UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÔPOLIS
PROGRAMA DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Caio Giesteira Cardoso

PROPOSTA DE SISTEMA DE GERENCIAMENTO PARA EMPRESA DE ASSESSORIA EM VISTO AMERICANO

Florianôpolis
2024

Caio Giesteira Cardoso

PROPOSTA DE SISTEMA DE GERENCIAMENTO PARA EMPRESA DE ASSESSORIA EM VISTO AMERICANO

Trabalho de Conclusão de Curso submetida ao Programa de Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do título de Graduação em Ciência da Computação.
Orientador: Prof. Carlos Becker Westphall, Dr.

Florianôpolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Cardoso, Caio Giesteira
PROPOSTA DE SISTEMA DE GERENCIAMENTO PARA EMPRESA DE ASSESSORIA EM VISTO AMERICANO / Caio Giesteira Cardoso ; orientador, Carlos Becker Westphall, 2024.
150 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2024.

Inclui referências.

1. Ciências da Computação. 2. Visto Americano. 3. Ciência da Computação. 4. Software. I. Westphall, Carlos Becker. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

Caio Giesteira Cardoso

PROPOSTA DE SISTEMA DE GERENCIAMENTO PARA EMPRESA DE ASSESSORIA EM VISTO AMERICANO

O presente trabalho em nível de Graduação foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof.Carlos Becker Westphall, Dr.
Universidade Federal de Santa Cararina

Prof.(a) Carla Merkle Westphall, Dr(a).
Universidade Federal de Santa Catarina

Ricardo do Nascimento Boing, Me
Universidade Federal de Santa Cararina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Graduação em Ciência da Computação.

_____

Coordenação do Programa de Graduação

_____

Prof. Carlos Becker Westphall, Dr.
Orientador

Florianôpolis, 2024.

Este trabalho é dedicado aos meus colegas de classe e
aos meus queridos pais.

AGRADECIMENTOS

Expresso minha sincera gratidão, a todos que contribuíram para que eu pudesse concluir essa fase importante da minha vida. Em primeiro lugar, agradeço meu orientador Prof. Carlos Whestphall, que foi quem me encorajou a persistir, teve paciência e me deu as diretrizes e correções necessárias para que eu conseguisse concluir essa etapa. À universidade, professores e toda a rede de apoio institucional, que me proporcionou conhecimento e metodologias, sou verdadeiramente grato por todas as oportunidades ao longo desses anos. À minha família, que sempre acreditou em mim, sempre investiu na minha educação e nunca desistiram, em especial a minha mãe, Luciane, que abdicou de muitas coisas para tornar isso possível e ao meu avô, Sr. Luís, que infelizmente não pôde presenciar esse momento, mas sempre se orgulhou e tinha certeza que teria um neto formado em uma universais pública. Agradeço a minha namorada Laryssa, que me apoiou durante os momentos mais difíceis do projeto. Agradeço também os meus amigos, que apesar da distância, sempre se faziam presentes e proporcionaram esse suporte emocional. No mais, antes de qualquer coisa, serei eternamente grato à Deus, sem Ele nada disso seria possível. A todos, meu mais sincero muito obrigado, sem vocês, nada disso seria possível.

RESUMO

O objetivo deste trabalho é apresentar uma proposta de solução para uma assessoria de visto americano que tem encontrado problemas em diversas áreas de seu funcionamento. Além disso, este trabalho tem a intenção de abordar todas as etapas do desenvolvimento de *software*, desde a definição dos requisitos do projeto até a entrega final. Por fim, será apresentado o produto final entregue ao cliente e as possibilidades de melhorias futuras.

Palavras-chave: Visto Americano. Software. Cîencia da Computação.

ABSTRACT

The objective of this work is to present a proposed solution for an American visa consulting firm that has encountered problems in various areas of its operations. Additionally, this work aims to cover all stages of software development, from defining project requirements to the final delivery. Finally, the final product delivered to the client and the possibilities for future improvements will be presented.

Keywords: American Visa. Software. Computer Science.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O visto americano é uma permissão oficial emitida pelo governo dos Estados Unidos que permite que cidadãos estrangeiros entrem no país por um determinado período de tempo e para um propósito específico. Este documento é essencial para viajantes que desejam visitar os Estados Unidos para turismo, negócios, estudo, trabalho temporário ou outras atividades autorizadas (VISTOS. . . , s.d.).

O processo de obtenção de um visto americano pode variar de acordo com o tipo de visto solicitado e a nacionalidade do requerente, e geralmente envolve o preenchimento de formulários, a participação em entrevistas consulares e a apresentação de documentos que comprovem a elegibilidade do solicitante para a entrada no país (VISTOS. . . , s.d.).

O visto americano desempenha um papel crucial no controle de fronteiras e na segurança nacional dos Estados Unidos, ao mesmo tempo em que facilita o intercâmbio cultural, econômico e educacional entre os Estados Unidos e o resto do mundo (VISTOS. . . , s.d.).

1.1 MOTIVAÇÃO

De acordo com a Embaixada dos Estados Unidos no Brasil, o ano de 2023 registrou a emissão de mais de 1,1 milhão de Vistos Americanos no país, marcando um recorde histórico pós-pandemia (EUA. . . , s.d.). No entanto, apenas 5,8 milhões de brasileiros possuem o visto americano até o momento. Além disso, há previsão de que mais de 1 milhão de vistos serão emitidos em 2024, com potencial para alcançar o recorde histórico de 2,2 milhões de vistos emitidos durante o ano de 2019 (PROCURA. . . , s.d.).

Com isso, a demanda por empresas de auxílio no processo para obtenção de visto americano tende a crescer em 2024, conforme indicado pelo aumento no número de ligações em algumas empresas, que aumentou quatro vezes em comparação com o mesmo período do ano anterior(PROCURA. . . , s.d.).

1.2 JUSTIFICATIVA

Atualmente, o processo de solicitação do visto americano é inteiramente manual, colocando a responsabilidade em todas as etapas, desde a abertura até a conclusão, sobre o solicitante. Em empresas de assessoria, um assessor é designado para executar essas etapas para dezenas de clientes. Com o aumento da demanda por esses serviços, os assessores podem se ver sobrecarregados, o que pode levar a erros devido ao acúmulo de trabalho. Esses erros potenciais representam um risco significativo, podendo prejudicar o processo de um cliente ou até mesmo resultar na negação do

visto americano para o mesmo (ASSESSORIA. . . , s.d.).

1.3 OBJETIVOS

1.3.1 Objetivo Geral

O objetivo deste trabalho é empregar a computação para automatizar as etapas do processo de obtenção do visto americano, visando a otimização do trabalho do assessor de visto. O foco principal está nos desafios associados a todas as fases do processo de obtenção do visto americano, o gerenciamento de bases de dados de clientes e o estabelecimento de um relacionamento eficaz entre o assessor e o cliente.

1.3.2 Objetivos Específicos

Os objetivos especifícos desse trabalho são os seguintes:

- Integrar os diferentes sistemas para a realização do visto americano;
- Introduzir um modelo de banco de dados para efetuar o gerenciamento de clientes;
- Fornecer mecanismos que facilitem a comunicação entre o cliente e o assessor;
- Validar a proposta através da implementação de um sistema.

1.4 MÉTODO DE PESQUISA E TRABALHO

A seguir são apresentados os métodos de estudos de trabalho:

1. Descrever os principais desafios e problemas existentes nas etapas do visto americano e no gerenciamento de clientes.
2. Realizar uma pesquisa em como os diferentes sistemas funcionam e como é possível automatizar o processo.
3. Realizar uma revisão sistemática para encontrar soluções existentes, as quais possam ser aprimoradas ou que sirvam de base para novas soluções.
4. Buscar tecnologias e conceitos que possam auxiliar na elaboração da proposta.
5. Desenvolver e testar o sistema com base no que foi estudado.

1.5 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado em 6 capítulos. No Cap. 2 é descritos todos conceitos e etapas relacionadas ao processo para realização do visto americano. No Cap. 3

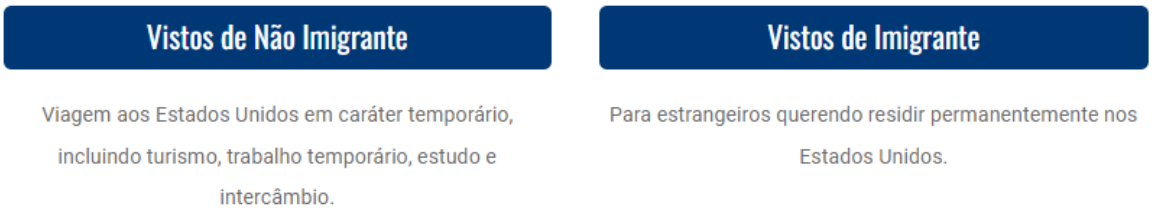
é descritos o funcionamento atual de uma empresa de assessoria em visto americano. No Cap. 4 é apresentada uma breve descrição sobre os artigos encontrados na revisão bibliográfica. No Cap. 5 é apresentada a proposta do sistema, cuja implementação é descrita no Cap. 6. Por fim, no Cap. 7 são apresentadas as conclusões e os trabalhos futuros.

2 VISTO AMERICANO

O visto consular funciona como uma pré-autorização de viagem, na qual o país de destino confirma que o viajante cumpriu as exigências de embarque. Contudo, possuir um visto não assegura a entrada no país. A decisão é tomada pelo oficial de imigração na chegada, ao verificar se o viajante satisfaz todos os requisitos, inclusive ter o visto em posse. Embora fundamental, a posse do visto não é o único critério para adentrar o país (VISTOS. . . , s.d.).

Um indivíduo estrangeiro que deseje ingressar nos Estados Unidos geralmente precisa obter previamente um visto dos Estados Unidos, o qual é inserido em seu passaporte, um documento de viagem emitido pelo país de origem do viajante. Basicamente, existem duas classes de visto americano, sendo elas: Vistos de Imigrante e os Vistos de Não Imigrante (VISTOS. . . , s.d.).

Figura 1 – Categorias de Vistos.



Fonte: `https://br.usembassy.gov/pt/visas-pt/`

2.1 TIPOS DE VISTO

2.1.1 Visto de Imigrante

O visto de imigrante é uma autorização concedida a indivíduos que desejam morar permanentemente nos Estados Unidos. Ao contrário do visto de não imigrante, que é para estadias temporárias, o visto de imigrante é destinado àqueles que desejam residir permanentemente nos EUA e receber o status de residente legal permanente, também conhecido como *"green card"*. (VISTOS. . . , s.d.).

Existem várias categorias de visto de imigrante, cada uma designada para diferentes situações, como reunificação familiar, emprego, refúgio, entre outras. Alguns exemplos de categorias de visto de imigrante são (VISTOS. . . , s.d.):

- Visto de Parente Imediato (IR): Destinado a cônjuges de cidadãos americanos, filhos solteiros menores de 21 anos de cidadãos americanos e pais de cidadãos americanos com mais de 21 anos de idade.

- Visto de Trabalhador Permanente (EB): Designado para trabalhadores estrangeiros com habilidades específicas e experiência profissional, incluindo profissionais altamente qualificados, trabalhadores com habilidades excepcionais, empresários e investidores.
- Visto de Refugiado (RS): Emitido para indivíduos que foram obrigados a deixar seus países de origem devido a perseguição, guerra ou violência, e que têm o status de refugiado reconhecido pelo governo dos EUA.
- Visto de Investidor (EB-5): Para investidores estrangeiros que desejam investir em projetos comerciais nos Estados Unidos e criar empregos para trabalhadores americanos.

O processo para obter um visto de imigrante geralmente é mais complexo e demorado do que o processo para obter um visto de não imigrante. Normalmente, envolve a apresentação de petições, cumprimento de requisitos específicos e passar por entrevistas e exames médicos. Assim, uma vez concedido o visto de imigrante e após a entrada nos Estados Unidos, o indivíduo torna-se um residente legal permanente e pode desfrutar de muitos dos mesmos direitos e responsabilidades que os cidadãos americanos, incluindo o direito de viver e trabalhar permanentemente nos EUA (VISTOS. . . , s.d.).

2.1.2 Visto de Não Imigrante

O visto de não imigrante é uma autorização concedida a indivíduos que desejam entrar temporariamente nos Estados Unidos para uma finalidade específica e por um período limitado de tempo. Ao contrário do visto de imigrante, que é para aqueles que buscam residência permanente nos EUA, o visto de não imigrante é designado para pessoas que não têm intenção de permanecer no país e que planejam retornar ao seu país de origem após a conclusão da atividade autorizada pelo visto. Existem várias categorias de vistos de não imigrante, cada uma designada para diferentes finalidades temporárias, as princiáis categorias são (VISTOS. . . , s.d.):

- Visto de Turismo ou Negócios (B-1/B-2): Destinado a pessoas que desejam visitar os Estados Unidos para fins de turismo, lazer, visita a amigos ou parentes, atividades comerciais temporárias, como conferências, negociações de contratos, consultas de negócios, treinamento curto, entre outros.
- Visto de Estudante (F-1, M-1): Para estudantes estrangeiros matriculados em instituições educacionais nos Estados Unidos. O visto F-1 é para estudantes acadêmicos, enquanto o visto M-1 é para estudantes de programas vocacionais.
- Visto de Trabalho Temporário (H-1B, H-2B): Para profissionais estrangeiros altamente qualificados que vão trabalhar temporariamente nos Estados Uni-

dos em ocupações especializadas que exigem conhecimento especializado. O visto H-2B é para trabalhadores temporários não agrícolas.

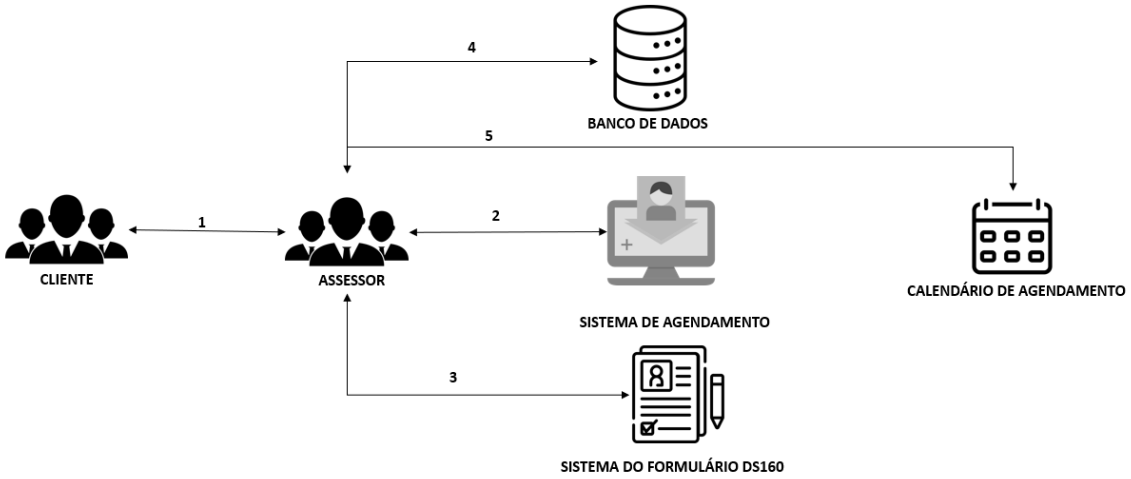
- Visto de Intercâmbio (J-1): Destinado a participantes de programas de intercâmbio cultural, educacional ou profissional, incluindo estudantes de intercâmbio, professores, pesquisadores, médicos residentes, entre outros.

O visto de não imigrante é emitido mediante solicitação prévia e aprovação do Serviço de Cidadania e Imigração dos Estados Unidos (USCIS) ou do Departamento de Estado dos Estados Unidos, dependendo da categoria de visto e do país de residência do requerente (VISTOS. . . , s.d.).

2.2 FUNCIONAMENTO ATUAL DE UM ASSESSORIA EM VISTO AMERICANO

Nesta sessão são descritas as etapas realizadas por uma empresa de acessoria de pequeno porte, especializada em vistos americanos, para conceder os vistos a seus clientes. Um único acessor está envolvido durante todas as etapas envolvidas no atendimento a um determinado cliente, conforme ilustrado pelo fluxograma da Fig. 2. Através deste fluxograma é possível observar a ordem da realização destas etapas, as quais são descritas com mais detalhes no restante deste capítulo.

Figura 2 – Fluxograma simplificado de uma empresa de assessoria em visto.



Fonte: Próprio

2.2.1 Etapa 1: Atendimento ao Cliente

A primeira etapa a ser compreendida é o atendimento ao cliente, como descrito na interação número um do fluxograma da Figura 2. Nessa fase, o assessor é encarregado de estabelecer e manter contato com o cliente ao longo de todo o processo de obtenção do visto. Isso implica em executar uma série de tarefas, tais como:

- Fornecer uma explicação detalhada das etapas envolvidas no processo de obtenção do visto americano ao cliente.
- Acompanhar o cliente durante a negociação dos honorários pelos serviços prestados, discutindo os valores envolvidos e detalhando todos os custos associados ao processo.
- Solicitar todas as informações essenciais ao cliente para preencher adequadamente os formulários exigidos.
- Oferecer orientações detalhadas e instruções claras sobre os procedimentos em cada uma das etapas do processo. Além disso, é fundamental resolver quaisquer dúvidas ou preocupações que o cliente possa ter ao longo do caminho.

2.2.2 Etapa 2: Cadastro do Cliente no Sistema de Agendamento

Nesta fase do processo, o assessor, utilizando as informações fornecidas pelo cliente, dá início ao procedimento junto ao Sistema de Agendamento, que é responsável pelo processamento da Taxa Consular. É crucial que, nesse momento, o assessor já tenha determinado o tipo de visto consular com base no motivo de viagem informado pelo cliente, uma vez que o valor da taxa pode variar de acordo com a categoria do visto. Após o cadastro do cliente no sistema, o assessor deve proceder com o pagamento da taxa consular de acordo com a escolha do cliente entre as opções disponíveis na localidade. No Brasil, por exemplo, as opções comuns são Cartão de Crédito e Boleto Bancário.

2.2.3 Etapa 3: Preenchimento de Formulário

Esta etapa é mais importante, pois é aqui que o assessor preenche o formulário DS-160 em nome do cliente com base nas informações fornecidas. O formulário DS-160 é importante, pois sintetiza todas as informações pertinentes para a obtenção do visto do cliente, sendo posteriarmente minuciosamente analisado pelo agente consular durante o processo de solicitação do visto americano.

