INSTITUTO SUPERIOR POLITÉCNICO DE TECNOLOGIAS E CIÊNCIAS
DEPARTAMENTO DE ENGENHARIA E TECNOLOGIAS

Cândido Ucuahamba

Karapinha XPTO – Sistema de Gestão de Salão

LUANDA 2024

Cândido Ucuahamba

Karapinha XPTO – Sistema de Gestão de Salão

Documentação do projecto Karapinha XPTO apresentado na disciplina de Aplicações Web do curso de Engenharia Informática do Instituto Politécnico de Tecnologias e Ciências como exame normal.
Docente: Sediangani Sofrimento

LUANDA 2024

I. OBJECTIVO

Desenvolver uma aplicação web para a rede de salões Karapinha XPTO para melhorar o gerenciamento do salão e de todos os intervenientes do processo de atendimento, bem como armazenando e gerenciando os clientes, profissionais, serviços prestados e a marcação dos serviços solicitados por clientes..

II. DESCRIÇÃO DO PROJECTO

O projecto consiste desenvolvimento de um sistema de gestão de salões para a Karapinha XPTO, uma empresa proprietária de uma rede de salões em Angola. O sistema será uma aplicação web que permitirá o gerenciamento eficiente e organizado de todos os intervenientes de um salão, desde a solicitação do serviço pelo cliente até o acompanhamento do desempenho. A aplicação web terá os seguintes perfis de utilizadores distintos nomeadamente:

- Utilizador não registado: que poderá consultar a informação pública disponibilizada pelo salão como: os serviços disponíveis com os seus respectivos preços e outras informações relevantes sobre o salão, bem com o realizado um registo ou entrar com uma conta (caso já tenha se registado previamente) para ter acesso aos serviços;
- Utilizador registado: que poderá solicitar quantos serviços e marcações quiser, escolher o profissional e o horário em que deseja ser atendido (isto em função dos horários em que o profissional trabalha) e a data. De igual modo poderá acompanhar o seu processo de atendimento e verificar se o pagamento já foi validado.
- Administrativo: que poderá realizar o gerenciar (adicionar, editar e remover) de elementos como: categoria, serviços e profissionais e a validação de marcações feitas pelos clientes.
- Administrador: que poderá activar ou desactivar a conta de todo o cliente que se inscreve na aplicação bem como realização o registo de administrativos para o sistema e gerenciamento do mesmo.

III. INTRODUÇÃO

No cenário atual, a gestão de salões de beleza enfrenta diversos desafios que podem comprometer a eficiência e a qualidade do atendimento. Problemas comuns incluem a dificuldade em gerenciar marcações de serviços, a falta de visibilidade sobre a disponibilidade dos profissionais, a ausência de uma visão do desempenho dos salões. Para superar esses desafios, propõe-se o desenvolvimento de uma aplicação web abrangente que permitirá à Karapinha XPTO gerenciar eficientemente todos os aspectos dos seus salões. O sistema será composto por diferentes perfis de utilizadores: **Utilizador Não Registado**, **Utilizador Registado**, **Administrativo** e **Administrador**. Cada perfil terá funcionalidades específicas para atender às suas necessidades e contribuir para uma gestão mais eficaz e organizada. Este relatório detalha a arquitetura do sistema, o desenvolvimento das funcionalidades, os desafios enfrentados e as técnicas implementadas para atender às necessidades específicas da Karapinha XPTO, visando melhorar o gerenciamento e a operação dos salões de beleza.

