CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE FRANCA

“Dr. THOMAZ NOVELINO”

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

FRANCISCO ANTÔNIO DE QUEIROZ JUNIOR

YAGO HENRIQUE SILVA TERENCIO

FASTBARBER:

Aplicativo para organização e gerenciamento de tarefas

Trabalho de Graduação apresentado à Faculdade de Tecnologia de Franca - “Dr. Thomaz Novelino”, como parte dos requisitos obrigatórios para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Me. Leonardo Henrique Raiz

FRANCA/SP

2024

FASTBARBER

Software para gestão e controle de agendamento de cortes

Francisco Antônio de Queiroz Junior

Yago Terêncio

Resumo

Esta monografia analisa os desafios enfrentados pelos barbeiros em Franca no processo de agendamento de cortes. Identifica-se que a perda de tempo durante o trabalho devido à necessidade de reservar horários para os clientes é uma das principais dificuldades enfrentadas. Em resposta a essa problemática, é apresentado o FastBarber, um software desenvolvido especificamente para atender às demandas das barbearias, oferecendo uma solução eficiente e simplificada para a gestão de agendamentos. O FastBarber possibilita que os barbeiros estabeleçam facilmente os horários de funcionamento da barbearia, ao passo que os clientes desfrutam da conveniência de agendar cortes de cabelo de maneira rápida e acessível por meio de um sistema automatizado. Esta abordagem inovadora visa resolver os desafios enfrentados pelas barbearias, promovendo uma modernização significativa em suas operações, impulsionando a eficiência e elevando a qualidade do serviço prestado. Dessa forma, o FastBarber busca proporcionar uma experiência satisfatória tanto para os profissionais quanto para os clientes.

Palavras-chave: Perda de tempo, Eficiência no serviço, Agendamento de cortes, FastBarber, Experiência do cliente e Satisfação dos profissionais.

Abstract

Upon analyzing the service processes in barbershops in Franca, numerous challenges faced by barbers during haircut scheduling become evident. Conversations with these professionals revealed that one of the main obstacles was the time lost while working due to the need to book appointments for clients. Addressing this issue, FastBarber emerges as a software specifically designed to meet the demands of barbershops by offering an efficient and simplified solution for appointment management. FastBarber allows barbers to easily define their shop's operating hours while providing clients the convenience of booking haircuts quickly and conveniently through an accessible automated system. With FastBarber's innovative approach, the aim is to solve barbershops' challenges by significantly modernizing their operations, enhancing efficiency, and improving the quality of service provided, thus ensuring a satisfactory experience for both professionals and clients.

Keywords*: Efficiency in service, FastBarber, Haircut scheduling, Time management, Customer experience, Professional satisfaction*

1 Introdução

Os softwares se apresentam, na modernidade, como uma ferramenta indispensável em inúmeros cenários cotidianos. Muitas vezes, funcionam como alternativa para solucionar desafios e dificuldades enfrentadas por pequenos empreendedores, que necessitam de praticidade e rapidez para aumentar suas vendas e resultados.

Em uma pesquisa de mercado, foi encontrado a aplicação BestBarbers, focada em serviços com o plano de pagamento mensal e totalmente focado no cliente do barbeiro (BestBarbers, 2020). Pensando nisso, o presente trabalho apresenta o FastBarber, um software pensado para auxiliar os barbeiros autônomos que, trabalhando sozinhos, precisam otimizar seus processos para evitar sobrecarga e perda de clientes, com um serviço focado no barbeiro, forma de pagamento único e melhorias previstas, para ambiente desktop.

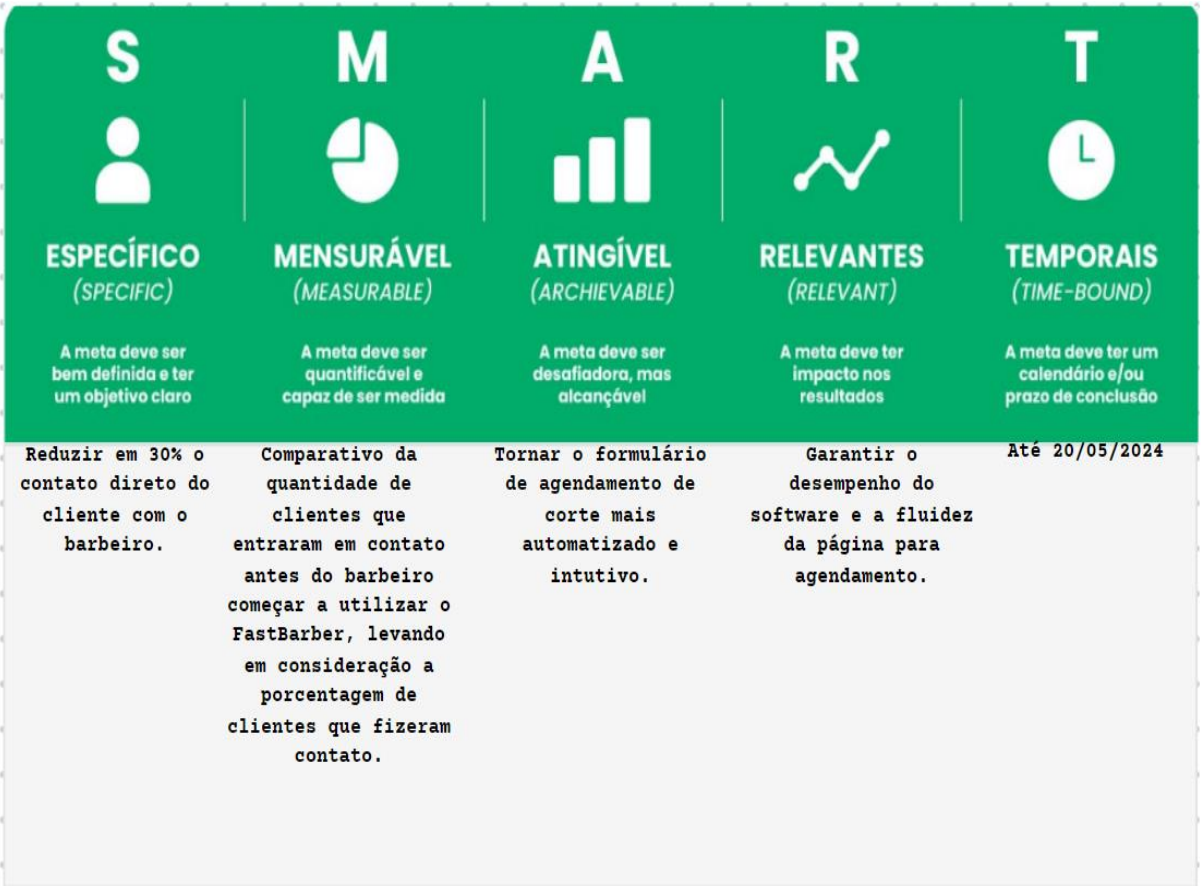
Neste contexto, a ferramenta criada busca simplificar o agendamento de horários, tornando possível que o próprio cliente, através do aplicativo Web, escolha o horário disponível que preferir. Com isso, obtemos a melhora da administração do tempo e recursos por parte dos barbeiros, que conseguem atender seus clientes sem se preocupar com a parte do agendamento, que passaria a ser automatizada.

Em busca de consistência, estabilidade e flexibilidade referente a atualizações, a ferramenta foi desenvolvida com as principais metodologias e linguagens C#, Javascript e Sql Server, reconhecidas por sua eficácia e relevância no mercado (Bruna, 2024). Dessa forma, o FastBarber foi pensado para proporcionar aos barbeiros um serviço de qualidade, capaz de aprimorar seus métodos e alavancar suas vendas através de uma maior agilidade no agendamento de novos atendimentos.

1.1 Termo da Abertura do Projeto (TAP)

O Termo de Abertura do Projeto (TAP) é fundamental, delineando objetivos, escopo e stakeholders. Além de ser uma ferramenta essencial respaldada por fontes especializadas, o TAP assegura transparência e uma estrutura sólida para o desenvolvimento, permitindo avaliações contínuas e adaptações ao longo do projeto. Este documento é a âncora que fundamenta o entendimento e sucesso do empreendimento, guiando os capítulos subsequentes para uma compreensão mais detalhada do projeto.

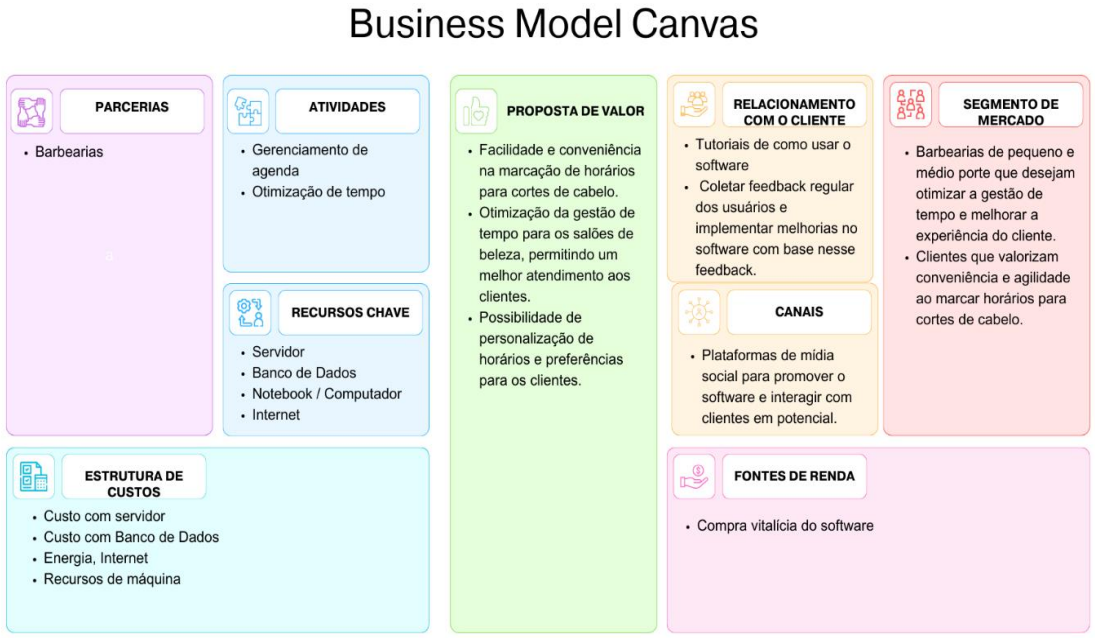
Figura 1 - Modelo SMART



Fonte: elaborado pelos autores

2 Viabilidade do projeto

Figura 1 - Modelo Canvas



Fonte: elaborado pelos autores

3 Levantamento de Requisitos

3.1 Elicitação e especificação dos Requisitos

O processo de levantamento de requisitos foi conduzido em duas fases. Inicialmente, realizando uma pesquisa das necessidades e expectativas do cliente com base em perguntas específicas (Fernanda, 2018). Com isso identificou-se a necessidade de aprimorar o controle de agendamento dos barbeiros, observando que muitos deles enfrentam uma otimização inadequada do tempo devido à execução manual do processo de agendamento de cortes. Essa prática resulta frequentemente na perda de clientes devido à falta de organização.

Após reconhecer essa demanda, realizou-se uma reunião envolvendo dois barbeiros do mesmo salão. Nesse encontro, entendemos as possíveis melhorias que seria preciso fazer para resolver o problema da barbearia. Vimos que é necessário que o cliente consiga fazer o agendamento de corte de maneira simples e automática, para que o barbeiro não perca tempo respondendo de forma manual os horários disponíveis.

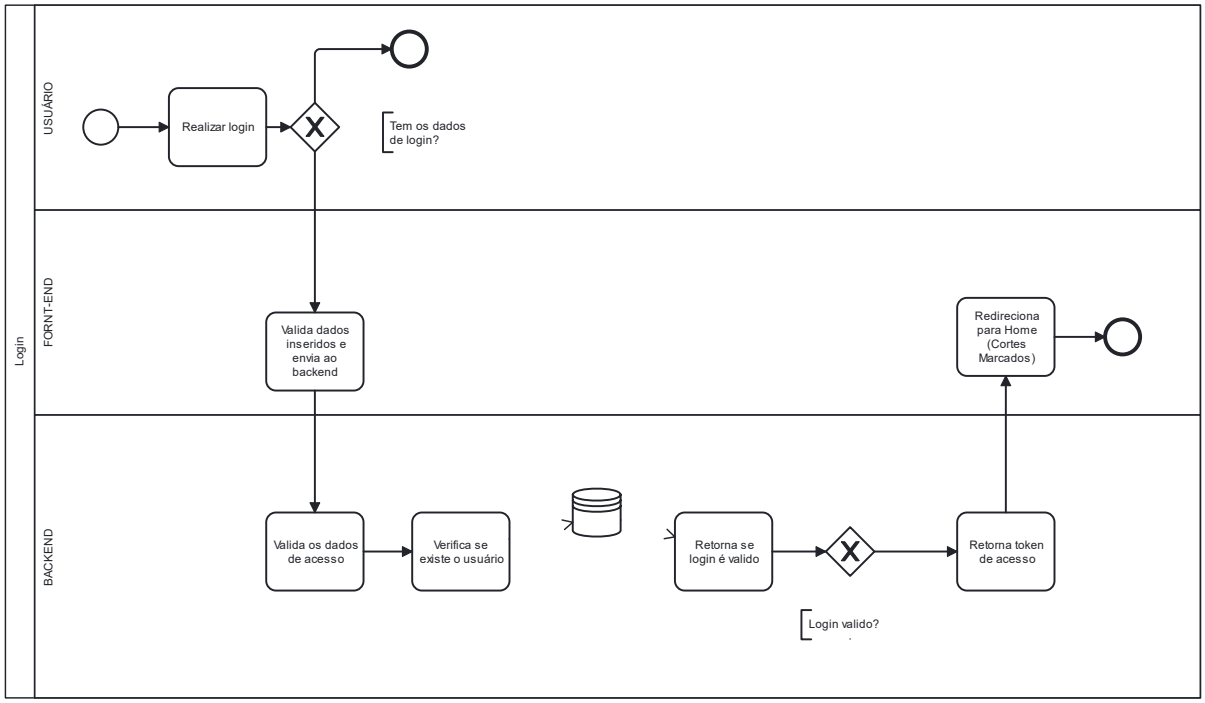
Na reunião, foram destacadas as principais funcionalidades desejadas para a aplicação. A ferramenta ideal deveria automatizar o agendamento por parte dos clientes, visando otimizar os horários dos cortes, e também permitir que o barbeiro possa ajustar sua grade de horários disponíveis, proporcionando o máximo de autonomia e automação possível.

Com base nas informações coletadas e nas funcionalidades concebidas para o sistema, procedeu-se à prototipação da aplicação, delineando a visão de como ela seria implementada. Simultaneamente, foram definidas a metodologia e as tecnologias a serem empregadas durante o processo de desenvolvimento.