2.2.4 Etapa 4: Atualizações do Banco de Dados

Como responsável pelo processamento de diversos pedidos de visto americano, é crucial que o assessor mantenha sua base de dados organizada e atualizada. Esta base de dados deve conter informações cruciais sobre os processos e clientes, como detalhes de login e senha, informações de contato dos clientes, agendamentos e locais dos processos de visto, bem como registros precisos de valores pagos e pendentes, entre outros dados relevantes. A falta de atualização dessas informações pode acarretar diversos problemas no processo, incluindo:

- Perda de contato com o cliente devido à falta de informações de contato atualizadas, o que pode resultar em dificuldades para fornecer atualizações importantes sobre o processo do visto e perder oportunidades de comunicação vital.
- Perda do agendamento consular do cliente devido a informações desatualizadas sobre datas e horários, o que pode resultar em atrasos significativos ou até mesmo na necessidade de reagendar todo o processo, gerando inconveniências e custos adicionais para o cliente.
- Pagamento duplicado de taxas consulares devido à falta de inclusão do cliente no banco de dados, o que pode levar a problemas financeiros para o cliente, além de resultar em atrasos desnecessários e complicações administrativas.
- Não recebimento de valores devidos, causado pela falta de controle financeiro na planilha sobre tais informações, o que pode resultar em problemas de fluxo de caixa para a empresa de assessoria e, potencialmente, em desentendimentos com os clientes devido a cobranças incorretas ou ausência de pagamentos.

2.2.5 Etapa 5: Calendário de Agendamento

Por fim, cabe ao assessor planejar o agendamento de acordo com as datas e localidades disponíveis no sistema, levando em consideração as necessidades individuais de cada cliente. Em muitas situações, os clientes têm urgência no processo, e é responsabilidade do assessor agir com rapidez máxima para atender a essas necessidades. Isso envolve priorizar agendamentos que se encaixem nas demandas específicas do cliente e garantir que o processo seja conduzido o mais eficientemente possível.

2.2.6 Identificação do Problema

Após analíse de todas as etapas de realização do visto americano, é preciso identificar os principais problemas enfrentados pelo estabelecimento, os quais poderiam ser resolvidos por meio do sistema. Com isso, identificamos os seguintes problemas:

- **Falta de Gerenciamento de agendamentos**: Atualmente, os agendamentos são controlados por meio de uma planilha, na qual todos os dados são mantidos e atualizados manualmente. Como resultado, a planilha está sujeita a erros, que podem levar à perda de agendamentos, já que uma modificação pode ser feita e não atualizada na planilha de controle, resultando em informações .
- **Acúmulo de funções dos funcionários**: No funcionamento atual, o assessor é responsável por todas as etapas do processo. Isso significa que o assessor deve negociar com o cliente, abrir o processo junto às entidades envolvidas, preencher os formulários necessários, manter a comunicação com o cliente, entre outras tarefas. No entanto, o acúmulo de funções pode prejudicar o desempenho nas tarefas mais importantes.
- **Contato com o cliente constante**: Devido à grande quantidade de clientes, frequentemente a comunicação entre os assessores e os clientes não é ideal, o que pode resultar em problemas, como a falta de aviso aos clientes sobre seus agendamentos com antecedência, falta de informações sobre a documentação necessária ou a localização de seus compromissos.
- **Controle financeiro** No momento, não há controle financeiro no estabelecimento. Como resultado, os assessores são responsáveis pelo pagamento de seus clientes, o que significa que não há disponibilidade para efetuar todas as cobranças necessárias, caso seja preciso.

3 ESTADO DA ARTE

Neste capítulo é apresentada a revisão bibliográfica. Palavras-chave foram definidas para a realização de uma busca na base de dados IEEE Xplorer e na ferramenta de buscas Google Scholar, cuja soma do número de trabalhos encontrados é apresentado na Tab. 1. Ao todo foram identificados 3 trabalhos com objetivos relacionados a proposta do presente trabalho. Um resumo, sobre cada um dos 3 trabalhos é apresentado no restante deste capítulo.

Tabela 1 – Resultado da revisão sistemática sobre as palavras chaves

Palava chave	Resultado	Categoria
Sistemas de Informação	2.640.00	1
Sistemas de Gerenciamento	837.000	1
Aplicações Web	160.000	1
Banco de Dados	97.300	1
Sistemas de Informação Sistemas de Gerenciamento	6.550	2
Sistemas de Informação Aplicações Web	6.140	2
Sistemas de Informação Banco de Dados	59.600	2
Sistemas de Gerenciamento Aplicações Web	1.120	2
Sistemas de Gerenciamento Banco de Dados	11.600	2
Sistemas de Informação Sistemas de Gerenciamento Aplicações Web	570	3
Sistemas de Informação Sistemas de Gerenciamento Aplicações Web Banco de Dados	537	4

Fonte: Próprio(2024).

3.1 PILA FÁCIL: SISTEMA DE GERENCIAMENTO DE PEDIDOS

O Pila Fácil revoluciona o gerenciamento de clientes em estabelecimentos comerciais do ramo alimentício, bares e casas de festa. Com ele, os clientes têm acesso a um sistema completo que facilita sua experiência. Através do aplicativo, é possível consultar o cardápio, acompanhar sua comanda de gastos em tempo real e efetuar o pagamento de forma simples e conveniente, tudo diretamente pelo celular (ZIMMERMANN, 2010).

O sistema é acessado pela Internet tanto pelo cliente, que realiza seus pedidos e o acompanhamento dos gastos através da aplicação, como pelo estabelecimento, que realiza o gerenciamento dos clientes pelo estabelecimento. O Pila Facil é encarregado de tarefas como o processo de pagamento, envio de pedidos a cozinha e cadastro do cliente. Dessa forma, facilita para o cliente, que precisará se cadastrar uma única vez, e para o próprio estabelecimento, que não terá a preocupação de manter uma lista dos clientes cadastrados e com sua manutenção, uma vez que os dados não serão mantidos localmente.

O modelo do Pila Fácil apresentado opera a partir de duas perspectivas distintas: a do cliente e a do restaurante. Do ponto de vista do cliente, ao chegar em um restaurante, é necessário inicialmente se identificar por meio de uma chave exclusiva do cliente para acessar as ferramentas disponíveis no sistema, como o cardápio, registro de pedidos e a visualização da conta. Já do ponto de vista do restaurante, o sistema proporciona uma visão geral de todos os clientes, incluindo o registro de pedidos, status das contas e informações sobre produtos e estoque disponíveis.

Durante a modelagem do sistema Pila Fácil, o autor destacou alguns desafios que devem ser enfrentados ao desenvolver um sistema de gerenciamento, dos quais podemos citar (ZIMMERMANN, 2010):

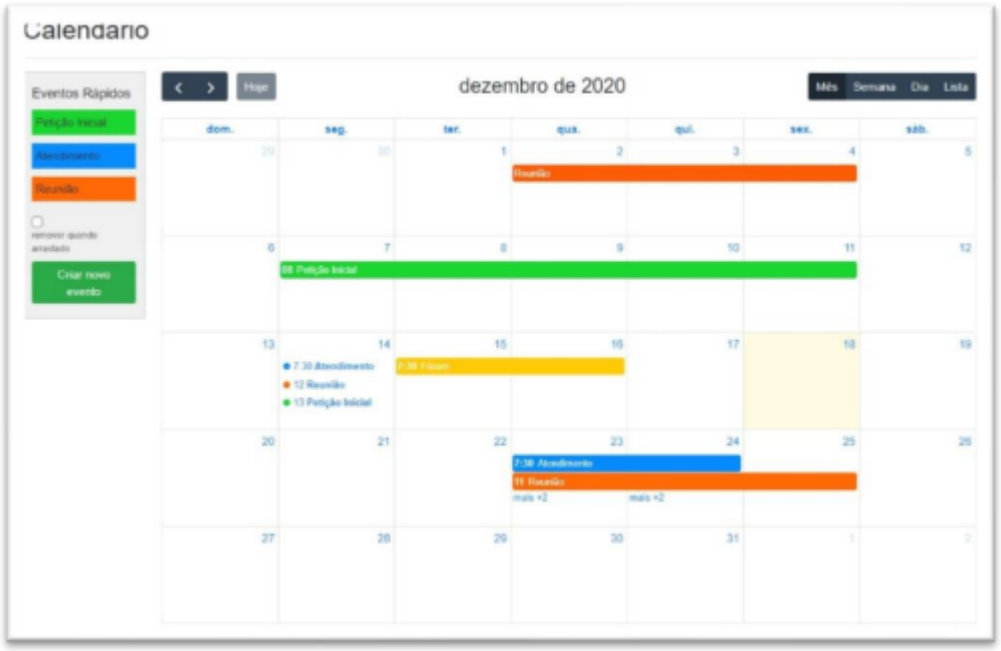
- Segurança: É crucial garantir a segurança dos dados do cliente, uma vez que informações sensíveis como dados pessoais e de pagamento ficam registradas, além de dados financeiros dos restaurantes, entre outros. Além disso, é essencial prevenir possíveis fraudes, como a tentativa de um usuário se passar por outro cliente.
- Alta Disponibilidade do Sistema: O sistema deve estar disponível para todos os clientes a todo momento. Portanto, não deve ser limitado apenas àqueles conectados à rede local, o que restringiria o acesso. Além disso, é essencial prevenir possíveis problemas, como a falta de conexão com a internet, falhas de energia, entre outros imprevistos.
- Localização: Em casos específicos, como em bares ou baladas, é comum que o cliente se desloque pelo espaço. Portanto, um dos desafios do sistema seria como localizar ou manter o registro do cliente para a entrega de um pedido, especialmente quando o cliente não se mantém em uma localidade fixa.

3.2 SISTEMA DE INFORMAÇÃO PARA CONTROLE DE AGENDAMENTO DE TAREFAS PARA ESCRITÓRIOS DE ADVOCACIA

O artigo explora o funcionamento de um software, detalhando suas funcionalidades com foco no aprimoramento do controle de agendamentos de prazos processuais em escritórios de advocacia, além de discutir sua disponibilidade para os usuários(CASTRO; KURTZ, s.d.).

O sistema inclui uma agenda que exibe datas agendadas para clientes ou processos, junto com seus respectivos prazos; recursos para controle financeiro da empresa, monitorando gastos e ganhos; um módulo de gestão de clientes, fornecendo detalhes cadastrais dos clientes; e a capacidade para o usuário armazenar uma variedade de arquivos, como peças processuais, documentos escaneados, entre outros.

Figura 3 – Modelo de Calendário Proposto



Fonte: (CASTILHO; KAMIENSKI, 2018)

A Figura 5 apresenta a agenda proposta pelo artigo, na qual é possível identificar a fase em que o processo se encontra por meio da cor, e a duração dessa etapa do processo é visualizada através da etiqueta que percorre os dias. Para criação desenvolvimento dessa interface foi utilizado o framework *Laravel*. Por fim, durante o desenvolvimento, foram encontrados algumas sugestões para trabalhos futuros, como:

- Propor uma visão para o cliente, onde ele posso ter um ambiente que possa acompanhar o andamento dos seu processos.
- Adicionar mais funcionalidades ao calendário, para dispobilizar mais opções de agendamento e especificidades a cada agendamento.

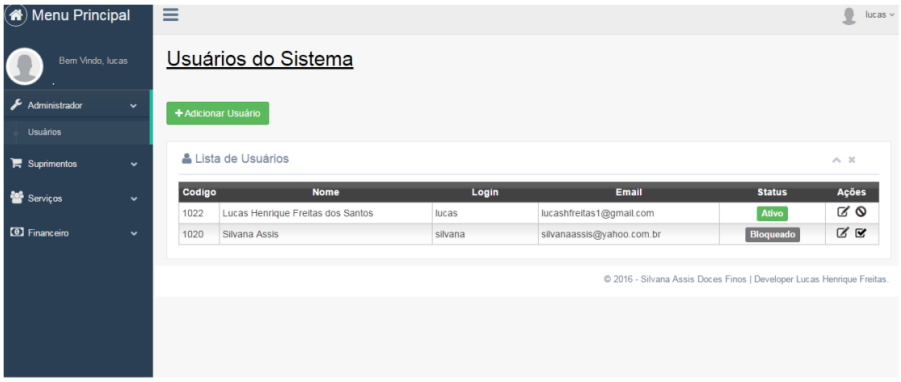
3.3 DESENVOLVIMENTO DE UM SISTEMA DE INFORMAÇÃO WEB PARA GERENCIAR OS PROCESSOS DE UM BUFFET DE DOCES E BOMBONS

O artigo propõe o desenvolvimento e implementação de um sistema de informação web para auxiliar um buffet de doces e bombons a gerenciar seu negócio(Fig. 4), com o objetivo de aumentar a eficiência dos processos e resolver os principais desafios enfrentados pela empresa (SANTOS, 2016).

Além de propor o desenvolvimento do programa, o intuito do artigo é descrever cada uma das etapas essenciais do desenvolvimento de software, conforme definido por (SOMMERVILLE, 2011), utilizando técnicas que visam facilitar o processo de de-

senvolvimento. Dessa forma, o projeto tem início com a identificação do problema e dos requisitos funcionais e não funcionais, passando por etapas como a modelagem dos modelos e funcionalidades, identificação dos casos de uso, entre outras.

Figura 4 – Sistema para gerenciamento de Estabelecimento de Doces e Bolos



Fonte: (SANTOS, 2016)

Durante o desenvolvimento do projeto, percebeu-se a necessidade, para projetos futuros,de disponibilizar *software* para acesso remoto, de modo que a propritário possa gerir seus estabelicimento de maneira remota. Além disso, percebeu-se que havia a necessiade de um controle de acessos e autorizações, ou seja, os funcionários não deveriam ter acesso a todos os dados do sistema, a depender de sua posição.

3.4 PROPOSTA X TRABALHOS CORRELATOS

Durante este capítulo, foram analisados três trabalhos correlatos, com o objetivo de nos subsidiar no desenvolvimento da aplicação proposta neste artigo. Em (ZIMMERMANN, 2010), é proposto um sistema de gerenciamento de pedidos para restaurantes. A ideia de citar este artigo é ter como base como deve ser feita a comunicação do sistema junto às interfaces do cliente e da administração do restaurante. Além disso, o artigo identifica algumas das principais dificuldades durante o desenvolvimento de sistemas de gerenciamento.

Em (CASTRO; KURTZ, s.d.), é proposto um sistema de informação para o gerenciamento de tarefas em escritórios de advocacia, com o objetivo de aprimorar o controle da agenda e das etapas processuais. Portanto, ao citar esse artigo, pretende-se utilizá-lo para auxiliar no desenvolvimento do calendário de agendamento dos solicitantes de visto americano, uma vez que um dos aspectos cruciais do sistema proposto é manter um controle organizado dos agendamentos dos solicitantes.

Em (SANTOS, 2016), é proposto sistema de informação web para auxiliar um buffet de doces e bombons para genciar o negócios, mas muito além do sistema

propsoto, esse artigo pode nos auxiliar nas etapas do desenvolvimento de um *software* descritas por (SOMMERVILLE, 2011) como essencias para um desenvolvimento eficaz.

Por fim, é necessário ressaltar que não existem ainda propostas similares a descrita nesse trabalho, por esse motivos, foram escolhidos três trabalhos que têm como proposta a criação de um sistema de gerenciamento para fins específicos, com objetivo de comparar sua ideais e utilizá-las em nosso projeto.

4 DESENVOLVIMENTO DO PROJETO

Este capítulo apresenta todas as etapas de desenvolvimento do sistema e os artefatos gerados em cada fase.

4.1 ANALISE E LEVANTAMENTO DE REQUISITOS

De acordo com (BARBOSA; SILVA, 2010), as principais técnicas utilizadas para coletar dados e levantar requisitos dos usuários são:

- entrevistas;
- grupos de foco;
- questionários;
- brainstorming de necessidades e desejos dos usuários;
- classificação de cartões (card sorting);
- estudos de campo;
- investigação contextual.

Nesse contexo (BARBOSA; SILVA, 2010) diz que cada técnica é caracterizada quanto ao seu objetivo, suas vantagens e o nível de esforço necessário para sua aplicação. Para nosso projeto utilizamos da técnicas de estudo de campo e a investigação contextual para fazer o levantamento dos requisitos do cliente.

No estudo de campo temos como objetivo entender o comportamento do usuário final no contexto de seu própio ambiente de atuação (COURAGE; BAXTER, 2005), enquanto a investigação contextual busca obter dados sobre a estrutura do trabalho na prática e conhecer os detalhes do trabalho que se tornam habituais e invisíveis.

Durante um período de seis meses fizemos parte da equipe do escritório de assessoria para entender sobre o processo de trabalho e a partir da analíse do trabalho definimos os requisitos do sistema, que foram divididos em dois tipos: funcionais e não funcionais.

4.1.1 Requisitos Funcionais

De acordo com (SOMMERVILLE, 2011), os requisitos funcionais explicitam os serviços que o sistema deve fornecer, como o sistema deve reagir a entradas e como deve se comportar em determinadas situações. A seguir, na Tabela 2, mostramos as funcionalidades que o software deve solucionar com base nos problemas citados na Seção 3.1, onde foram descritas cinco requisitos funcionais, sendo a coluna "*Nome do Requisito*"descritos o nome do requisito, na coluna "*Descrição*"está descrito a funcionalidade desse requisito, e por fim, na coluna "*Prioridade*"é apresentado a prioridado do requisito para o sistema.

NOME DO REQUISITO	DESCRIÇÃO	PRIORIDADE
RF001 - Gerenciar a entrada de novos processos	O sistema deve iniciar novos processos com base nas informações fornececidas.	OBRIGATÓRIO
RF002 - Gerenciar os agendamentos dos solicitantes	O sistema deve manter atualizados os dados sobre os agendamentos e realizar a antecipação dos agendamento conforme a necessidade.	OBRIGATÓRIO
RF003 - Gerenciar a comunição com o solitantes	O sistema deve fornecer os dados cruciais aos clientes como data e local do agendamento.	OBRIGATÓRIO
RF004 - Visualizar a planilha de clientes	O sistema deve fornecer a visualização dos dados dos clientes, como as datas de seus agendamentos.	OBRIGATÓRIO
RF005 - Gerenciamento Financeiro	O sistema deve fornecer o valor cobrado ao cliente e o saldo pago pelo mesmo	OBRIGATÓRIO

Tabela 2 – Requisitos Funcionais

4.1.2 Requisitos Não Funcionais

Em (SOMMERVILLE, 2011) é dito que os requisitos não funcionais são aqueles que não são diretamente relacionados ás funções específicas forncecidas pelo sistema, sendo aplicados frequentemente ao sistema como um todo. A seguir, na Tabela 3, é apresentado os requisitos não funcionais do sistema, onde na coluna "*Código*"é descrito o código do requisito não funcional, na coluna "*Descrição*"está descrito a restrição referente ao requisito e, por fim, na coluna "*Categoria*"é descrita a categoria referente a essa descrição.