IV. DIAGRAMAS

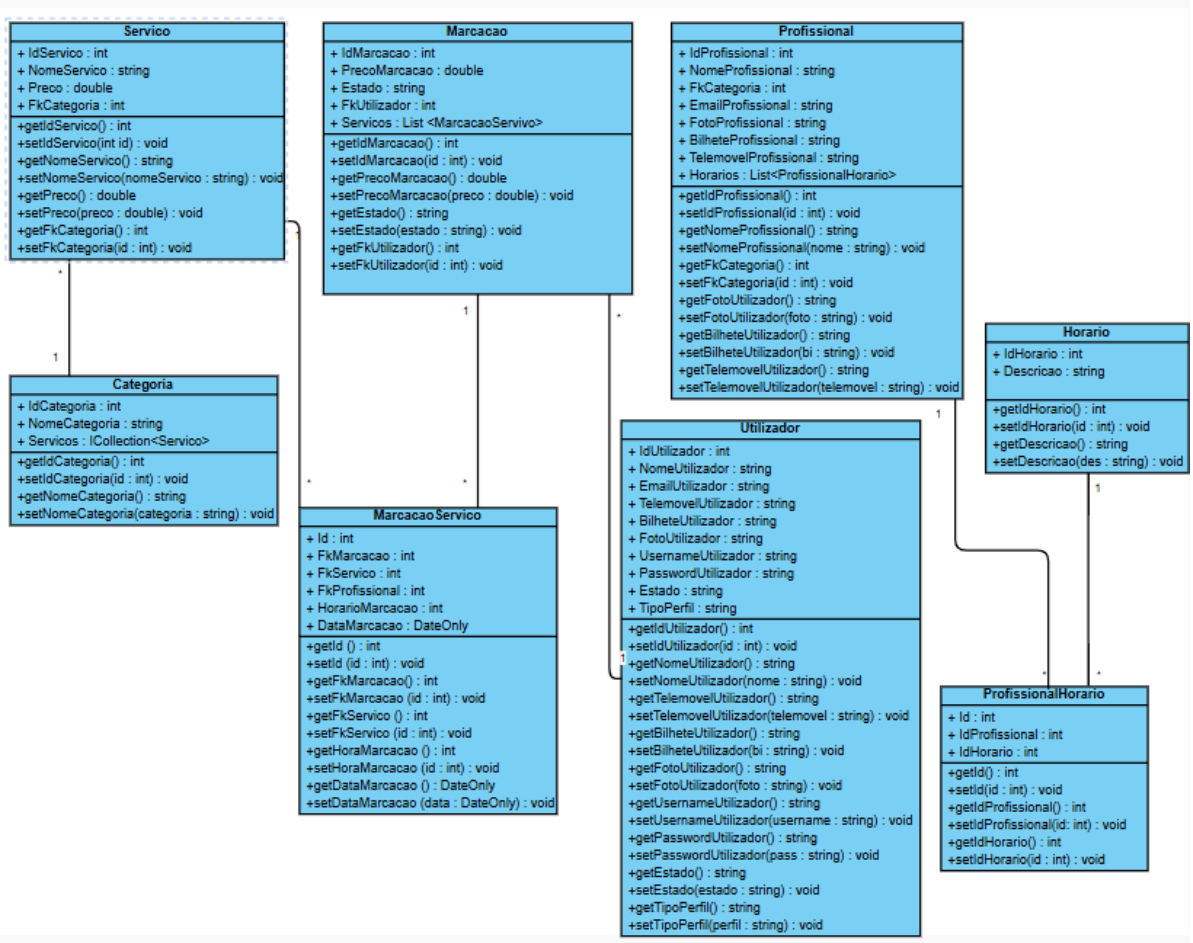

Figure 1. Diagrama de classes do projecto.

Figure 2. Diagrama de caso de uso do projecto

V. IMPLEMENTAÇÃO

- **Tecnologias e Linguagens**: para a construção da aplicação, foi usado Rect TypeScript, React Bootstrap, CSS para a parte do front-end, ASP.NET (C#), Entity FrameWork para a lógica de negócio e funcionalidade da aplicação (back-end) e para armazenar as informações foi usado o banco de dados SQL Server.
- **Arquitetura do Software**: foi usado a arquitetura Repository Pattern constituido por DAL ,DTO, Model, Shared, Services e Controller que dá uma boa organização, flexibilidade, testabilidade e reutilização de código. A abordagem utilizada para o desenvolvimento foi a Code-First onde as tabelas da base de dados se tornaram espelho das classes criadas na Model.
- **Módulos e Funcionalidades**: como foi descrito no problema temos algumas funcionalidades primordiais para as diferentes funções da aplicação. Abaixo serão listadas as funcionalidades fundamentais da aplicação:
    - **Utilizador Não Registado**: Consultar as informações públicas do salão sobre os serviços; Realizar o registo ou login para solicitar um serviço;

- **Utilizador Registado**: Consultar as serviços disponíveis, solicitar marcação de serviços, consultar o estado das marcações feitas e actualizador os dados de registo;
- **Administrativo**: Gerenciamento de profissionais, serviços e categoria; confirmação e reagendamento de marcações feitas por clientes;
- **Administrador**: Activação e Bloqueio das contas cliente e registo do administrativos.

