específicas de cada estabelecimento. É possível definir diferentes menus, e preços, tornando-o adequado para uma ampla variedade de tipos de restaurantes e públicos.

PESSOAS 2030 PORTUGAL 2030 Cofinanciado pela União Europeia SELO DE CONFORMIDADE EQAVET uma escola Rumos education Knowledge sharing
Os Fundos Europeus mais próximos de si.

1.4 Cronograma

Cronograma		
Fase	Calendarização	Elenco de atividades nas diversas fases
Conceção	ABRIL - 2023	Idealização da PAP
	MAIO - 2023	Primeiros rascunhos
	SETEMBRO - 2023	Inicio do Projeto
	OUTUBRO - 2023	Estruturação Inicial
	NOVEMBRO - 2023	Design Inicial
	DEZEMBRO - 2023	Criação da Base de Dados
Desenvolvimento	JANEIRO - 2024	Base de dados a funcionar
	FEVEREIRO - 2024	Conexões entre cliente/restaurante
	MARÇO - 2024	Correções e melhorias
	ABRIL - 2024	Testes
	MAIO - 2024	Finalização
Autoavaliação e Relatório		

2. Logotipo

O nome XSYSTEMS foi escolhido por sua universalidade e não define claramente seu propósito inicial, que é a restauração. O "X" no nome simboliza um garfo e uma faca cruzados, embora essa representação não seja imediatamente aparente. A escolha de um nome genérico foi intencional para permitir que, no futuro, XSYSTEMS possa abranger outros programas e sistemas sem estar diretamente associado apenas ao setor de restauração.

Figura 1- Logotipo 1

Figura 2 - Logotipo 2

Figura 3 - Logotipo 3

Até que optei por algo mais simples e genérico:

Figura 4- Logotipo 4

Figura 5- Logotipo 5

3. Idealizações e Manuscritos

O projeto foi concebido com o objetivo de registrar pedidos e monitorar todo o processo de entrega ao cliente de forma otimizada. Tendo experiência na área, reconheci a complexidade de gerenciar pedidos com especificações detalhadas, que precisam respeitar certos limites e restrições.

Iniciamos o desenvolvimento com uma interface de login que inclui botões e campos simples, buscando a máxima otimização. Optamos por utilizar o Firebase como base de dados, seguindo recomendações de colegas e professores, o que nos permitiu ter uma conexão rápida com o TPA e a Caixa de Pagamento.

A organização da base de dados mostrou-se um desafio considerável, mas eventualmente chegamos a um consenso que atendia às necessidades funcionais do sistema. Quanto à estética do programa, optamos por um design simples desde o início, já tendo uma visão clara de como queríamos que o design fosse.

3.1 Firebase

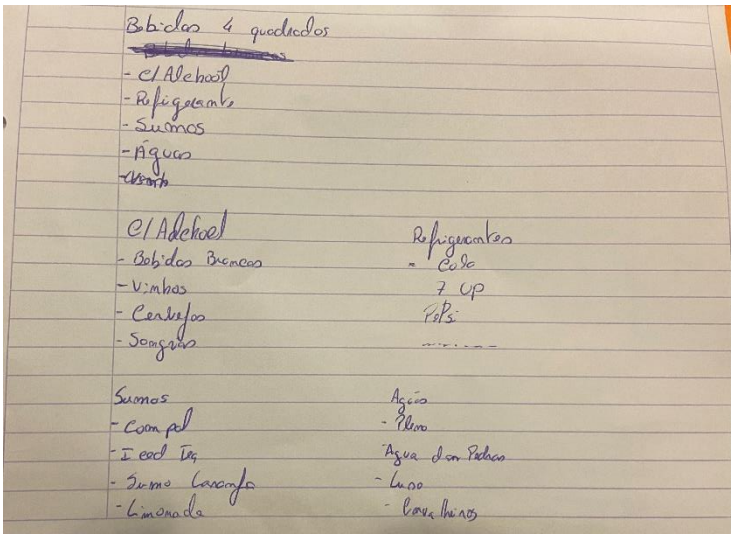


Figura 6 - Manuscritos Firebase

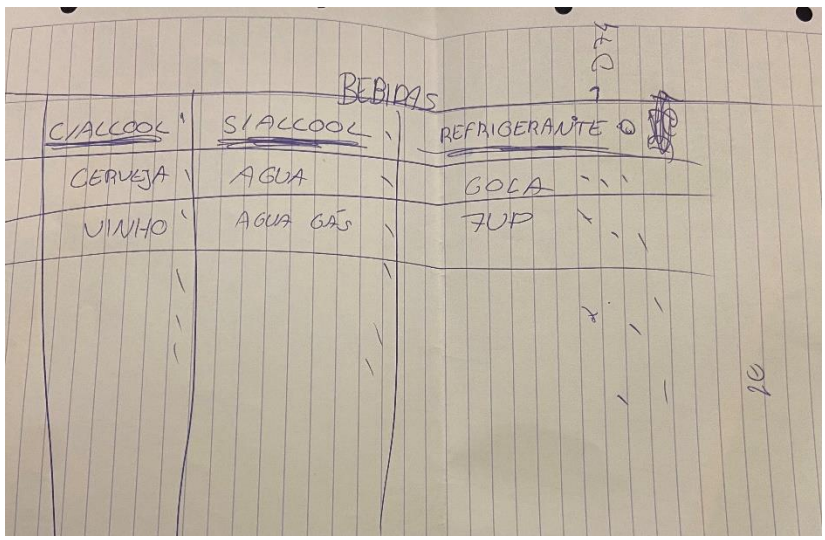


Figura 7- Organização da Firebase

3.2 Software

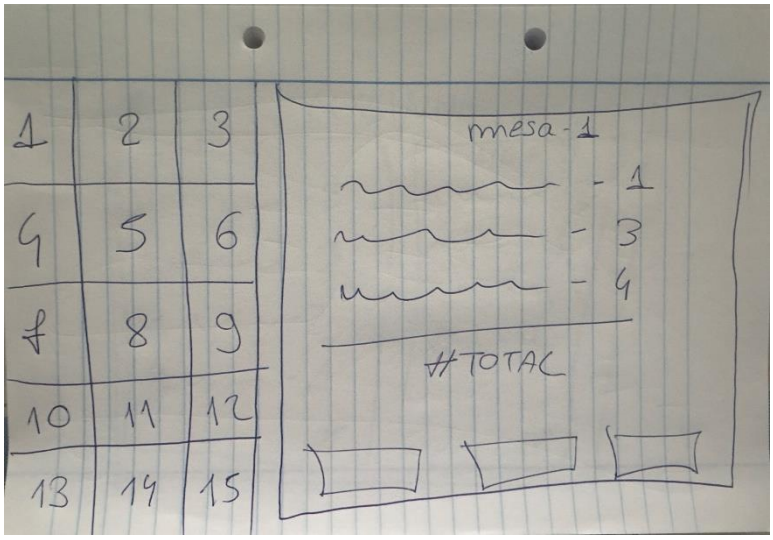


Figura 9 - Manuscritos Software

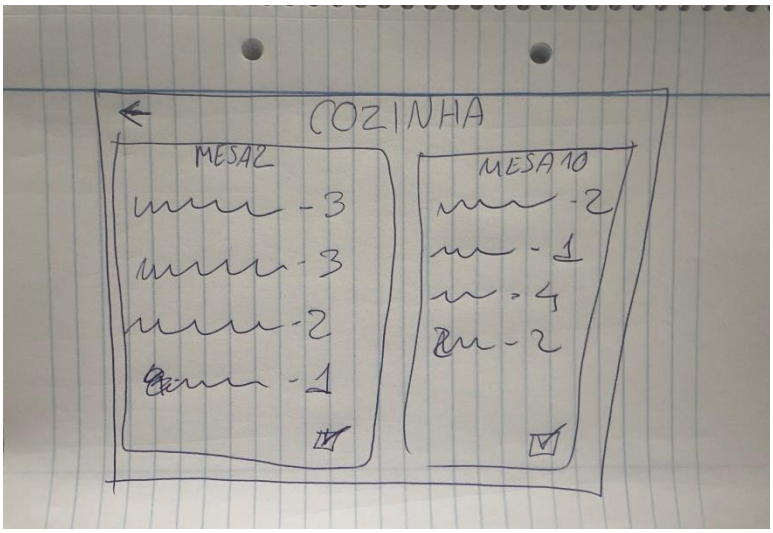


Figura 8 - Manuscritos Software Cozinha

A criação do software foi claramente um desafio. Tive que me familiarizar com uma linguagem nova e complexa, o que exigiu muitos tutoriais e pesquisas. No entanto, apesar das dificuldades, o resultado ficou bastante próximo da idealização inicial. Claro que houve algumas mudanças ao longo do caminho para aprimorar e otimizar o projeto, como ajustes na interface do utilizador e na arquitetura do sistema.

Além disso, alguns colegas de turma foram fundamentais para me ajudar a superar alguns problemas e questões, oferecendo dicas e perspetivas que enriqueceram o processo de desenvolvimento.

Durante o processo de criação, também aprendi a lidar com bugs e problemas de compatibilidade, o que exigiu muita dedicação e persistência para superá-los.

A documentação do código foi uma parte importante do projeto, garantindo que o código fosse claro e bem comentado para facilitar a compreensão.

4. Recursos Técnicos

4.1 Visual Studio Code

Figura 10 - Visual Studio Code

O Visual Studio Code, é um ambiente de desenvolvimento integrado de código aberto, altamente popular e amplamente utilizado pelos desenvolvedores de software. Desenvolvido pela Microsoft, é uma ferramenta robusta e flexível que atende às necessidades diversificadas dos desenvolvedores modernos, proporcionando uma experiência de programação eficiente e enriquecedora. A combinação de facilidade de uso, extensibilidade e recursos avançados o torna uma escolha preferida para muitos profissionais da área de desenvolvimento de software.

A preferência pelo Visual Studio Code reside na sua interface intuitiva, flexibilidade através de extensões, e integração eficaz com Git e serviços em nuvem. Sua comunidade ativa e atualizações frequentes evidenciam um compromisso constante com a inovação, tornando-o uma escolha destacada para desenvolvedores em busca de eficiência e versatilidade no desenvolvimento de software.

4.2 Firebase

Figura 11 - Firebase

A Firebase é uma plataforma de desenvolvimento de aplicativos móveis e web fornecida pelo Google. Ela oferece uma variedade de serviços e ferramentas para ajudar os desenvolvedores a criar, melhorar e dimensionar os seus projetos com mais facilidade e eficiência. Isso inclui recursos como banco de dados em tempo real, autenticação de utilizadores, armazenamento em nuvem, hospedagem na web, funções em nuvem, análise e monitoramento. Em resumo, a Firebase simplifica o desenvolvimento de aplicativos, permitindo que os desenvolvedores se concentrem na criação de experiências excecionais para os utilizadores, sem se preocupar com infraestrutura complexa.

4.3 Node.js

Figura 12- Node.js

O Node.js é uma plataforma JavaScript utilizada para criar aplicativos escaláveis. No meu projeto, aproveitei o Node.js para simplificar a instalação do ambiente de desenvolvimento do Flutter, agilizando a configuração de ferramentas essenciais como o Flutter SDK, Android Studio e Xcode. Isso permitiu uma configuração mais eficiente e rápida do ambiente de desenvolvimento necessário para começar a trabalhar com Flutter.

4.4 Android Studio

Figura 13 - Android Studio

O Android Studio é um ambiente de desenvolvimento integrado (IDE) criado especificamente para desenvolver aplicações para dispositivos Android. Ele fornece ferramentas abrangentes para escrever, depurar e testar aplicações, além de oferecer recursos como emuladores de dispositivos para simular diferentes configurações de hardware. O Android Studio também é compatível com várias linguagens de programação, incluindo Flutter (a linguagem que utilizei) Java e Kotlin, e oferece suporte para a integração de bibliotecas e frameworks populares. Em resumo, é uma plataforma essencial para os desenvolvedores de aplicações Android.