CÓDIGO	DESCRIÇÃO	CATEGORIA
RNF001	O sistema deve ser acessado via navegador.	PORTABILIDADE
RNF002	O sistema deve ser desenolvido nas liguagens de programação Javascript, HTML e PHP.	ARQUITETURA
RNF003	O sistema deve estar disponivel para apenas para acesso local.	DISPONIBILIDADE
RNF004	O sistema deve ser adaptado para usar apenas em computadores.	PORTABILIDADE
RNF005	O sistema deve ter boa fluidez, de modo a não prejudicar a experiência do usuário.	DESEMPENHO
RNF006	O sistema deve ser adaptado a tratar possíveis falhas ao acesso de informações, mantendo copias atualizadas dos dados.	CONFIABILIDADE

Tabela 3 – Requisitos Não Funcionais

4.1.3 Validação de Requisitos

Após a realização do levantamento de requisitos, foi definido um documento com objetivo de especificar os requisitos e as funcionalidades do sistema para a proprietária do estabelecimento. O documento, anexado na apêndice A desse trabalho, específica todos os requisitos e condições para desenvolvimento do sistema, que já foi aprovado pela proprieária.

4.2 MÓDULOS E FUNCIONALIDADES

Após a definição de todos os requisitos funcionais e não funcionais serem aprovadas, começou a ser definida a estrutura do sistema com os módulos e funcionalidades.

4.2.1 Módulo Administrador

O módulo administrador conterá os serviços que permitiram gerenciar os assessores que irão acessar as demais funionalidades do sistema. O modulo possuirá a seguinte funcionalidade.

- **FA01 - Gerenciar Assessores** - A funcionalidade deve permitir a criação dos usuários que vão ser autorizados acessar as funcionalidades do sistema.

4.2.2 Módulo Calendário

O módulo cliente é responsável pelos serviços que permiram a empresa gerenciar todos os dados e os agendamentos dos solicitantes do visto americano. O módulo possuirá as seguintes funcionalidades.

- **FA02 - Gerenciar Agendamentos** - A funcionalidade deve permitir atualizar as informações referente as datas de agendamento dos clientes, além ser responsavél por realizar a antecipação dos clientes, conforme as datas disponíveis e as informações fornecidas.

4.2.3 Módulo Assessor

O módulo assessor é responsável pelo cadastro de solicitantes na plaforma, além de ser responsável pelas informações financeiras de cada solicitante. O módulo possuirá as seguintes funcionalidades:

- **FA03 - Gerenciar Solicitantes** - A funcionalidade deve permitir criar novos clientes, alterar dados dos clientes ou excluir um cliente.
- **FA04 - Gerenciar Pagamentos** - A funcionalidade deve permitir que o usuário insira as informações sobre os valores pagos, os valores cobrados e os valores devidos pelos solicitantes.
- **FA05 - Gerenciar Comunicação** - A funcionalidade deve permitir a criação de um canal de comunicação entre o solicitante e o usuário para provir informações sobre seus agendamentos.

4.2.4 Módulo Formulário

O módulo formulário é responsável pelos preenchimentos dos formulários oficiais. O módulo deve conter as seguintes funcionalidades:

- **FA06 - Gerenciar Abertura de Processo** - A funcionalidade deve permitir a criação de novos processos de visto americano junto ao consulado.

4.3 CASOS DE USO

Conforme dito em (PRESSMAN; MAXIM, 2016), a partir da reunião dos requisitos, uma visão geral das funções e características começam a se materializar. Entretando, é difícil passar para atividades mais técnicas, da engenharia de software até que se entenda como tais características serão usadas por diferentes classes de usuários. Com isso, os desenvolvedores e usuários podem criar um conjunto de cenários que identifique um roteiro de uso para sistema, esses cenários também são conhecidos como *casos de uso*. A seguir, na Figura 5, será apresentando o diagrama de casos de uso do sistema.

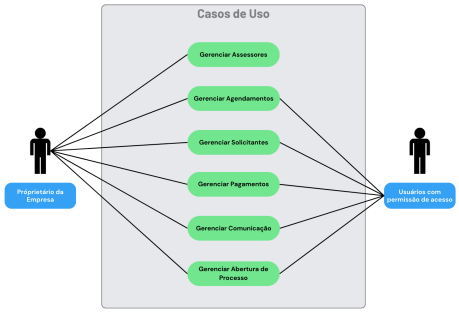


Figura 5 – Casos de Uso

Em seguida foi criado um documento onde são descritos de maneira detalhada os casos de uso para cada funcionalidade descritos na Seção 6.2, conforme os códigos descritos na Tabela 4. Nesse documento são descritos todo o contexto e o fluxo de interações e restriçoes de cada funcionalidade, que pode ser encontrados na Sessão Apêndice B deste trabalho.

CÓDIGO	FUNCIONALIDADE
UC001	Gerenciar Assessores
UC002	Gerenciar Agendamentos
UC003	Gerenciar Solicitantes
UC004	Gerenciar Pagamentos
UC005	Gerenciar Comunicação
UC006	Gerenciar Abertura de Processos

Tabela 4 – Casos de Uso Detalhados

4.4 PROTOTIPAÇÃO

Após realizarmos a elaboração dos casos de uso detalhados de todas as funcionalidades, iniciamos a fase da prototipação. Como dito em (PRESSMAN; MAXIM, 2016), a prototipação é um modelo de processo isolado que tem como objetivo auxiliar os envolvidos a compreender melhor o está para ser contruído.A prototipação se concetra em ser uma representação dos aspectos do *software* que serão visivéis ao usuário, que leva a contrução de um protótipo que é entre e avaliado pelos envolvidos.

Desse modo, no Apêndice C desse trabalho é apresentando protótipos de interface criados e aprovados pelos envolvidos. Para criação foi utilizado a ferramenta de prototipapação Pencil que é uma ferramenta gratuita que permite projetar interface de diversos tipos de sistemas.

4.5 BANCO DE DADOS

Após definidos todos os requisitos e os protótipos da interface, foi realizado a construção do banco de dados das funcionalidade dos sistemas. O fato de termos um

compreensão adequada do projeto, facilitou no desenvolvimentos do banco de dados do sistema. Para desenvolver o banco de dados utilizamos a ferramenta *SQL Server Management Studio*, que possui um recurso que permite que os diagramas lógicos sejam criados de maneira visual para posteriormente serem convertidos em tabelas no servidor de banco de dados.

A seguir, na Figura 6, é mostrado o diagrama lógico do banco de dados do sistema, contendo um total de cinco tabelas necessárias para construir o sistema. A tabela *Tabela Assessores* refere-se aos assessores cadastrados no sistema. Nela, é importante conter o nome do assessor, o login para acessar o sistema, a senha de acesso, o email para recuperação da senha e um campo que indica se ele é ou não o administrador do sistema. A *Tabela Clientes* é a tabela principal do banco de dados, pois contém o registro de todos os clientes da empresa. Além disso, essa tabela se relaciona com a *Tabela Pagamentos*, que é responsável por manter os registros dos pagamentos, a *Tabela Contato*, que mantém os registros dos contatos dos clientes, e a *Tabela Cidade*, que mantém os locais onde estão abrigados os consulados americanos, com o intuito de minimizar a repetição de dados. Por fim, a *Tabela Clientes* se relaciona também com a própria *Tabela Assessores*, visto que cada cliente deve possuir o assessor responsável.

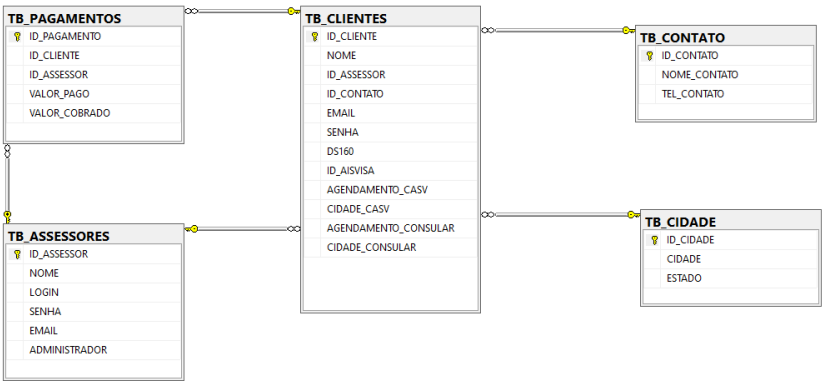


Figura 6 – Diagrama Lógico do Banco de Dados

4.6 CODIFICAÇÃO

Após a construção inicial do banco de dados do sistema, iniciamos a fase de implementação, utilizando como base os casos de uso detalhados nas etapas anteriores para definir as restrições que o sistema deveria seguir e os fluxos de interação com os usuários. Além disso, os protótipos de interface serviram como modelos para o desenvolvimento das interfaces da aplicação, visto que todos os elementos de cada tela já haviam sido projetados previamente.

Outro fator importante, por se tratar de um projeto incremental, é que conforme avanços no desenvolvimento foram realizados nesta fase de codificação, também foram

definidas algumas alterações no projeto inicial previsto junto ao cliente. As seguintes mudanças foram realizadas em comparação ao projeto inicial:

- Referente ao Caso de Uso UC02 - Gerenciar os Agendamentos, foi definido junto ao cliente que não haveria a necessidade de realizar a antecipação e o cancelamento dos agendamentos, devido ao fato de que o sistema de agendamentos possui um mecanismo de segurança contra aplicações web, o que poderia acarretar em bloqueio do processo e, consequentemente, perda financeira.
- Referente ao Caso de Uso UC06 - Gerenciar Aberturas de Processos, foi definido que a abertura de processos deve ser feita pelo usuário, pelos mesmos motivos mencionados no primeiro item desta seção, visto que pode ocasionar prejuízos financeiros. Assim, o sistema manterá os dados essenciais do usuário para a realização do processo.

4.7 TESTES E VALIDAÇÃO

De acordo com (SOMMERVILLE, 2011), uma das principais vantagens de usar o desenvolvimento incremental é permitir que as funcionalidades sejam testadas à medida que ficam prontas, descartando a necessidade de esperar o sistema ficar completamente pronto. Além disso, isso permite que o sistema seja testado diversas vezes, visto que o desenvolvimento de uma funcionalidade pode acarretar mudanças no desenvolvimento inicial de outra funcionalidade.

Os testes foram realizados antes de promover a entrega de cada funcionalidade ao cliente para a realização da etapa da validação, que tem como objetivo analisar se o sistema está suprindo as necessidades do cliente. Como dito (SOMMERVILLE, 2011), a etapa da validade é definida como um série de atividades que são executas para verificar se o sistema atende a todos os requisitos do cliente.

4.7.1 Testes Exploratórios

Para a realização dos testes do sistema, foram realizados testes exploratórios, que são testes manuais com o objetivo de seguir o fluxo de interações de cada funcionalidade prevista nas etapas anteriores. Durante esses testes, os cenários de uso foram repetidos diversas vezes para verificar se o sistema se comportava conforme o esperado.

Os testes exploratórios são uma abordagem essencial, pois permitem a identificação de problemas e inconsistências que podem não ser detectados. Ao seguir os fluxos de interação previstos, os teste puderam explorar o sistema de maneira intuitiva e identificar comportamentos inesperados, falhas de usabilidade e possíveis bugs.

4.7.2 Validação

A validação foi realizada por meio de reuniões. Após a conclusão de cada funcionalidade do projeto, era organizada uma reunião com o cliente para validar a nova funcionalidade, verificando se atendia às expectativas. Desse modo, a cada reunião poderiam ser definidas melhorias a serem implementadas no próximo escopo de desenvolvimento. Se a funcionalidade fosse aprovada por completo, ela era considerada finalizada. Após a entrega da última funcionalidade, era feito um encontro final para apresentar o sistema geral para validação e, se necessário, definir as últimas mudanças a serem realizadas antes da entrega final.

4.8 IMPLEMENTAÇÃO

Após o desenvolvimento e a validação, o sistema foi implementando para utilização na empresa, porém foi decidido que o sistema vai passar por um período de experiência, onde o sistema será utilizado como sistema principal, porém o sistema anterior continurá a ser usado, que é baseado em planilhas, como um mecanismo de segurança para manter as informação caso seja verificado algum erro durante a utilização do novo sistema.

5 SISTEMA DE GERENCIAMENTO DO VISTO AMERICANO

Nesse capítulo são apresantados as interfaces do sistema juntamente com as principais funcionalidades. Cada sessão será dividada conforme suas funcionalidades.

5.1 LOGIN E RECUPERAÇÃO DE SENHA

Quando o usuário acessa o sistema, ele se depara com a *Tela de Login* (Fig.7), nesse caso ele deve colocar suas credencias no sistema para acessar o sistema.

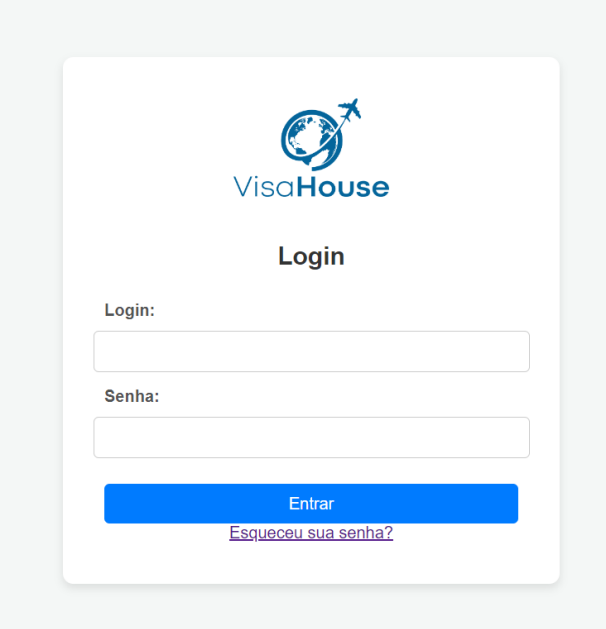


Figura 7 – Tela de Login

Caso o usuário tenha esquicido das informações de login ou senha, ele tem a opção de realizar a recuperação da sua senha, nesse caso ele irá acessar *Tela de Recuperação de Senha* (Fig. 8), onde ele deve entrar com o email cadastrado para recuperar as informações.

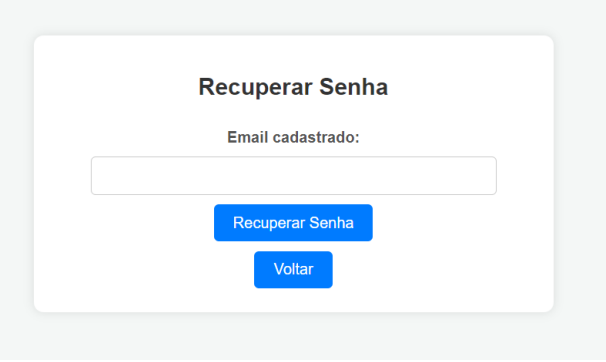


Figura 8 – Tela de Recuperação de Senha

Se obtiver êxito na verificação, ele será levado a *Tela de REcuperação de Dados do Assessor* (Fig. 9), onde poderá observar todos os dados do seu cadastro.

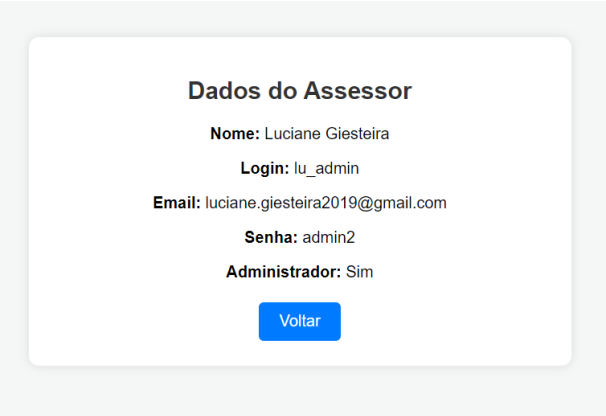


Figura 9 – Recuperação de Dados do Assessor

5.2 MENU PRINCIPAL E GERENCIAMENTO DE ASSESSORES

Ao realizar o login, o usuário será levado para o menu principal (Fig. 10), na *Tela Menu Principa*l são apresentados todas as funcionalidades. Inclusive, duas funcionalidade que são presente apenas aos administradores, que são de criação de assessores (Fig. 11) e de exclusão de assessores (Fig.12). Uma confirguração importante dessas funcionalidades se deve ao fato que o administrador pode criar novos administradores, porém os administradores não podem ser excluídos do sistema.

Além disso, no *Menu Principal*, (Fig. 10), é apresentado a tabela de clientes, com todas as informações importantes sobre cada um dos clientes, onde, através da opção *Ordenar*, é possível realizar três tipos de ordenação: por nome, por agendamento na CASV e por agendamento no consulado. Por fim, através opção de Atualizar, é possível realizar a atualização automática das informações dos agendamentos dos clientes, onde o sistema acessa individualmente cada cadastro e verifica as informações sobre as datas da agendamentos e as localidades.