VI. FLUXO DE DADOS

- **Registo de um Utilizador**: Quando um utilizador faz o registro na aplicação, é guardado o seu tipo, ou seja, um utilizador pode ser cliente, gestor ou administrador. Caso seja cliente, será enviado um email para o administrador poder ativar a sua conta, caso seja administrativo será enviado email ao mesmo com as suas credencias de login.
- **Login de um Utilizador**: Caso seja cliente e estiver com a conta activada, o login será permitido e será redirecionado para a página logada onde poderá ter acesso à todos os privilégios anteriormente descritos sobre este tipo de utilizador, se estiver desactivada o login não será permitido. Caso seja administrativo deverá trocar a palavra-passe no primeiro login, de seguida terá acesso as funcionalidades anteriormente descritas para este tipo de utlizador. Caso seja administrador será redirecionado para a sua página e poderá ter acesso ao que é devido conforme descrito anteriormente;
- **Inserção de Profissionais**: O administrativo regista profissionais com a respecitva, nome, email, telemóvel, bilhete e os horários em que o mesmo trabalha. Os profissionais de uma categoria podem atender todos os serviços da categoria na qual estão associados;
- **Inserção de serviços**: O administrativo regista serviços com a respectiva categoria, descrição e preço. Quando o serviço é registado, o mesmo fica automaticamente disponível para todos os clientes.
- **Marcação de serviços**: Sendo que todos os serviços disponíveis ficam a vista do cliente, o mesmo pode adicionar quantos quiser no seu carrinho, e posteriormente personalizar a sua marcação selecionado quais os profissionais irão atendê-lo em cada serviço com respectivo horario em que os mesmos trabalham, bem como a data de marcação, feito isso o estado da marcação fica pendente.
- **Confirmação e reagendamento de marcação de serviços**: Após a marcação ser feita o cliente aguardará pela resposta do administrativo, que pode resultar na confirmação ou reagendamento e todas as respostas são enviadas para o email do cliente.

VII. TÉCNICAS USADAS

- **Para o envio de e-mails**: Utilizou-se as bibliotecas **MailKit** e **System.Net.Mail**. Estas bibliotecas foram integradas em todas as funcionalidades que necessitavam de envio de notificações por e-mail, seja para ativação d bloqueio de contas, confirmação de marcações ou outras comunicações com o usuário. As classes `EmailService` e `EmailReceiver` foram criadas para centralizar e gerenciar o envio de e-mails, garantindo assim a reutilização do código e a consistência nas mensagens enviadas.
- **Mapeamento de modelos e DTO**: De acordo com a regra de negócio da aplicação, implementou-se um sistema de mapeamento entre os modelos e os Data Transfer Objects (DTOs). Este mapeamento é realizado dentro da pasta **Converter** na camada **DAL** e é essencial para garantir que os dados das entidades sejam transportados correctamente entre as diferentes camadas da aplicação. O mapeamento é particularmente importante para a segurança e a integridade dos dados. Este processo é realizado na camada **Service**, que actua como intermediário entre a camada de Data Access Layer) e o usuário final.
- **Fectch API**: Para o consumo das requisições e envio de respostas que foi a parte que mais constituiu esforço e dedicação, utilizou-se a API fetch do JavaScript, integrada na biblioteca **React**. A escolha pelo fetch foi motivada pela sua simplicidade e eficácia, não necessitando da instalação de dependências adicionais. A API fetch facilita o manuseio de promessas, permitindo que as requisições sejam feitas de forma assíncrona e tornando o código mais limpo e fácil de manter.
- **Hooks em React**: Para gerenciar o estado dos componentes de forma eficiente, utilizamos **hooks** do React, como useState e useEffect. Os hooks permitem que os componentes funcionais do React tenham um estado interno e efetuem efeitos colaterais, como a realização de fetch de dados. O uso de hooks tornou o código mais modular e reutilizável, permitindo uma clara separação de preocupações dentro dos componentes funcionais do React.