4.5 Pintrest

Figura 14- Pintrest

O Pinterest é uma plataforma de mídia social centrada em imagens, onde os utilizadores podem descobrir e salvar ideias em seus quadros virtuais. Utilizei o Pinterest como fonte de inspiração para o design do meu software, explorando uma variedade de categorias, desde design de interface até paletas de cores e layouts criativos. Através da ampla gama de imagens e ideias disponíveis, encontrei inspiração para criar uma experiência de utilizador visualmente atraente e intuitiva para o meu software.

Os Fundos Europeus mais próximos de si.

5. Linguagens de Programação

5.1 Dart

Figura 15 - Dart

Dart é uma linguagem de programação desenvolvida pelo Google, reconhecida pela sua integração com o framework Flutter. Ela é moderna, eficiente e fácil de aprender, com uma sintaxe limpa e simples. Dart é orientada a objetos e suporta tipagem estática, proporcionando eficiência e deteção de erros de tipo em tempo de compilação. Reconhecida por sua performance, é frequentemente usada no desenvolvimento de uma variedade de aplicativos, incluindo aplicativos móveis com Flutter, que oferece uma abordagem moderna para criar interfaces de usuário nativas e responsivas.

5.2 Flutter

Figura 16 - Flutter

Flutter é um framework de código aberto do Google para criar interfaces de usuário nativas para dispositivos móveis, web e desktop a partir de uma única base de código. Ele usa a linguagem de programação Dart e oferece uma experiência de desenvolvimento eficiente, rápida e consistente em várias plataformas. Com o Flutter, os desenvolvedores podem criar aplicativos com desempenho rápido e interfaces de usuário atrativas, aproveitando um conjunto abrangente de ferramentas e widgets personalizáveis.

5.3 Experiência

Figura 17- Flutter / Dart

Optei por utilizar a linguagem de programação Flutter, uma extensão do Dart, para desenvolver o meu software, influenciado pelas recomendações de colegas e pela crescente popularidade da plataforma. Flutter é uma estrutura de código aberto desenvolvida pelo Google que permite a criação de aplicações nativas para dispositivos móveis e desktop a partir de uma única base de código.

Dart, a linguagem de programação por trás do Flutter, é uma linguagem moderna, orientada a objetos e orientada a eventos, conhecida por sua sintaxe limpa e simples. Ela é projetada para ser rápida, previsível e eficiente para o desenvolvimento de aplicativos.

A escolha do Flutter e do Dart para meu projeto foi motivada pela capacidade de criar interfaces de utilizadores ricas e responsivas de forma rápida e eficiente. Além disso, o Flutter oferece um conjunto robusto de widgets personalizáveis e ferramentas de desenvolvimento poderosas que facilitam a criação de aplicações visualmente atraentes e funcionais.

Ao optar pelo Flutter, também levei em consideração sua capacidade de oferecer uma experiência consistente em diferentes plataformas, permitindo que meu software fosse facilmente portado para dispositivos móveis e desktop, economizando tempo e recursos de desenvolvimento. Em resumo, escolhi o Flutter e o Dart para meu projeto devido à sua eficiência, versatilidade e à recomendação positiva de colegas.

Os Fundos Europeus mais próximos de si.
Cofinanciado pela União Europeia

6. Software

6.1 Login

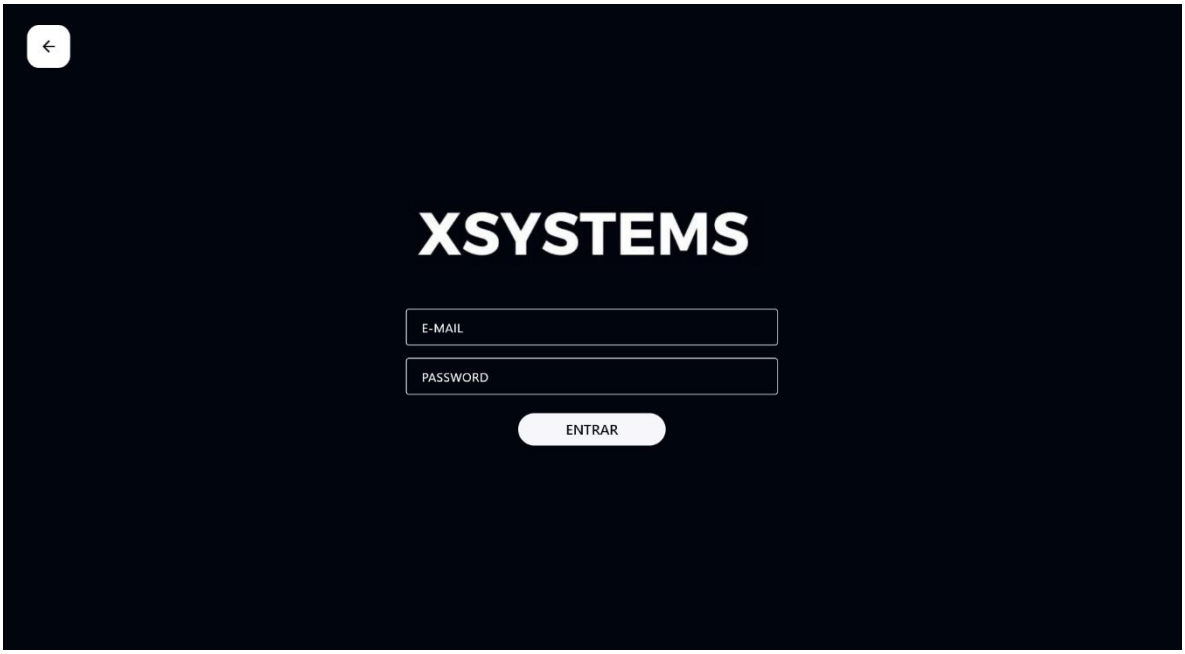


Figura 20 - Página de Login

```dart
// Campo de E-mail
Container(
  constraints: const BoxConstraints(maxWidth: 500),
  padding: const EdgeInsets.all(8.0),
  child: TextField(
    controller: _emailController,
    style: const TextStyle(color: Colors.white),
    decoration: const InputDecoration(
      labelText: 'E-MAIL',
      labelStyle: TextStyle(color: Colors.white),
      focusedBorder: OutlineInputBorder(
        borderSide: BorderSide(color: Colors.white),
      ), // OutlineInputBorder
      enabledBorder: OutlineInputBorder(
        borderSide: BorderSide(color: Colors.white),
      ), // OutlineInputBorder
      contentPadding:
          EdgeInsets.symmetric(vertical: 10, horizontal: 20),
    ), // InputDecoration
  ), // TextField
), // Container
```

Figura 19 - Campo E-mail / Linguagem Flutter

```dart
// Tenta realizar o login com Firebase Auth
await FirebaseAuth.instance.signInWithEmailAndPassword(
  email: _emailController.text,
  password: _passwordController.text,
);
// Se bem-sucedido, redireciona para a página de usuários
redirecionar();
catch (e) {
// Em caso de erro de autenticação, exibe uma mensagem de erro
final snackBar = SnackBar(
  elevation: 10,
  behavior: SnackBarBehavior.floating,
  backgroundColor: Colors.transparent,
  content: AwesomeSnackbarContent(
    title: 'ERRO DE AUTENTICAÇÃO !',
    message: 'VERIFIQUE AS SUAS CREDÊNCIAIS E TENTE NOVAMENTE !',
    contentType: ContentType.failure,
  ), // AwesomeSnackbarContent
); // SnackBar
ScaffoldMessenger.of(context)
  ..hideCurrentSnackBar()
  ..showSnackBar(snackBar);
```

Figura 18 - Função de Login / Linguagem Flutter

A interface de login é projetada para ser intuitiva e eficiente, sendo constituída apenas um campo para inserção de e-mail e outro para password, ambos integrados à base de dados no Firebase. Quando um utilizador tenta acessar o sistema, esses campos verificam se ele já possui um registo na base de dados. Se as credenciais forem reconhecidas, o utilizador é automaticamente redirecionado para a página seguinte. No entanto, se não houver correspondência, uma mensagem de erro é exibida na tela. Esta mensagem orienta o utilizador a verificar e reconfirmar o e-mail e a senha digitados. Em casos onde o utilizador ainda não está registado, a mensagem também oferece a opção de registrar o estabelecimento na base de dados, garantindo assim que apenas utilizadores autorizados tenham acesso ao sistema.

6.2 Utilizadores

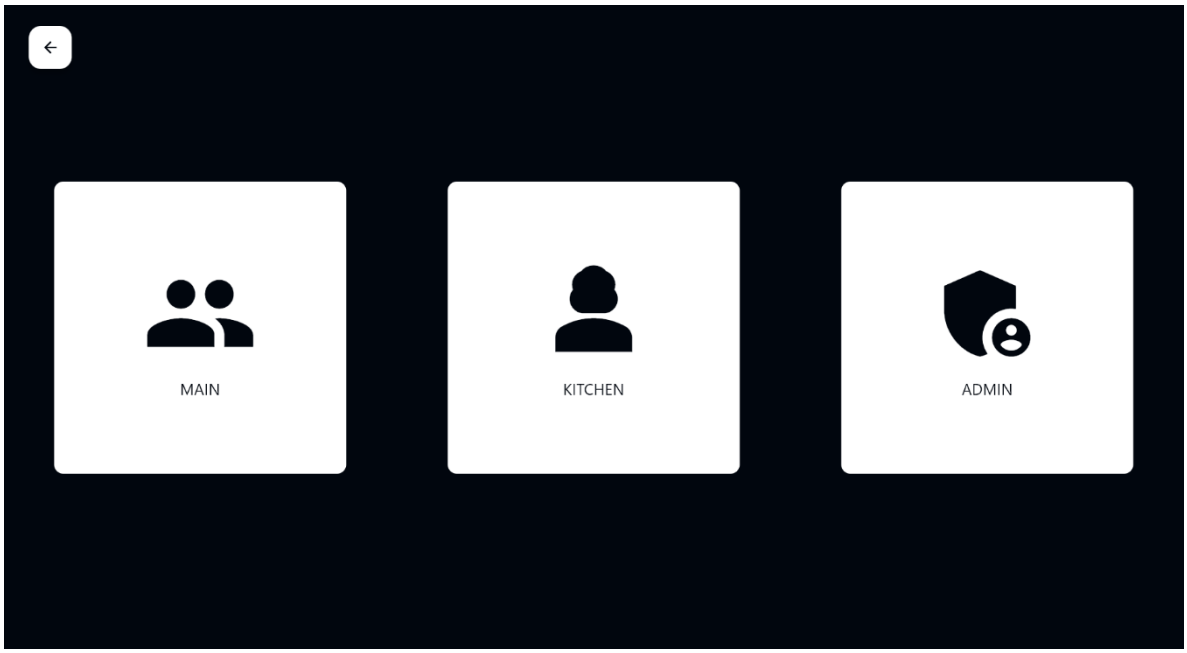


Figura 21 - Página de Utilizadores

```
// Botão "COZINHA"
InkWell(
  onTap: () {
    // Navega para a tela da cozinha
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => const KitchenScreen()),
    );
  },
  child: Container(
    // Container do botão
    width: 380,
    height: 380,
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(12.0),
    ), // BoxDecoration
    child: const Column(
      // Conteúdo do botão
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        // Ícone
        Icon(
          Icons.person_3,
          size: 150,
          color: Color.fromRGBO(1, 6, 14, 1),
        ), // Icon
        SizedBox(height: 8.0),
        // Texto
        Text(
          'COZINHA',
          style: TextStyle(
            fontSize: 20,
            color: Color.fromRGBO(1, 6, 14, 1),
          ), // TextStyle
        ), // Text
      ],
    ), // Column
  ), // Container
), // InkWell
```

Figura 22 - Botão Cozinha / Linguagem Flutter

Temos uma página simples com três botões: "PRINCIPAL", "COZINHA" ou "ADMINISTRADOR", que representam as funções principais do software. Com um design simples, esta página permite selecionar em qual dos utilizadores ou pontos do restaurante desejo entrar.

Além disso, há um botão simples no canto superior esquerdo que nos permite retornar para a página de login com facilidade.