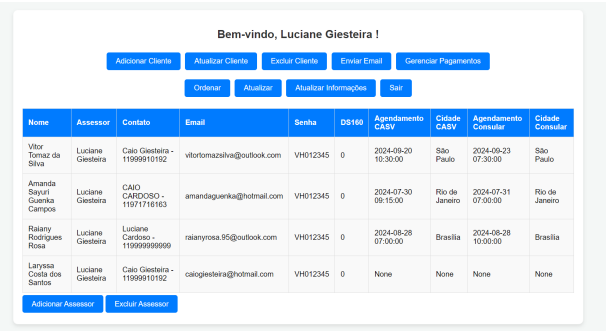


Figura 10 – Menu Principal

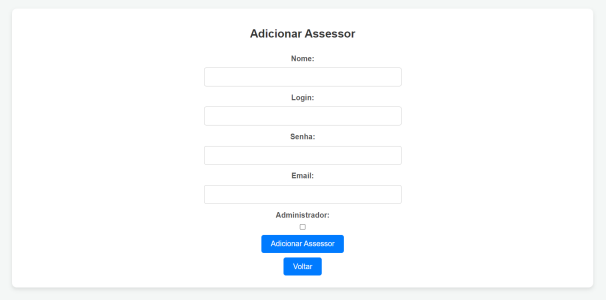


Figura 11 – Adicionar Assessor

Figura 12 – Excluir Assessor

5.3 GERENCIAR CLIENTES

Nesse modúlo são apresentados todas as manipulações que podem ser realizados sobre os clientes. Na Fig. 13, está sendo mostrado a Tela de Criação de Novos Clientes, onde devemos entrar com os dados dos clientes. Nas Fig. 14 e Fig. 15, respectivamente, são apresentados o menu *drop-down* dos clientes, onde devemos escolher qual cliente desejamos modificar e a página de edição dos dados do clientes, que devem ser salvos quando finalizadas as alterações. Por fim, na Fig. 16, é apresentado o menu para excluir um cliente, para isso, basta o usuario selecionar o cliente e selecionar o botão excluir.

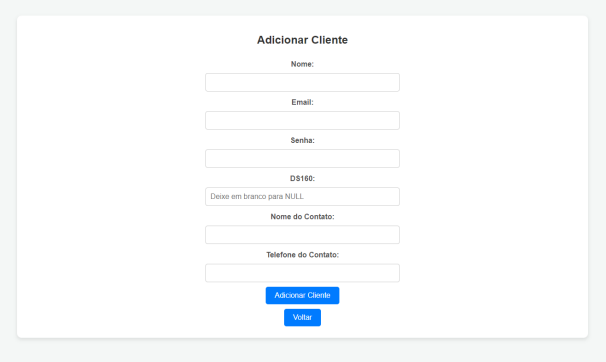


Figura 13 – Criação de Clientes

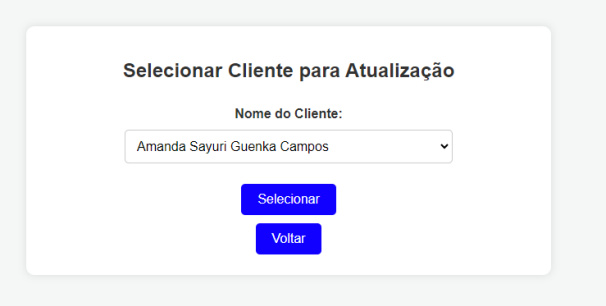


Figura 14 – Selecionar Cliente para Edição

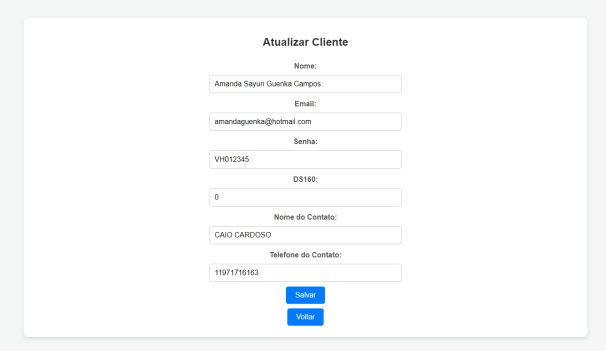


Figura 15 – Editar Cliente

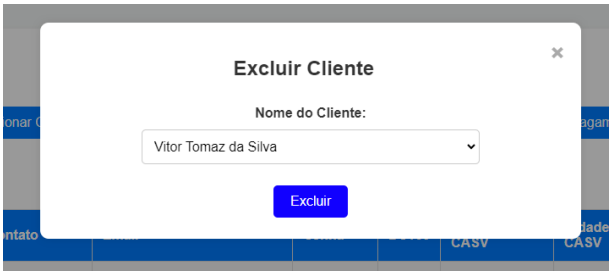


Figura 16 – Excluir Cliente

5.4 GERENCIAMENTO DE COMUNICAÇÃO

Nessa sessão vamos apresentar o mecanismo do comunicação proposto, que seria através funcionalidade *Enviar E-mail*, Fig. 17, onde o usuário pode enviar email ao cliente com base no email cadastrado, ou podendo alterar o email para qual deseja enviar. O usuário deve incluir o título da mensagem, o corpo da mensagem e também se necessário, anexar até no máximo dois documentos.

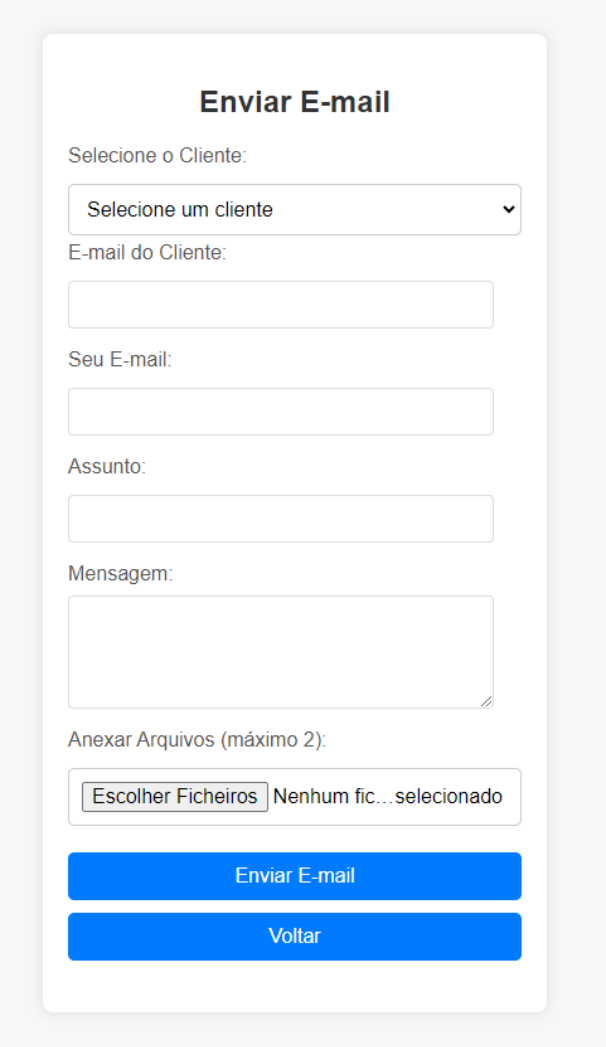


Figura 17 – Enviar Email

5.5 GERENCIMENTO DE PAGAMENTOS

Nessa sessão vamos apresentar os mecanismos de gerenciamento de pagamentos, onde apresentaremos o menu para criação de pagamentos, Fig. 18, onde o usuário deve selecionar o cliente e cadastrar o pagamento, Fig. 19 , caso esse cliente já possua um pagamento, o sistema emite um alerta informando que esse cliente já tem um pagamento cadastrado, nesse caso, o usuário pode editar o pagamento desse cliente através do menu de edição de pagamento, Fig. 20, onde serão apresentados os dados do último registro de pagamento.

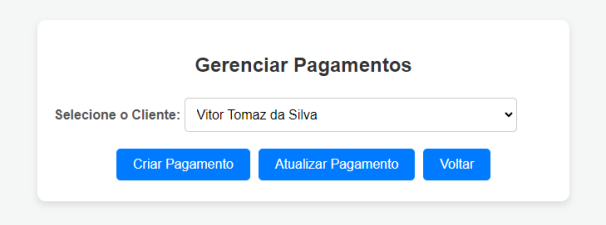


Figura 18 – Gerenciar Pagamento

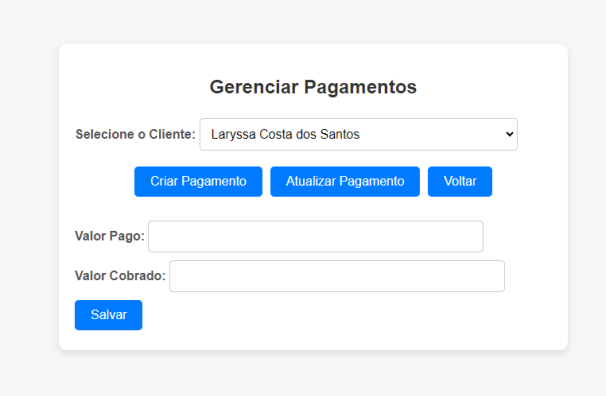


Figura 19 – Criar Pagamento

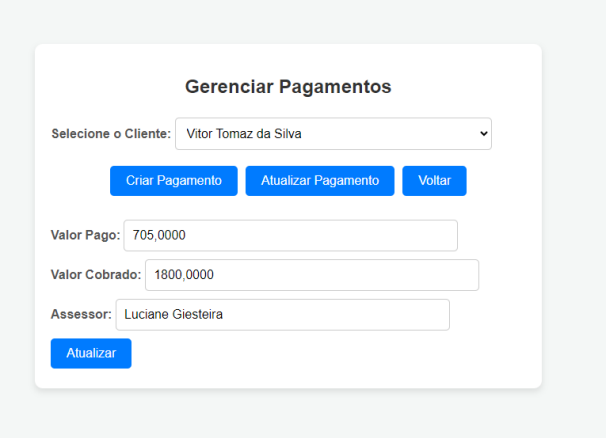


Figura 20 – Editar Pagamento

5.6 EDITAR INFORMAÇÕES PESSOAIS DO ASSESSOR

Por fim, temos o menu de edição das informações do assessor, Fig. 21, que está conectado, onde é possível alterar qualquer informação desejada de cadastrado, incluindo cadastrar um nova senha.

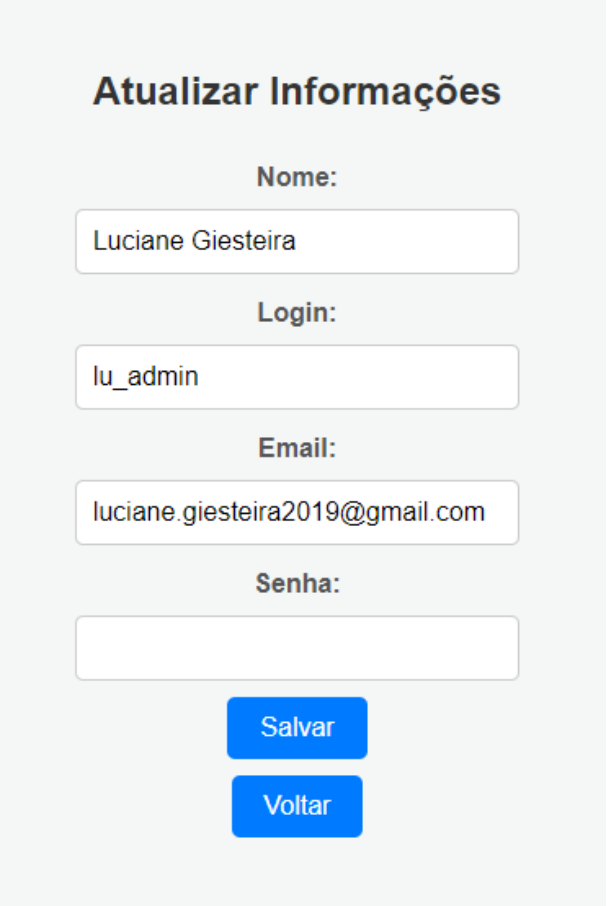


Figura 21 – Edição de Informações do Usuário

6 CONCLUSÃO E TRABALHOS FUTUROS

A assessoria de Visto Americano analisada durante o projeto possuía diversos desafios na organização de seu negócio e, por esse motivo, buscou uma solução digital personalizada. Atualmente, pequenas empresas estão buscando esse tipo de solução para potencializar o seu negócio, aumentando a eficácia dos processos.

Durante o desenvolvimento deste trabalho, a possibilidade de estar presente diariamente no ambiente de funcionamento do negócio facilitou a compreensão dos problemas e a identificação das necessidades para a aplicação do sistema, visando o aprimoramento das tarefas da empresa.

Os objetivos gerais e específicos do trabalho foram alcançados, pois os requisitos especificados foram atendidos durante o desenvolvimento. O controle sobre os dados dos clientes ficou mais simples, a atualização dos dados foi automatizada, permitindo que os usuários não percam tempo valioso com essa tarefa, e a comunicação entre os clientes e os assessores foi simplificada. Além disso, agora é possível ter um certo controle financeiro.

Por fim, para trabalhos futuros, seria ideal expandir o sistema para outros tipos de vistos, verificar a possibilidade de disponibilizar um calendário que gere avisos sobre o andamento dos processos dos clientes e, finalmente, tornar o sistema disponível para acesso web, criando também um perfil para o cliente, de modo que ele possa acompanhar o andamento de seu processo de maneira clara.

REFERÊNCIAS

ASSESSORIA visto americano: entenda como funciona! [*S.l.*: *s.n.*]. https://www4.mundodosvistos.com.br/p/quando-contratar-assessoria-para-visto-americano/. (Accessed on 01/07/2024).

BARBOSA, S.; SILVA, B. **Interação Humano-Computador**. [*S.l.*]: Elsevier Brasil, 2010. ISBN 9788535211207.

CASTILHO, Gustavo Uruguay; KAMIENSKI, Carlos Alberto. Aplicação de Computação em Névoa na Internet das Coisas para Cidades Inteligentes: da Teoria à Prática. *In*: ANAIS do XVI Workshop em Clouds e Aplicações. Campos do Jordão: SBC, 2018.

CASTRO, Alex Hyacon Carvalho de; KURTZ, Guilherme Chaga. Sistema de informação para controle de agendamento de tarefas para escritórios de advocacia.

COURAGE, C.; BAXTER, K. **Understanding Your Users: A Practical Guide to User Requirements Methods, Tools, and Techniques**. [*S.l.*]: Elsevier Science, 2005. (Interactive Technologies). ISBN 9781558609358.

EUA registram recorde de mais de 1,1 milhão de vistos emitidos para brasileiros em 2023. [*S.l.*: *s.n.*]. https://oglobo.globo.com/mundo/noticia/2024/01/05/eua-registram-recorde-de-mais-de-11-milhao-de-vistos-emitidos-para-brasileiros-em-2023.ghtml. (Acessado em 16/03/2024).

PRESSMAN, R.S.; MAXIM, B. **Engenharia De Software: UMA ABORDAGEM PROFISSIONAL**. [*S.l.*]: MCGRAW HILL - ARTMED, 2016. ISBN 9788580555332.

PROCURA por consultoria de vistos e cidadania dispara após vitória de Lula. [*S.l.*: *s.n.*]. https://www.correiobraziliense.com.br/brasil/2022/11/5050079-procura-por-consultoria-de-vistos-e-cidadania-cresce-apos-vitoria-de-lula.html. (Acessado em 16/03/2024).

SANTOS, Lucas Henrique Freitas dos. Desenvolvimento de um Sistema de Informação Web para gerenciar os processos de um buffet de doces e bombons, 2016.

SOMMERVILLE, Ian. **Engenharia de Software**. [*S.l.*]: Pearson, 2011.

VISTOS - Embaixada e Consulados dos EUA no Brasil. [*S.l.*: *s.n.*]. https://br.usembassy.gov/pt/visas-pt/. (Accessed on 03/17/2024).

ZIMMERMANN, Alexandre Ferronatto. **Pila fácil : sistema de gerenciamento de pedidos**. [*S.l.*: *s.n.*], 2010. `https://lume.ufrgs.br/handle/10183/28336?show=full`. (Accessed on 04/06/2024).

APÊNDICE A – DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITOS

Especificação de Requisitos - Visa House Assessoria em Documentação Consular

Figura A.1 – Logo - Visa House

A.1 OBJETIVO DO DOCUMENTO

O presente documento tem como objetivo especificar todos os requisitos do sistema e as necessidades do cliente em relação ao mesmo, de modo a firmar um acordo que especifique as funcionalidades e restrições para o desenvolvimento do projeto.

A.2 OBJETIVO DO SISTEMA

Atualmente, o cliente não possui um sistema que automatiza diversas tarefas, sendo assim, grande parte do processo é feita de maneira manual, onde os funcionários se encontram sobrecarregados. Além disso, o cliente não possui um sistema de controle de cadastro efetivo dos clientes, um controle efetivos dos agendamentos dos solicitantes e um canal de comunicação rápida entre solicitantes e os funcionários da empresa. O principal objetivo do sistema é fornecer ao cliente, um sistema que permita ao cliente otimizar as tarefas mais simples do processo, manter um controle adequado dos agendamentos dos solicitantes, fornecer um controle financeiro simples e criar um canal com o cliente para fornecer as principais informações ao cliente.

A.3 FUNCIONALIDADES DO SISTEMA

A partir das necessidades e características do cliente, podemos destacar as principais funcionalidades do sistema como sendo:

1. Cadastro de novos solicitantes: A funcionalidade permitirá ao acesso cadastro um novo solicitantes no sistema, com todas as informações do solicitante.

A partir das informações fornecidas, o sistema deve armezenar as principais informações, e gerar um novo cadastro junto ao sistemas do consulado americano e realizar os passos iniciais do processo.

2. Gerenciamento de Agendamentos: A funcionalidade do sistema deve realizar o gerenciamento dos agendamentos do solicitantes, de modo a mante-lo sempre atualizado. Além disso, deve ter como objetivo realizar agendamento e antecipar datas de agendamento caso seja preciso.
3. Cadastro de Pagamentos: A funcionalidade deve permitir ao usuário mantenha as informações de quando o solicitante já pagou, o valor que foi cobrado e o valor que ainda deve.
4. Visualização dos Agendamento: A funcionalidade deve permitir que o usuário verifique todos os solicitantes agendados, além das principais informações do solicitante, como, endereço eletrônico de cadastro, telefone de contato, datas do agendamento, e o local dos agendamentos.

A.4 PRAZO

O **prazo** total para desenvolvimeno do sistema desde seu início até a instalação do mesmo para o cliente, será de **até 6 meses após o início do desenvolvimento**, podendo ser entregue antes do prazo final. Conforme o desenvolvimento for avançando, e as funcionalidades forem sendo entregues, deve-se realizar reuniões para mostrar o andamento do processo e também para que o cliente possa dar sugestões a serem feitas que permitiram o aprimoramento do sistema antes da entrega final, além de dar autonomia ao cliente durante o processo e possa acompanhar o andamento do desenvolvimento.