VIII. ALGUNS MÉTODOS E FUNÇÕES

```
4 references
public interface IProfissionalService
{
    2 references
    Task<ProfissionalDTO> CreateProfissional(ProfissionalDTO dto, IFormFile foto);
    2 references
    Task<ProfissionalDTO> GetProfissionalById(int id);
    2 references
    Task<ProfissionalDTO> GetProfissionalByIdCategoria(int id);
    2 references
    Task<IEnumerable<ProfissionalDTO>> GetAllProfissionals();
    2 references
    Task<bool> DeleteProfissional(int id);
    2 references
    IEnumerable<dynamic> GetAllProfissionaisWithCategoria();
    2 references
    Task<IEnumerable<dynamic>> GetAllProfissionaisByIdCategoria(int idCategoria);
}
```

Figure 3. Interface dos profissionais

```
4 references
public interface IUtilizadorService
{
    2 references
    Task<UtilizadorDTO> CreateUser(UtilizadorDTO Utilizador, IFormFile foto);
    2 references
    Task<UtilizadorDTO> GetUserById(int id);
    2 references
    Task<IEnumerable<UtilizadorDTO>> GetAllUsers();
    2 references
    Task<IEnumerable<UtilizadorDTO>> GetAllClientes();
    2 references
    Task<IEnumerable<UtilizadorDTO>> GetAllAdministratives();
    2 references
    Task<bool> DeleteUser(int id);
    2 references
    Task UpdateUser(UtilizadorUpdateDTO Utilizador);
    2 references
    Task ActivateOrDesactivateClient(int cliente);
    2 references
    Task<UtilizadorDTO> Login(LoginDTO login);
    2 references
    Task<string> GetUserRole(string username);
    2 references
    Task<bool> VerifyAdministrativeStatus(UtilizadorDTO dto);
    2 references
    Task ActivateAndChangePassword(string username, string password);
    2 references
    Task<int> GetUserIdByUsername(string username);
    2 references
    Task<UtilizadorDTO> GetUserByUsername(string username);
}
```

Figure 4. Interface do utilizador.

```
4 references
public interface IMarcacaoService
{
    2 references
    Task<MarcacaoDTO> CreateBooking(MarcacaoDTO marcacao);
    2 references
    Task<MarcacaoGetDTO> GetBookingById(int id);
    2 references
    Task<IEnumerable<MarcacaoGetDTO>> GetAllBookingByUserId(int idUtilizador);
    2 references
    IEnumerable<MarcacaoGetDTO> GetAllBookings();
    2 references
    Task<bool> ConfirmBooking(int id);
    2 references
    Task<bool> RescheduleBooking(int id, DateOnly data);
    0 references
    Task<bool> DeleteBooking(int id);
    0 references
    Task UpdateBooking(MarcacaoDTO marcacao);
}
```

Figure 5. Interface de marcações.

IX. CONCLUSÃO E BREVES CONSIDERAÇÕES

O desenvolvimento do sistema de gestão de salões para a Karapinha XPTO visa otimizar a administração de sua rede de salões em Angola, melhorando a eficiência operacional e a satisfação dos clientes. A aplicação web atende a diferentes perfis de utilizadores, proporcionando uma experiência personalizada e eficaz. Durante o desenvolvimento, enfrentei dificuldades ao trabalhar com o ASP.NET, sendo o primeiro framework com que tive contato. A robustez do framework e minha falta de experiência tornaram o processo desafiador, especialmente no consumo de requisições e respostas. No entanto, a implementação de envio de e-mails com MailKit e System.Net.Mail garantiu uma comunicação eficiente, enquanto o mapeamento de modelos para DTOs assegurou a integridade dos dados. O uso do fetch simplificou as requisições HTTP no frontend, e os hooks do React permitiram uma gestão eficiente do estado dos componentes, resultando em uma aplicação robusta e de fácil manutenção.

REFERÊNCIAS BIBLIOGRÁFICAS

ENTITY FRAMEWORK TUTORIAL : https://www.entityframeworktutorial.net/code-first/inheritance-strategy-in-code-first.aspx

REACT : https://react.dev/learn/typescript

REACT BOOTSTRAP : https://react-bootstrap.netlify.app/

GITHUB : https://github.com/eniodev/dotnet-core-mininal-api

STACKOVERFLOW : https://stackoverflow.com/

YOUTUBE : https://www.youtube.com/@PatrickGod

YOUTUBE: https://www.youtube.com/@amantinband

MICROSOFT LEARN : https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0

VISUAL PARADIGM: https://online.visual-paradigm.com/

GEEKS FOR GEEKS : GeeksforGeeks | A computer science portal for geeks

SQL TUTORIAL : https://www.w3schools.com/sql/default.asp

REACT TUTORIAL : https://www.w3schools.com/react/default.asp

JS TUTORIAL : https://www.w3schools.com/js/default.asp