6.2.1 Principal

O perfil "PRINCIPAL" tem a função de gerenciar a sala onde o software é aplicado. Isso inclui visualizar as mesas ocupadas ou disponíveis, rever os pedidos das mesas, realizar pagamentos e acessar um menu de consulta e serviços adicionais, como take-away.

Além disso, é possível consultar o histórico de pedidos, acompanhar o processo do serviço através de uma dashboard que mostra o número de pedidos realizados, o faturamento até o momento e realizar o fechamento de caixa, zerando o sistema diariamente.

6.2.1.1 Sala

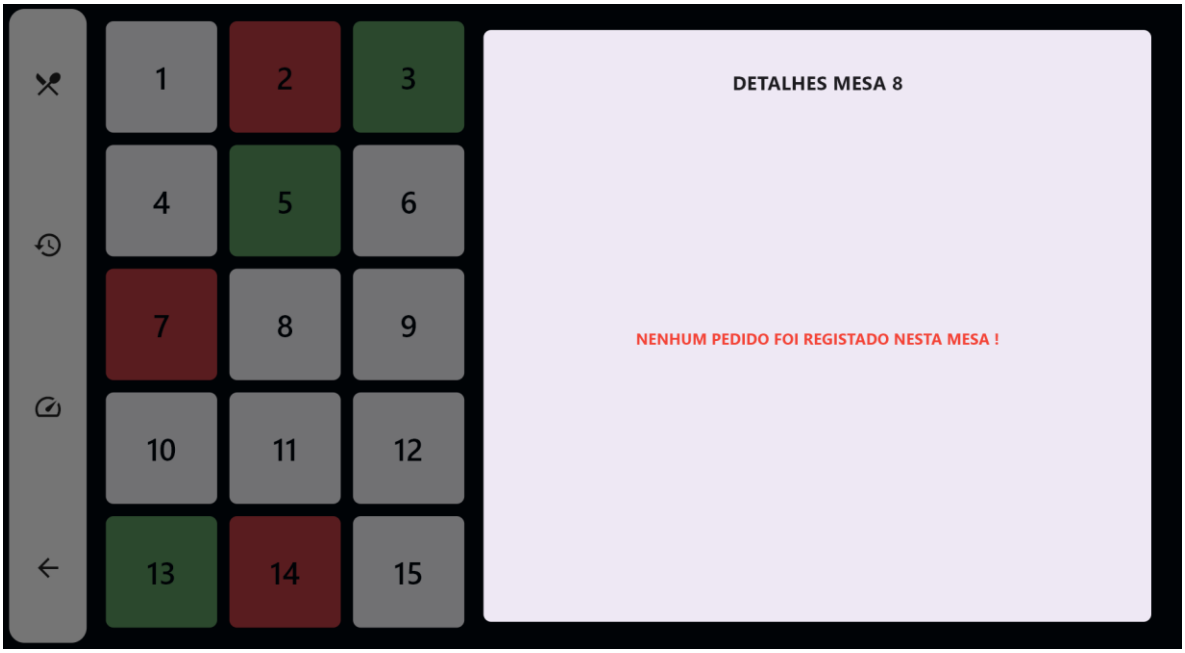


Figura 23 - Página Principal

```
Padding(
  padding: const EdgeInsets.only(left: 110),
  child: Container(
    padding: const EdgeInsets.all(16.0),
    color: const Color.fromRGBO(1, 6, 14, 1),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.start,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: List.generate(5, (rowIndex) {
        return Row(
          children: List.generate(3, (colIndex) {
            final mesaIndex = rowIndex * 3 + colIndex + 1;
            final mesaId = 'mesa_$mesaIndex';
            return Padding(
              padding: const EdgeInsets.all(8.0),
              child: SizedBox(
                width: 145,
                height: 145,
                child: StreamBuilder<QuerySnapshot>(
                  stream: FirebaseFirestore.instance.collection('mesas').doc(mesaId).collection('pedidos').snapshots(),
                  builder: (context, snapshot) {
                    Color buttonColor = Colors.white;
                    if (snapshot.hasData) {
                      final pedidos = snapshot.data!.docs;
                      for (var pedido in pedidos) {
                        final status = pedido['status'];
                        if (status == 'pendente') {
                          buttonColor = const Color.fromARGB(255, 200, 62, 62);
                          break;
                        } else if (status == 'enviado') {
                          buttonColor = const Color.fromARGB(255, 93, 161, 96);
                          break;
                        }
                      }
                    }
```

Figura 24 - Consulta Firebase / Linguagem Flutter

```
final pedidos = snapshot.data!.docs;
final itensQuantidades = <String, int>{};
final itensPrecos = <String, double>{};
double totalPreco = 0.0;
for (var pedido in pedidos) {
  final pedidoData = pedido.data() as Map<String, dynamic>;
  if (pedidoData.containsKey('itens')) {
    final itensPedido = pedidoData['itens'] as List<dynamic>;
    for (var item in itensPedido) {
      final nome = item['nome'] as String;
      final quantidade = item['quantidade'] as int?;
      final preco = item['preco'];
      if (quantidade != null && preco != null) {
        final precoUnitario = preco is int
            ? preco.toDouble()
            : preco is double
                ? preco
                : double.tryParse(preco.toString()) ?? 0.0;
        if (itensQuantidades.containsKey(nome)) {
          itensQuantidades[nome] = itensQuantidades[nome]! + quantidade;
        } else {
          itensQuantidades[nome] = quantidade;
        }
        itensPrecos[nome] = precoUnitario;
        totalPreco += (quantidade * precoUnitario);
      }
    }
  }
}

if (itensQuantidades.isEmpty) {
  return const Text('Nenhum item encontrado.');
}
final rows = itensQuantidades.entries.map((entry) {
  final nome = entry.key;
  final quantidade = entry.value;
  final precoUnitario = itensPrecos[nome] ?? 0.0;
  final precoTotal = quantidade * precoUnitario;
```

Figura 25 - Função Cálculos Pedido / Linguagem Flutter

Na página principal, apresentamos a numeração das mesas até 15, onde cada botão representa uma mesa específica. Os botões são visualmente diferenciados: branco indica que a mesa está vazia, vermelho indica que está ocupada e o pedido está em preparação, e verde indica que a mesa está ocupada, o pedido foi atendido e os clientes estão desfrutando dos itens solicitados.

Os Fundos Europeus mais próximos de si.

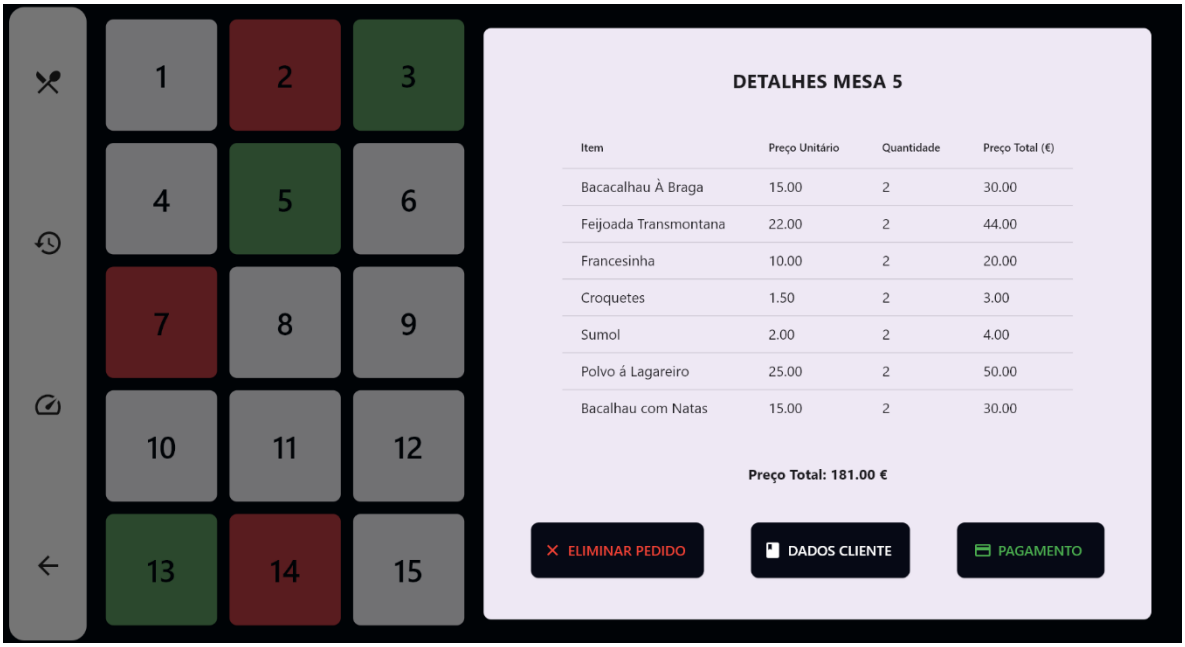


Figura 26- Página Principal

Ao selecionar uma mesa, uma caixa de texto à direita da tela mostra o número da mesa selecionada. Abaixo dessa caixa, uma tabela detalha os itens pedidos, exibindo seus nomes, preços unitários, quantidades e o preço total. Essa tabela oferece uma visão abrangente dos pedidos e suas composições, proporcionando uma experiência informativa ao usuário.

No final da tabela, encontramos três botões para realizar ações específicas: "Eliminar Pedido", que permite remover o pedido sem deixar registo no histórico ou faturamento, "Dados Cliente", para inserir informações como o NIF do cliente, e "Pagamento".

Os Fundos Europeus mais próximos de si.

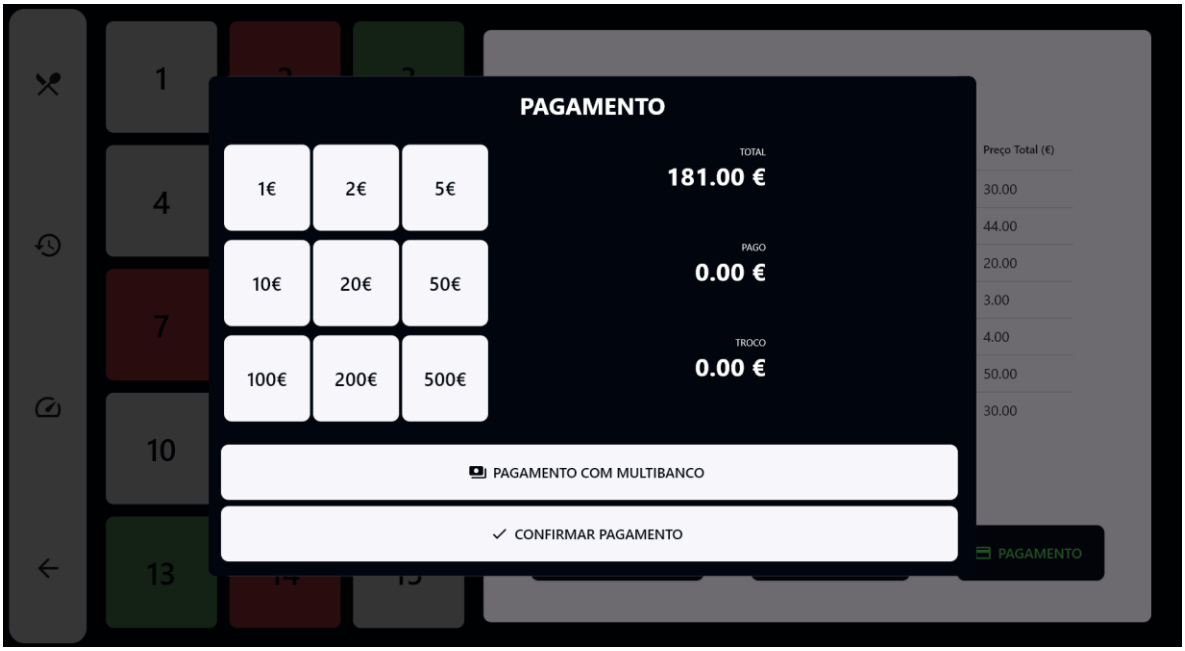


Figura 29 -Página Pagamento

```
@override
Widget build(BuildContext context) {
  return AlertDialog(
    elevation: 0,
    insetPadding: EdgeInsets.zero,
    contentPadding: EdgeInsets.zero,
    content: Container(
      width: widget.width,
      height: widget.height,
      decoration: BoxDecoration(
        color: const Color.fromRGBO(1, 6, 14, 1),
        borderRadius: BorderRadius.circular(10.0),
      ), // BoxDecoration
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            const Text(
              'PAGAMENTO',
              style: TextStyle(fontWeight: FontWeight.bold, fontSize: 30, color: Colors.white),
              textAlign: TextAlign.center,
            ), // Text
            const SizedBox(height: 30),
            Padding(
              padding: const EdgeInsets.only(right: 250.0),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  _buildPaymentRow([1, 2, 5]),
                  const Spacer(),
                  Column(
                    crossAxisAlignment: CrossAxisAlignment.end,
                    children: [
                      const Text(
                        'TOTAL',
                        style: TextStyle(fontSize: 12, color: Colors.white),
                      ), // Text
                      Text(
                        '${widget.total.toStringAsFixed(2)} €',
                        style: const TextStyle(fontWeight: FontWeight.bold, fontSize: 32, color: Colors.white),
                      ), // Text
                    ],
```