A.4.1 Necessidade de Equipamento

O sistema desenvolvido será um sistema Web, que poderá ser acessado através de qualquer navegador de internet. Para hospedar o sistema e o banco de dados, será necessário um computador com as configurações mínimas de:

- 8GB de memória
- 50GB de disco rígido (HD)

A.4.2 6. Considerações Importantes

1. O foco do sistema deve ser em fornecer um solução simples e eficiência, que permita o cliente ter um visão geral de todos os solicitantes do visto americano.

2. O sistema deve estar disponível para ser utilizado em qualquer computador na rede local da agência.
3. O sistema deve possuir boa fluidez.
4. A interface do sistema deve ser adaptada apenas para utilização em computadores.
5. A cada etapa do processo de desenvolvimento o cliente deve ser notificado para que possa acompanhar o processo, podendo dar seu feedback de cada uma das funcionalidades que possam haver aprimoramentos.
6. O presente documentos é apenas o reconhecimentos da proposta e para firmar um acordo entre partes, sem valor legal.

Assinaturas:

Caio Giesteira Cardoso (Desenvolvedor)	
Luciane Giesteira Moyses (Proprietário)	

APÊNDICE B – CASOS DE USO

B.1 ESPECIFICAÇÃO DO CASO DE USO: UC-001 GERENCIAR ASSESSOR

B.1.1 Objetivos

O objetivo da funcionalidade é permitir cadastrar usuários para acessar o sistema de assessoria.

B.1.2 Atores

Administrador do sistema que deseja gerenciar os usuários do sstema.

B.1.3 Contexto

Este caso de uso é iniciado quando o ator deseja gerenciar os usuários.

B.1.4 Pré-Condições

Deve-se estar utilizando o acesso de administrador do sistema.

B.1.5 Fluxos de Execução

- **FE01 - Cadastrar Usuário**

1. O sistema exibe a lista de todos os usuários cadastrados. (RN001)
2. O ator seleciona o comando de cadastro de novo usuário.
3. O sistema exibe a tela cadastro de novo usuário.
4. O ator preenche as informações do novo usuário e acionar o comando cadastrar. (RN002, RN002, RN003)
5. O sistema salva o usuário na base de dados e retorna a lista de todos os usuários.
6. O fluxo de execução é encerrado.

- **FE02 - Excluir Usuário**

1. O sistema exibe a lista de todos os usuários cadastrados. (RN001)
2. O ator seleciona um dos usuários da lista de usuários cadastrados.
3. O ator seleciona o comando excluir. (RN005)
4. O sistema apaga o usuário da base de dados e retorna a lista de todos os usuários.
5. O fluxo de execução é encerrado.

- **FE03 - Alterar Usuário**

1. O sistema exibe a lista de todos os usuários cadastrados. (RN001)
2. O ator seleciona um dos usuários da lista de usuários cadastraods.
3. O ator seleciona o comando editar. (RN005)
4. O sistema exibe a tela de edição com os dados do usuário.
5. O ator altera as informações desejadas e aciona o botão salvar. (RN003, RN004)
6. O sistema atualiza as informações no banco de dados e retorna a lista de todos os usuários.
7. O fluxo de execução é encerrado.

B.1.5.1 Regras de Negócios

ID	Descrição
RN001	O sistema deve lista todos os usuários em ordem alfabética.
RN002	O sistema não permite cadastrar o usuário caso exista outro usuário com o mesmo C.P.F. que o informado.
RN003	O sistema não permite salvar o usuário sem os campos obrigatórios (Nome, C.P.F, Email, Senha, Confirmação da Senha, Contato) estiverem pre-enchidos.
RN004	O sistema não deve permitir salvar o usuário se a senha informada for diferente da confirmação da senha.
RN005	Aos usuários ativos, o sistema deve disponibilixar um comando para editá-los ou excluir-los.

Tabela 5 – Regras de Negócio (UC-001)

B.2 ESPECIFICAÇÃO DO CASO DE USO: UC-002 GERENCIAR AGENDAMENTOS

B.2.1 Objetivos

O objetivo da funcionalidade é permitir gerenciar os agendamentos dos clientes.

B.2.2 Atores

Usuários autenticados que desejam fazer alterações nas informações referente aos agendamentos dos clientes.

B.2.3 Contexto

Este caso de uso é iniciado quando o ator deseja inserir, alterar ou atualizar informações sobre de agendamento do cliente.

B.2.4 Pré-Condições

Para acessar essa funcionalidade deve-se estar devidamento autenticado no sistema.

B.2.5 Fluxos de Execução

- **FE001 - Atualizar Agendamentos**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001, RN006)
2. O ator seleciona o comando de atualizar agendamentos. (RN002)
3. O sistema atualiza as informações dos agendamentos com base no sistema de agendamento.
4. O sistema informa que os agendamentos foram atualizados.
5. O sistema exibe a lista de clientes cadastrados atualizada.
6. O fluxo de execução é encerrado.

- **FE002 - Antecipar Agendamento**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001, RN006)
2. O ator seleciona o cliente e, em seguida, seleciona o comando antecipar.
3. O sistema disponibiliza a tela com informações necessárias para antecipação. (RN003)
4. O ator fornece as informações necessários para antecipar o agendamento. (RN005)
5. O sistema realiza a antecipação do agendamento do cliente, e quando houve sucesso informa o cliente e o usuário sobre nova data de agendamento. Além disso, atualiza as informações no banco de dados.
6. O fluxo de execução é encerrado.

- **FE003 - Cancelar Agendamento**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001, RN006)
2. O ator seleciona o cliente e, em seguida, seleciona o comando cancelar agendamento. (RN004)
3. O sistema realiza o cancelamento do agendamento e atualiza a informação no banco de dados.
4. O fluxo de execução é encerrado.

B.2.6 Regras de Negócios

ID	Descrição
RN001	O sistema deve permitir ordenar a lista de clientes em ordem alfabética, ou pela data dos agendamentos.
RN002	O sistema deve disponibilizar um comando para atualizar os agendamentos do sistema.
RN003	O sistema deve disponibilizar uma ferramenta de antecipação de agendamento a partir das preferências do cliente.
RN004	O sistema deve disponibilizar uma ferramenta para realizar o cancelamento do usuário.
RN005	O sistema não deve antecipar o agendamento enquanto o usuário não preencher os campos "Cidade Desejada"e a "Data Máxima".
RN006	O sistema deve disponibilizar na lista de clientes as informações dos datas e locais dos agendamentos de cada cliente.

Tabela 6 – Regras de Negócio (UC-002)

B.3 ESPECIFICAÇÃO DO CASO DE USO: UC-003 GERENCIAR SOLICITANTES

B.3.1 Objetivos

O objetivo da funcionalidade é permitir criar novos clientes, alterar os dados dos clientes cadastrados e excluir clientes no sistema.

B.3.2 Atores

Usuários autenticados que desejam fazer alterações nas informaçõeos dos clientes.

B.3.3 Contexto

Este caso de uso é iniciado quando o ator deseja alterar alguma informaççao do cliente.

B.3.4 Pré-Condições

Para acessar essa funcionalidade deve-se estar devidamento autenticado no sistema.

B.3.5 Fluxos de Execução

- **FE001 - Criar Cliente**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona o comando de criar no cliente.
3. O sistema exibe a tela de criação do cliente.
4. O ator preenche as informações do cliente e seleciona o comando criar. (RN003)
5. O sistema salva as informações no banco de dados
6. O fluxo de execução é encerrado.

- **FE002 - Editar Dados**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona o cliente e, em seguida, seleciona o comando editar. (RN002)
3. O sistema disponibiliza a tela com as informações do cliente.
4. O ator altera as informações desejadas e seleciona o comando salvar. (RN003)
5. O sistema salva as alterações na base dados e retorna a lista de clientes.
6. O fluxo de execução é encerrado.

- **FE003 - Excluir Cliente**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona o cliente e, em seguida, seleciona o comando excluir. (RN002)
3. O sistema exibe um tela solicitante que o usuário confirme que deseja excluir esse cliente. (RN004)
4. O usuário seleciona o comando sim.
5. O sistema remova o cliente da base de dados e retorna a lista de clientes.
6. O fluxo de execução é encerrado.

B.3.6 Regras de Negócios

B.4 ESPECIFICAÇÃO DO CASO DE USO: UC-004 GERENCIAR PAGAMENTOS

B.4.1 Objetivos

O objetivo da funcionalidade é permitir gerenciar os pagamentos dos cliente.

ID	Descrição
RN001	O sistema deve permitir ordenar a lista de clientes em ordem alfabética, ou pela data dos agendamentos;
RN002	O sistema deve permitir pesquisar por nome ou C.P.F.
RN003	O sistema não deve permitir que as informações sejam salvas sem os campos obrigatórios (Nome, C.P.F, Email, Senha, Contato Telefônico) estarem preenchidos.
RN004	O sistema deve exigir uma confirmação do usuário antes da exclusão de um cliente.

Tabela 7 – Regras de Negócio (UC-003)

B.4.2 Atores

Usuários autenticados que desejam fazer alterações nas informações referente aos pagamentos dos cliente.

B.4.3 Contexto

Este caso de uso é iniciado quando o ator deseja inserir ou alterar informações sobre o pagamento de um cliente.

B.4.4 Pré-Condições

Para acessar essa funcionalidade deve-se estar devidamento autenticado no sistema.

B.4.5 Fluxos de Execução

- **FE001 - Inserir ou Atualizar Pagamento**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001, RN005)
2. O ator seleciona um cliente, e aciona o comando pagamento. (RN002)
3. O sistema exibe as informaçoes de pagamento do cliente.
4. O ator insere as informações de pagamento do cliente e aciona o botão salvar. (RN004, RN006, RN007)
5. O sistema atualiza as informações do banco de dados.
6. O fluxo de execução é encerrado.

ID	Descrição
RN001	O sistema deve permitir ordenar a lista de clientes em ordem alfabética, ou pela data dos agendamentos.
RN002	O sistema deve disponibilizar um comando para atualizar as informações de pagamento do cliente.
RN003	O sistema deve disponibilizar uma ferramenta para notificar o solicitante sobre o valor devido.
RN004	O sistema só deve atualizar as informações de pagamento se as informações refente ao "Valor Pago"e Valor Cobrado"estiverem preenchidos.
RN005	O sistema deve fornecer na lista de clientes as informações de pagamento de cada cliente.
RN006	O sistema não permite colocar o valor devido maior que o valor cobrado.
RN007	O sistema utilizada a moeda Reais (BRL) para todos os valores informados.

Tabela 8 – Regras de Negócio (UC-04)

B.4.6 Regras de Negócios

B.5 ESPECIFICAÇÃO DO CASO DE USO: UC-005 GERENCIAR COMUNICAÇÃO

B.5.1 Objetivos

O objetivo da funcionalidade é permitir criar um canal de comunicação entre o cliente, o usuário e o sistema.

B.5.2 Atores

Cliente e o Usuário responsavél pela orientação do cliente.

B.5.3 Contexto

Este caso de uso é iniciado quando algum informação relevante deve ser informada ao cliente pelo usuário através do sistema.

B.5.4 Pré-Condições

Para acessar essa funcionalidade deve-se estar devidamento autenticado no sistema.

B.5.5 Fluxos de Execução

- **FE001 - Notificar Usuário Responsável**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona um cliente, e aciona o comando criar canal de comunicação.
3. O sistema cria um canal entre o cliente e usuário para comunicarem e informa ao cliente as informações iniciais. (RN002, RN004)
4. O fluxo de execução é encerrado.

- **FE002 - Notificar Valor Devido**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona um cliente, e aciona o comando pagamento.
3. O sistema exibe as informaçoes de pagamento do cliente.
4. O ator seleciona o comando de enviar notificação de valor devido. (RN003)
5. O sistema envia uma mensagem ao cliente informando sobre o valor devido no canal de comunicação.
6. O fluxo de execução é encerrado.

- **FE003 - Notificar Agendamento**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona um cliente e aciona o comando a notificação de agendamento. (RN005)
3. O sistema envia uma mensagem ao cliente informando a data e local do agendamento do cliente.
4. O fluxo de execução é encerrado.

- **FE003 - Notificar Taxa Consular**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona um cliente e aciona o comando de enviar taxa consular.
3. O sistema exibe tela da criação de boleto da taxa consular. (RN006)
4. O ator fornece os daods necessários para criação do boleto da taxa consular.
5. O sistema cria o boleto e envia o documento ao cliente atravpes do canal de comunicação.
6. O fluxo de execução é encerrado.

B.5.6 Regras de Negócios

B.6 ESPECIFICAÇÃO DO CASO DE USO: UC-06 ABERTURA DE PROCESSO

B.6.1 Objetivos

O objetivo da funcionalidade é permitir a criação ou alteração de um processo do visto americano.

ID	Descrição
RN001	O sistema deve permitir ordenar a lista de clientes em ordem alfabética, ou pela data dos agendamentos.
RN002	O sistema deve criar um canal de comunicação apenas quando o usuário acionar o comando.
RN003	O sistema deve notifiar o valor devido em reais e informar os meios de pagamento.
RN004	O sistema deve enviar uma mensagem ao cliente através do *telegram*.
RN005	O sistema deve enviar os documentos necessários junto com a notificação de agendamento.
RN006	O sistema deve verificar se o cliente ja possui uma conta criada.

Tabela 9 – Regras de Negócio (UC-005)

B.6.2 Atores

Usuário que deseja realizar a criação ou alteração de um processo de um cliente.

B.6.3 Contexto

Este caso de uso é iniciado quando o usuário deseja realizar o preenchimento do DS-160 de um cliente.

B.6.4 Pré-Condições

Para acessar essa funcionalidade deve-se estar devidamento autenticado no sistema.

B.6.5 Fluxos de Execução

- **FE001 - Criar Conta**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona um cliente, e aciona o comando criar conta.
3. O sistema exibe a tela dos dados necessários para criar conta.
4. O ator realiza o preenchimento dos dados e, ao final, aciona comando iniciar processo.(RN003)
5. O sistema iniciar a criação de conta junto ao sistama do visto americano e solicita o link para confirmaçõ da crição de contao ao ator. (RN002)
6. O ator fornece o link disponível para o sistema.

7. O sistema finaliza a criação do processo e salva as informações no banco de dados.(RN004)
8. O sistema envia uma mensagem de sucesso ao usuário e atualiza as informações no banco de dados. (RN005)
9. O fluxo de execução é encerrado.

- **FE002 - Atualizar Local de Retirada**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona um cliente, e aciona o comando atualizar local de retirada.
3. O sistema exibe a tela com os locais de retirada disponivéis.
4. O ator seleciona qual local deseja e aciona o comando salvar.
5. O sistema acessa a conta do cliente, atualiza a informação e informar ao usuário. (RN005)
6. O fluxo de execução é encerrado.

- **FE003 - Atualizar Informações da conta.**

1. O sistema exibe a lista de clientes cadastrados nos sistema. (RN001)
2. O ator seleciona um cliente, e aciona o comando atualizar informações da conta.
3. O sistema exibe a tela com os dados da conta.
4. O ator realiza o preenchimento dos dados e, ao final, aciona comando salvar.
5. O sistema acessa a conta do usuário, atualiza as informações necessária e informa ao usuário.(RN005)
6. O fluxo de execução é encerrado.

B.6.6 Regras de Negócios

ID	Descrição
RN001	O sistema deve permitir ordenar a lista de clientes em ordem alfabética, ou pela data dos agendamentos.
RN002	O sistema deve sempre solicitar ao usuário que forneça os dados necessários que são enviar por email para criação da conta.
RN003	O sistema deve criar a conta com base no email e senha fornecido pelo usuário
RN004	O sistema deve manter salvo as informações cruciais para acesso ao processo.
RN005	O sistema deve sempre informar ao usuário se a operação foi bem sucedida ou não.

Tabela 10 – Regras de Negócio (UC-005)

APÊNDICE C – PROTÓTIPOS DE INTERFACE

Nessa sessão serão apresentados os protótipos das interfaces de cada uma das funcionalidades criadas.