3.2 BPMN

As figuras abaixo, apresentam os diagramas de BPMN, que consistem na modelagem dos processos do negócio. A partir destes diagramas é possível entender o contexto dos processos que serão automatizados pelo sistema em desenvolvimento (Vinicius L. de Almeida, 2017). (Figura 2) Representa a ação de login do usuário.

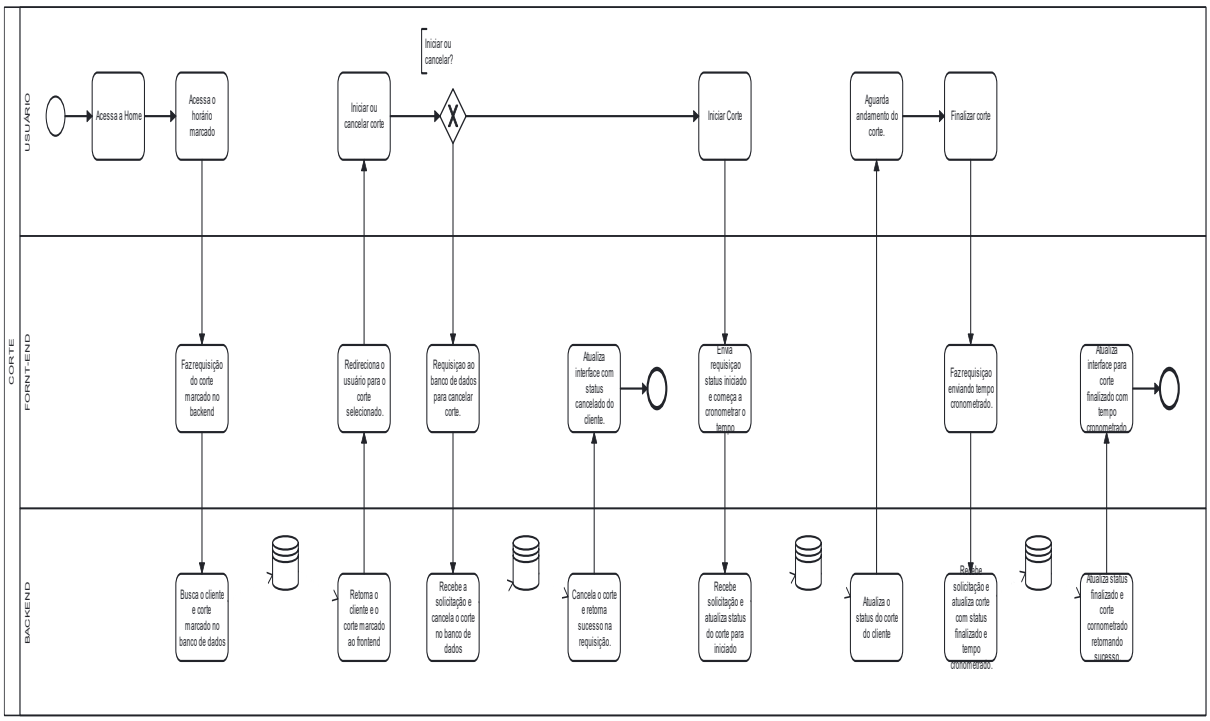
Figura 2 – BPMN Login



Fonte: elaborado pelos autores

(Figura 3) Representa o fluxo para se iniciar ou cancelar um corte realizado pelo barbeiro.

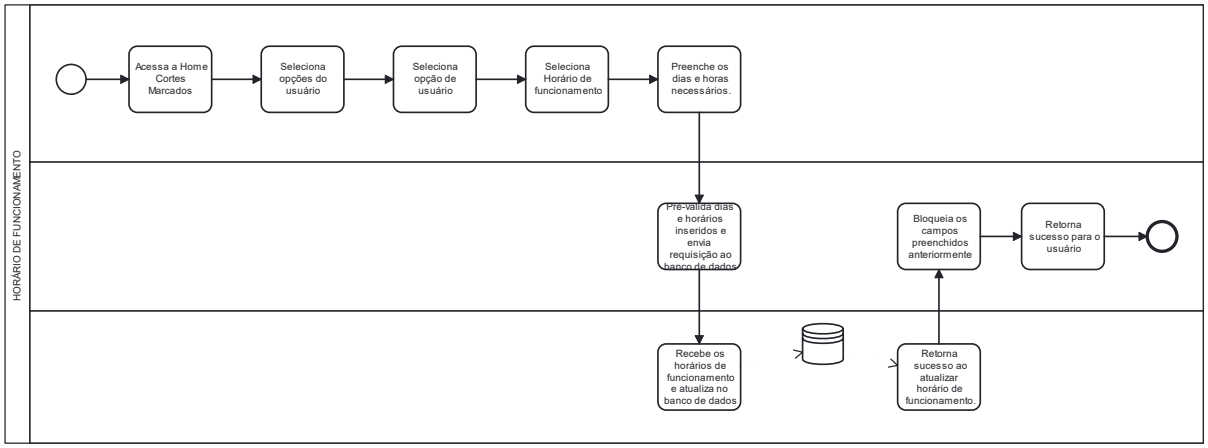
Figura 3 – BPMN Corte



Fonte: elaborado pelos autores

(Figura 4) Representa o fluxo realizado pelo barbeiro para conseguir alterar o horário de funcionamento.

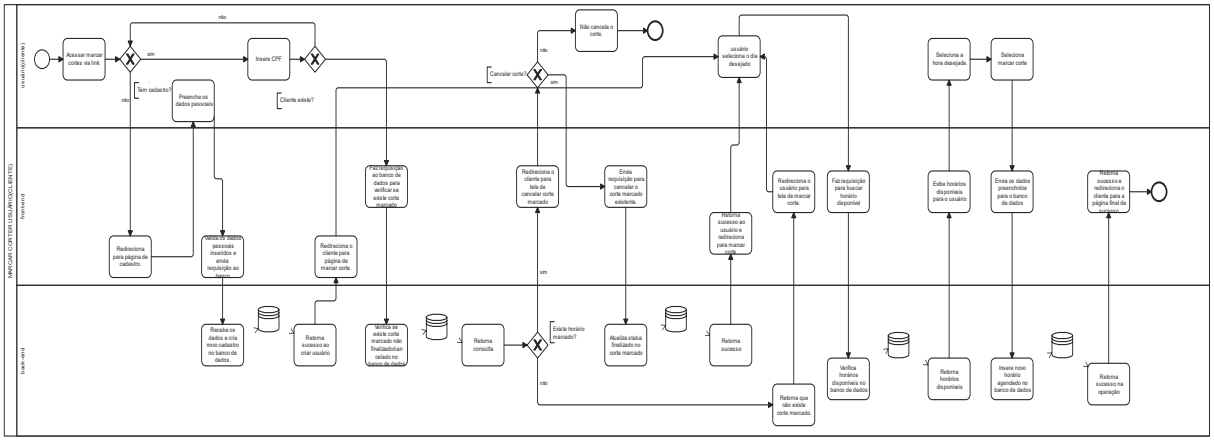
Figura 4 – BPMN Horário de funcionamento



Fonte: elaborado pelos autores

(Figura 4) Representa o fluxo realizado pelo usuário(cliente) para conseguir agendar um horário.

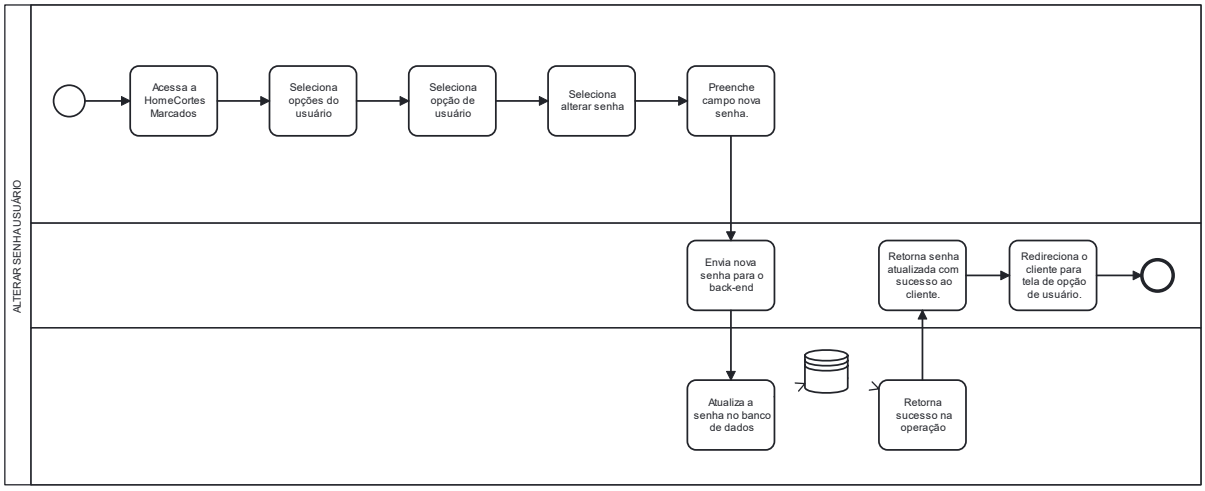
Figura 4 – BPMN Marcar corte usuário(cliente)



Fonte: elaborado pelos autores

(Figura 5) Representa o fluxo realizado pelo usuário(barbeiro) para conseguir alterar a senha de acesso.

Figura 5 – BPMN Alterar senha



Fonte: elaborado pelos autores

3.3 Requisitos Funcionais

Quadro 1 – Requisitos Funcionais do sistema

RF001-Definir horário de funcionamento	Categoria: ( ) Oculto (X)Evidente	Prioridade: (X) Altíssima ( ) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário(barbeiro) defina o período em que os horários podem ser marcados.		
RF002- Iniciar corte	Categoria: () Oculto (X) Evidente	Prioridade: ( ) Altíssima (X) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que ao usuário(barbeiro) possa iniciar um corte, fazendo com que o tempo seja cronometrado através de um timer.		
RF003- Cancelar corte	Categoria: ( ) Oculto (X) Evidente	Prioridade: (X) Altíssima ( ) Alta ( ) Média ( ) Baixa
Descrição: O usuário(barbeiro) realiza o cancelamento do horário agendado de forma manual.		

RF004- Realizar cadastro no sistema	Categoria: ( ) Oculto (X) Evidente	Prioridade: (X) Altíssima () Alta ( ) Média ( ) Baixa
Descrição: O usuário(cliente) realiza o cadastro pessoal através de um formulário.		
RF005- Realizar agendamento de horário	Categoria: ( ) Oculto (X) Evidente	Prioridade: () Altíssima (X) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário(cliente) realize agendamento de horário somente com perfil cadastrado, que não possua corte pendente.		
RF006- Cancelamento de horário	Categoria: ( ) Oculto (X) Evidente	Prioridade: () Altíssima (X) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário(cliente) realize o cancelamento do horário agendado		
RF007- Exibir cortes agendados do dia	Categoria: ( ) Oculto (X) Evidente	Prioridade: (X) Altíssima () Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário(barbeiro) possa visualizar os cortes agendados do dia		

RF008- O sistema deve exibir informações do cliente	Categoria: ( ) Oculto (X) Evidente	Prioridade: () Altíssima (X) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve exibir as informações relacionadas ao cliente, como cpf e telefone.		

3.4 Requisitos Não Funcionais

Quadro 2 – Requisitos Não Funcionais do sistema

RNF001- Horários Dinâmicos	O sistema deve calcular automaticamente os horários disponíveis para serem agendados com base nos horários que já estão marcados.	Tipo Implementação	() Desejável (X) Obrigatório	(X) Perma-nente ( ) Transitório

RNF002- Uso simples	O usuário do sistema deve ser capaz de navegar mesmo sem treinamento prévio.	Tipo: Usabilidade	( ) Desejável (X) Obrigatório	(X) Permanente ( ) Transitório
RNF003 - Online	O sistema deve ser acessado online, através do site da aplicação.	Tipo: Usabilidade	( ) Desejável (X) Obrigatório	(X) Permanente ( ) Transitório
RNF004 – Login	O aplicativo deverá autenticar o usuário através do e-mail e senha previamente cadastrados pelos desenvolvedores do software, após pagamento do serviço.	Tipo: Segurança	( ) Desejável (X) Obrigatório	(X)Perma-nente ( ) Transitório
RNF005 – Manter timer de corte	Caso o usuário(barbeiro) saia da tela de timer do cliente, o timer deverá continuar contando	Tipo: Usabilidade	( ) Desejável (X) Obrigatório	(X)Perma-nente ( ) Transitório

3.5 Regras de negócio

As Regras de Negócio são um conjunto de diretrizes que definem como as atividades de uma organização devem ser realizadas. Elas representam as políticas, práticas, procedimentos e requisitos que governam o comportamento da organização e de seus colaboradores. No (Quadro 4) é apresentado as regras de negócio do FastBarber.

Quadro 3 – Regras de Negócio do sistema.

RN001 – Cancelamento de horários agendados
Descrição: Os horários agendados que ultrapassarem 10 minutos de seu horário serão cancelados automaticamente.
RN002 – Período de cortes

Descrição: O cliente poderá somente marcar um só corte durante o período de uma semana.
RN003 – Iniciar cortes
Descrição: O usuário(barbeiro) poderá iniciar somente um corte por vez.
RN004 – Horários dinâmicos
Descrição: Os horários apresentados para o usuário(cliente) deveram ser apresentados dinamicamente do horário em diante em que ele acessar a aplicação.