Figura 27 - Dialog Pagamento / Linguagem Flutter

```
FirebaseFirestore.instance
  .collection('mesas')
  .doc(mesaId)
  .collection('pedidos')
  .get()
  .then((querySnapshot) {
List<Map<String, dynamic>> items = [];
for (var doc in querySnapshot.docs) {
  items.add(doc.data());
  doc.reference.delete();
}

String estadoPagamento;

if (paymentType == 'multibanco') {
  estadoPagamento = 'multibanco';
} else {
  estadoPagamento = 'numerário';
}

FirebaseFirestore.instance.collection('historico_pedidos').add({
  'total': widget.total,
  'pagamento': _paymentAmount,
  'troco': _change,
  'timestamp': Timestamp.now(),
  'estado_pagamento': estadoPagamento,
  'mesa': mesaId,
  'itens': items, // Incluindo os itens do pedido
}).then((value) {
  print('Pedido registrado com sucesso!');
}).catchError((error) {
  print('Erro ao registrar pedido: $error');
});
}).catchError((error) {
  print("Erro ao limpar pedidos da mesa: $error");
});
```

Figura 28 - Função Dados Histórico / Linguagem Flutter

Ao clicar em "Pagamento", o valor total a ser pago é exibido. Para facilitar a escolha do valor a ser pago, existem botões rápidos com valores comumente utilizados. Após inserir o valor, o sistema calcula o troco automaticamente.

Ao confirmar o pagamento, ou optar pelo pagamento com multibanco, o pedido é efetivamente fechado. A mesa é então limpa e zerada, e os dados são enviados para o histórico, incluindo o total, a data, a hora, o método de pagamento e o número da mesa. Este processo mantém um registro detalhado das transações realizadas no restaurante, contribuindo para uma gestão eficiente e organizada do estabelecimento.

6.2.1.2 Histórico

Figura 30- Página Histórico Pedidos

```dart
// Constrói a lista de histórico de pedidos
Widget _buildHistoricoPedidos() {
  return StreamBuilder<QuerySnapshot>(
    stream: FirebaseFirestore.instance.collection('historico_pedidos').orderBy('timestamp', descending: true).snapshots(),
    builder: (context, snapshot) {
      // Verifica o estado da conexão
      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(child: CircularProgressIndicator());
      }
      // Verifica se não há dados ou se a lista está vazia
      if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
        return const Center(
          child: Text(
            'AINDA NÃO HOUVE NENHUM FATURAMENTO',
            style: TextStyle(fontSize: 30, fontWeight: FontWeight.bold, color: Colors.white),
          ), // Text
        ); // Center
      }
      // Verifica se há erro
      if (snapshot.hasError) {
        return Center(
          child: Text(
            'Erro: ${snapshot.error}',
            style: const TextStyle(fontSize: 18.0, color: Colors.white),
          ), // Text
        ); // Center
      }
      // Se houver dados, constrói a lista
      final pedidos = snapshot.data!.docs;
      return ListView.builder(
        itemCount: pedidos.length,
        itemBuilder: (context, index) {
          final pedido = pedidos[index];
          final data = pedido.data() as Map<String, dynamic>;
          final formattedDate = DateFormat('dd/MM/yyyy').format(data['timestamp'].toDate());
          final formattedTime = DateFormat('HH:mm').format(data['timestamp'].toDate());
          final isMultibanco = data['estado_pagamento'] == 'multibanco';
```

Figura 32- Lista Pedidos / Linguagem Flutter

```dart
// Lista de histórico geral
Expanded(
  child: StreamBuilder<QuerySnapshot>(
    stream: FirebaseFirestore.instance.collection('historico_geral').orderBy('timestamp', descending: true).snapshots(),
    builder: (context, snapshot) {
      // Verifica o estado da conexão
      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(child: CircularProgressIndicator());
      }
      // Verifica se não há dados ou se a lista está vazia
      if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
        return const Center(
          child: Text(
            'AINDA NÃO HOUVE NENHUM FATURAMENTO NO HISTÓRICO GERAL',
            style: TextStyle(fontSize: 30, fontWeight: FontWeight.bold, color: Colors.white),
          ), // Text
        ); // Center
      }
      // Verifica se há erro
      if (snapshot.hasError) {
        return Center(
          child: Text(
            'Erro: ${snapshot.error}',
            style: const TextStyle(fontSize: 18.0, color: Colors.white),
          ), // Text
        ); // Center
      }
      // Filtra os dados pela data selecionada
      final historicoGeral = snapshot.data!.docs.where((doc) {
        if (_selectedDate == null) return true;
        final DateTime docDate = (doc['timestamp'] as Timestamp).toDate();
        return docDate.year == _selectedDate!.year &&
            docDate.month == _selectedDate!.month &&
            docDate.day == _selectedDate!.day;
      }).toList();
```

Figura 31 - Lista Histórico Geral / Linguagem Flutter

Na página "Histórico de Pedidos", é possível consultar os pedidos fechados ao longo do dia. Nesta aba, são listados todos os pedidos que foram servidos durante o dia (antes do fechamento de caixa), mostrando o total da conta, a data e hora do pedido, o método de pagamento e o número da mesa associada a cada pedido.

Figura 33 -Página Histórico Geral

Ao selecionar a opção "Histórico Geral", temos a possibilidade de consultar os pedidos por dias específicos. Podemos escolher a data desejada e visualizar todos os pedidos realizados naquele dia, incluindo o total da conta, a data e hora do pedido, o método de pagamento e o número da mesa associada a cada pedido, os pedidos passam para o Histórico Geral quando o fecho de caixa é realizado, saindo de “Histórico Pedidos e indo para “Histórico Geral”

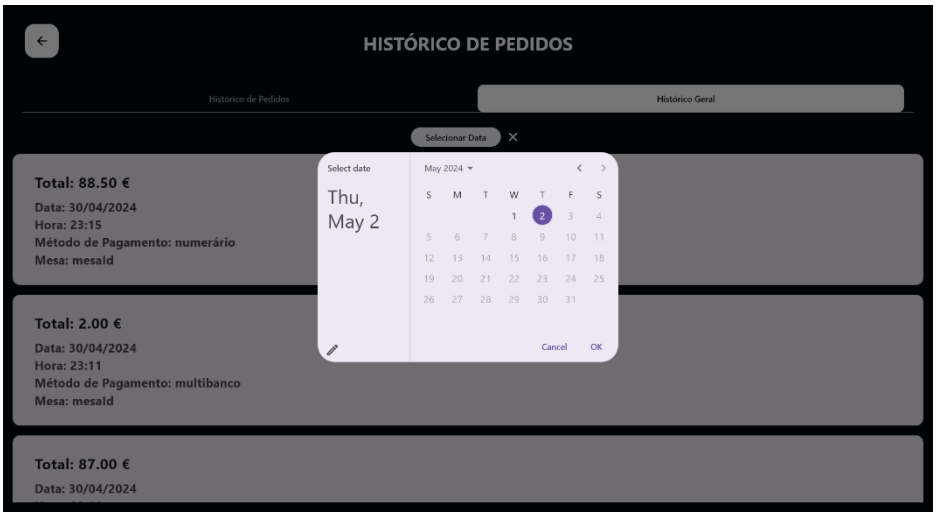


Figura 34- Página Histórico Geral

```
// Filtra os dados pela data selecionada
final historicoGeral = snapshot.data!.docs.where((doc) {
  if (_selectedDate == null) return true;
  final DateTime docDate = (doc['timestamp'] as Timestamp).toDate();
  return docDate.year == _selectedDate!.year &&
      docDate.month == _selectedDate!.month &&
      docDate.day == _selectedDate!.day;
}).toList();
// Constrói a lista
return ListView.builder(
  itemCount: historicoGeral.length,
  itemBuilder: (context, index) {
    final pedido = historicoGeral[index];
    final data = pedido.data() as Map<String, dynamic>;
    final formattedDate = DateFormat('dd/MM/yyyy').format(data['timestamp'].toDate());
    final formattedTime = DateFormat('HH:mm').format(data['timestamp'].toDate());
```

Figura 35 - Filtro Data / Linguagem Flutter

6.2.1.3 Dashboard

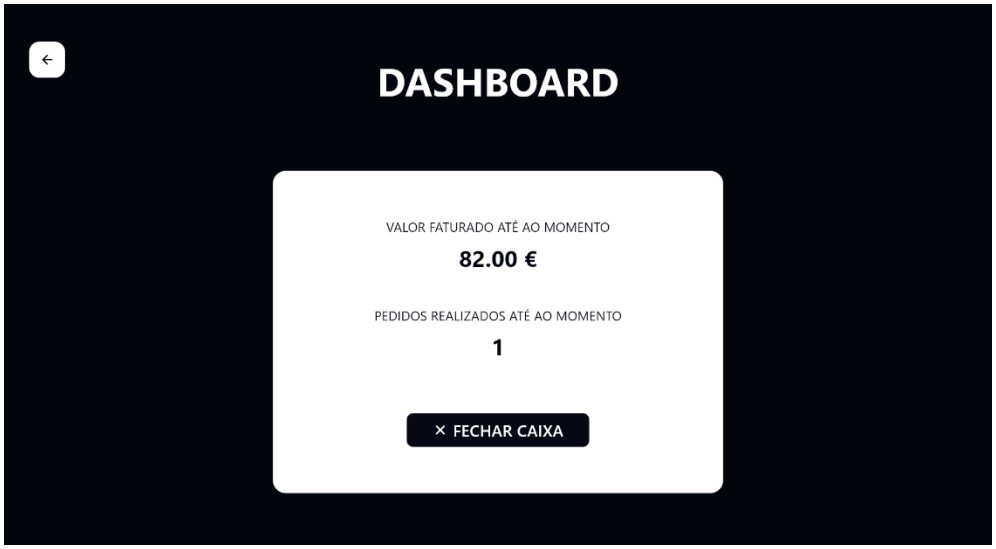


Figura 36 -Página Dashboard

```dart
// Definine os valores de faturamento e quantidade de pedidos
double totalFaturado = 0.0;
int quantidadePedidos = 0;

if (snapshot.hasData && snapshot.data!.docs.isNotEmpty) {
  // Calcular o valor faturado e a quantidade de pedidos
  totalFaturado = 0.0;
  quantidadePedidos = snapshot.data!.docs.length;

  for (var pedido in snapshot.data!.docs) {
    final data = pedido.data() as Map<String, dynamic>;
    totalFaturado += data['total'] as double;
  }
```

Figura 37- Função Calcular Faturamento / Linguagem Flutter

A página de dashboard exibe o valor de faturamento desde o último fecho de caixa, mostrando-o na tela para consulta a cada pedido encerrado. Além disso, há um botão para realizar o fechamento da caixa, onde todos os valores, históricos e tabelas são zerados após o fechamento para uma nova contagem. Ao clicar em "Fechar caixa", surge uma caixa de diálogo com o valor do faturamento e uma confirmação se realmente deseja fechar a caixa.