C.1 PRÓTIPO DE TELA - LOGIN

Figura C.1 – Protótipo: Tela de Login

C.2 PRÓTIPO DE TELA - MENU PRINCIPAL

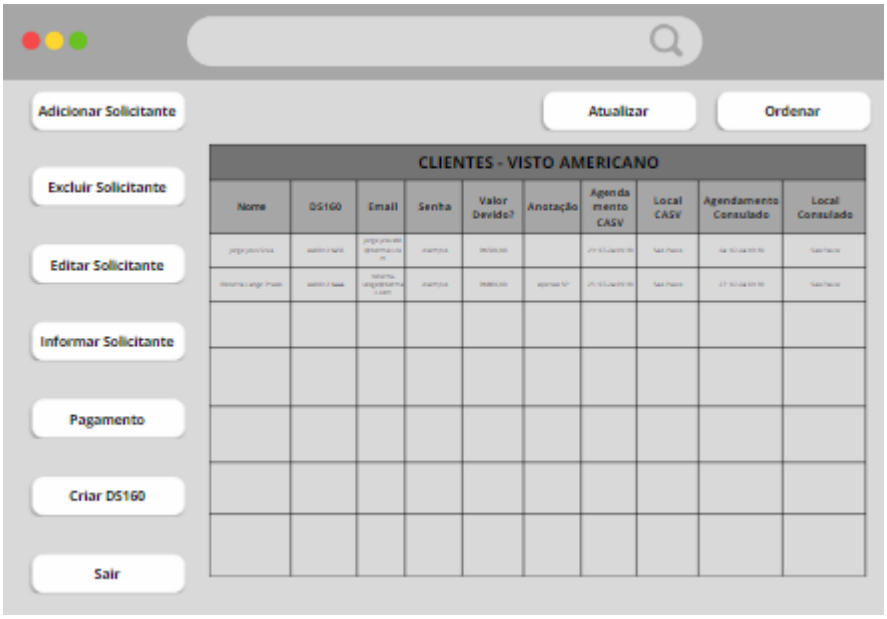


Figura C.2 – Protótipo: Menu Principal

C.3 PRÓTIPO DE TELA - ADICIONAR SOLICITANTE

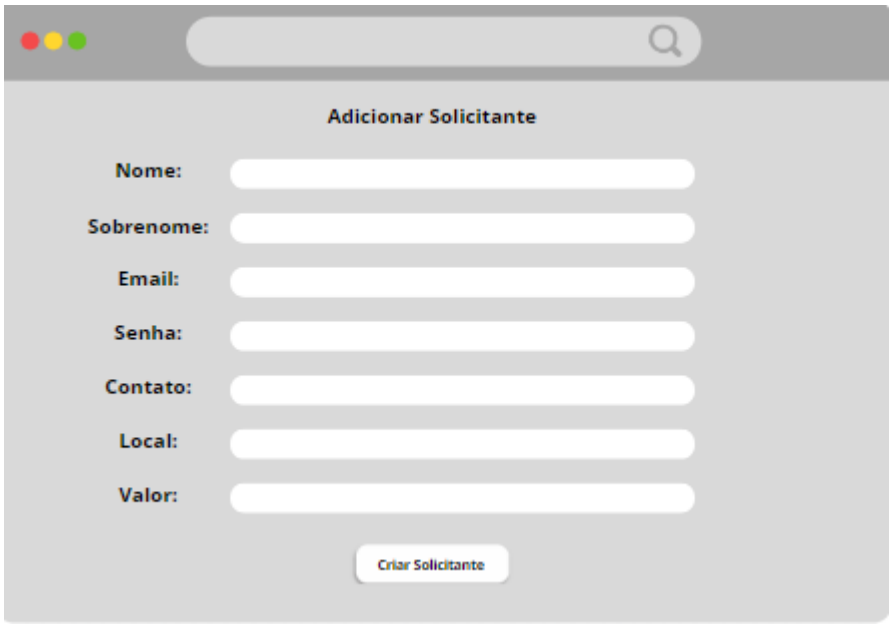


Figura C.3 – Protótipo: Adicionar Solcitante

C.4 PRÓTIPO DE TELA - EDITAR SOLICITANTE

Figura C.4 – Protótipo: Editar Solicitante

C.5 PRÓTIPO DE TELA - EXCUIR SOLICITANTE

Figura C.5 – Protótipo: Excluir Solcitante

C.6 PRÓTIPO DE TELA - INFORMAR SOLICITANTE

Figura C.6 – Protótipo: Informar Solicitante

C.7 PRÓTIPO DE TELA - INFORMAR PAGAMENTO

Figura C.7 – Protótipo: Informar Pagamento

ANEXO A – ARTIGO

Proposta de Sistema de Gerencimento para Empresa de Assessoria em Visto Americano

Caio G. Cardoso[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
88.040-900 – Florianópolis – SC – Brasil

caio.g.cardoso@grad.ufsc.br

Abstract. *The objective of this work is to present a proposed solution for an American visa consulting firm that has encountered problems in various areas of its operations. Additionally, this work aims to cover all stages of software development, from defining project requirements to the final delivery. Finally, the final product delivered to the client and the possibilities for future improvements will be presented.*

Resumo. *O objetivo deste trabalho é apresentar uma proposta de solução para uma assessoria de visto americano que tem encontrado problemas em diversas áreas de seu funcionamento. Além disso, este trabalho tem a intenção de abordar todas as etapas do desenvolvimento de software, desde a definição dos requisitos do projeto até a entrega final. Por fim, será apresentado o produto final entregue ao cliente e as possibilidades de melhorias futuras.*

1. Introdução

O visto americano é uma permissão oficial emitida pelo governo dos Estados Unidos que permite que cidadãos estrangeiros entrem no país por um determinado período de tempo e para um propósito específico. Este documento é essencial para viajantes que desejam visitar os Estados Unidos para turismo, negócios, estudo, trabalho temporário ou outras atividades autorizadas [Vis].

O processo de obtenção de um visto americano pode variar de acordo com o tipo de visto solicitado e a nacionalidade do requerente, e geralmente envolve o preenchimento de formulários, a participação em entrevistas consulares e a apresentação de documentos que comprovem a elegibilidade do solicitante para a entrada no país [Vis].

O visto americano desempenha um papel crucial no controle de fronteiras e na segurança nacional dos Estados Unidos, ao mesmo tempo em que facilita o intercâmbio cultural, econômico e educacional entre os Estados Unidos e o resto do mundo [Vis].

1.1. Motivação

De acordo com a Embaixada dos Estados Unidos no Brasil, o ano de 2023 registrou a emissão de mais de 1,1 milhão de Vistos Americanos no país, marcando um recorde histórico pós-pandemia [EUA]. No entanto, apenas 5,8 milhões de brasileiros possuem o visto americano até o momento. Além disso, há previsão de que mais de 1 milhão de vistos serão emitidos em 2024, com potencial para alcançar o recorde histórico de 2,2 milhões de vistos emitidos durante o ano de 2019 [Pro].

Com isso, a demanda por empresas de auxílio no processo para obtenção de visto americano tende a crescer em 2024, conforme indicado pelo aumento no número de ligações em algumas empresas, que aumentou quatro vezes em comparação com o mesmo período do ano anterior[Pro].

1.2. Justificativa

Atualmente, o processo de solicitação do visto americano é inteiramente manual, colocando a responsabilidade em todas as etapas, desde a abertura até a conclusão, sobre o solicitante. Em empresas de assessoria, um assessor é designado para executar essas etapas para dezenas de clientes. Com o aumento da demanda por esses serviços, os assessores podem se ver sobrecarregados, o que pode levar a erros devido ao acúmulo de trabalho. Esses erros potenciais representam um risco significativo, podendo prejudicar o processo de um cliente ou até mesmo resultar na negação do visto americano para o mesmo [Ass].

1.3. Objetivo Geral

O objetivo deste trabalho é empregar a computação para automatizar as etapas do processo de obtenção do visto americano, visando a otimização do trabalho do assessor de visto. O foco principal está nos desafios associados a todas as fases do processo de obtenção do visto americano, o gerenciamento de bases de dados de clientes e o estabelecimento de um relacionamento eficaz entre o assessor e o cliente.

1.4. Objetivos Específicos

Os objetivos especifícos desse trabalho são os seguintes:

- Integrar os diferentes sistemas para a realização do visto americano;
- Introduzir um modelo de banco de dados para efetuar o gerenciamento de clientes;
- Fornecer mecanismos que facilitem a comunicação entre o cliente e o assessor;
- Validar a proposta através da implementação de um sistema.

1.5. Método de pesquisa e trabalho

A seguir são apresentados os métodos de estudos de trabalho:

1. Descrever os principais desafios e problemas existentes nas etapas do visto americano e no gerenciamento de clientes.
2. Realizar uma pesquisa em como os diferentes sistemas funcionam e como é possível automatizar o processo.
3. Realizar uma revisão sistemática para encontrar soluções existentes, as quais possam ser aprimoradas ou que sirvam de base para novas soluções.
4. Buscar tecnologias e conceitos que possam auxiliar na elaboração da proposta.
5. Desenvolver e testar o sistema com base no que foi estudado.

2. Visto Americano

O visto consular funciona como uma pré-autorização de viagem, na qual o país de destino confirma que o viajante cumpriu as exigências de embarque. Contudo, possuir um visto não assegura a entrada no país. A decisão é tomada pelo oficial de imigração na chegada, ao verificar se o viajante satisfaz todos os requisitos, inclusive ter o visto em posse. No caso dos Estados Unidos, existem duas classes de visto americano, sendo elas: Vistos de Imigrante e os Vistos de Não Imigrante.

Figure 1. Tipos de Visto

2.1. Tipos de Visto

2.1.1. Visto de Imigrante

O visto de imigrante é uma autorização concedida a indivíduos que desejam morar permanentemente nos Estados Unidos. Ao contrário do visto de não imigrante, que é para estadias temporárias, o visto de imigrante é destinado àqueles que desejam residir permanentemente nos EUA e receber o status de residente legal permanente, também conhecido como *"green card"*. [Vis].

Existem várias categorias de visto de imigrante, cada uma designada para diferentes situações, como reunificação familiar, emprego, refúgio, entre outras. Alguns exemplos de categorias de visto de imigrante são [Vis]:

- Visto de Parente Imediato (IR): Destinado a cônjuges de cidadãos americanos, filhos solteiros menores de 21 anos de cidadãos americanos e pais de cidadãos americanos com mais de 21 anos de idade.
- Visto de Trabalhador Permanente (EB): Designado para trabalhadores estrangeiros com habilidades específicas e experiência profissional, incluindo profissionais altamente qualificados, trabalhadores com habilidades excepcionais, empresários e investidores.
- Visto de Refugiado (RS): Emitido para indivíduos que foram obrigados a deixar seus países de origem devido a perseguição, guerra ou violência, e que têm o status de refugiado reconhecido pelo governo dos EUA.
- Visto de Investidor (EB-5): Para investidores estrangeiros que desejam investir em projetos comerciais nos Estados Unidos e criar empregos para trabalhadores americanos.

O processo para obter um visto de imigrante geralmente é mais complexo e demorado do que o processo para obter um visto de não imigrante. Normalmente, envolve a apresentação de petições, cumprimento de requisitos específicos e passar por entrevistas e exames médicos. Assim, uma vez concedido o visto de imigrante e após a entrada nos Estados Unidos, o indivíduo torna-se um residente legal permanente e pode desfrutar de muitos dos mesmos direitos e responsabilidades que os cidadãos americanos, incluindo o direito de viver e trabalhar permanentemente nos EUA [Vis].

2.1.2. Visto de Não Imigrante

O visto de não imigrante é uma autorização concedida a indivíduos que desejam entrar temporariamente nos Estados Unidos para uma finalidade específica e por um período limitado de tempo. Ao contrário do visto de imigrante, que é para aqueles que buscam residência permanente nos EUA, o visto de não imigrante é designado para pessoas que

não têm intenção de permanecer no país e que planejam retornar ao seu país de origem após a conclusão da atividade autorizada pelo visto. Existem várias categorias de vistos de não imigrante, cada uma designada para diferentes finalidades temporárias, as princiáis categorias são [Vis]:

- Visto de Turismo ou Negócios (B-1/B-2): Destinado a pessoas que desejam visitar os Estados Unidos para fins de turismo, lazer, visita a amigos ou parentes, atividades comerciais temporárias, como conferências, negociações de contratos, consultas de negócios, treinamento curto, entre outros.
- Visto de Estudante (F-1, M-1): Para estudantes estrangeiros matriculados em instituições educacionais nos Estados Unidos. O visto F-1 é para estudantes acadêmicos, enquanto o visto M-1 é para estudantes de programas vocacionais.
- Visto de Trabalho Temporário (H-1B, H-2B): Para profissionais estrangeiros altamente qualificados que vão trabalhar temporariamente nos Estados Unidos em ocupações especializadas que exigem conhecimento especializado. O visto H-2B é para trabalhadores temporários não agrícolas.
- Visto de Intercâmbio (J-1): Destinado a participantes de programas de intercâmbio cultural, educacional ou profissional, incluindo estudantes de intercâmbio, professores, pesquisadores, médicos residentes, entre outros.

O visto de não imigrante é emitido mediante solicitação prévia e aprovação do Serviço de Cidadania e Imigração dos Estados Unidos (USCIS) ou do Departamento de Estado dos Estados Unidos, dependendo da categoria de visto e do país de residência do requerente [Vis].

2.2. Funcionamento Atual de um Assessoria em Visto Americano

Nesta sessão são descritas as etapas realizadas por uma empresa de acessoria de pequeno porte, especializada em vistos americanos, para conceder os vistos a seus clientes. Um único acessor está envolvido durante todas as etapas envolvidas no atendimento a um determinado cliente, conforme ilustrado pelo fluxograma da Fig. 2. Através deste fluxograma é possível observar a ordem da realização destas etapas, as quais são descritas com mais detalhes no restante deste capítulo.

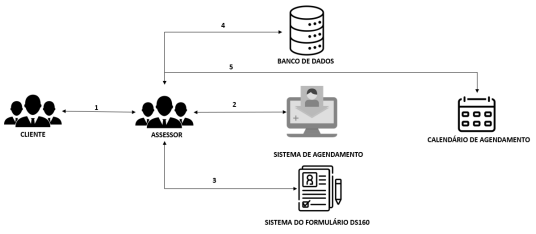


Figure 2. Fluxograma simplificado de uma empresa de assessoria em visto.

2.2.1. Etapa 1: Atendimento ao Cliente

A primeira etapa a ser compreendida é o atendimento ao cliente, como descrito na interação número um do fluxograma da Figura 2. Nessa fase, o assessor é encarregado de

estabelecer e manter contato com o cliente ao longo de todo o processo de obtenção do visto. Isso implica em executar uma série de tarefas, tais como:

- Fornecer uma explicação detalhada das etapas envolvidas no processo de obtenção do visto americano ao cliente.
- Acompanhar o cliente durante a negociação dos honorários pelos serviços prestados, discutindo os valores envolvidos e detalhando todos os custos associados ao processo.
- Solicitar todas as informações essenciais ao cliente para preencher adequadamente os formulários exigidos.
- Oferecer orientações detalhadas e instruções claras sobre os procedimentos em cada uma das etapas do processo. Além disso, é fundamental resolver quaisquer dúvidas ou preocupações que o cliente possa ter ao longo do caminho.

2.2.2. Etapa 2: Cadastro do Cliente no Sistema de Agendamento

Nesta fase do processo, o assessor, utilizando as informações fornecidas pelo cliente, dá início ao procedimento junto ao Sistema de Agendamento, que é responsável pelo processamento da Taxa Consular. É crucial que, nesse momento, o assessor já tenha determinado o tipo de visto consular com base no motivo de viagem informado pelo cliente, uma vez que o valor da taxa pode variar de acordo com a categoria do visto. Após o cadastro do cliente no sistema, o assessor deve proceder com o pagamento da taxa consular de acordo com a escolha do cliente entre as opções disponíveis na localidade. No Brasil, por exemplo, as opções comuns são Cartão de Crédito e Boleto Bancário.

2.2.3. Etapa 3: Preenchimento de Formulário

Esta etapa é mais importante, pois é aqui que o assessor preenche o formulário DS-160 em nome do cliente com base nas informações fornecidas. O formulário DS-160 é importante, pois sintetiza todas as informações pertinentes para a obtenção do visto do cliente, sendo posteriormente minuciosamente analisado pelo agente consular durante o processo de solicitação do visto americano.

2.2.4. Etapa 4: Atualizações do Banco de Dados

Como responsável pelo processamento de diversos pedidos de visto americano, é crucial que o assessor mantenha sua base de dados organizada e atualizada. Esta base de dados deve conter informações cruciais sobre os processos e clientes, como detalhes de login e senha, informações de contato dos clientes, agendamentos e locais dos processos de visto, bem como registros precisos de valores pagos e pendentes, entre outros dados relevantes. A falta de atualização dessas informações pode acarretar diversos problemas no processo, incluindo:

- Perda de contato com o cliente devido à falta de informações de contato atualizadas, o que pode resultar em dificuldades para fornecer atualizações importantes sobre o processo do visto e perder oportunidades de comunicação vital.

- Perda do agendamento consular do cliente devido a informações desatualizadas sobre datas e horários, o que pode resultar em atrasos significativos ou até mesmo na necessidade de reagendar todo o processo, gerando inconveniências e custos adicionais para o cliente.
- Pagamento duplicado de taxas consulares devido à falta de inclusão do cliente no banco de dados, o que pode levar a problemas financeiros para o cliente, além de resultar em atrasos desnecessários e complicações administrativas.
- Não recebimento de valores devidos, causado pela falta de controle financeiro na planilha sobre tais informações, o que pode resultar em problemas de fluxo de caixa para a empresa de assessoria e, potencialmente, em desentendimentos com os clientes devido a cobranças incorretas ou ausência de pagamentos.

2.2.5. Etapa 5: Calendário de Agendamento

Por fim, cabe ao assessor planejar o agendamento de acordo com as datas e localidades disponíveis no sistema, levando em consideração as necessidades individuais de cada cliente. Em muitas situações, os clientes têm urgência no processo, e é responsabilidade do assessor agir com rapidez máxima para atender a essas necessidades. Isso envolve priorizar agendamentos que se encaixem nas demandas específicas do cliente e garantir que o processo seja conduzido o mais eficientemente possível.

2.2.6. Identificação do Problema

Após analíse de todas as etapas de realização do visto americano, é preciso identificar os principais problemas enfrentados pelo estabelecimento, os quais poderiam ser resolvidos por meio do sistema. Com isso, identificamos os seguintes problemas:

- **Falta de Gerenciamento de agendamentos**: Atualmente, os agendamentos são controlados por meio de uma planilha, na qual todos os dados são mantidos e atualizados manualmente. Como resultado, a planilha está sujeita a erros, que podem levar à perda de agendamentos, já que uma modificação pode ser feita e não atualizada na planilha de controle, resultando em informações .
- **Acúmulo de funções dos funcionários**: No funcionamento atual, o assessor é responsável por todas as etapas do processo. Isso significa que o assessor deve negociar com o cliente, abrir o processo junto às entidades envolvidas, preencher os formulários necessários, manter a comunicação com o cliente, entre outras tarefas. No entanto, o acúmulo de funções pode prejudicar o desempenho nas tarefas mais importantes.
- **Contato com o cliente constante**: Devido à grande quantidade de clientes, frequentemente a comunicação entre os assessores e os clientes não é ideal, o que pode resultar em problemas, como a falta de aviso aos clientes sobre seus agendamentos com antecedência, falta de informações sobre a documentação necessária ou a localização de seus compromissos.
- **Controle financeiro** No momento, não há controle financeiro no estabelecimento. Como resultado, os assessores são responsáveis pelo pagamento de seus clientes, o que significa que não há disponibilidade para efetuar todas as cobranças necessárias, caso seja preciso.

3. Estado da Arte

Nesse capítulo faremos os estudos de artigos que possuem objetivos similares aos projeto que está sendo desenvolvido nesse trabalho.

3.1. Pila Fácil: Sistema de Gerenciamento de Pedidos

O Pila Fácil revoluciona o gerenciamento de clientes em estabelecimentos comerciais do ramo alimentício, bares e casas de festa. Com ele, os clientes têm acesso a um sistema completo que facilita sua experiência. Através do aplicativo, é possível consultar o cardápio, acompanhar sua comanda de gastos em tempo real e efetuar o pagamento de forma simples e conveniente, tudo diretamente pelo celular [Zimmermann 2010].