3.6 Casos de Uso

Índice de casos de uso e Diagrama de casos de uso

UC001- Efetuar Login

UC002- Definir horário de funcionamento

UC003- Gerar link de formulário

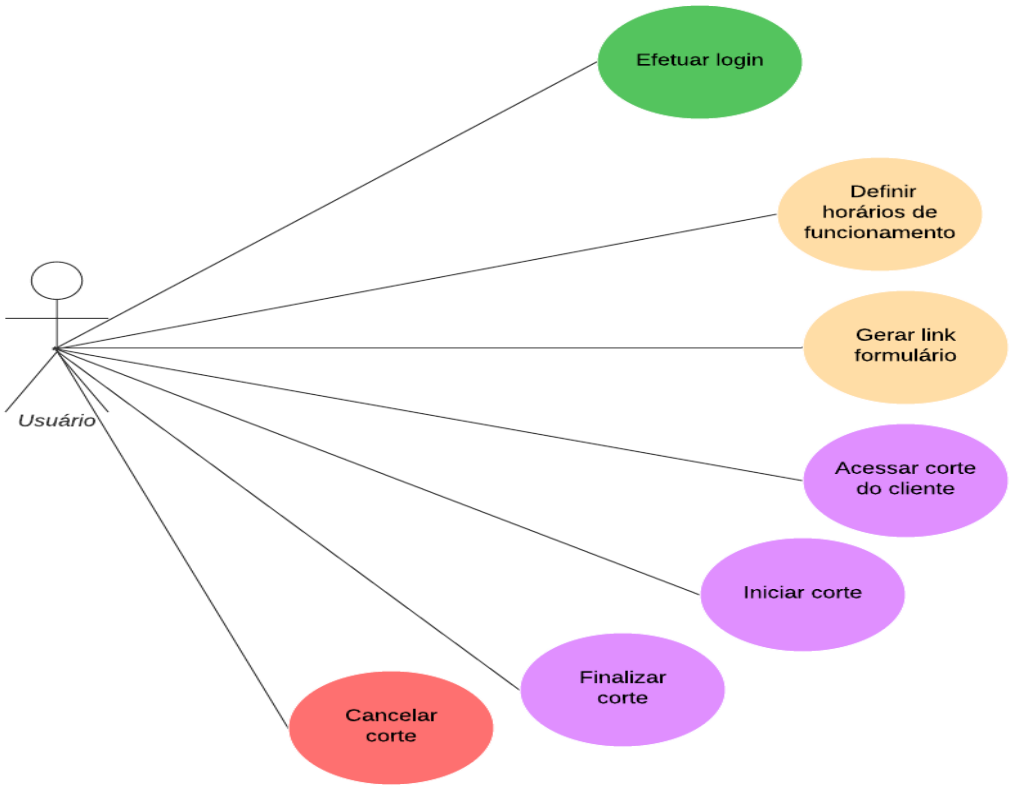
UC004- Acessar corte do cliente

UC005- Iniciar corte

UC006- Finalizar corte

UC007- Cancelar horário marcado

Figura 3 – Diagrama de Caso de Uso



Fonte: elaborado pelos autores

Quadro 4 – Use Case Login

Caso de Uso – Efetuar Login	
ID	UC 001
Descrição	Este caso de uso tem por objetivo efetuar login do usuário.
Ator Primário	Usuário do sistema
Pré-condição	Usuário precisa possuir os dados de acesso.
Cenário Principal	1. O use case inicia quando o usuário preenche os campos e-mail e senha e seleciona o botão “entrar”. 2. O sistema busca no banco de dados o usuário e verifica se as credenciais estão corretas. 3. O sistema retorna token de acesso. 4. O sistema autoriza o acesso. 5. O sistema o redireciona para a página inicial com usuário informado logado. 6. O sistema encerra o caso de uso.
Pós-condição	Nenhuma.
Cenário Alternativo	1a – O usuário não preenche os campos e-mail e senha. 1a.1 – o sistema retorna que é obrigatório inserir os dados para acesso. 1a.2 – retorna o passo 2 do cenário principal. 1b – O usuário não preenche os campos corretamente. 1b.1 – O sistema informa que os campos foram preenchidos de maneira indevida. 1b.2 – retorna ao passo 2 do cenário principal. 2a – O sistema não encontra nenhum e-mail e/ou senha cadastrado. 2a.1 – O sistema retorna acesso negado.

Quadro 5 – Use Case Definir Horário

Caso de Uso – Definir horário de funcionamento	
ID	UC 002
Descrição	Este caso de uso tem por objetivo possibilitar o usuário a efetuar a alteração do tempo de corte disponível para realizar o agendamento.
Ator Primário	Usuário do sistema
Pré-condição	Usuário precisa estar logado.
Cenário Principal	1. O use case inicia quando o usuário seleciona a opção "Período de horário disponível para agendamento." 2. O sistema busca o período disponível e redireciona o usuário para a respectiva tela. 3. O usuário altera os campos para o horário de disponibilidade desejada. 4. O sistema irá armazenar e atualizar no banco de dados o novo horário de disponibilidade. 5. O sistema retorna sucesso na alteração. 6. O sistema encerra o caso de uso.
Pós-condição	Nenhuma.
Cenário Alternativo	3a- O usuário informa um período inválido. 3ª. 1 – O sistema solicita ao usuário para inserir um horário válido

Quadro 6 – Use Case Gerar Link

Caso de Uso – Gerar link de formulário	
ID	UC 003
Descrição	Este caso de uso tem por objetivo possibilitar que o usuário solicite ao sistema o link de formulário.
Ator Primário	Usuário do sistema
Pré-condição	Usuário precisa estar logado.
Cenário Principal	1. O use case inicia quando o usuário seleciona a opção "Marcar cortes via link." 2. O sistema gera o link de formulário. 3. O usuário copia o link gerado. 4. O sistema encerra o caso de uso.
Pós-condição	Nenhuma.
Cenário Alternativo	2a – Usuário perdeu conexão com a internet 2a.1 O sistema exibe um erro de falha com conexão com a internet.

Quadro 7 – Use Case Acessar Perfil

Caso de Uso – Acessar corte do cliente	
ID	UC 004
Descrição	Este caso de uso tem por objetivo possibilitar o usuário acesse os perfis de seus clientes.
Ator Primário	Usuário do sistema
Pré-condição	Usuário precisa estar logado.
Cenário Principal	1. O use case inicia quando o usuário seleciona a opção "Acessar perfil do cliente." 2. O sistema carrega a lista de clientes para o usuário. 3. Caso o usuário clique em algum item da lista, este é aberto em outra tela com mais informações do planeta. 4. O sistema encerra o caso de uso.
Pós-condição	Nenhuma.
Cenário Alternativo	2a – Usuário perdeu conexão com a internet 2a.1 O sistema exibe um erro de falha com conexão com a internet.

Quadro 8 – Use Case Iniciar corte

Caso de Uso – Iniciar corte	
ID	UC 005
Descrição	Este caso de uso tem por objetivo possibilitar o usuário iniciar a contagem de um Timer.
Ator Primário	Usuário do sistema
Pré-condição	Usuário precisa estar no perfil do cliente.
Cenário Principal	1. O use case inicia quando o usuário seleciona a opção "INICIAR" 2. O sistema inicia a contagem de um Timer. 3. O sistema encerra o caso de uso.
Pós-condição	Nenhuma.
Cenário Alternativo	2a – Usuário perdeu conexão com a internet 2a.1 O sistema exibe um erro de falha com conexão com a internet.

Quadro 9 – Use Case finalizar corte

Caso de Uso – Finalizar corte	
ID	UC 006
Descrição	Este caso de uso tem por objetivo possibilitar o usuário finalize a contagem de um Timer.
Ator Primário	Usuário do sistema
Pré-condição	O timer precisa ter sido iniciado pelo usuário.
Cenário Principal	1. O use case inicia quando o usuário seleciona a opção “Finish Haircut.” 2. O sistema finaliza a contagem de um Timer. 3. O sistema encerra o caso de uso.
Pós-condição	Nenhuma.
Cenário Alternativo	1a – Usuário seleciona a opção “Finish Haircut”, antes de selecionar a opção” Start Haircut” 1a. 1 O sistema não executa a ação de finalizar. 2a – Usuário perdeu conexão com a internet 2a.1 O sistema exibe um erro de falha com conexão com a internet.

Quadro 10 – Use Case Cancelar corte

Caso de Uso – Cancelar horário marcado	
ID	UC 007
Descrição	Este caso de uso tem por objetivo possibilitar o usuário cancelar um horário agendado.
Ator Primário	Usuário do sistema
Pré-condição	Usuário precisa estar no perfil do cliente.
Cenário Principal	1. O use case inicia quando o usuário seleciona a opção “cancelar corte.” 2. O sistema solicita a confirmação do cliente sobre a ação. 3. O usuário confirma selecionando a opção “SIM”. 4. O horário é cancelado da agenda. 5. O sistema encerra o caso de uso.
Pós-condição	Nenhuma.
Cenário Alternativo	2a – Usuário seleciona a opção “Não” 2a. 1 O sistema não executa a ação de cancelar horário. 2b – Usuário perdeu conexão com a internet 2b.1 O sistema exibe um erro de falha com conexão com a internet.

4 Ferramentas e Métodos

4.1 Ferramentas

a) **Figma (Figma, 2024).** Aplicação responsável pela prototipação das telas, foi utilizada devido à previa familiaridade e disponibilidade de conteúdo na Internet para estudo de novas funcionalidades.

b) **Microsoft SQL Server (Microsoft, 2022).** Banco de dados relacional utilizado para gerenciar dados e realizar a autenticação de usuários na aplicação. A ferramenta foi escolhida por ser um banco de dados relacional confiável e amplamente utilizado, que oferece recursos avançados para armazenar e manipular dados. Além disso, o SQL Server possui suporte para procedimentos armazenados, funções, gatilhos e outras funcionalidades que facilitaram a implementação de lógica do negócio no banco de dados.

c) **Visual Studio (Microsoft, 2022).** Ambiente de desenvolvimento integrado (IDE) completo e poderoso para desenvolvedores de software. A ferramenta oferece suporte a uma ampla variedade de linguagens de programação. Além disso, possui recursos avançados de edição, depuração e gerenciamento de código, permitindo que os desenvolvedores criem, testem e implementem aplicativos com eficiência.

d) **jQuery (jQuery, 2006).** Biblioteca JavaScript popular que facilita a manipulação do DOM, a manipulação de eventos, a animação e a realização de requisições AJAX. Com jQuery, o uso de AJAX para carregar ou enviar dados de forma assíncrona é mais simples e intuitivo.

e) **iziToast (iziToast, 2023).** Biblioteca JavaScript que facilita a criação de notificações e mensagens de toast para a interface do usuário. Com iziToast, você pode exibir notificações dinâmicas e elegantes para comunicar informações, alertas, avisos ou erros aos usuários de uma aplicação web.

f) **ASP.NET 4.5.1 (Microsoft, 2017).** Versão do framework ASP.NET, parte da plataforma .NET, que facilita o desenvolvimento de aplicativos web modernos, dinâmicos e eficientes. Ele oferece suporte para o desenvolvimento de aplicativos em C# e Visual Basic, com uma variedade de recursos e melhorias para tornar o desenvolvimento web mais ágil e robusto.

g) **SQL Server 2022 Management Studio (Microsoft, 2022).** é uma ferramenta de gerenciamento poderosa e essencial para administradores de banco de dados e desenvolvedores que trabalham com o SQL Server 2022. Esta ferramenta fornece uma interface gráfica para executar várias tarefas.

4.2 Métodos ou Desenvolvimento

No desenvolvimento do projeto, foram priorizadas ferramentas que geralmente mantêm um padrão consistente, tornando-as a base ideal para o desenvolvimento da aplicação. Essas ferramentas proporcionam uma estrutura sólida que pode ser facilmente adaptada e atualizada conforme as necessidades do cliente.

Nas linguagens de programação usadas para o desenvolvimento, de acordo com o site hostinger (Bruna, 2024), renomado em serviços de data-center, destacam-se o C# em segundo lugar e o Javascript em quarto. O C# (Microsoft, 2000) é uma linguagem de programação desenvolvida pela Microsoft e lançada em 2000 como parte da plataforma .NET, sua forte tipagem estática e orientação a objetos, é frequentemente usado para desenvolvimento de aplicativos Windows, serviços da web e jogos. Desde então, tem sido continuamente atualizado, mantendo-se relevante e adaptável às demandas da indústria de software.

Javascript (MDN, 2024), por outro lado, é uma linguagem de programação amplamente utilizada para desenvolvimento web. Inicialmente concebida para tornar as páginas web interativas, o Javascript agora é uma linguagem de programação de propósito geral, com vasta aplicabilidade em diferentes contextos de desenvolvimento, incluindo front-end, back-end e até mesmo desenvolvimento de aplicativos móveis.

Para o armazenamento de dados não logo atrás, ainda na mesma pesquisa o Microsoft SQL Server (Microsoft, 2022) é umas das melhores opções listado 9 e sendo a terceira linguagem mais usada no mundo da programação. Desde seu lançamento em 1989 pela Microsoft, tem se mantido como uma escolha proeminente e confiável no mundo do armazenamento de dados. Ao longo das décadas, o SQL Server evoluiu significativamente para atender às crescentes demandas por eficiência, segurança e escalabilidade nos sistemas de gerenciamento de banco de dados.

Após definir as ferramentas base para o front e back-end, partiu-se para a modelagem DER do banco de dados, realizada com base nas funcionalidades solicitadas pelo cliente e também nos requisitos não funcionais definidos, aplicados a terceira forma normal (3FN) definida por Edgar F. Codd (Wikipedia, 2022) **(Figura 4).**

4.2.1 Estrutura das pastas

Para estruturação do projeto foi utilizado o SOLID (João. R, 2019), introduzido por Robert C. Martin em seu livro “Desgin Patterns: Elements of Reusable Object-Oriented Software", separando as pastas de acordo com sua responsabilidade, podendo assim ser organizadas para permitir a extensão de classes sem modificar o código que já existe. Além de seguir o padrão do MVC (Model, View, Controller):

Model: representa os dados e a lógica de negócios da aplicação. Ele manipula a lógica de armazenamento, recuperação e manipulação dos dados. O modelo geralmente encapsula a interação com o banco de dados ou outros mecanismos de armazenamento de dados.