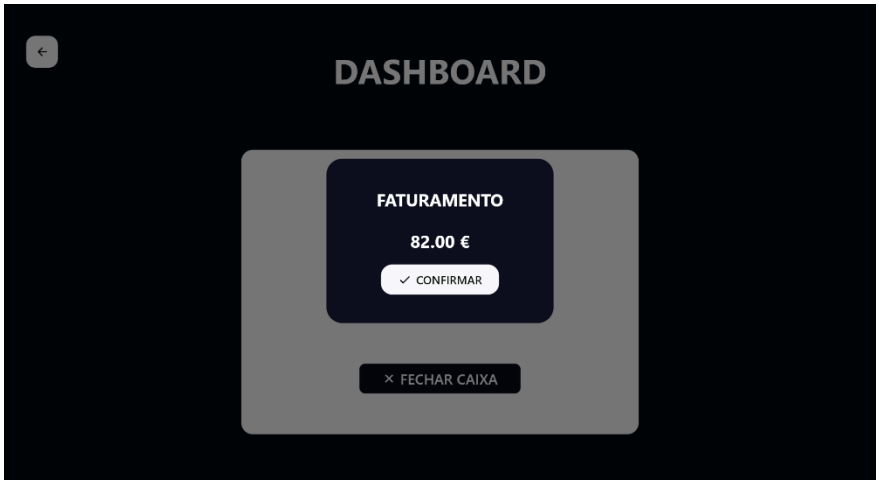


Figura 38- Página Fechar Caixa

6.2.1.4 Menu / Take-Away

Figura 39 -Página Take-Away

```
      controller: _tabController,
      labelColor: const Color.fromRGBO(1, 6, 14, 1),
      unselectedLabelColor: Colors.grey,
      indicator: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(10),
      ), // BoxDecoration
      tabs: const [
        Tab(text: '                    Entradas                    '),
        Tab(text: '                    Pratos Principais                    '),
        Tab(text: '                    Sobremesas                    '),
        Tab(text: '                    Bebidas                    '),
      ],
    ), // TabBar
  ), // Container
  Expanded(
    child: Row(
      children: [
        Expanded(
          child: TabBarView(
            controller: _tabController,
            children: [
              _buildMenuCategory('entradas'), // Entradas
              _buildMenuCategory('pratos principais'), // Pratos Principais
              _buildMenuCategory('sobremesas'), // Sobremesas
              _buildMenuCategory('bebidas'), // Bebidas
            ],
          ), // TabBarView
        ), // Expanded
      ],
    ), // Row
  ), // Expanded
 ],
), // Column
// Padding
```

```
// Função para construir a categoria do menu
Widget _buildMenuCategory(String categoryName) {
  return StreamBuilder(
    stream: FirebaseFirestore.instance.collection('itens').where('categoria', isEqualTo: categoryName).snapshots(),
    builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
      if (snapshot.hasError) {
        return Text('Erro: ${snapshot.error}');
      }

      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(child: CircularProgressIndicator());
      }

      List<DocumentSnapshot> sortedDocs = snapshot.data!.docs;
      sortedDocs.sort((a, b) => (a['nome'] as String).compareTo(b['nome'] as String));

      return GridView.count(
        crossAxisCount: 6, // Organiza em 6 colunas
        padding: const EdgeInsets.all(10),
        childAspectRatio: 181 / 181, // Proporção definida para largura e altura iguais
        mainAxisSpacing: 10,
        crossAxisSpacing: 10,
        children: sortedDocs.map((DocumentSnapshot document) {
          Map<String, dynamic> data = document.data() as Map<String, dynamic>;
          String nome = data['nome'];
          String preco = data['preco']; // Preco unitário diretamente do Firestore
```

Figura 40 - Cria Categorias / Linguagem Flutter

Figura 41 - Navegação Categorias / Linguagem Flutter

Nesta página, podemos realizar pedidos, seja para o balcão ou para take-away, que é o principal propósito desta página. Além disso, ela serve como uma consulta de menu ou preços. Aqui são listados todos os itens da base de dados, organizados por categorias como "bebidas", "entradas", "pratos principais" e "sobremesas", ao clicar no item é possível adicioná-lo ao pedido para mais tarde realizar o pagamento.

Ao clicar no ícone do pedido no canto superior direito, abre-se um campo de texto onde podemos consultar os itens adicionados ao pedido, mostrados em uma tabela organizada por Nome, Preço Unitário, Quantidade e Preço Total. No final da tabela, há o total do pedido e dois botões: "Dados Cliente" para adicionar o NIF do cliente e "Pagamento".

```dart
// Abre o diálogo do carrinho de compras
showDialog(
  context: context,
  builder: (BuildContext context) {
    return StatefulBuilder(
      builder: (BuildContext context, StateSetter setState) {
        return AlertDialog(
          title: Text('Itens do Pedido (${orderItems.length})'),
          content: SingleChildScrollView(
            child: DataTable(
              columns: const [
                DataColumn(label: Text('Item')),
                DataColumn(label: Text('Preço Unitário')),
                DataColumn(label: Text('Quantidade')),
                DataColumn(label: Text('Preço Total')),
                // Adicionei uma coluna extra para o botão de remover
                DataColumn(label: Text('Remover')),
              ],
              rows: [
                ...orderItems.map((item) {
                  final itemName = item['nome'];
                  final itemPrice = item['precoUnitario'];
                  final itemQuantity = item['quantidade'];
                  final totalPrice = double.parse(itemPrice) * itemQuantity;
                  return DataRow(cells: [
                    DataCell(Text(itemName)),
                    DataCell(Text(itemPrice)),
                    DataCell(Text(itemQuantity.toString())),
                    DataCell(Text(totalPrice.toStringAsFixed(2))),
                    DataCell(IconButton(
                      icon: const Icon(Icons.delete),
                      onPressed: () {
                        setState(() {
                          removeItemFromOrder(itemName); // Remove o item da lista
                        });
                      },
                    )), // IconButton // DataCell
                  ]); // DataRow
                }).toList(),
                DataRow(cells: [
                  const DataCell(Text('Total', style: TextStyle(fontWeight: FontWeight.bold))),
                  const DataCell(Text('')),
                  const DataCell(Text('')),
                  DataCell(Text(calculateTotalPrice().toStringAsFixed(2), style: const TextStyle(fontWeight: FontWeight.bold))),
                  const DataCell(Text('')),
                ]), // DataRow
              ],
            ), // DataTable
```

Figura 43 - Tabela Pedido / Linguagem Flutter

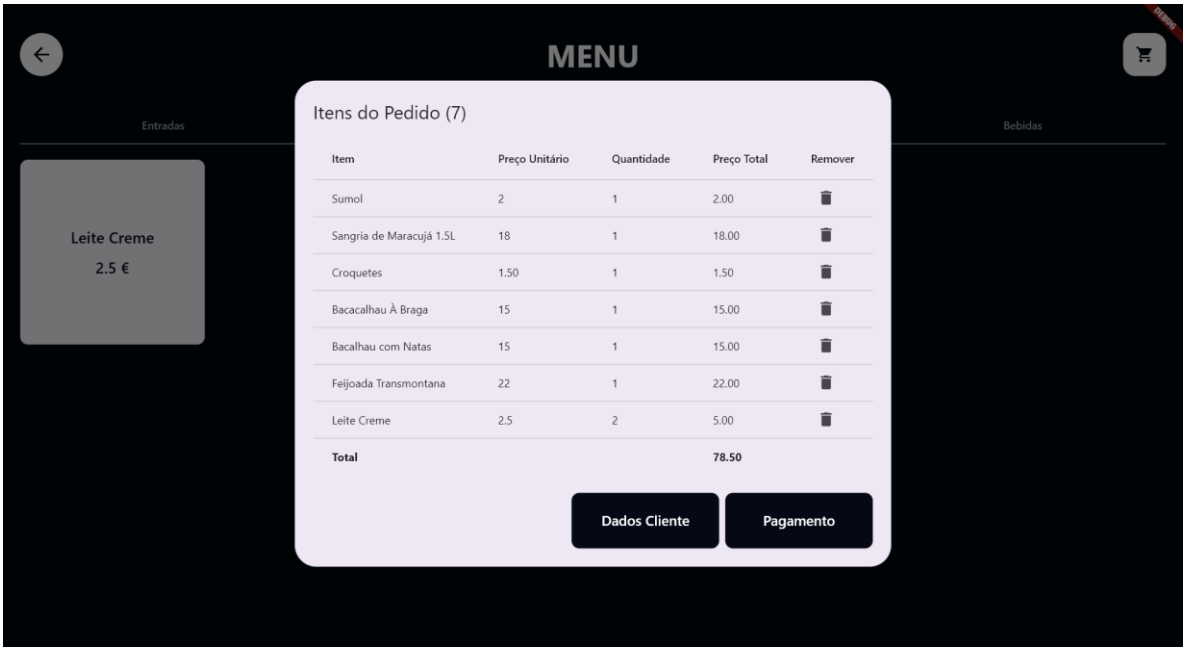


Figura 42 - Página de Pagamento Take-Away

Ao clicar em "Pagamento", o valor total a ser pago é exibido. Para facilitar a escolha do valor a ser pago, existem botões rápidos com valores comumente utilizados. Após inserir o valor, o sistema calcula o troco automaticamente.

Ao confirmar o pagamento, ou optar pelo pagamento com multibanco, o pedido é efetivamente fechado. A tabela é então limpa e zerada, e os dados são enviados para o histórico, incluindo o total, a data, a hora, o método de pagamento e o local do pedido no caso “take-away”.

Figura 44 - Página Pagamento

```
Padding(
  padding: const EdgeInsets.only(right: 250.0),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      _buildPaymentRow([10, 20, 50]),
      const Spacer(),
      Column(
        crossAxisAlignment: CrossAxisAlignment.end,
        children: [
          const Text(
            'PAGO',
            style: TextStyle(fontSize: 12, color: Colors.white),
          ), // Text
          Text(
            '${_paymentAmount.toStringAsFixed(2)} €',
            style: const TextStyle(fontWeight: FontWeight.bold, fontSize: 32, color: Colors.white),
          ), // Text
        ],
      ), // Column
      const SizedBox(width: 8),
```

Figura 45 - Botões Pagamento / Linguagem Flutter

6.2.2 Cozinha

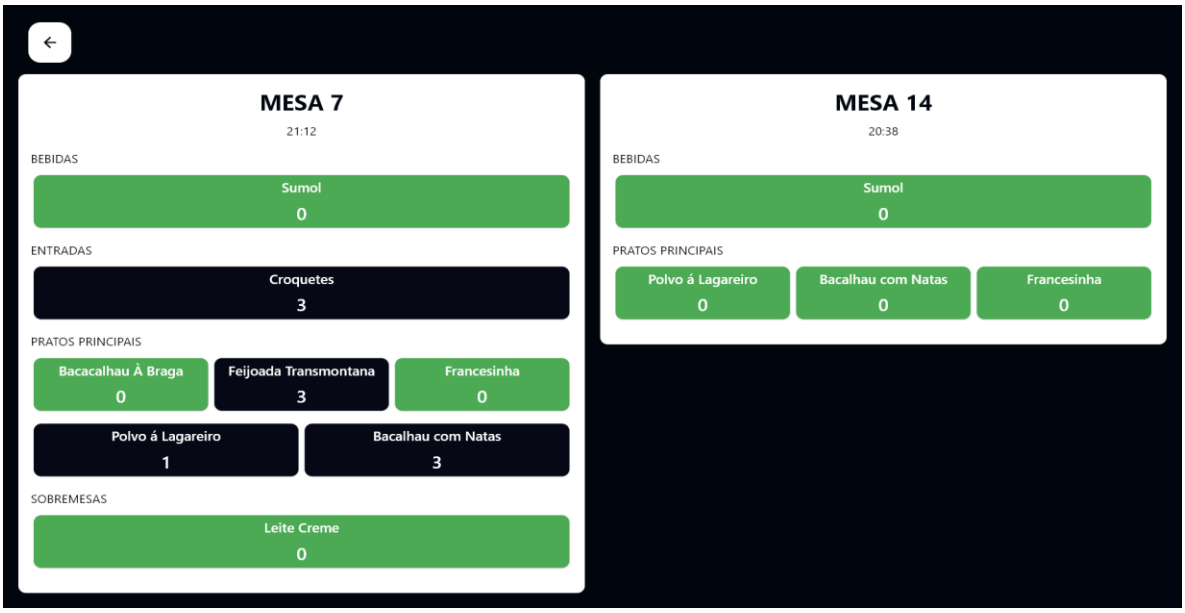


Figura 46 - Página da Cozinha

Na página da cozinha, acessamos os pedidos registados de acordo com a mesa. Ou seja, o pedido é feito na mesa e, após ser enviado, chega à tela da cozinha, onde aparece o número da mesa e a hora em que o pedido foi feito. A hora é crucial para gerenciar o tempo de espera do pedido.