O sistema é acessado pela Internet tanto pelo cliente, que realiza seus pedidos e o acompanhamento dos gastos através da aplicação, como pelo estabelecimento, que realiza o gerenciamento dos clientes pelo estabelecimento. O Pila Facil é encarregado de tarefas como o processo de pagamento, envio de pedidos a cozinha e cadastro do cliente. Dessa forma, facilita para o cliente, que precisará se cadastrar uma única vez, e para o próprio estabelecimento, que não terá a preocupação de manter uma lista dos clientes cadastrados e com sua manutenção, uma vez que os dados não serão mantidos localmente.

O modelo do Pila Fácil apresentado opera a partir de duas perspectivas distintas: a do cliente e a do restaurante. Do ponto de vista do cliente, ao chegar em um restaurante, é necessário inicialmente se identificar por meio de uma chave exclusiva do cliente para acessar as ferramentas disponíveis no sistema, como o cardápio, registro de pedidos e a visualização da conta. Já do ponto de vista do restaurante, o sistema proporciona uma visão geral de todos os clientes, incluindo o registro de pedidos, status das contas e informações sobre produtos e estoque disponíveis.

Durante a modelagem do sistema Pila Fácil, o autor destacou alguns desafios que devem ser enfrentados ao desenvolver um sistema de gerenciamento, dos quais podemos citar [Zimmermann 2010]:

- Segurança: É crucial garantir a segurança dos dados do cliente, uma vez que informações sensíveis como dados pessoais e de pagamento ficam registradas, além de dados financeiros dos restaurantes, entre outros. Além disso, é essencial prevenir possíveis fraudes, como a tentativa de um usuário se passar por outro cliente.
- Alta Disponibilidade do Sistema: O sistema deve estar disponível para todos os clientes a todo momento. Portanto, não deve ser limitado apenas àqueles conectados à rede local, o que restringiria o acesso. Além disso, é essencial prevenir possíveis problemas, como a falta de conexão com a internet, falhas de energia, entre outros imprevistos.
- Localização: Em casos específicos, como em bares ou baladas, é comum que o cliente se desloque pelo espaço. Portanto, um dos desafios do sistema seria como localizar ou manter o registro do cliente para a entrega de um pedido, especialmente quando o cliente não se mantém em uma localidade fixa.

3.2. Sistema de informação para controle de agendamento de tarefas para escritórios de advocacia

O artigo explora o funcionamento de um software, detalhando suas funcionalidades com foco no aprimoramento do controle de agendamentos de prazos processuais em escritórios

de advocacia, além de discutir sua disponibilidade para os usuários[de Castro and Kurtz].

O sistema inclui uma agenda que exibe datas agendadas para clientes ou processos, junto com seus respectivos prazos; recursos para controle financeiro da empresa, monitorando gastos e ganhos; um módulo de gestão de clientes, fornecendo detalhes cadastrais dos clientes; e a capacidade para o usuário armazenar uma variedade de arquivos, como peças processuais, documentos escaneados, entre outros.

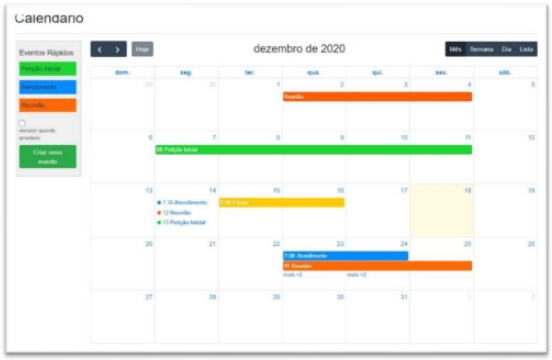


Figure 3. Modelo de Calendário Proposto

A Figura 5 apresenta a agenda proposta pelo artigo, na qual é possível identificar a fase em que o processo se encontra por meio da cor, e a duração dessa etapa do processo é visualizada através da etiqueta que percorre os dias. Para criação desenvolvimento dessa interface foi utilizado o framework *Laravel*. Por fim, durante o desenvolvimento, foram encontrados algumas sugestões para trabalhos futuros, como:

- Propor uma visão para o cliente, onde ele posso ter um ambiente que possa acompanhar o andamento dos seu processos.
- Adicionar mais funcionalidades ao calendário, para dispobilizar mais opções de agendamento e especificidades a cada agendamento.

3.3. Desenvolvimento de um Sistema de Informação Web para gerenciar os processos de um buffet de doces e bombons

O artigo propõe o desenvolvimento e implementação de um sistema de informação web para auxiliar um buffet de doces e bombons a gerenciar seu negócio, com o objetivo de aumentar a eficiência dos processos e resolver os principais desafios enfrentados pela empresa [Santos 2016].

Além de propor o desenvolvimento do programa, o intuito do artigo é descrever cada uma das etapas essenciais do desenvolvimento de software, conforme definido por [Sommerville 2011], utilizando técnicas que visam facilitar o processo de desenvolvimento. Dessa forma, o projeto tem início com a identificação do problema e dos requisitos funcionais e não funcionais, passando por etapas como a modelagem dos modelos e funcionalidades, identificação dos casos de uso, entre outras.

Durante o desenvolvimento do projeto, percebeu-se a necessidade, para projetos futuros,de disponibilizar *software* para acesso remoto, de modo que a propritário possa gerir seus estabelicimento de maneira remota. Além disso, percebeu-se que havia a necessiade de um controle de acessos e autorizações, ou seja, os funcionários não deveriam ter acesso a todos os dados do sistema, a depender de sua posição.

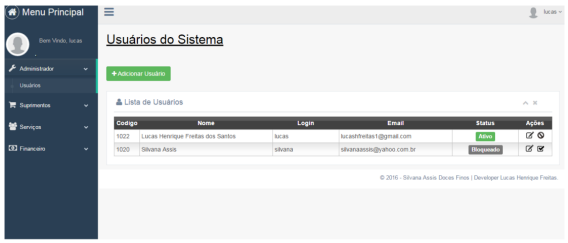


Figure 4. Sistema para gerenciamento de Estabelecimento de Doces e Bolos

3.4. Proposta x Trabalhos Correlatos

Durante este capítulo, foram analisados três trabalhos correlatos, com o objetivo de nos subsidiar no desenvolvimento da aplicação proposta neste artigo. Em [Zimmermann 2010], é proposto um sistema de gerenciamento de pedidos para restaurantes. A ideia de citar este artigo é ter como base como deve ser feita a comunicação do sistema junto às interfaces do cliente e da administração do restaurante. Além disso, o artigo identifica algumas das principais dificuldades durante o desenvolvimento de sistemas de gerenciamento.

Em [de Castro and Kurtz], é proposto um sistema de informação para o gerenciamento de tarefas em escritórios de advocacia, com o objetivo de aprimorar o controle da agenda e das etapas processuais. Portanto, ao citar esse artigo, pretende-se utilizá-lo para auxiliar no desenvolvimento do calendário de agendamento dos solicitantes de visto americano, uma vez que um dos aspectos cruciais do sistema proposto é manter um controle organizado dos agendamentos dos solicitantes.

Em [Santos 2016], é proposto sistema de informação web para auxiliar um buffet de doces e bombons para genciar o negócios, mas muito além do sistema propsoto, esse artigo pode nos auxiliar nas etapas do desenvolvimento de um *software* descritas por [Sommerville 2011] como essencias para um desenvolvimento eficaz.

Por fim, é necessário ressaltar que não existem ainda propostas similares a descrita nesse trabalho, por esse motivos, foram escolhidos três trabalhos que têm como proposta a criação de um sistema de gerenciamento para fins específicos, com objetivo de comparar sua ideais e utilizá-las em nosso projeto.

4. Desenvolvimento do Projeto

Este capítulo apresenta todas as etapas de desenvolvimento do sistema e os artefatos gerados em cada fase.

4.1. Analise e levantamento de requisitos

De acordo com [Barbosa and Silva 2010], as principais técnicas utilizadas para coletar dados e levantar requisitos dos usuários são:

- entrevistas;
- grupos de foco;
- questionários;
- brainstorming de necessidades e desejos dos usuários;
- classificação de cartões (card sorting);

- estudos de campo;
- investigação contextual.

Nesse contexo [Barbosa and Silva 2010] diz que cada técnica é caracterizada quanto ao seu objetivo, suas vantagens e o nível de esforço necessário para sua aplicação. Para nosso projeto utilizamos da técnicas de estudo de campo e a investigação contextual para fazer o levantamento dos requisitos do cliente.

No estudo de campo temos como objetivo entender o comportamento do usuário final no contexto de seu própio ambiente de atuação [Courage and Baxter 2005], enquanto a investigação contextual busca obter dados sobre a estrutura do trabalho na prática e conhecer os detalhes do trabalho que se tornam habituais e invisíveis.

Durante um período de seis meses fizemos parte da equipe do escritório de assessoria para entender sobre o processo de trabalho e a partir da analíse do trabalho definimos os requisitos do sistema, que foram divididos em dois tipos: funcionais e não funcionais.

4.1.1. Requisitos Funcionais

De acordo com [Sommerville 2011], os requisitos funcionais explicitam os serviços que o sistema deve fornecer, como o sistema deve reagir a entradas e como deve se comportar em determinadas situações. A seguir, na Tabela 2, mostramos as funcionalidades que o software deve solucionar com base nos problemas citados na Seção 3.1, onde foram descritas cinco requisitos funcionais, sendo a coluna "*Nome do Requisito*" descritos o nome do requisito, na coluna "*Descrição*" está descrito a funcionalidade desse requisito, e por fim, na coluna "*Prioridade*" é apresentado a prioridado do requisito para o sistema.

Table 1. Requisitos Funcionais

NOME DO REQUISITO	DESCRIÇÃO	PRIORIDADE
RF001 - Gerenciar a entrada de novos processos	O sistema deve iniciar novos processos com base nas informações fornececidas.	OBRIGATÓRIO
RF002 - Gerenciar os agendamentos dos solicitantes	O sistema deve manter atualizados os dados sobre os agendamentos e realizar a antecipação dos agendamento conforme a necessidade.	OBRIGATÓRIO
RF003 - Gerenciar a comunição com o solitantes	O sistema deve fornecer os dados cruciais aos clientes como data e local do agendamento.	OBRIGATÓRIO
RF004 - Visualizar a planilha de clientes	O sistema deve fornecer a visualização dos dados dos clientes, como as datas de seus agendamentos.	OBRIGATÓRIO
RF005 - Gerenciamento Financeiro	O sistema deve fornecer o valor cobrado ao cliente e o saldo pago pelo mesmo	OBRIGATÓRIO

4.1.2. Requisitos Não Funcionais

Em [Sommerville 2011] é dito que os requisitos não funcionais são aqueles que não são diretamente relacionados ás funções específicas forncecidas pelo sistema, sendo aplicados frequentemente ao sistema como um todo. A seguir, na Tabela 3, é apresentado os requisitos não funcionais do sistema, onde na coluna "*Código*" é descrito o código do requisito não funcional, na coluna "*Descrição*" está descrito a restrição referente ao requisito e, por fim, na coluna "*Categoria*" é descrita a categoria referente a essa descrição.

Table 2. Requisitos Não Funcionais

CÓDIGO	DESCRIÇÃO	CATEGORIA
RNF001	O sistema deve ser acessado via navegador.	PORTABILIDADE
RNF002	O sistema deve ser desenolvido nas liguagens de programação Javascript, HTML e PHP.	ARQUITETURA
RNF003	O sistema deve estar disponivel para apenas para acesso local.	DISPONIBILIDADE
RNF004	O sistema deve ser adaptado para usar apenas em computadores.	PORTABILIDADE
RNF005	O sistema deve ter boa fluidez, de modo a não prejudicar a experiência do usuário.	DESEMPENHO
RNF006	O sistema deve ser adaptado a tratar possíveis falhas ao acesso de informações, mantendo copias atualizadas dos dados.	CONFIABILIDADE

4.1.3. Validação de Requisitos

Após a realização do levantamento de requisitos, foi definido um documento com objetivo de especificar os requisitos e as funcionalidades do sistema para a proprietária do estabelecimento. O documento, anexado na apêndice A desse trabalho, específica todos os requisitos e condições para desenvolvimento do sistema, que já foi aprovado pela proprieária.

4.2. Módulos e Funcionalidades

Após a definição de todos os requisitos funcionais e não funcionais serem aprovadas, começou a ser definida a estrutura do sistema com os módulos e funcionalidades.

4.2.1. Módulo Administrador

O módulo administrador conterá os serviços que permitiram gerenciar os assessores que irão acessar as demais funionalidades do sistema. O modulo possuirá a seguinte funcionalidade.

- **FA01 - Gerenciar Assessores** - A funcionalidade deve permitir a criação dos usuários que vão ser autorizados acessar as funcionalidades do sistema.

4.2.2. Módulo Calendário

O módulo cliente é responsável pelos serviços que permiram a empresa gerenciar todos os dados e os agendamentos dos solicitantes do visto americano. O módulo possuirá as seguintes funcionalidades.

- **FA02 - Gerenciar Agendamentos** - A funcionalidade deve permitir atualizar as informações referente as datas de agendamento dos clientes, além ser responsavél por realizar a antecipação dos clientes, conforme as datas disponíveis e as informações fornecidas.

4.2.3. Módulo Assessor

O módulo assessor é responsável pelo cadastro de solicitantes na plaforma, além de ser responsável pelas informações financeiras de cada solicitante. O módulo possuirá as seguintes funcionalidades:

- **FA03 - Gerenciar Solicitantes** - A funcionalidade deve permitir criar novos clientes, alterar dados dos clientes ou excluir um cliente.
- **FA04 - Gerenciar Pagamentos** - A funcionalidade deve permitir que o usuário insira as informações sobre os valores pagos, os valores cobrados e os valores devidos pelos solicitantes.
- **FA05 - Gerenciar Comunicação** - A funcionalidade deve permitir a criação de um canal de comunicação entre o solicitante e o usuário para provir informações sobre seus agendamentos.

4.2.4. Módulo Formulário

O módulo formulário é responsável pelos preenchimentos dos formulários oficiais. O módulo deve conter as seguintes funcionalidades:

- **FA06 - Gerenciar Abertura de Processo** - A funcionalidade deve permitir a criação de novos processos de visto americano junto ao consulado.

4.3. Casos de Uso

Conforme dito em [Pressman and Maxim 2016], a partir da reunião dos requisitos, uma visão geral das funções e características começam a se materializar. Entretando, é difícil passar para atividades mais técnicas, da engenharia de software até que se entenda como tais características serão usadas por diferentes classes de usuários. Com isso, os desenvolvedores e usuários podem criar um conjunto de cenários que identifique um roteiro de uso para sistema, esses cenários também são conhecidos como *casos de uso*. A seguir, na Figura 5, será apresentando o diagrama de casos de uso do sistema.

Em seguida foi criado um documento onde são descritos de maneira detalhada os casos de uso para cada funcionalidade descritos na Seção 6.2, conforme os códigos descritos na Tabela 4. Nesse documento são descritos todo o contexto e o fluxo de interações e restriçoes de cada funcionalidade, que pode ser encontrados na Sessão Apêndice B deste trabalho.

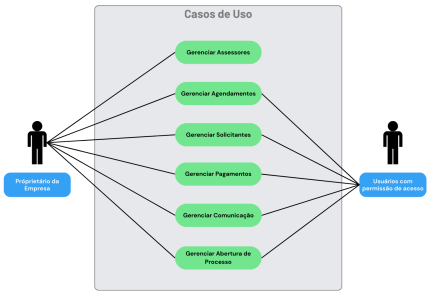


Figure 5. Casos de Uso

Table 3. Casos de Uso Detalhados

CÓDIGO	FUNCIONALIDADE
UC001	Gerenciar Assessores
UC002	Gerenciar Agendamentos
UC003	Gerenciar Solicitantes
UC004	Gerenciar Pagamentos
UC005	Gerenciar Comunicação
UC006	Gerenciar Abertura de Processos

4.4. Prototipação

Após realizarmos a elaboração dos casos de uso detalhados de todas as funcionalidades, iniciamos a fase da prototipação. Como dito em [Pressman and Maxim 2016], a prototipação é um modelo de processo isolado que tem como objetivo auxiliar os envolvidos a compreender melhor o está para ser contruído.A prototipação se concetra em ser uma representação dos aspectos do *software* que serão visivéis ao usuário, que leva a contrução de um protótipo que é entre e avaliado pelos envolvidos.

Desse modo, no Apêndice C desse trabalho é apresentando protótipos de interface criados e aprovados pelos envolvidos. Para criação foi utilizado a ferramenta de prototipapação Pencil que é uma ferramenta gratuita que permite projetar interface de diversos tipos de sistemas.

4.5. Banco de Dados

Após definidos todos os requisitos e os protótipos da interface, foi realizado a construção do banco de dados das funcionalidade dos sistemas. O fato de termos um compreensão adequada do projeto, facilitou no desenvolvimentos do banco de dados do sistema. Para desenvolver o banco de dados utilizamos a ferramenta *SQL Server Management Studio*, que possui um recurso que permite que os diagramas lógicos sejam criados de maneira visual para posteriormente serem convertidos em tabelas no servidor de banco de dados.

A seguir, na Figura 6, é mostrado o diagrama lógico do banco de dados do sistema, contendo um total de cinco tabelas necessárias para construir o sistema. A tabela *Tabela Assessores* refere-se aos assessores cadastrados no sistema. Nela, é importante conter o nome do assessor, o login para acessar o sistema, a senha de acesso, o email para recuperação da senha e um campo que indica se ele é ou não o administrador do sistema. A *Tabela Clientes* é a tabela principal do banco de dados, pois contém o registro de todos os clientes da empresa. Além disso, essa tabela se relaciona com a *Tabela Pagamentos*,

que é responsável por manter os registros dos pagamentos, a *Tabela Contato*, que mantém os registros dos contatos dos clientes, e a *Tabela Cidade*, que mantém os locais onde estão abrigados os consulados americanos, com o intuito de minimizar a repetição de dados. Por fim, a *Tabela Clientes* se relaciona também com a própria *Tabela Assessores*, visto que cada cliente deve possuir o assessor responsável.