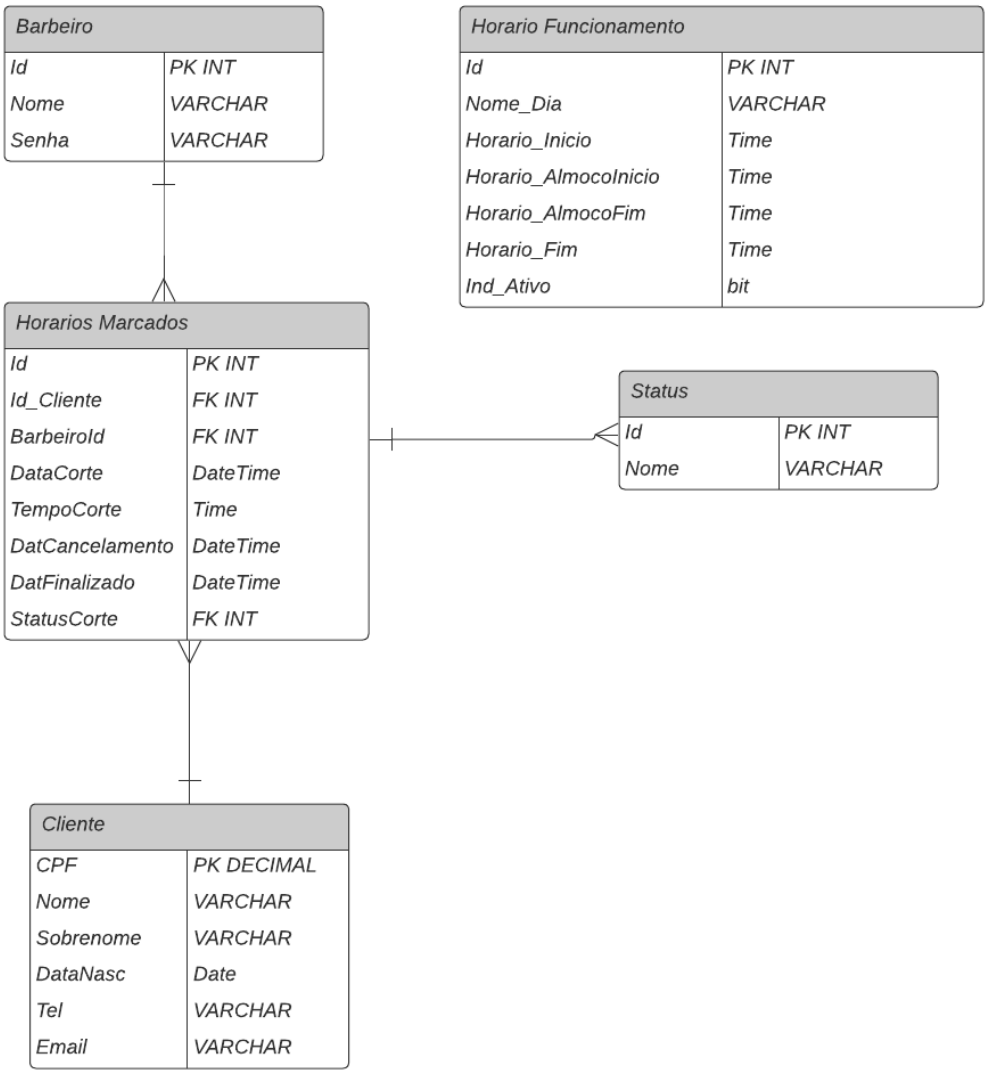
View: responsável pela apresentação dos dados ao usuário. Ela exibe a interface do usuário e interage com o usuário para coletar entrada. As visualizações são passivas e geralmente não têm conhecimento da lógica de negócios subjacente.

Controller: atua como intermediário entre o model e a view. Ele recebe entradas do usuário através da visualização, processa essas entradas e coordena as interações entre o modelo e a visualização. O controlador é responsável por atualizar o modelo com base nas ações do usuário e por selecionar a visualização apropriada para exibir os resultados**.**

A separação desses componentes permite uma maior modularidade, reutilização de código e facilita a manutenção e extensão do software. Por exemplo, ao adicionar uma nova funcionalidade à aplicação, pode precisar apenas atualizar o controller e/ou o modelo correspondente, sem precisar modificar a vi, facilitando assim a organização de pastas.

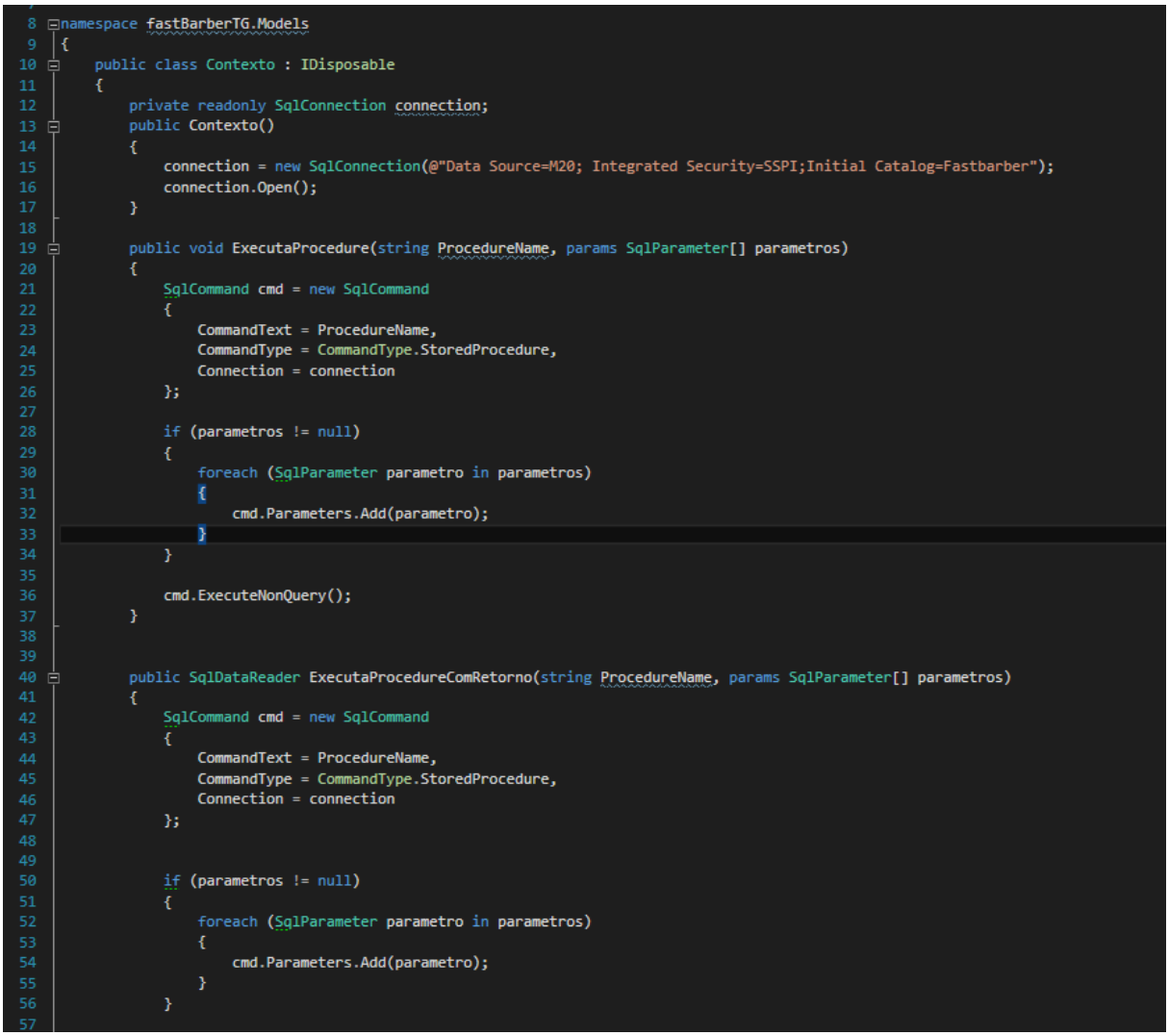
No próximo passo foi realizado a configuração da conexão do banco de dados SQL Server com o C#, utilizado juntamento com a biblioteca nativa que realiza a conexão do banco de dados. Foi necessário a criação de uma classe responsável por ser o “Contexto” da aplicação, podendo assim realizar insert’s, create’s, delete’s e executar procedimentos armazenados. **(Figura 5)**

Figura 4 – DER



Fonte: elaborado pelos autores

Figura 5 – Contexto da aplicação



```
namespace fastBarberTG.Models
{
    public class Contexto : IDisposable
    {
        private readonly SqlConnection connection;
        public Contexto()
        {
            connection = new SqlConnection(@"Data Source=M20; Integrated Security=SSPI;Initial Catalog=Fastbarber");
            connection.Open();
        }

        public void ExecutaProcedure(string ProcedureName, params SqlParameter[] parametros)
        {
            SqlCommand cmd = new SqlCommand
            {
                CommandText = ProcedureName,
                CommandType = CommandType.StoredProcedure,
                Connection = connection
            };

            if (parametros != null)
            {
                foreach (SqlParameter parametro in parametros)
                {
                    cmd.Parameters.Add(parametro);
                }
            }

            cmd.ExecuteNonQuery();
        }


        public SqlDataReader ExecutaProcedureComRetorno(string ProcedureName, params SqlParameter[] parametros)
        {
            SqlCommand cmd = new SqlCommand
            {
                CommandText = ProcedureName,
                CommandType = CommandType.StoredProcedure,
                Connection = connection
            };


            if (parametros != null)
            {
                foreach (SqlParameter parametro in parametros)
                {
                    cmd.Parameters.Add(parametro);
                }
            }

```

Fonte: elaborado pelos autores

Para o realizar queries no banco de dados foram criadas os Repositories para cada tabela que irá ser manipulada, responsável por manipular e realizar a query no banco de dados de acordo com a sua respectiva função.

Para persistir os dados recuperados do banco de dados pelo repository foram criadas as respectivas classes para que fossem populadas com os dados retornados do banco para exibição em tela.

Na imagem **(Figura 6)** é representado o repository de horários agendados, onde é realizado a busca no banco de dados pelos horários que estão agendados utilizando uma stored procedure do SQL Server para retornar os dados para a tela, este repository, realiza as seguintes funções:

- Buscar os horários agendados.
- Busca um corte especifico selecionado em tela.
- Desmarca um corte.
- Retorna lista de cortes com horários já agendados para manipulação do calendário de agendamento para o cliente do barbeiro.
- Finalizar corte.

Figura 6 – Repository horários agendados

```
namespace fastBarberTG.Models
{
    public class HorariosAgREPO
    {
        private Contexto contexto;

        public IEnumerable<HorariosMarcadosModel> HorariosMarcados()
        {
            using (contexto = new Contexto())
            {
                var reader = contexto.ExecutaProcedureComRetorno("FBSP_MostraHorariosMarc");
                var obj = new List<HorariosMarcadosModel>();
                while (reader.Read())
                {
                    var hasObj = new HorariosMarcadosModel()
                    {
                        HorarioId = int.Parse(reader["Id"].ToString()),
                        Id_Cliente = int.Parse(reader["Id_Cliente"].ToString()),
                        DataCorte = DateTime.Parse(reader["DataCorte"].ToString()),
                        StatusCorte = int.Parse(reader["StatusCorte"].ToString()),
                        Cpf = decimal.Parse(reader["CPF"].ToString()),
                        Nome = reader["Nome"].ToString(),
                        Sobrenome = reader["SNome"].ToString(),
                        DataNasc = DateTime.Parse(reader["DataNasc"].ToString()),
                        Tel = reader["Tel"].ToString(),
                        Email = reader["Email"].ToString()
                    };
                    obj.Add(hasObj);
                }

                return obj;
            }
        }

        public HorariosMarcadosModel BuscarCorteCliente(Costumer costumer)
        {
            using (contexto = new Contexto())
            {
                var Cpf = new SqlParameter("@Cpf", SqlDbType.Decimal) { Value = costumer.Cpf };
                var reader = contexto.ExecutaProcedureComRetorno("FBSP_BuscarCorteCliente", Cpf);

                if (reader.Read())
                {
                    var obj = new HorariosMarcadosModel()
                    {
                        HorarioId = int.Parse(reader["Id"].ToString()),
                        Id_Cliente = int.Parse(reader["Id_Cliente"].ToString()),
                        StatusCorte = int.Parse(reader["StatusCorte"].ToString()),
                        BarberId = int.Parse(reader["BarberId"].ToString()),
                        DataCorte = DateTime.Parse(reader["DataCorte"].ToString()),
                        TempoCorte = reader["TempoCorte"].ToString()
                    };
```

Fonte: elaborado pelos autores

Para a persistência dos dados retornados pelo repository de horários agendados, também é criado a model de horários agendados, onde os valores serão armazenados e manipulados no front-end **(Figura 7)**.

Figura 7 – Model horários agendados

```
namespace fastBarberTG.Models
{
    public class HorariosMarcadosModel: Costumer
    {
        public int HorarioId { get; set; }
        public int Id_Cliente { get; set; }
        public DateTime DataCorte { get; set; }
        public int StatusCorte { get; set; }
        public int BarberId { get; set; }
        public string TempoCorte { get; set; }

        public string RetornoIdade()
        {
            DateTime dataApenas = DataNasc.Date;
            DateTime _dataAtual = Geral.DataAtual;

            int idade = _dataAtual.Year - DataNasc.Year;

            if (_dataAtual < DataNasc.AddYears(idade))
            {
                idade--;
            }

            return idade.ToString();
        }

        public string HoraCorte()
        {
            return DataCorte.ToShortTimeString();
        }

        public string DataCorteFormatado() => DataCorte.ToShortDateString();

        public string RetornaStatus()
        {
            if (StatusCorte == 2)
                return "yellow";

            if (StatusCorte == 1)
                return "green";

            if (StatusCorte == 3)
                return "red";

            return "";
        }

        public int RetornaHoraDiaInt() => int.Parse(DataCorte.Hour.ToString());
    }
```

Fonte: elaborado pelos autores

Na imagem **(Figura 8)** é representado o repository do barbeiro, onde realizado todas ações em que o barbeiro é necessário. Sendo assim só é utilizado para realizar o login e a alteração de senha do usuário.