Os itens estão divididos por categorias: bebidas, entradas, pratos principais e sobremesas. Cada item corresponde a um botão com a quantidade pedida logo abaixo do texto no mesmo botão. Quando cada unidade de um item é servida, clicamos no botão correspondente, e 1 é subtraído da quantidade. Quando a quantidade chega a zero, o botão fica verde, indicando que aquele item e todas as suas quantidades foram enviadas para a mesa.

Após todas as quantidades chegarem a zero ou todos os botões estiverem verdes, clicar em qualquer área do pedido o marca como finalizado e o retira da tela. Os pedidos são organizados em duas colunas na tela, em ordem temporal. Os primeiros pedidos realizados ficam no topo da tela, organizados de cima para baixo e da esquerda para a direita. Também há um botão para retornar à página anterior, a "página de utilizadores".

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color.fromRGBO(1, 6, 14, 1),
    body: StreamBuilder<QuerySnapshot>(
      stream: FirebaseFirestore.instance.collectionGroup('pedidos').snapshots(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const Center(child: CircularProgressIndicator());
        }
        if (snapshot.hasError) {
          return Center(child: Text('Erro: ${snapshot.error}', style: const TextStyle(color: Colors.white)));
        }
        _pedidos = snapshot.data!.docs.toList();

        // Filtrar pedidos com status diferente de 'enviado'
        _pedidos = _pedidos.where((pedido) => pedido['status'] != 'enviado').toList();

        if (_pedidos.isEmpty) {
          return const Center(child: Text('NÃO EXISTE NENHUM PEDIDO PARA CONFECÇÃO !', style: TextStyle (
            fontSize: 30, fontWeight: FontWeight.bold,color: Colors.white))); // TextStyle // Text // Center
        }

        // Ordenar os pedidos por horário de envio
        _pedidos.sort((a, b) {
          final horaEnvioA = (a['hora_envio'] as Timestamp).toDate();
          final horaEnvioB = (b['hora_envio'] as Timestamp).toDate();
          return horaEnvioA.compareTo(horaEnvioB);
        });

        // Dividir a lista de pedidos em duas colunas
        final List<QueryDocumentSnapshot> pedidosEsquerda = [];
        final List<QueryDocumentSnapshot> pedidosDireita = [];

        for (int i = 0; i < _pedidos.length; i++) {
          if (i % 2 == 0) {
            pedidosEsquerda.add(_pedidos[i]);
          } else {
            pedidosDireita.add(_pedidos[i]);
          }
        }
```

Figura 47 - Organização dos Pedidos / Linguagem Flutter

```dart
margin: const EdgeInsets.only(bottom: 20.0),
padding: const EdgeInsets.all(16.0),
width: MediaQuery.of(context).size.width / 2, // Define a largura como metade da largura da tela
decoration: BoxDecoration(
  color: Colors.white,
  borderRadius: BorderRadius.circular(10.0),
  boxShadow: [
    BoxShadow(
      color: Colors.black.withOpacity(0.2),
      blurRadius: 5.0,
      spreadRadius: 2.0,
      offset: const Offset(0, 3),
    ), // BoxShadow
  ],
), // BoxDecoration
child: Column(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Text(
      'MESA $mesaId',
      style: const TextStyle(fontSize: 30, fontWeight: FontWeight.bold, color: Color.fromRGBO(1, 6, 14, 1)),
    ), // Text
    const SizedBox(height: 8),
    Text(
      horaEnvio,
      style: const TextStyle(fontSize: 16, color: Colors.black),
    ), // Text
    const SizedBox(height: 15),
    Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        for (final categoria in ['bebidas', 'entradas', 'pratos principais', 'sobremesas'])
          if (itens.any((item) => item['categoria'] == categoria))
            ...[
              Text(
                categoria.toUpperCase(),
                style: const TextStyle(fontSize: 16, color: Colors.black),
              ), // Text
```

Figura 48 - Organização do Pedido / Linguagem Flutter

PESSOAS 2030 PORTUGAL 2030 Cofinanciado pela União Europeia SELO DE CONFORMIDADE EQAVET uma escola Rumos education Knowledge sharing
Os Fundos Europeus mais próximos de si.

6.2.3 Administrador

No perfil "ADMINISTRADOR", podemos gerenciar partes do software e da base de dados, como adicionar, remover ou editar itens. Também é possível reportar bugs e outros problemas.

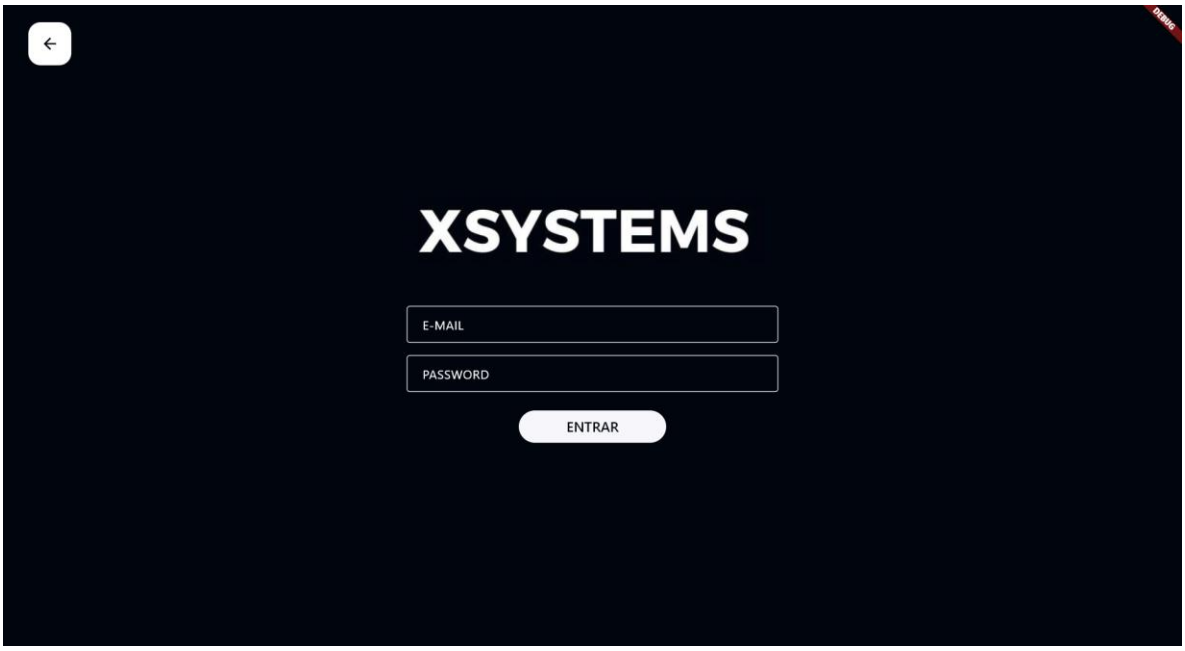


Figura 49 - Página Login Administrador

```dart
// Função para redirecionar para a página de administração
Future<void> redirecionar() async {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const Admin()),
  );
}

// Função para efetuar o login
Future<void> signIn(context) async {
  String email = _emailController.text.trim();
  String password = _passwordController.text.trim();

  // Verifica se os campos estão vazios
  if (email.isEmpty || password.isEmpty) {
    final snackBar = SnackBar(
      elevation: 10,
      behavior: SnackBarBehavior.floating,
      backgroundColor: Colors.transparent,
      content: AwesomeSnackbarContent(
        title: 'Erro ocorrido!',
        message: 'Por favor, preencha todos os dados.',
        contentType: ContentType.failure,
      ), // AwesomeSnackbarContent
    ); // SnackBar
    ScaffoldMessenger.of(context)
      ..hideCurrentSnackBar()
      ..showSnackBar(snackBar);
    return;
```

Figura 50 - Função Login / Linguagem Flutter

Na tela de administrador, há um login com e-mail e senha para restringir os acessos, já que nesta área é possível alterar grande parte do sistema. Após a confirmação do login, é possível acessar as configurações do sistema.

Após o login na página de administração, você é direcionado para a interface de administração, onde a função principal é registrar novos itens para o software. Você preenche campos como nome, preço e categoria para cada novo item. Ao clicar em "Confirmar", os itens são submetidos.

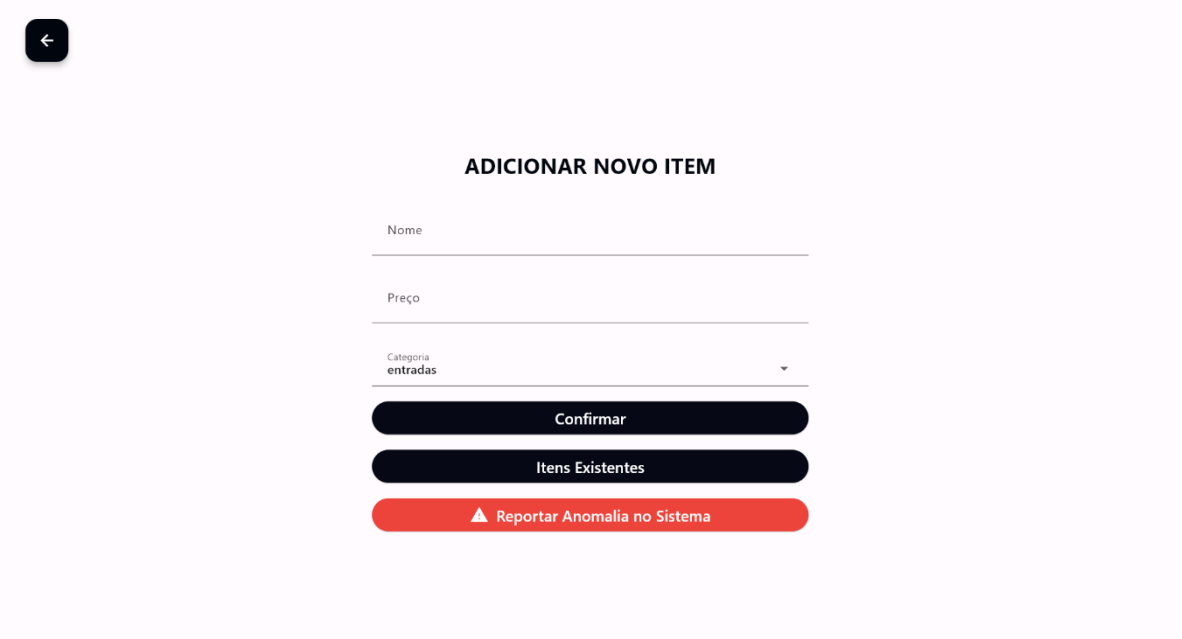


Figura 51 - Página Administrador