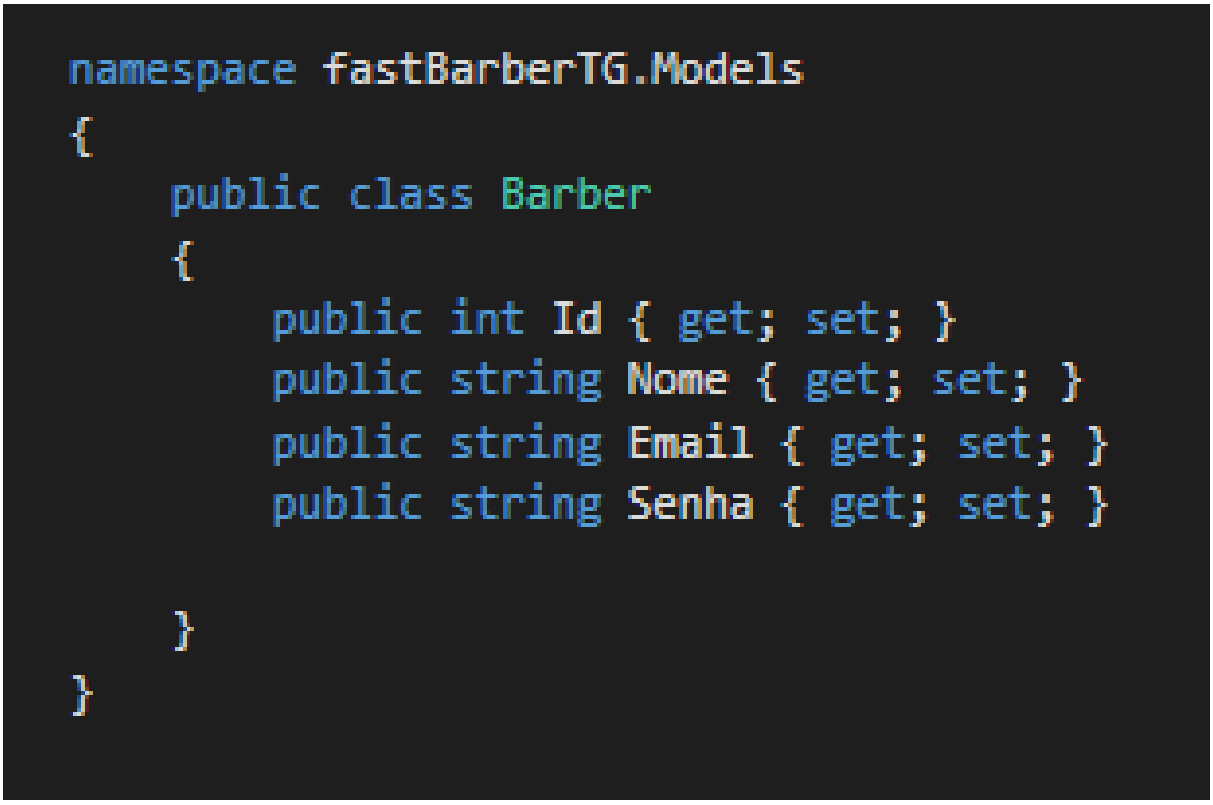
Figura 8 – Repository barbeiro

```
namespace fastBarberTG.Models.Repositories
{
    public class BarberREPO
    {
        private Contexto contexto;

        public bool LoginUsuario(string email, string senha)
        {
            using (contexto = new Contexto())
            {
                var Email = new SqlParameter("@Email", SqlDbType.NVarChar) { Value = email };
                var Senha = new SqlParameter("@Senha", SqlDbType.NVarChar) { Value = senha };
                var reader = contexto.ExecutaProcedureComRetorno("FBSP_LoginUsuario", Email, Senha);
                var hasObj = new Barber();
                while (reader.Read())
                {

                    {
                        hasObj.Id = int.Parse(reader["Id"].ToString());
                        hasObj.Nome = reader["Nome"].ToString();
                        hasObj.Email = reader["Email"].ToString();
                        hasObj.Senha = reader["Senha"].ToString();
                    };
                }

                if (hasObj.Email == null)
                    return false;

                return true;
            }
        }

        public void AlterarSenha(string Senha)
        {
            using (contexto = new Contexto())
            {
               var senha = new SqlParameter("@Password", SqlDbType.VarChar) { Value = Senha};
               contexto.ExecutaProcedure("FBSP_AlterarSenha", senha);
            }
        }
    }
}
```

Fonte: elaborado pelos autores

Para persistências dos dados do repository barbeiro é realizado a criação da model para persistência dos dados. **(Figura 9).**

Figura 9 – Model barbeiro



```
namespace fastBarberTG.Models
{
    public class Barber
    {
        public int Id { get; set; }
        public string Nome { get; set; }
        public string Email { get; set; }
        public string Senha { get; set; }

    }
}
```

Fonte: elaborado pelos autores

Na seguinte imagem **(Figura 10)** é apresentado o repository de horário de funcionamento, onde é possível realizar a alteração dos dias de funcionamento conseguindo definir os horários em que o cliente do barbeiro pode agendar um corte.

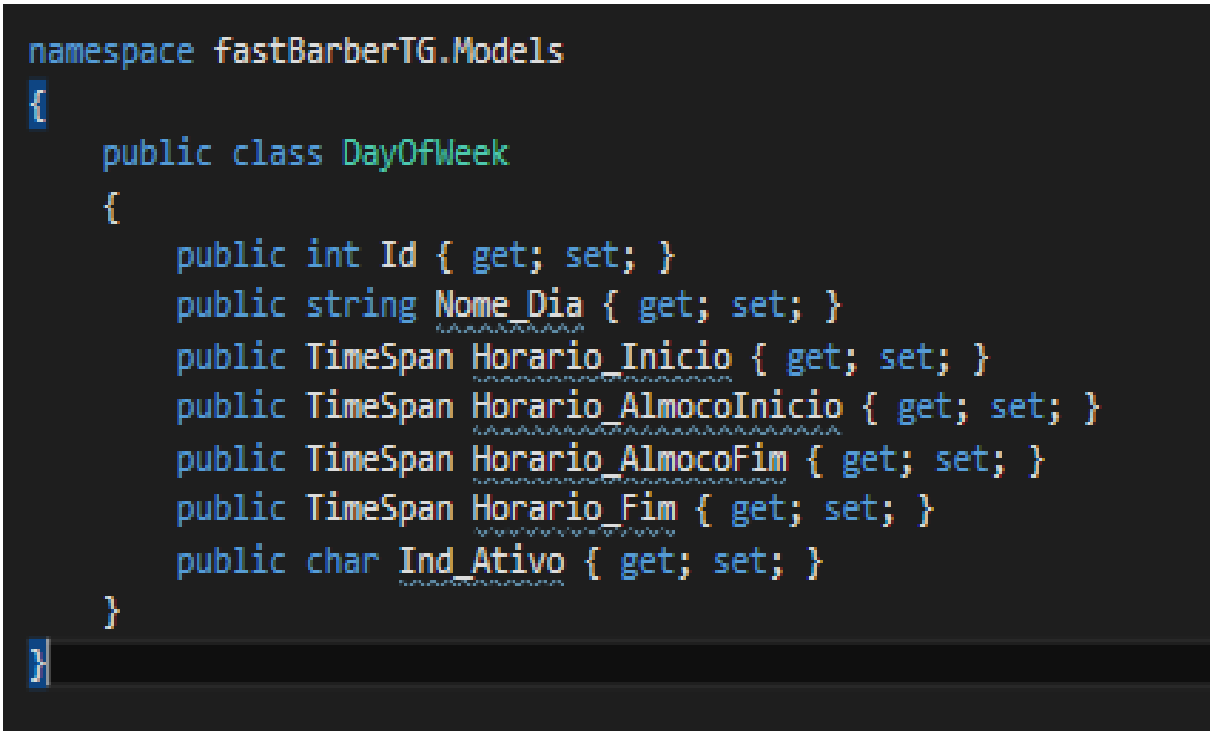
Figura 10 – Repository Horário de funcionamento

```
namespace fastBarberTG.Models.Repositories
{
    public class DayOfWeekREPO
    {
        private Contexto contexto;

        public IEnumerable<DayOfWeek> DaysOfWeek()
        {
            using (contexto = new Contexto())
            {
                var reader = contexto.ExecutaProcedureComRetorno("FBSP_BuscaHorarioFunc");
                var obj = new List<DayOfWeek>();
                while (reader.Read())
                {
                    var hasObj = new DayOfWeek()
                    {
                        Id = int.Parse(reader["Id"].ToString()),
                        Nome_Dia = reader["Nome_Dia"].ToString(),
                        Horario_Inicio = TimeSpan.Parse(reader["Horario_Inicio"].ToString()),
                        Horario_AlmocoInicio = TimeSpan.Parse(reader["Horario_AlmocoInicio"].ToString()),
                        Horario_AlmocoFim = TimeSpan.Parse(reader["Horario_AlmocoFim"].ToString()),
                        Horario_Fim = TimeSpan.Parse(reader["Horario_Fim"].ToString()),
                        Ind_Ativo = char.Parse(reader["Ind_Ativo"].ToString())
                    };
                    obj.Add(hasObj);
                }

                return obj;
            }
        }
    }
}
```

Fonte: elaborado pelos autores

Na imagem **(Figura 11)** é demonstrado a model armazenar e retornar os dados persistidos na tela.

Figura 11 – Model Horário de funcionamento



```
namespace fastBarberTG.Models
{
    public class DayOfWeek
    {
        public int Id { get; set; }
        public string Nome_Dia { get; set; }
        public TimeSpan Horario_Inicio { get; set; }
        public TimeSpan Horario_AlmocoInicio { get; set; }
        public TimeSpan Horario_AlmocoFim { get; set; }
        public TimeSpan Horario_Fim { get; set; }
        public char Ind_Ativo { get; set; }
    }
}
```

Fonte: elaborado pelos autores

A imagem **(Figura 12)** representa o repository de cliente do barbeiro, onde é realizado um novo cadastro e também a validação se o cliente já existe.

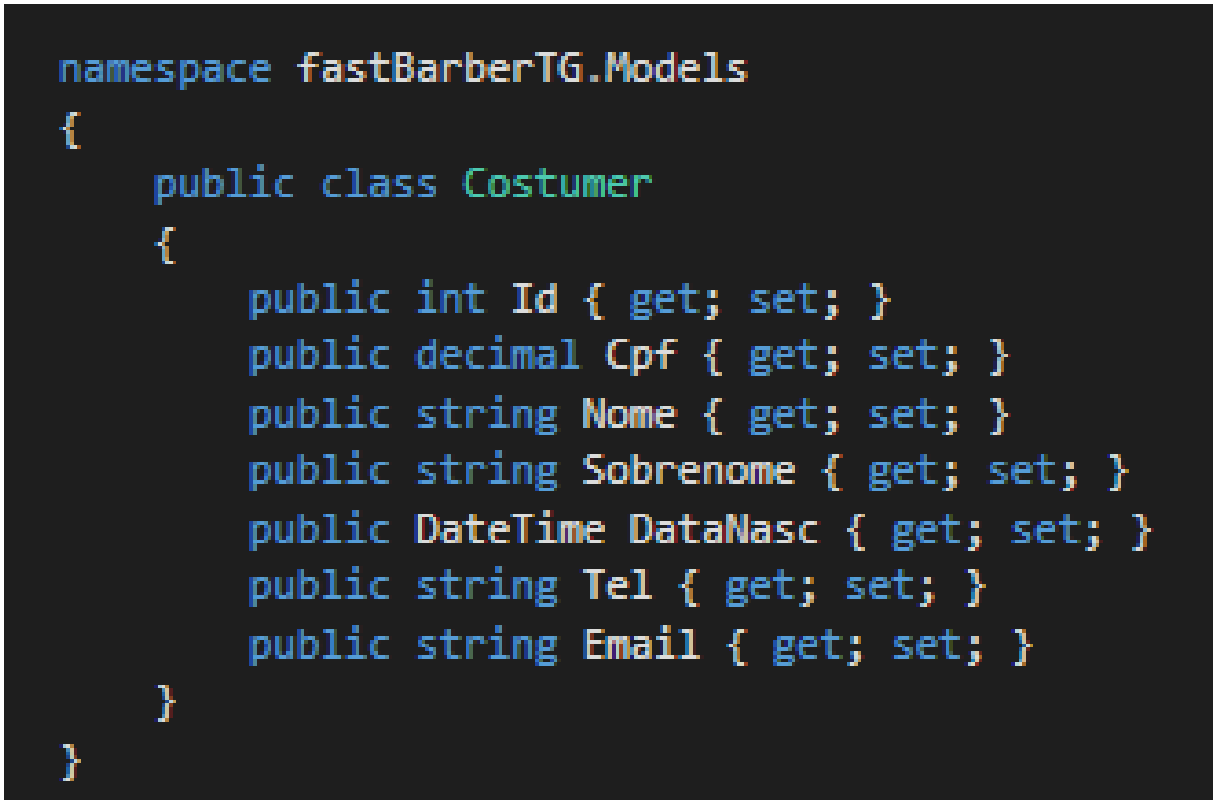
Figura 12 – Repository Cliente

```
public class NewCostumerREPO
{
    private Contexto contexto;

    public string AddCostumer(Costumer costumer)
    {
        using (contexto = new Contexto())
        {
            var Cpf = new SqlParameter("@Cpf", SqlDbType.Decimal) { Value = costumer.Cpf };
            var Nome = new SqlParameter("@Nome", SqlDbType.NVarChar, 20) { Value = costumer.Nome };
            var SNome = new SqlParameter("@SNome", SqlDbType.NVarChar, 10) { Value = costumer.Sobrenome };
            var DataNasc = new SqlParameter("@DataNasc", SqlDbType.Date) { Value = costumer.DataNasc };
            var Tel = new SqlParameter("@Tel", SqlDbType.NVarChar, 20) { Value = costumer.Tel };
            var Email = new SqlParameter("@Email", SqlDbType.NVarChar, 50) { Value = costumer.Email };

            var resultParameter = new SqlParameter("@Result", SqlDbType.NVarChar, 100);
            resultParameter.Direction = ParameterDirection.Output;

            var parameters = new SqlParameter[] { Cpf, Nome, SNome, DataNasc, Tel, Email, resultParameter };
            SqlDataReader reader = contexto.ExecutaProcedureComRetorno("FBSP_AddOrDenyCostumer", parameters);
            string resultMessage = resultParameter.Value.ToString();
            return resultMessage;
        }
    }


    public int ExistsCostumer(Costumer costumer)
    {
        using (contexto = new Contexto())
        {
            var Cpf = new SqlParameter("@Cpf", SqlDbType.Decimal) { Value = costumer.Cpf };
            var reader = contexto.ExecutaProcedureComRetorno("FBSP_ExistsCostumer", Cpf);

            int result = 0;
            while (reader.Read())
            {
                result = int.Parse(reader["Result"].ToString());
            };

            return result;

        }
    }
```

Fonte: elaborado pelos autores

Apresentação da model do cliente, onde é persistido os dados para apresentação em tela referente a um horário marcado.

Figura 13 – Model Cliente



```
namespace fastBarberTG.Models
{
    public class Costumer
    {
        public int Id { get; set; }
        public decimal Cpf { get; set; }
        public string Nome { get; set; }
        public string Sobrenome { get; set; }
        public DateTime DataNasc { get; set; }
        public string Tel { get; set; }
        public string Email { get; set; }
    }
}
```

Fonte: elaborado pelos autores

O responsável pelas estruturas html e css foi o razor pages, parte do ASP.NET MVC (Microsoft, 2013) responsável pela criação da view, que torna a exibição dos dados mais dinâmica e prática, sendo uma abordagem eficaz para gerar HTML e CSS possibilitando a introdução de dados de maneira eficiente.

A figura abaixo apresenta o layout criado com o razor pages utilizado na grande maioria das telas. **(Figura 14)**

Figura 14 – Layout da aplicação.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title</title>
    <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
    <link href="~/Content/Layout/_Layout.css" rel="stylesheet" type="text/css" />
    <link href="~/Content/Layout/UserStyle.css" rel="stylesheet" type="text/css" />
    <link href="~/Content/iziToast.css" rel="stylesheet" type="text/css" />
    <link href="~/Content/iziToast.min.css" rel="stylesheet" type="text/css" />
    <link href="~/Content/libcss.css" rel="stylesheet" type="text/css" />
    <script src="~/Scripts/jquery-1.10.2.min.js"></script>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FILL,GRAD@24,400,0,0" />
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FILL,GRAD@20..48,100..700,0..1,-50..200" />
    <script src="~/scripts/iziToast.min.js" type="text/javascript"></script>
    <script src="~/scripts/iziToast.js" type="text/javascript"></script>
</head>
<body>
    <header>
        <a href="/BarberControl/">
            <img src="~/Content/Logo/LoginLogo.png"/>
        </a>

        <a href="/User" class="displayUser">
            <span class="material-symbols-outlined">
                menu
            </span>
            <span>Olá, user!</span>
        </a>
    </header>

    <main>
        <div id="renderDiv">
            @RenderBody()
        </div>
    </main>

    <script src="~/Scripts/jquery-1.10.2.min.js"></script>
</body>
</html>
```

Fonte: elaborado pelos autores

Para realizar as requisições de dados, foi utilizado o Ajax do jQuery (Wikipedia, 2024), criado por John Resig em 2006. Essa tecnologia permite enviar dados para os controllers, que por sua vez retornam respostas de sucesso ou erro, funcionando essencialmente como uma API. Exemplo de controller **(Figura 16).**

Figura 15 – Requisição Ajax

```
var cadastrarCostumer = function () {
    event.preventDefault();
    var model = {
        Id: 0,
        Nome: $("#input-Nome").val(),
        Sobrenome: $("#input-subnome").val(),
        Cpf: formatCpf($("#input-cpf").val()),
        DataNasc: $("#input-datanasc").val(),
        Tel: $("#input-tel").val(),
        Email: $("#input-email").val()
    }

    $.post(config.urls.newCostumer, model).done(function (data) {

        if (data === 'Sucesso, Cliente cadastrado!') {
            iziToast.success({
                color: 'blue',
                title: 'Success',
                message: data,
            });

            //limpando campos do formulário de cadastro cliente
            $("#cpf-cliente").val(model.Cpf);
            $("#input-Nome").val("");
            $("#input-subnome").val("");
            $("#input-cpf").val("");
            $("#input-datanasc").val("");
            $("#input-tel").val("");
            $("#input-email").val("");

            $("#DontHaveAcc").hide();
            $("#HaveAcc").show("slow");

        }

        if (data === 'Cadastro inválido, Cliente já existe!') {
            iziToast.error({
                title: 'Error',
                message: "CPF já cadastrado!",
            });
        }

    }).fail(function (msg) {
        alert(msg.toString())
    })
}
```

Fonte: elaborado pelos autores

Figura 16 – Controller

```
public string NewCostumer(Costumer costumer)
{
    try
    {
        return _newCostumerREPO.AddCostumer(costumer);