```
// Formulário de entrada de dados
Form(
  key: _formKey,
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: <Widget>[
      // Campo de entrada para o nome do item
      TextFormField(
        controller: _nameController,
        style: const TextStyle(fontSize: 20.0),
        decoration: const InputDecoration(
          labelText: 'Nome',
          contentPadding: EdgeInsets.symmetric(
            vertical: 15.0,
            horizontal: 20.0,
          ), // EdgeInsets.symmetric
        ), // InputDecoration
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Por favor, insira um nome para o item';
          }
          return null;
        },
      ), // TextFormField
      const SizedBox(height: 20.0),
      // Campo de entrada para o preço do item
      TextFormField(
        controller: _priceController,
        style: const TextStyle(fontSize: 20.0),
        decoration: const InputDecoration(
          labelText: 'Preço',
          contentPadding: EdgeInsets.symmetric(
            vertical: 15.0,
            horizontal: 20.0,
          ), // EdgeInsets.symmetric
        ), // InputDecoration
        keyboardType: TextInputType.number,
        validator: (value) {
```

Figura 52 - Formulário Novo Item / Linguagem Flutter

Cada item registrado recebe um identificador único, como "item_1", "item_2" e assim por diante. O sistema automaticamente atribui o próximo número disponível para garantir a otimização dos itens e da base de dados, evitando itens duplicados.

Além disso, você pode consultar os itens existentes na base de dados. Eles são listados, permitindo editar o nome, preço ou categoria conforme necessário. Isso é útil especialmente para atualizar preços. Também há a opção de excluir um item, por exemplo, se ele for descontinuado do restaurante.

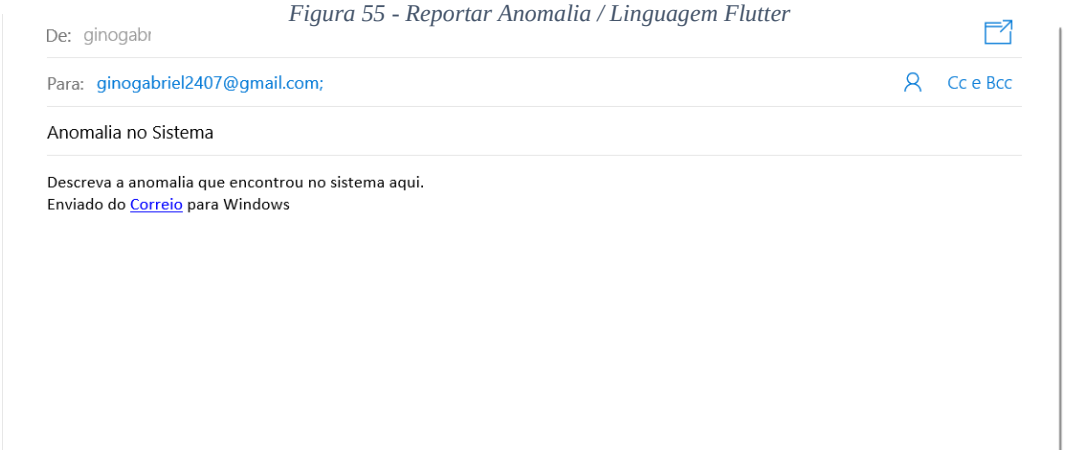
Figura 54 - Listagem de Itens Existentes

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Editar Item'),
    ), // AppBar
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          // Campo de edição para o nome do item
          TextFormField(
            controller: _nameController,
            decoration: const InputDecoration(labelText: 'Nome'),
          ), // TextFormField
          // Campo de edição para o preço do item
          TextFormField(
            controller: _priceController,
            decoration: const InputDecoration(labelText: 'Preço'),
          ), // TextFormField
          // Dropdown para seleção da categoria do item
          DropdownButtonFormField<String>(
            value: _category,
            items: <String>[
              'entradas',
              'pratos principais',
              'sobremesas',
              'bebidas'
            ].map<DropdownMenuItem<String>>((String value) { // <String>[]
              return DropdownMenuItem<String>(
                value: value.toLowerCase(),
                child: Text(value),
              ); // DropdownMenuItem
            }).toList(),
            onChanged: (value) {
              setState(() {
                _category = value!;
              });
            },
```

Figura 53- Editar Itens / Linguagem Flutter

Há um botão para reportar anomalias ou bugs. Ao clicar nele, você é redirecionado para enviar um e-mail para o endereço associado ao software, permitindo relatar o problema por escrito.

Figura 55 - Reportar Anomalia / Linguagem Flutter




```
// Função para lançar um email para reportar anomalia no sistema
Future<void> _launchEmail(String email) async {
  final Uri params = Uri(
    scheme: 'mailto',
    path: email,
    query: 'subject=Anomalia no Sistema&body=Descreva a anomalia que encontrou no sistema aqui.',
  );
  String url = params.toString();
  if (await canLaunch(url)) {
    await launch(url);
  } else {
    throw 'Could not launch $url';
  }
}
```

Figura 56 - Página Reportar Anomalia

No canto superior esquerdo, há um botão para retornar à página anterior, facilitando a navegação na interface de administração.

6.3 Base de Dados

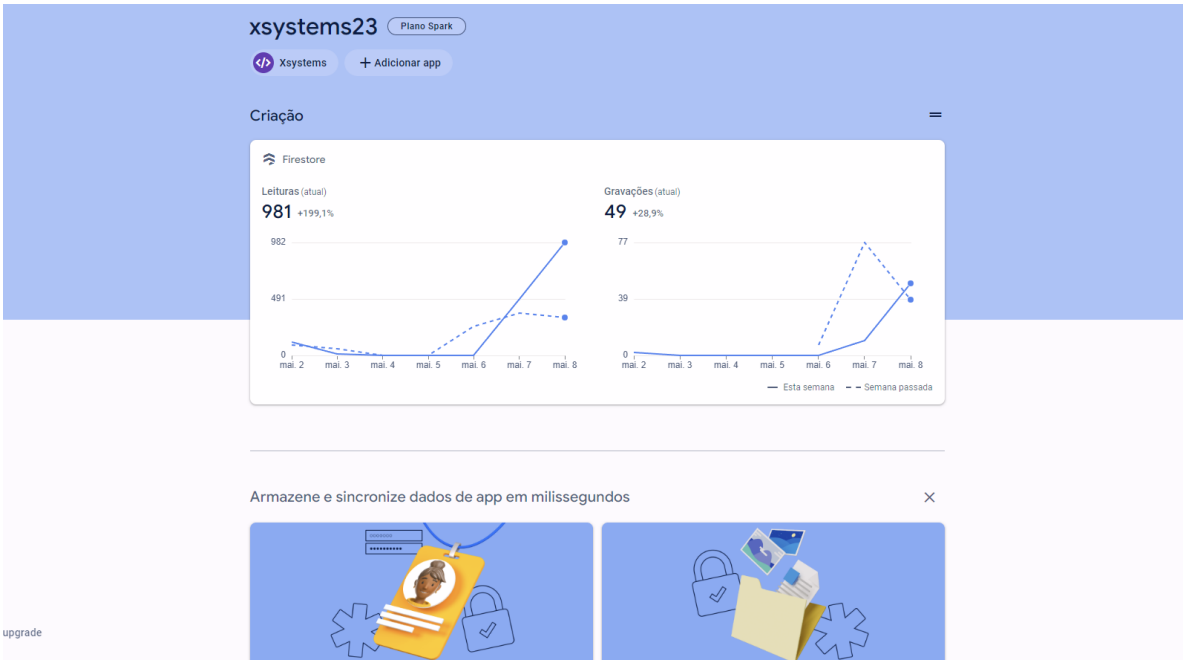


Figura 57 - Página Firebase

Decidi utilizar o Firebase devido à sua capacidade de otimização de dados. No Firebase, a estrutura é organizada utilizando o formato JSON em vez de tabelas convencionais. A minha base de dados Firebase é composta por 4 coleções principais

6.3.1 Coleção Mesas

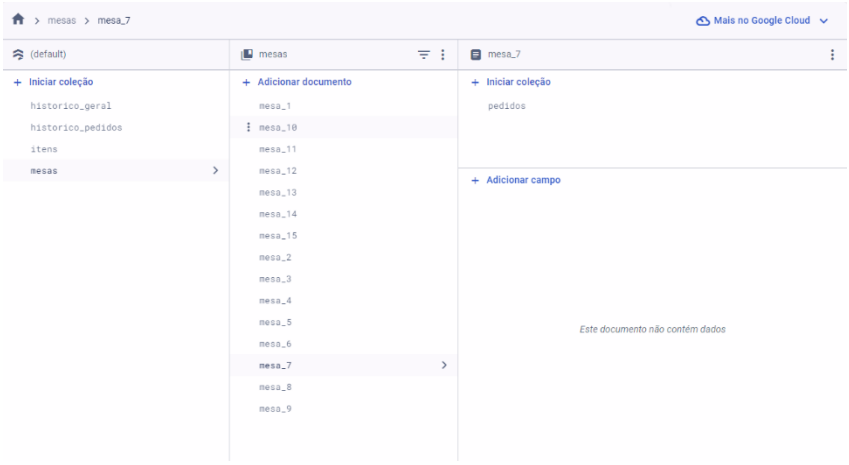


Figura 58 - Coleção Mesas

Esta coleção contém subcolecções para cada mesa, até à mesa número 15 (por exemplo, "mesa_1", "mesa_2", etc.). Quando um pedido é feito, é criado um documento "pedido" na subcolecção correspondente à mesa. Cada vez que um pedido é encerrado, uma nova subcolecção é criada com um ID automático do Firebase. Neste documento, são registrados o horário de envio, os itens pedidos e o status do pedido.

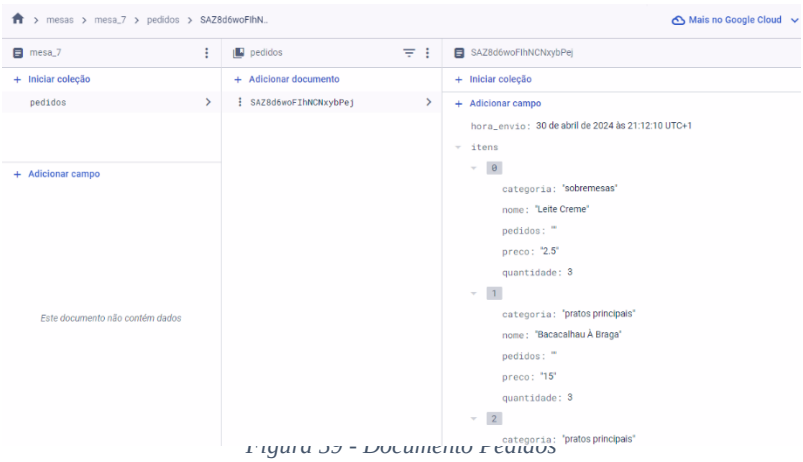


Figura 59 - Documento Pedidos

6.3.2 Coleção Itens

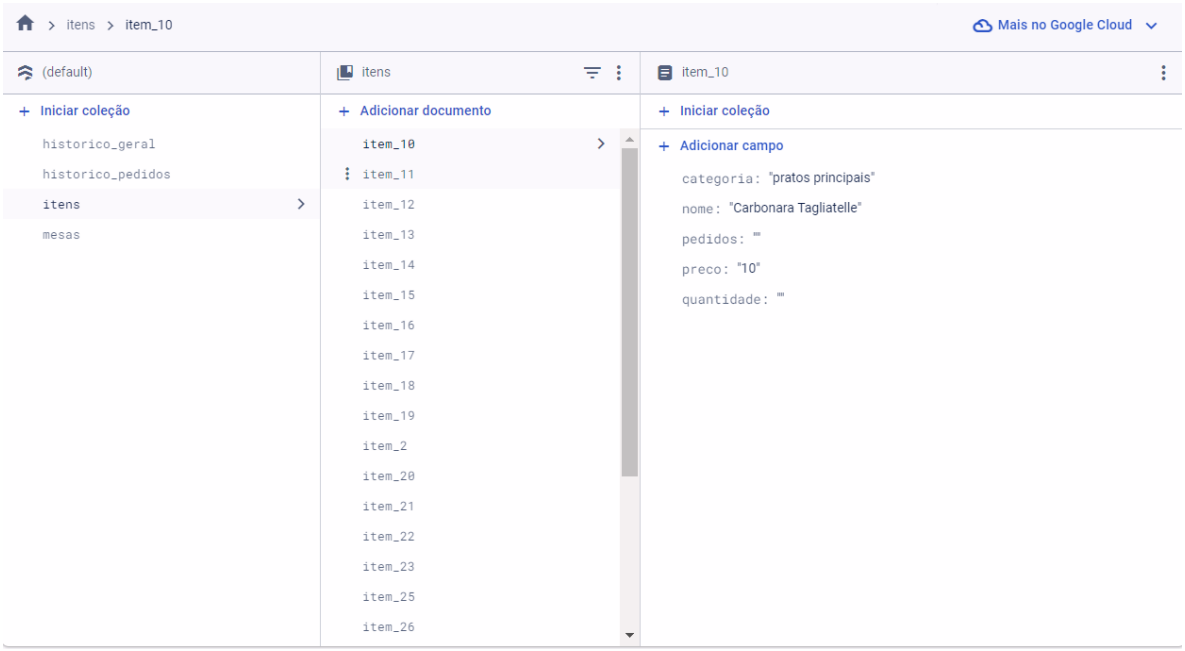


Figura 60 - Coleção Itens

Nesta coleção, são criados os novos itens do software através da área de administração. Cada item é representado por um documento com um identificador automático ("item_+1"). Os campos incluem categoria, nome, pedidos, preço e quantidade, os campos são criados automaticamente quando o item é registado no software.