    } catch (SqlException ex)
    {
        return ex.Message.ToString();
    }
}
```

Fonte: elaborado pelos autores

5 Resultados e Discussão

No início do projeto FastBarber, o objetivo principal era resolver, de forma geral, o desafio enfrentado pelas barbearias de Franca no tocante ao agendamento de horários. Para melhor entendimento do cenário, foi realizado um levantamento de requisito na barbearia Nova Geração, onde foi possível compreender as principais funcionalidades necessárias para o desenvolvimento do software.

O principal problema, o desperdício de tempo cotidiano em que o barbeiro precisa realizar o agendamento de forma manual, foi solucionado pelo software desenvolvido. Todo o relato do cliente foi transformado em uma automação para agendamento de corte, fazendo com que o tempo para realizar a atividade se diminuísse drasticamente, uma vez que o próprio cliente seria capaz de agendar seu horário.

O teste da aplicação fora realizado barbearia Nova Geração, no dia 01/04/2024. Para esse teste foi combinado com o barbeiro para que fosse enviado um aviso prévio aos seus clientes no dia anterior, para que o agendamento fosse realizado por meio da aplicação pelo link do FastBarber para agendar seu corte.

Por se tratar de algo atípico na barbearia, os clientes ainda não estavam acostumados, portanto, ocorreram alguns desencontros pela falta de informação propagada para todos. Nesse sentido, houveram alguns casos que entraram em contato com o barbeiro pelo WhatsApp para marcar o corte. Mesmo assim, o resultado foi satisfatório, onde 15 clientes fizeram o agendamento de maneira automática sem

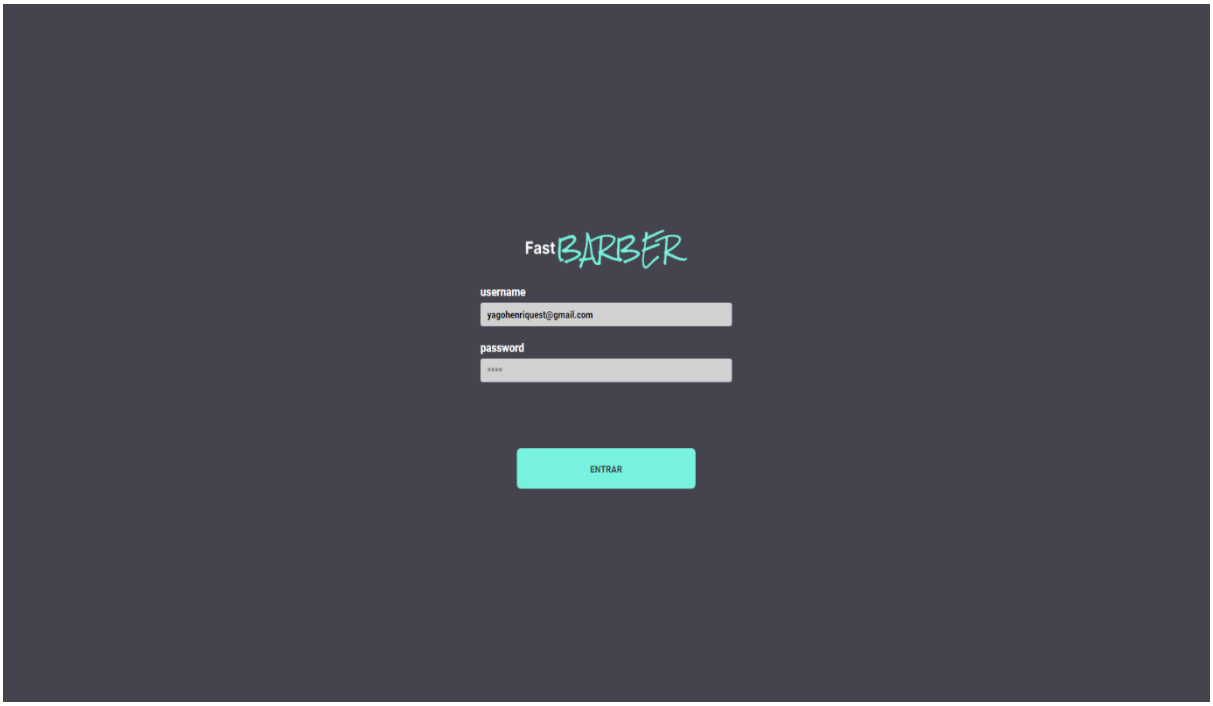
precisar realizar contato direto com o barbeiro e 5 clientes necessitaram do agendamento manual por não terem conhecimento da aplicação. Ao final do dia, o total de cortes foram 20, sendo essa a média diária de corte do barbeiro.

O agendamento de corte de forma manual foi reduzido em mais de 75%, demonstrando uma alta eficiência. Por conta desse resultado, o FastBarber recebeu um ótimo feedback por parte do funcionário da barbearia NovaGeração, e por conta disso o barbeiro teria interesse em realizar um investimento e até oferecer novas ideias de implementações. Assim, considera-se um sucesso a aplicação do software.

Para apresentação do resultado visual do front-end apresentando ao barbeiro, a aplicação se inicia na tela de login, onde o usuário (barbeiro) irá possuir um acesso disponibilizado pelos criadores do software, fazendo com que o ele seja totalmente controlado pelo próprio usuário **(Figura 17).**

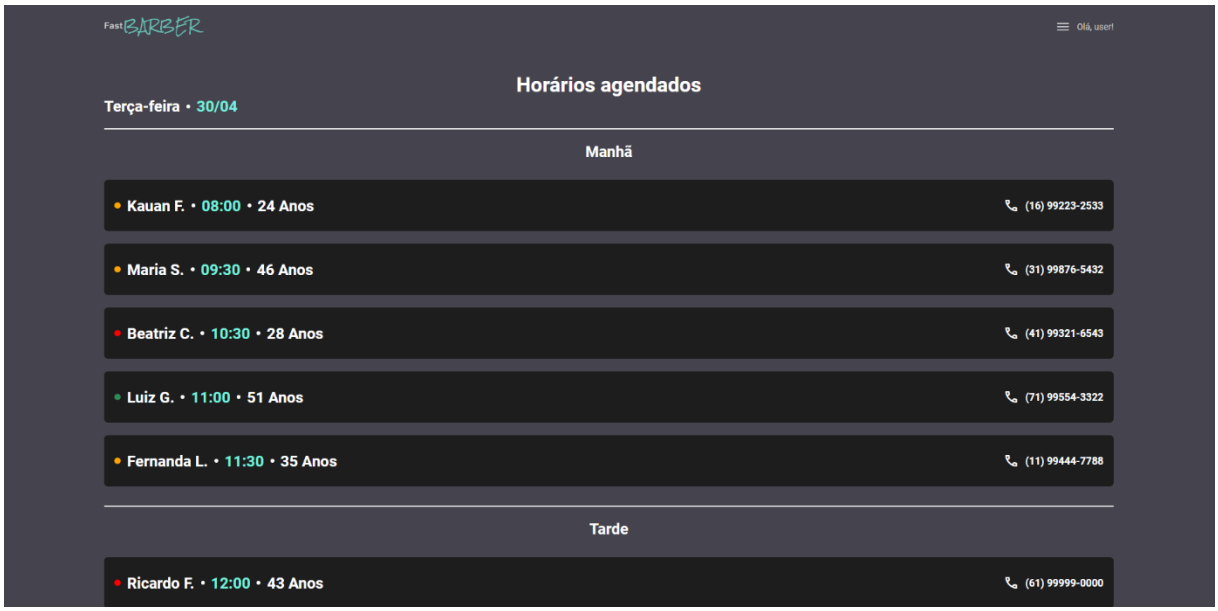
Logo após realizar o login, o usuário é redirecionado para a Home da aplicação, onde é possível visualizar os cortes marcados para o respectivo dia **(Figura 18)**.

Figura 17 – Tela inicial login



Fonte: elaborado pelos autores

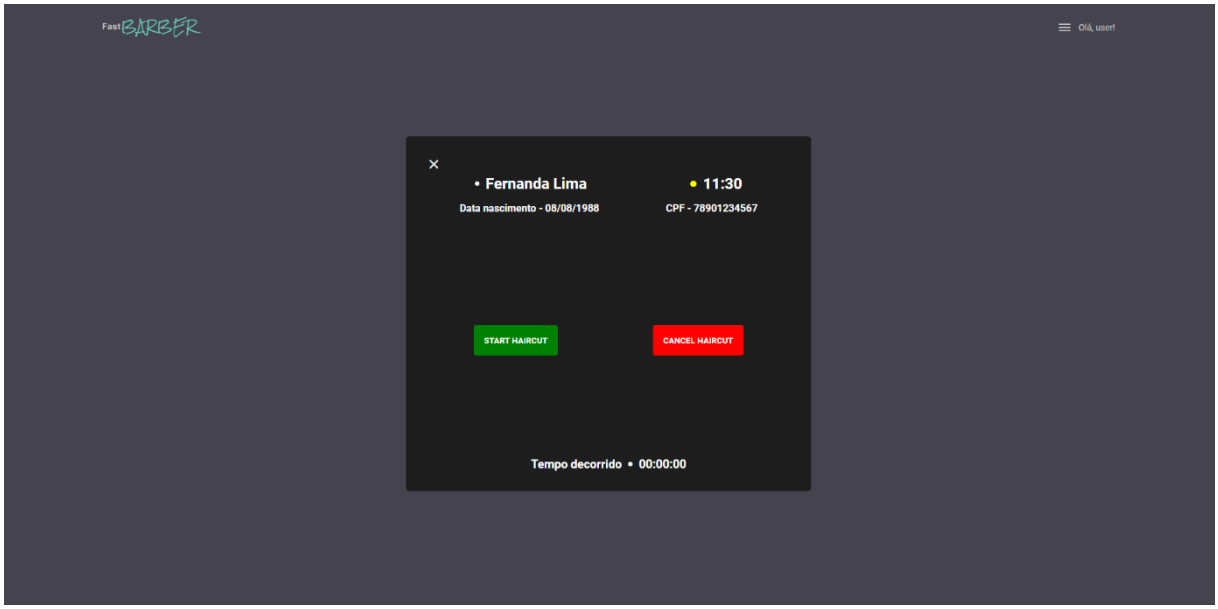
Figura 18 – Tela após login.



Fonte: elaborado pelos autores

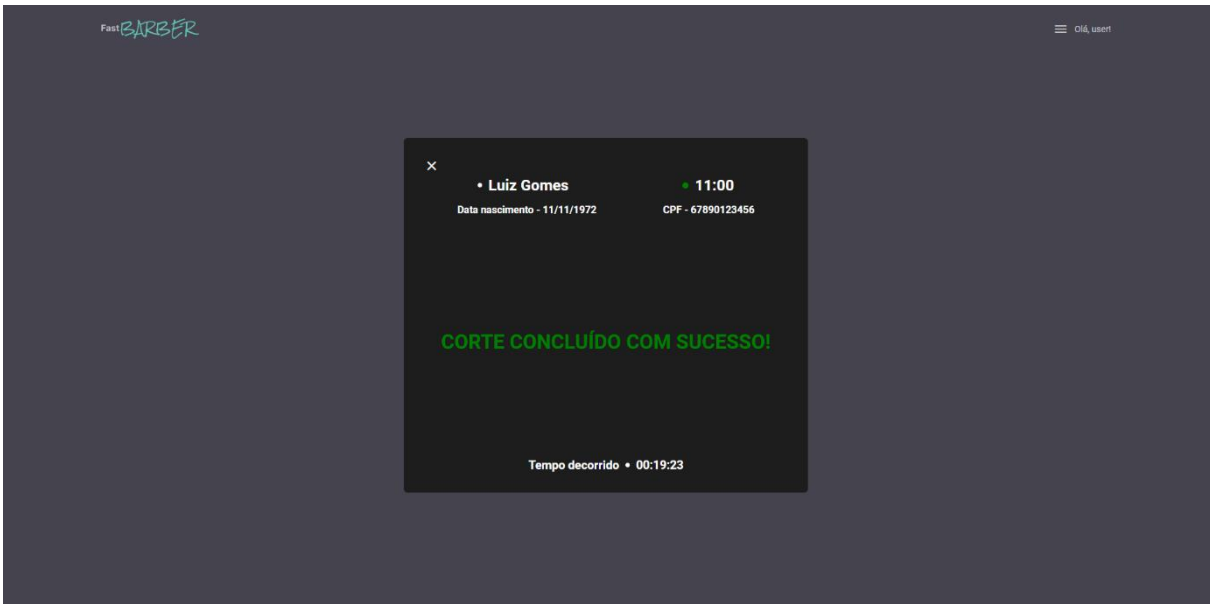
Nos cortes com horário marcado **(Figura 19)**, o software possibilita que o tempo seja cronometrado. Além disso, também é possível realizar o cancelamento do corte.

Figura 19 – Tela de acesso a um corte.



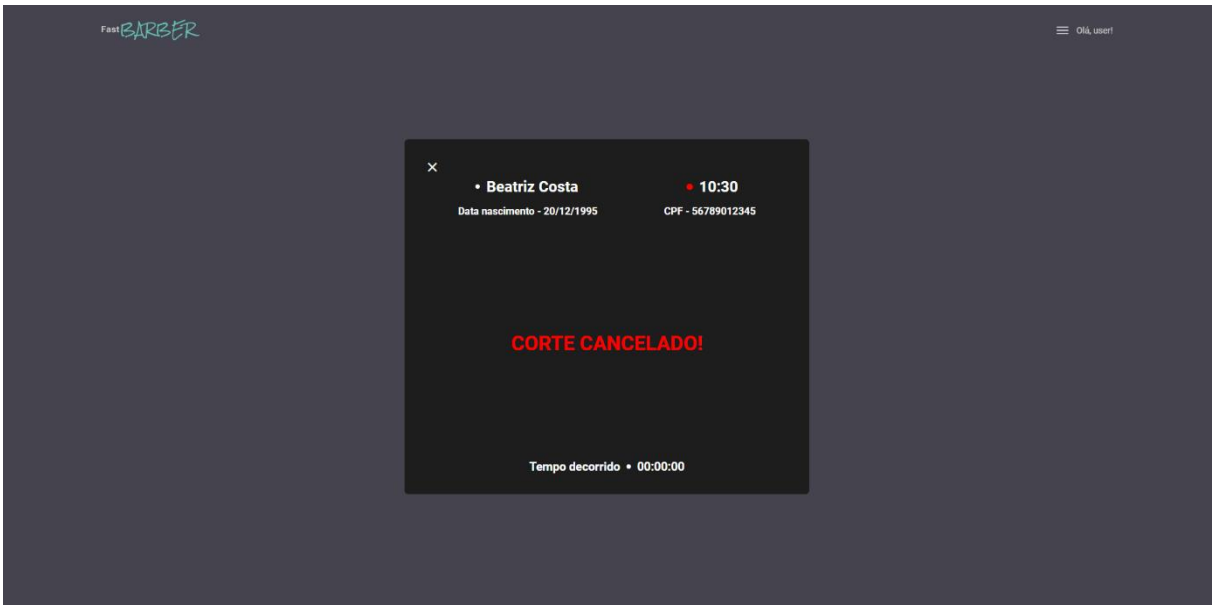
Fonte: elaborado pelos autores

Figura 20 – Corte finalizado



Fonte: elaborado pelos autores

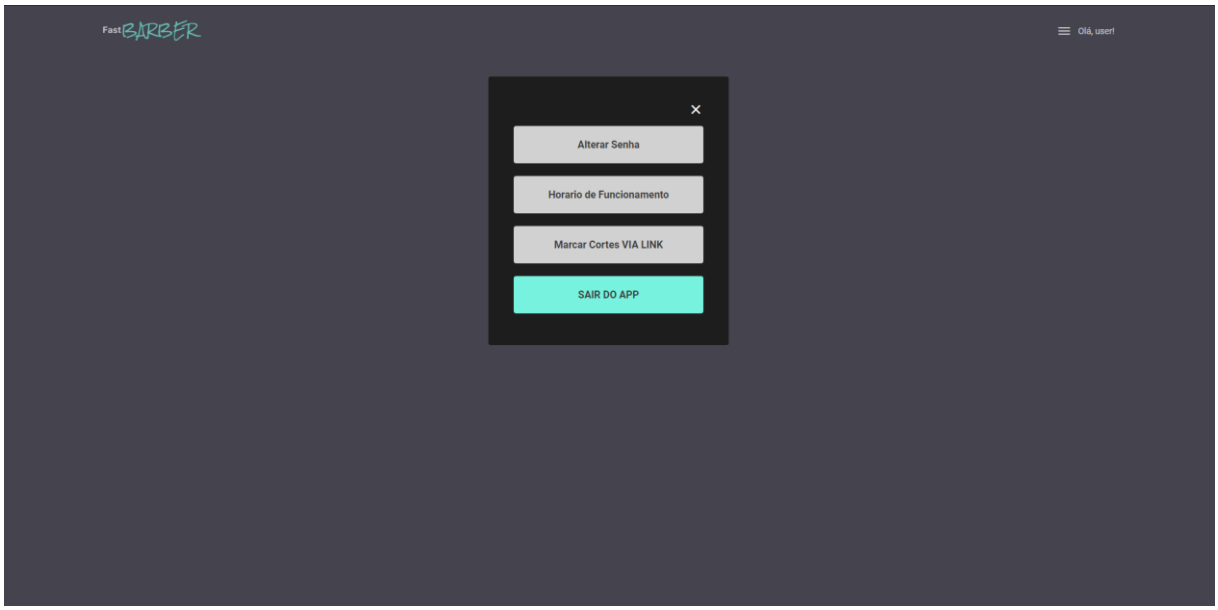
Figura 21 – Corte cancelado.



Fonte: elaborado pelos autores

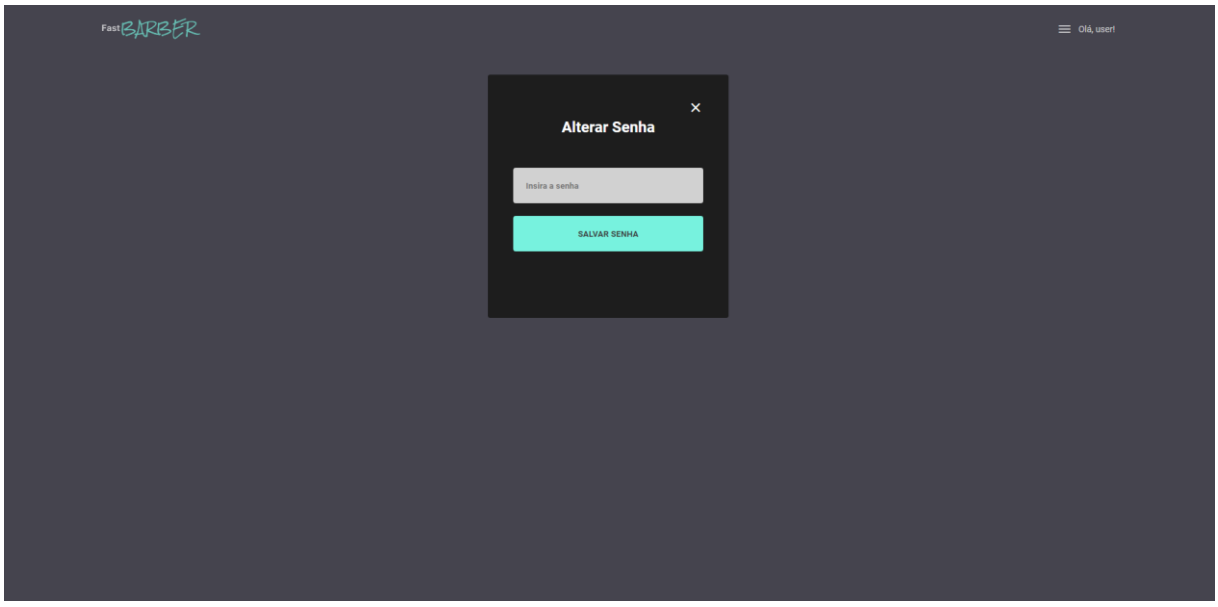
A aplicação conta também com a parte de suporte ao usuário, podendo acessar o menu de opções **(Figura 22).**

Figura 22 – Tela de opções disponíveis ao usuário



Fonte: elaborado pelos autores

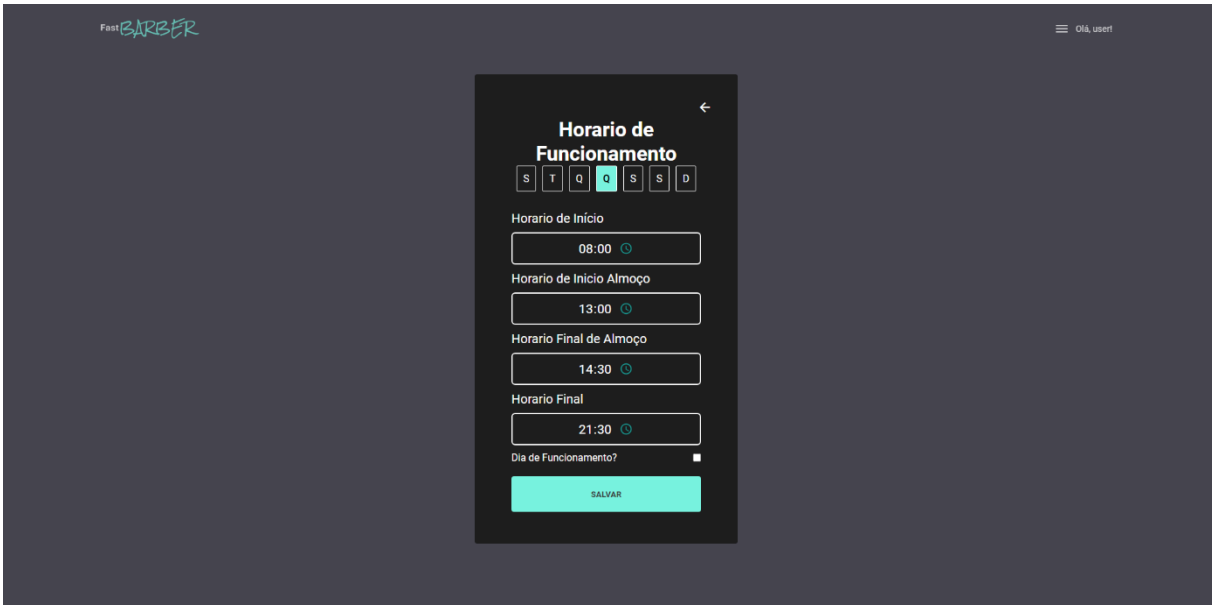
Figura 23 – Tela para alterar senha



Fonte: elaborado pelos autores

Nestas opções podemos alterar o horário em que a aplicação irá permitir que os cortes sejam marcados pelo cliente do usuário **(Figura 24)**.

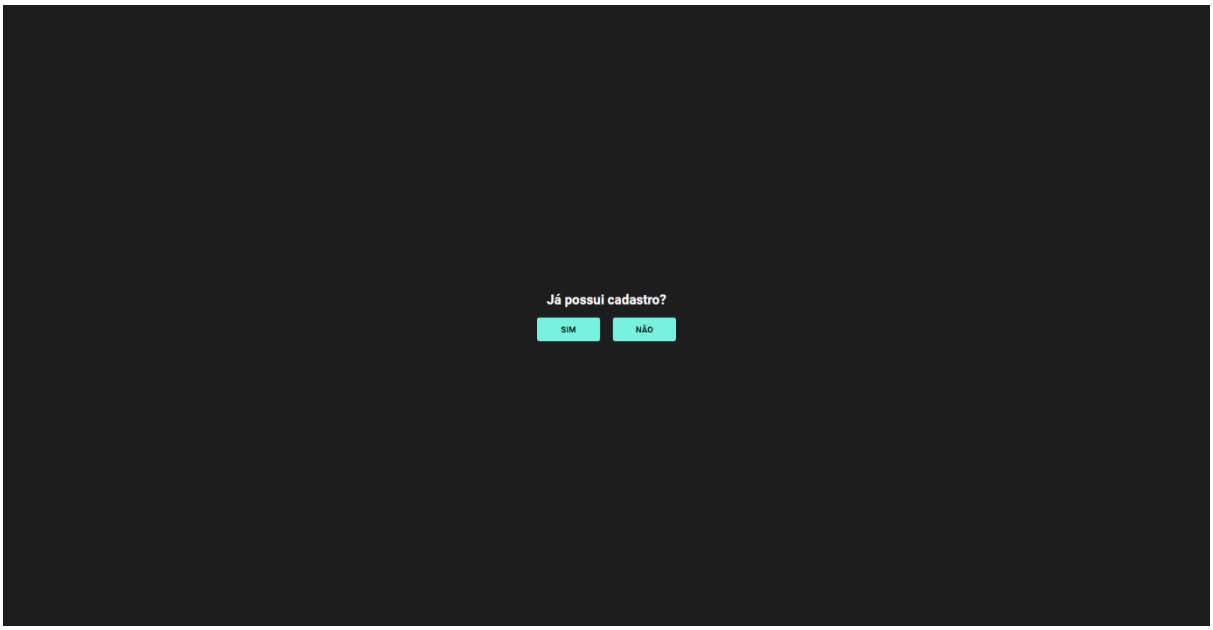
Figura 24 – Tela de horário de funcionamento



Fonte: elaborado pelos autores

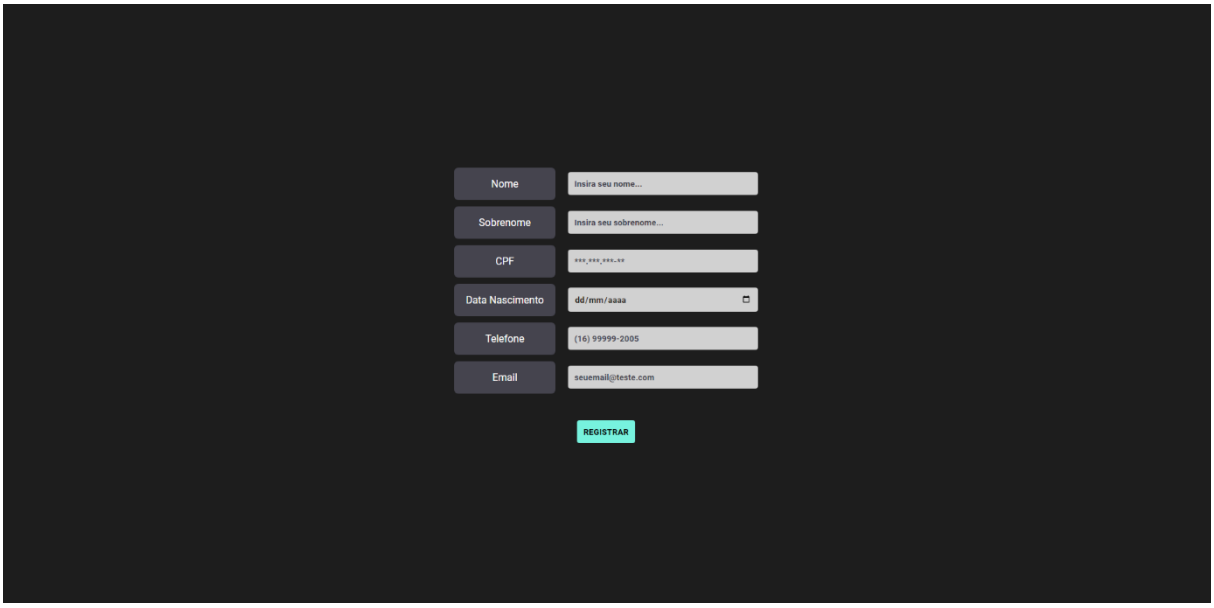
Na tela de cadastro de horários será disponibilizado o link para que o cliente do usuário possa realizar o agendamento do corte de cabelo. Caso possua um cadastro, somente selecionar a opção sim e redirecionar para a tela de agendamento. Caso não, o cliente será redirecionado para tela de cadastro e posteriormente para a tela de agendamento. **(Figura 25)**

Figura 25 – Marcar Cortes via link



Fonte: elaborado pelos autores

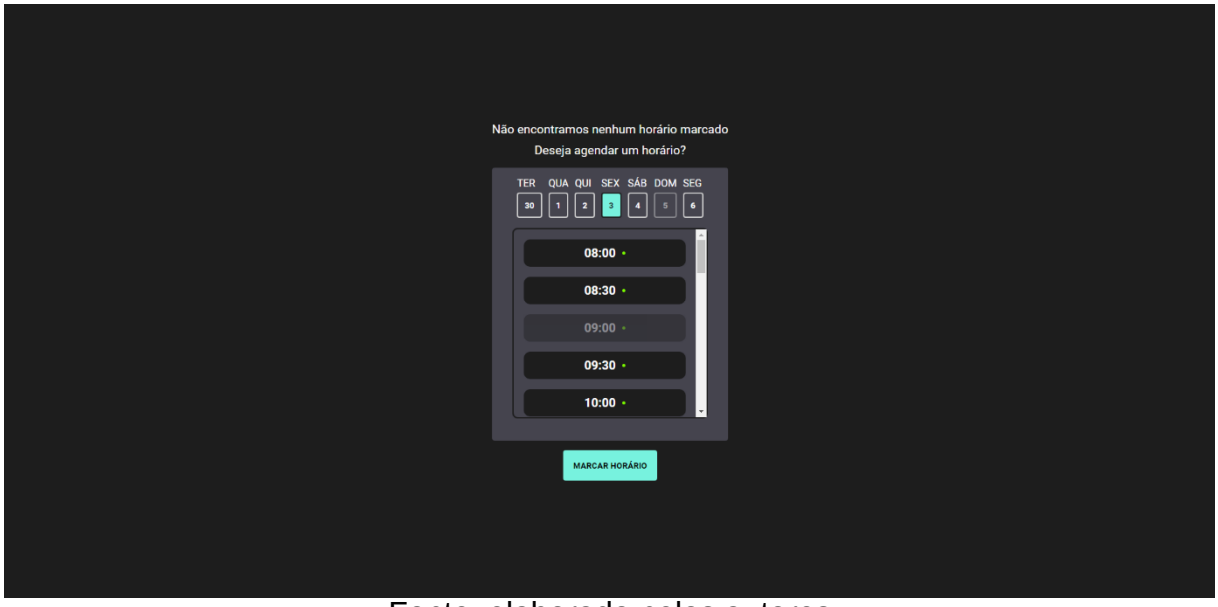
Figura 26 – Tela de registro de cliente.



Fonte: elaborado pelos autores

A imagem **(Figura 27)** apresenta a tela de visualização do cliente dos horários e dias disponíveis para o agendamento de corte com base nos dias definidos pelo usuário na figura 24.

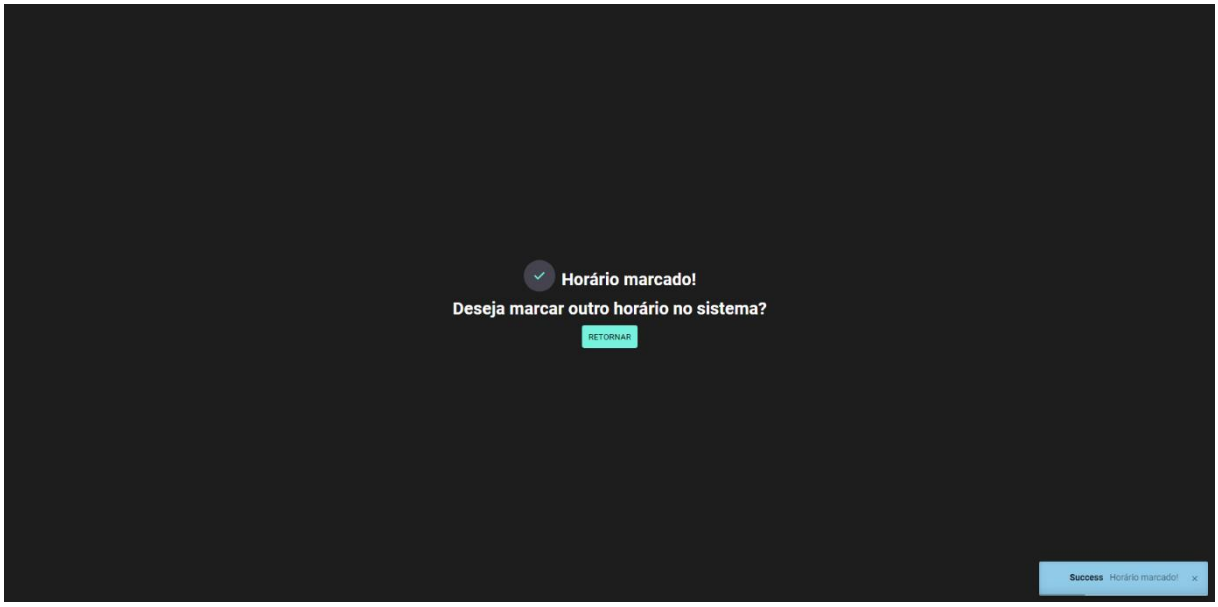
Figura 27 – Agendar corte



Fonte: elaborado pelos autores

Tela de redirecionamento, quando é apresentado sucesso ao agendar um corte **(Figura 28).**

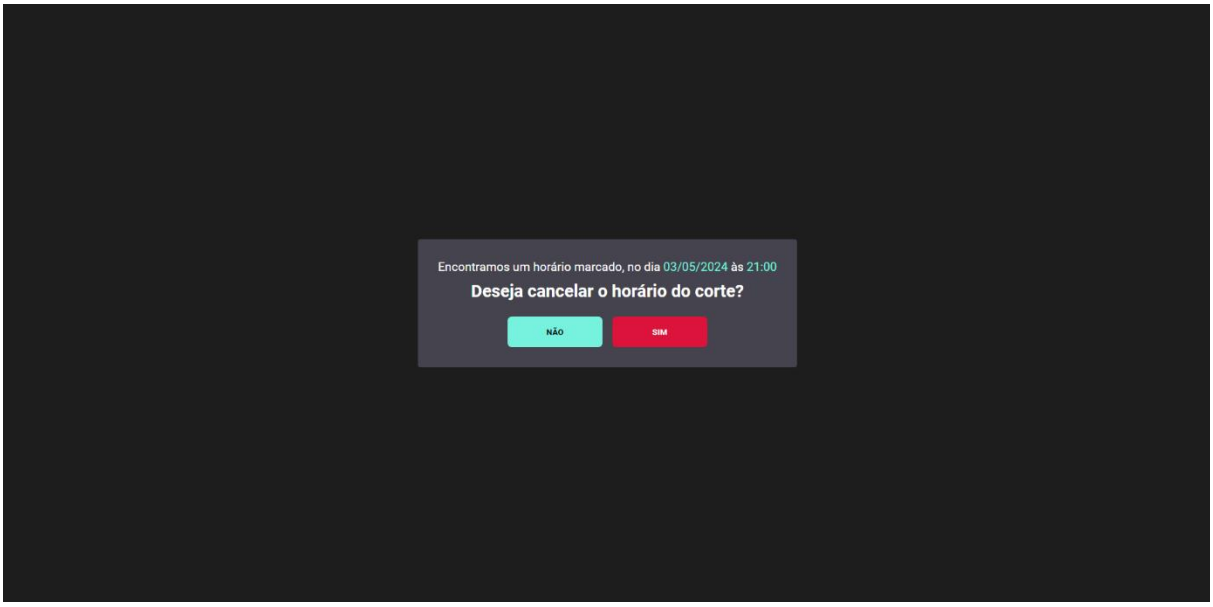
Figura 28 – Corte agendado



Fonte: elaborado pelos autores

Caso exista um horário marcado, será informado para o cliente que já se encontra um corte e se o mesmo deseja o cancelar. **(Figura 29)**

Figura 29 – Corte agendado existente



Fonte: elaborado pelos autores

Considerações finais