6.3.3 Histórico Pedidos

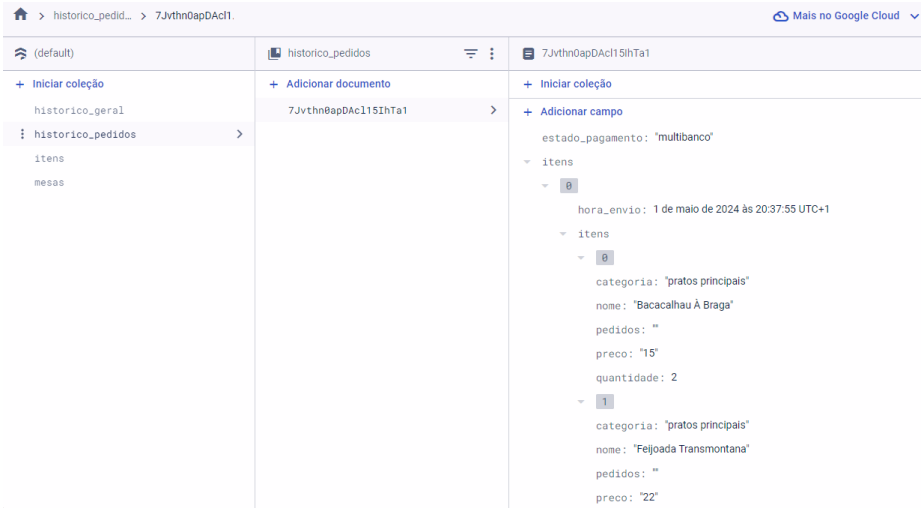


Figura 61 - Coleção Histórico Pedidos

Após o pagamento, as principais informações do pedido são listadas nesta coleção, representando o histórico diário.

6.3.4 Histórico Geral

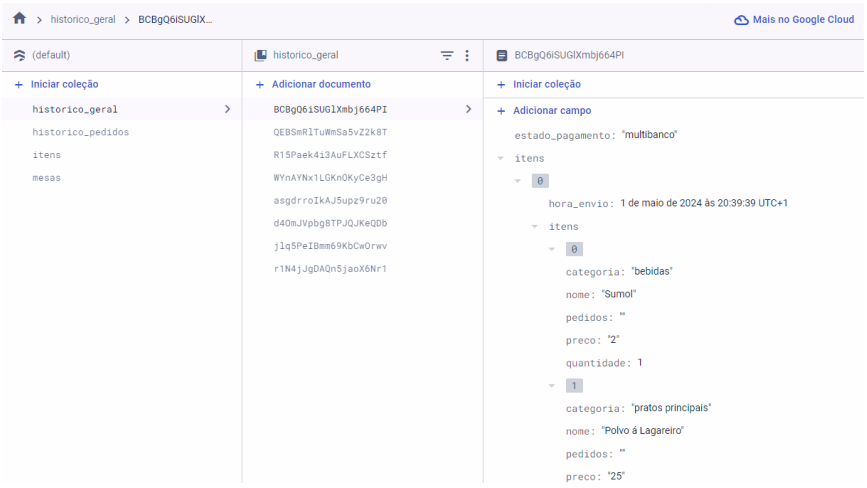


Figura 62 - Coleção Histórico Geral

Quando a caixa é encerrada, todos os itens presentes na coleção "histórico de pedidos" são transferidos para esta tabela, que representa um histórico total do software.

6.3.5 Authentication

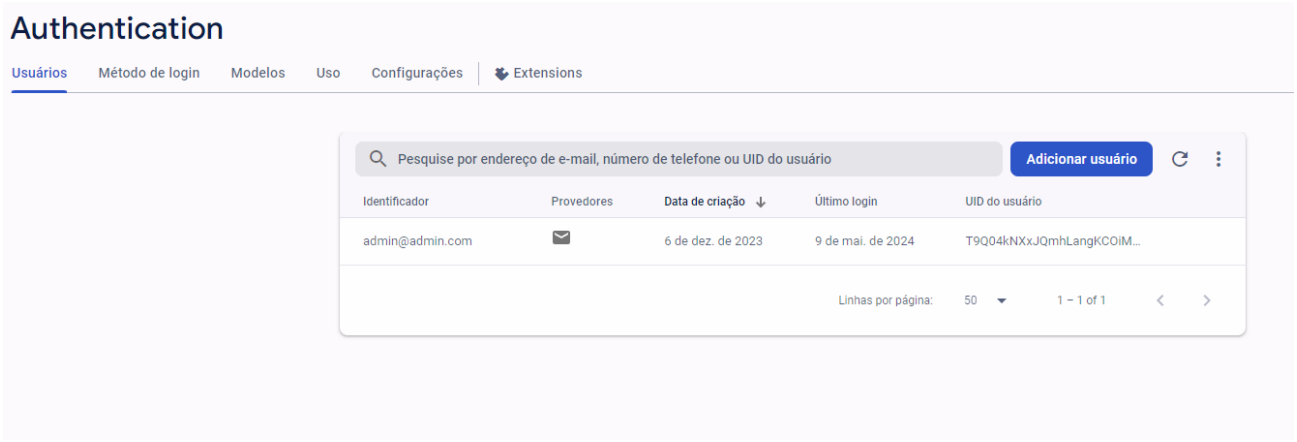


Figura 63 - Firebase Authentication

Utilizei o Firebase Authentication para gerar uma conta de login no sistema. Propositalmente, há apenas uma conta disponível, já que em um software dedicado não há necessidade de múltiplos utilizadores.

7. Alterações Futuras

No futuro, na minha PAP, pretendo aprimorar o design de algumas páginas, adicionando transições e splash screens. Além disso, quero concluir o botão "Dados do Cliente" para permitir a inserção do NIF do cliente, assim como melhorar a responsividade do projeto, pretendo também adicionar o campo de observações dos pedidos como alergénios e etc; Planejo integrar algumas APIs, como a do Multibanco, e tornar o software mais personalizável e adaptável às necessidades específicas dos utilizadores.

8. Conclusão

Ao concluir este projeto, percebo que estou preparado para dar o próximo passo e possivelmente ingressar no mercado de trabalho. Conseguir realizar o software que imaginei desde o surgimento da ideia foi uma jornada desafiadora, repleta de complicações que foram sendo superadas com muita paciência e dedicação. Criar um software não é tarefa fácil; exige um alto nível de organização, perseverança e habilidades técnicas.

Ao longo desse processo, aprendi a importância de planejar cuidadosamente, resolver problemas de forma criativa e manter o foco mesmo diante das adversidades. Além disso, a experiência me permitiu aprimorar as minhas habilidades de comunicação e trabalho em equipa, pois frequentemente precisei colaborar com outras pessoas para encontrar soluções eficazes.

Sinto-me confiante não apenas na minha capacidade de desenvolver software, mas também na minha capacidade de enfrentar desafios e aprender continuamente. Estou entusiasmado para aplicar o que aprendi neste projeto em um ambiente profissional e contribuir de forma significativa para equipas e projetos futuros.

Considero esta jornada como um marco importante em minha preparação para o mercado de trabalho, e estou ansioso para as oportunidades que virão pela frente.

PESSOAS 2030 PORTUGAL 2030 Cofinanciado pela União Europeia SELO DE CONFORMIDADE EQAVET uma escola Rumos education Knowledge sharing
Os Fundos Europeus mais próximos de si.

Na escola, adquiri uma base sólida que foi fundamental para entender a lógica da programação. Ao longo dos três anos do curso, o terceiro ano se destacou como o mais desafiador, mas também o mais cativante para mim. Foi nesse período que pude consolidar e aprofundar meus conhecimentos, enfrentando desafios que exigiram mais de mim, porém, ao mesmo tempo, despertaram em mim uma curiosidade ainda maior pela área.

Durante esse tempo, aprendi conceitos fundamentais que se tornaram a fundamentação do meu entendimento em programação. Desde os princípios básicos até os conceitos mais avançados, cada etapa do meu aprendizado contribuiu para fortalecer minha compreensão e habilidades práticas.

Além disso, as experiências compartilhadas com colegas e professores enriqueceram ainda mais minha jornada. A troca de ideias, a resolução de problemas em equipa e os projetos colaborativos foram aspetos importantes que me prepararam para enfrentar desafios no mercado de trabalho.

Considero estes anos de estudo como um investimento valioso no meu desenvolvimento profissional. Eles não apenas me deram as ferramentas necessárias para entender a programação, mas também me inspiraram a buscar novos desafios e aprimorar continuamente minhas habilidades. Estou ansioso para aplicar tudo o que aprendi e continuar a crescer neste campo empolgante da tecnologia.

9. Bibliografia

Web - flutter.dev

Web – docs.flutter.dev

Youtube – Dotcode

Youtube – RetroPortal Studio

Anexos

Anexo 1 – Curriculum Vitae

Anexo 2 – Microsoft Officce Specialist - Associate

Gino Gabriel Rodrigues Oliveira

Data de nascimento: 24/07/2006

Nacionalidade: Portuguesa

Sexo: Masculino

CONTACTO

Rua do Ribeirinho, 75,
4715 - 446 Braga, Portugal
(Casa)

ginogabriel2407@gmail.com

(+351) 935945719

europass

SOBRE MIM

Sou um entusiasta por tecnologia e programação. Estudei programação por 3 anos e desde então tenho me dedicado a aprender e criar projetos pessoais. Agora, busco oportunidades para aplicar minhas habilidades como desenvolvedor e contribuir para soluções inovadoras na área.

EDUCAÇÃO E FORMAÇÃO

13/09/2021 – 25/07/2024 Braga, Portugal

Gestão e Programação de Sistemas Informáticos Escola Profissional de Braga (EPB)

Sítio Web epb.pt

COMPETÊNCIAS DIGITAIS

LINGUAGENS DE PROGRAMAÇÃO

Dart (Flutter) | HTML | Python | Bootstrap | CSS | C# | JavaScript | PHP

BASES DE DADOS

Firebase | MySql | Access | SQL

SISTEMAS OPERATIVOS E SOFTWARES DE APLICAÇÃO

Windows Desktop | Office e Open Source

CARTA DE CONDUÇÃO

Carta de condução: A1

DISTINÇÕES E PRÉMIOS

02/03/2023 Microsoft

Microsoft Office Specialist - Associate

PESSOAS 2030 PORTUGAL 2030 Cofinanciado pela União Europeia SELO DE CONFORMIDADE EQAVET uma escola Rumos education | Knowledge sharing

Os Fundos Europeus mais próximos de si.

Microsoft Office Specialist - Associate

Gino Gabriel Oliveira

has successfully completed the requirements to be recognized as a

Microsoft Office Specialist - Associate

Excel 2019
PowerPoint 2019
Word 2019

Date of achievement: março 2, 2023
verify.certiport.com wEcBD-48uk

Satya Nadella
Chief Executive Officer

Microsoft

PESSOAS 2030 PORTUGAL 2030 Cofinanciado pela União Europeia SELO DE CONFORMIDADE EQAVET uma escola Rumos education Knowledge sharing
Os Fundos Europeus mais próximos de si.