Ao analisar o cotidiano dos barbeiros que trabalham com uma equipe e estrutura reduzida, nota-se uma dificuldade relacionada ao agendamento de horários. Isso ocorre devido à disponibilidade reduzida dos profissionais, que precisam atender os clientes presencialmente, na barbearia, e também respondê-los nas redes sociais. Dessa forma, o processo se torna desafiador, e a barbearia muitas vezes perde oportunidades de atendimento devido à demora para realizar o agendamento.

Assim, buscando solucionar essa questão, foi criado o FastBarber, um software especializado em facilitar os agendamentos das barbearias. Seu principal objetivo é otimizar os processos dos estabelecimentos, evitando a perda de clientes e reduzindo as responsabilidades dos barbeiros. Para isso, foi desenvolvido para que os próprios clientes se tornassem capazes de agendar seus horários, tornando o agendamento mais rápido e prático.

O software foi desenvolvido a partir da plataforma ASPNET 4.5.1 (Microsoft, 2013) e, buscando uma estrutura sólida e atualizada, as linguagens utilizadas foram C# (Microsoft, 2000) e JavaScript (MDN, 2012). Dessa forma, o FastBarber utiliza ferramentas imensamente relevantes, que proporcionam fáceis adaptações referentes a possíveis atualizações ou mudanças que possam ser solicitadas pelo cliente. Para o armazenamento de dados, a escolha foi o Microsoft SQL Server, capaz de atender satisfatoriamente as demandas das barbearias analisadas.

Para comprovar sua eficiência, um teste de um dia foi realizado na barbearia Nova Geração. Os resultados se mostraram satisfatórios, uma vez que o software foi capaz de reduzir em cerca de 50% os agendamentos manuais. Esse número ainda pode ser melhorado após mais alguns dias de utilização, já que muitos clientes não o utilizaram por não estarem cientes de seu funcionamento. Dessa forma, considera-se um sucesso a aplicação inicial do FastBarber.

Referências

Bruna. As 10 Linguagens de Programação Mais Usadas em 2024: Aprimore suas Habilidades em Desenvolvimento Web. 2024. Disponível em:< https://www.hostinger.com.br/tutoriais/linguagens-de-programacao-mais-usadas>. Acesso em: 22.fev.2024

BestBarber. **BestBarber**. Disponível em: <http://bestbarbers.app>. Acesso em: 10.mar.2024

CODEPROJETS, **Visual representation of SQL joins,** 10/01/2015. Disponível em: <http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins>. Acesso em: 05.out.2015.

DATE, C J. **Introdução a sistemas de banco de dados**. 8 ed. Rio de Janeiro: Elsevier, 2003.

Diego, Lima. **C# - Saiba O Que é, Como Surgiu E Para Que é Utilizado**. Disponível em: <https://www.dio.me/articles/c-saiba-o-que-e-como-surgiu-e-para-que-e-utilizado>. Acesso em: 23.fev.2024

Edmundo, F. **O que é Clean Code e Quais são Suas Regras Básicas?.** 2020. Disponível em:< https://blog.accurate.com.br/clean-code>. Acesso em: 23.fev.24.

Edwin, L. Eugênio, R. **C# E .NET – GUIA DO DESENVOLVEDOR.** 2002. Disponível em: <http://www.etelg.com.br/paginaete/downloads/informatica/apostila2.pdf>. Acesso em: 10.jan.2024

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistema de banco de dados**. 4 ed. São Paulo: Pearson Addison Wesley, 2005.

Fernanda. **Elicitação de requisitos**. Disponível em:< https://medium.com/@fnandaleite/elicita%C3%A7%C3%A3o-de-requisitos-ff98a998189a>. Acesso em 16.dez.2023

FIGMA. Figma. Disponível em:<https://www.figma.com>. Acesso em: 20.set.23.

FRANK, D. R. SEIBT, L. JavaScript. s.d. Disponível em: <https://fit.faccat.br/~leonardoseibt/ArtigoJavaScript.pdf>. Acesso em: 10.mar.24.

IBICT. INSTITUTO BRASILEIRO DE INFORMAÇÃO EM CIÊNCIA E TECNOLOGIA**. Bibliografia Brasileira de Ciência da Informação:** 2004/2006. Brasília: IBICT, 2007. 64pp.

João, R. **O que é SOLID: O guia completo para você entender os 5 princípios da POO.** 2019. Disponível em:<https://medium.com/desenvolvendo-com-paixao/o-que-%C3%A9-solid-o-guia-completo-para-voc%C3%AA-entender-os-5-princ%C3%ADpios-da-poo-2b937b3fc530>. Acesso em: 20/03/2024.

jQuery. **jQuery.** Disponível em:<https://jquery.com>. Acesso em: 20.set.23.

Liandro, S. **Introdução aos princípios do SOLID**. 2023. Disponível em: <https://www.dio.me/articles/introducao-aos-principios-do-solid>. Acesso em: 20.mar. 24.

Marcelo, D. **iziToast.** Disponível em:<https://izitoast.marcelodolza.com>. Acesso em: 20.set.23.
MARTIN, Robert C. Arquitetura Limpa: O Guia do Artesão para Estrutura e Design de Software. Rio de Janeiro: Alta Books, 2019.

MDN Web Docs. **JavaScript.** Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em 23.jun.2023

Microsoft. **ASP.NET 4.5.1.** Disponível em: <https://dotnet.microsoft.com/pt-br/download/dotnet-framework/net451>. Acesso em: 05.jun.2024

MICROSOFT. **Bem-vindo ao IDE do Visual Studio**. 2022. Disponível em: <https://docs.microsoft.com/pt-br/visualstudio/get-started/visual-studio-ide?view=vs2022>. Acesso em: 05.mar.24

MICROSOFT. **C#**. Disponível em: <https://dotnet.microsoft.com/pt-br/languages/csharp>. Acesso em 23.jun.2023

Microsoft**. Microsoft Sql Server 2022.** Disponível em: < https://www.microsoft.com/pt-br/sql-server/sql-server-downloads>. Acesso em: 25.jun.2023

Vinicius, N. de Almeida. **O que é BPMN (Business Process Model and Notation) e como aplicar essa notação na Modelagem de Processos** Disponível em: <https://www.euax.com.br/2017/02/o-que-e-bpmn-business-process-model-and-notation>. Acesso em: 10.jan.2024.