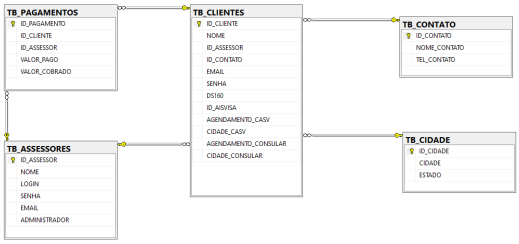


Figure 6. Diagrama Lógico do Banco de Dados

4.6. Codificação

Após a construção inicial do banco de dados do sistema, iniciamos a fase de implementação, utilizando como base os casos de uso detalhados nas etapas anteriores para definir as restrições que o sistema deveria seguir e os fluxos de interação com os usuários. Além disso, os protótipos de interface serviram como modelos para o desenvolvimento das interfaces da aplicação, visto que todos os elementos de cada tela já haviam sido projetados previamente.

Outro fator importante, por se tratar de um projeto incremental, é que conforme avanços no desenvolvimento foram realizados nesta fase de codificação, também foram definidas algumas alterações no projeto inicial previsto junto ao cliente. As seguintes mudanças foram realizadas em comparação ao projeto inicial:

- Referente ao Caso de Uso UC02 - Gerenciar os Agendamentos, foi definido junto ao cliente que não haveria a necessidade de realizar a antecipação e o cancelamento dos agendamentos, devido ao fato de que o sistema de agendamentos possui um mecanismo de segurança contra aplicações web, o que poderia acarretar em bloqueio do processo e, consequentemente, perda financeira.
- Referente ao Caso de Uso UC06 - Gerenciar Aberturas de Processos, foi definido que a abertura de processos deve ser feita pelo usuário, pelos mesmos motivos mencionados no primeiro item desta seção, visto que pode ocasionar prejuízos financeiros. Assim, o sistema manterá os dados essenciais do usuário para a realização do processo.

4.7. Testes e Validação

De acordo com [Sommerville 2011], uma das principais vantagens de usar o desenvolvimento incremental é permitir que as funcionalidades sejam testadas à medida que ficam prontas, descartando a necessidade de esperar o sistema ficar completamente pronto. Além disso, isso permite que o sistema seja testado diversas vezes, visto que o desenvolvimento de uma funcionalidade pode acarretar mudanças no desenvolvimento inicial de outra funcionalidade.

Os testes foram realizados antes de promover a entrega de cada funcionalidade ao cliente para a realização da etapa da validação, que tem como objetivo analisar se o sistema está suprindo as necessidades do cliente. Como dito [Sommerville 2011], a etapa da validade é definida como um série de atividades que são executas para verificar se o sistema atende a todos os requisitos do cliente.

4.7.1. Testes Exploratórios

Para a realização dos testes do sistema, foram realizados testes exploratórios, que são testes manuais com o objetivo de seguir o fluxo de interações de cada funcionalidade prevista nas etapas anteriores. Durante esses testes, os cenários de uso foram repetidos diversas vezes para verificar se o sistema se comportava conforme o esperado.

Os testes exploratórios são uma abordagem essencial, pois permitem a identificação de problemas e inconsistências que podem não ser detectados. Ao seguir os fluxos de interação previstos, os teste puderam explorar o sistema de maneira intuitiva e identificar comportamentos inesperados, falhas de usabilidade e possíveis bugs.

4.7.2. Validação

A validação foi realizada por meio de reuniões. Após a conclusão de cada funcionalidade do projeto, era organizada uma reunião com o cliente para validar a nova funcionalidade, verificando se atendia às expectativas. Desse modo, a cada reunião poderiam ser definidas melhorias a serem implementadas no próximo escopo de desenvolvimento. Se a funcionalidade fosse aprovada por completo, ela era considerada finalizada. Após a entrega da última funcionalidade, era feito um encontro final para apresentar o sistema geral para validação e, se necessário, definir as últimas mudanças a serem realizadas antes da entrega final.

4.8. Implementação

Após o desenvolvimento e a validação, o sistema foi implementando para utilização na empresa, porém foi decidido que o sistema vai passar por um período de experiência, onde o sistema será utilizado como sistema principal, porém o sistema anterior continurá a ser usado, que é baseado em planilhas, como um mecanismo de segurança para manter as informação caso seja verificado algum erro durante a utilização do novo sistema.

5. Sistema de Gerenciamento do Visto Americano

Nesse capítulo são apresantados as interfaces do sistema juntamente com as principais funcionalidades. Cada sessão será dividada conforme suas funcionalidades.

5.1. Login e Recuperação de Senha

Quando o usuário acessa o sistema, ele se depara com a *Tela de Login* (Fig.7), nesse caso ele deve colocar suas credencias no sistema para acessar o sistema.

Caso o usuário tenha esquicido das informações de login ou senha, ele tem a opção de realizar a recuperação da sua senha, nesse caso ele irá acessar *Tela de Recuperação*

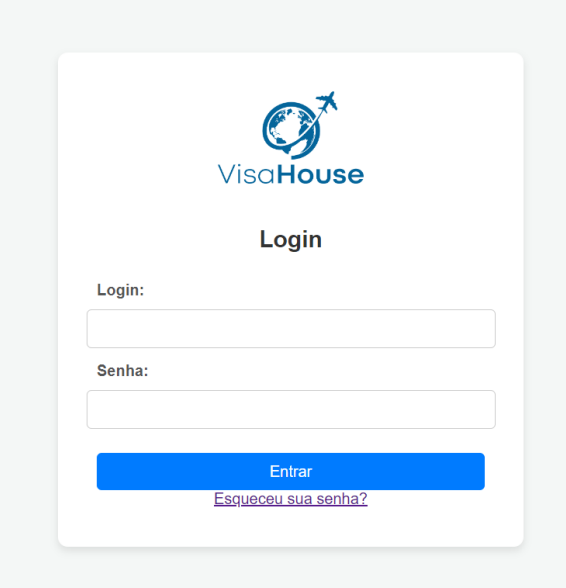


Figure 7. Tela de Login

de Senha (Fig. 8), onde ele deve entrar com o email cadastrado para recuperar as informações.

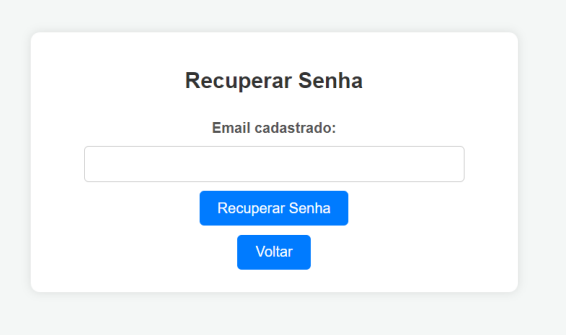


Figure 8. Tela de Recuperação de Senha

Se obtiver êxito na verificação, ele será levado a *Tela de REcuperação de Dados do Assessor* (Fig. 9), onde poderá observar todos os dados do seu cadastro.

5.2. Menu Principal e Gerenciamento de Assessores

Ao realizar o login, o usuário será levado para o menu principal (Fig. 10), na *Tela Menu Principa*l são apresentados todas as funcionalidades. Inclusive, duas funcionalidade que são presente apenas aos administradores, que são de criação de assessores (Fig. 11) e de exclusão de assessores (Fig.12). Uma confirguração importante dessas funcionalidades se deve ao fato que o administrador pode criar novos administradores, porém os administradores não podem ser excluídos do sistema.

Além disso, no *Menu Principal*, (Fig. 10), é apresentado a tabela de clientes, com todas as informações importantes sobre cada um dos clientes, onde, através da opção *Ordenar*, é possível realizar três tipos de ordenação: por nome, por agendamento na

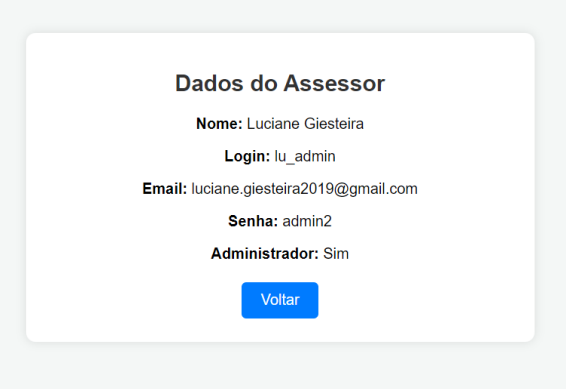


Figure 9. Recuperação de Dados do Assessor

CASV e por agendamento no consulado. Por fim, através opção de Atualizar, é possível realizar a atualização automática das informações dos agendamentos dos clientes, onde o sistema acessa individualmente cada cadastro e verifica as informações sobre as datas da agendamentos e as localidades.

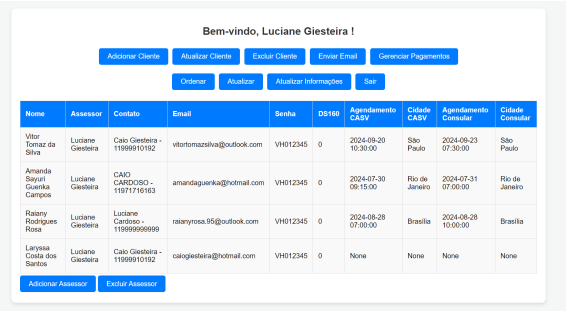


Figure 10. Menu Principal

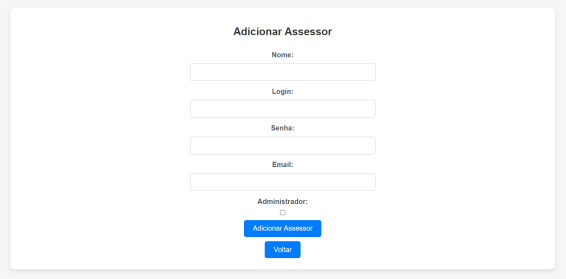


Figure 11. Adicionar Assessor

5.3. Gerenciar Clientes

Nesse modúlo são apresentados todas as manipulações que podem ser realizados sobre os clientes. Na Fig. 13, está sendo mostrado a Tela de Criação de Novos Clientes, onde devemos entrar com os dados dos clientes. Nas Fig. 14 e Fig. 15, respectivamente, são apresentados o menu *drop-down* dos clientes, onde devemos escolher qual cliente desejamos modificar e a página de edição dos dados do clientes, que devem ser salvos quando finalizadas as alterações. Por fim, na Fig. 16, é apresentado o menu para excluir um cliente, para isso, basta o usuario selecionar o cliente e selecionar o botão excluir.

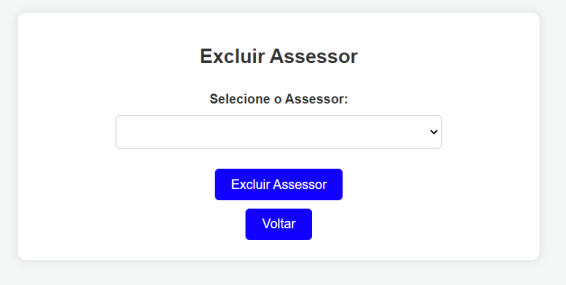


Figure 12. Excluir Assessor

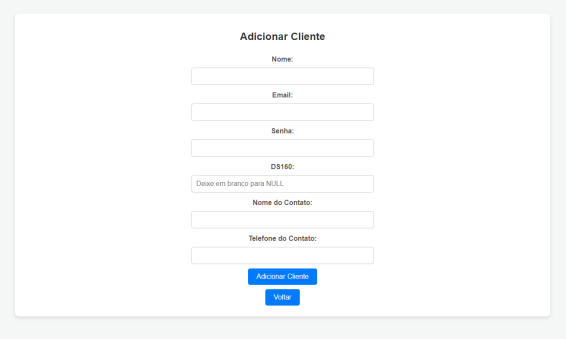


Figure 13. Criação de Clientes

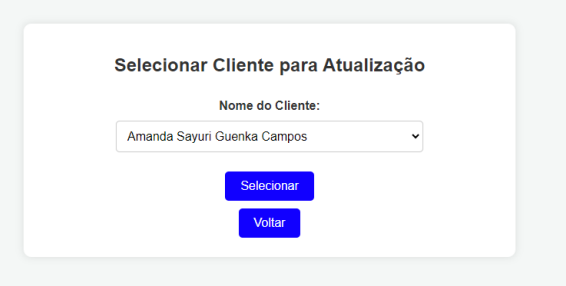


Figure 14. Selecionar Cliente para Edição

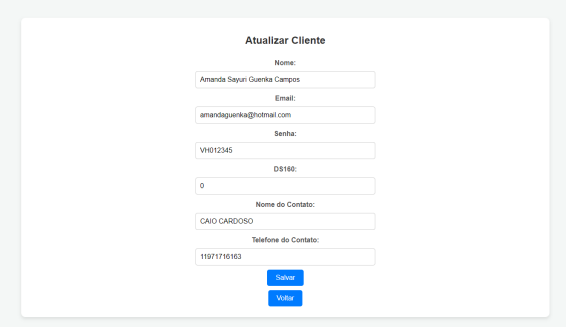


Figure 15. Editar Cliente

5.4. Gerenciamento de Comunicação

Nessa sessão vamos apresentar o mecanismo do comunicação proposto, que seria através funcionalidade *Enviar E-mail*, Fig. 17, onde o usuário pode enviar email ao cliente com

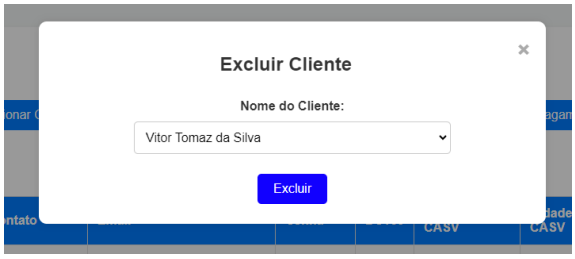


Figure 16. Excluir Cliente

base no email cadastrado, ou podendo alterar o email para qual deseja enviar. O usuário deve incluir o título da mensagem, o corpo da mensagem e também se necessário, anexar até no máximo dois documentos.

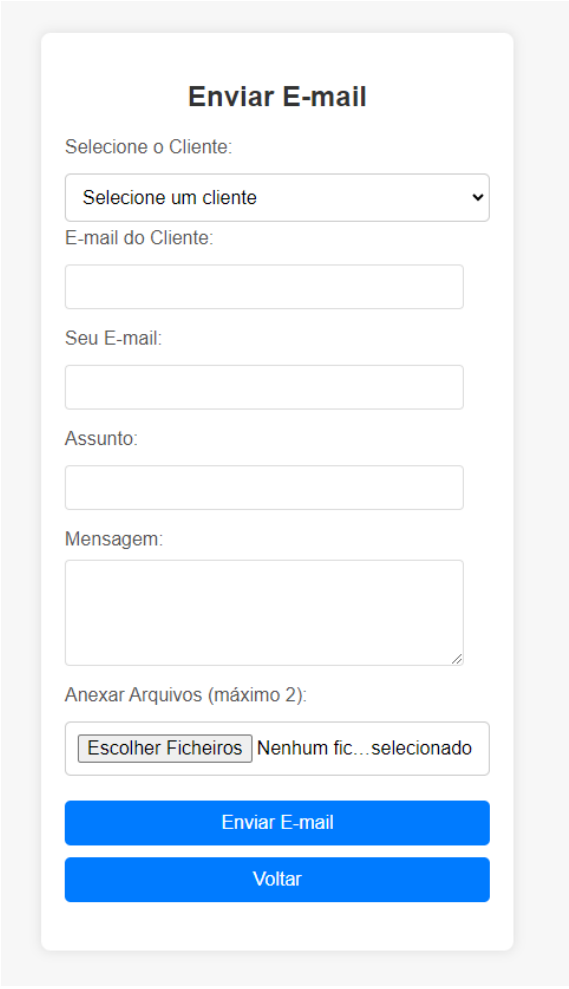


Figure 17. Enviar Email

5.5. Gerencimento de Pagamentos

Nessa sessão vamos apresentar os mecanismos de gerenciamento de pagamentos, onde apresentaremos o menu para criação de pagamentos, Fig. 18, onde o usuário deve selecionar o cliente e cadastrar o pagamento, Fig. 19 , caso esse cliente já possua um

pagamento, o sistema emite um alerta informando que esse cliente já tem um pagamento cadastrado, nesse caso, o usuário pode editar o pagamento desse cliente através do menu de edição de pagamento, Fig. 20, onde serão apresentados os dados do último registro de pagamento.

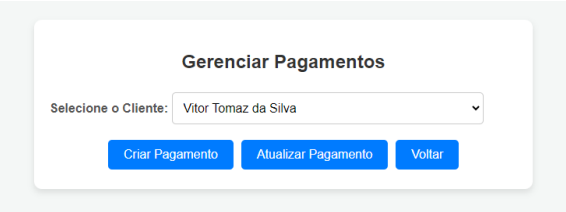


Figure 18. Gerenciar Pagamento

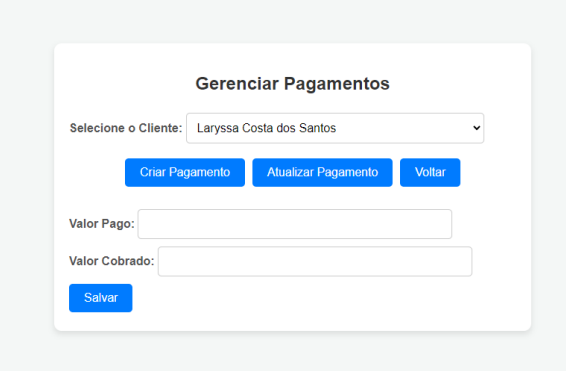


Figure 19. Criar Pagamento

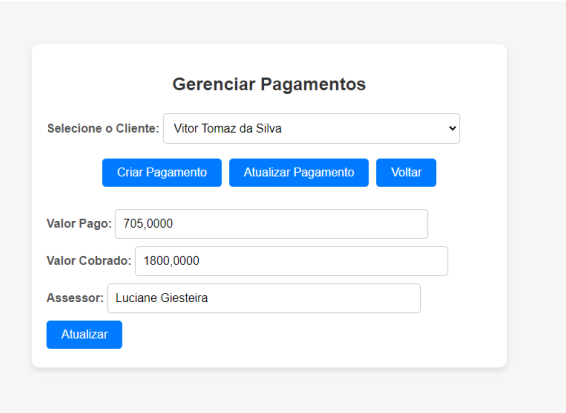


Figure 20. Editar Pagamento

5.6. Editar Informações Pessoais do Assessor

Por fim, temos o menu de edição das informações do assessor, Fig. 21, que está conectado, onde é possível alterar qualquer informação desejada de cadastrado, incluindo cadastrar um nova senha.

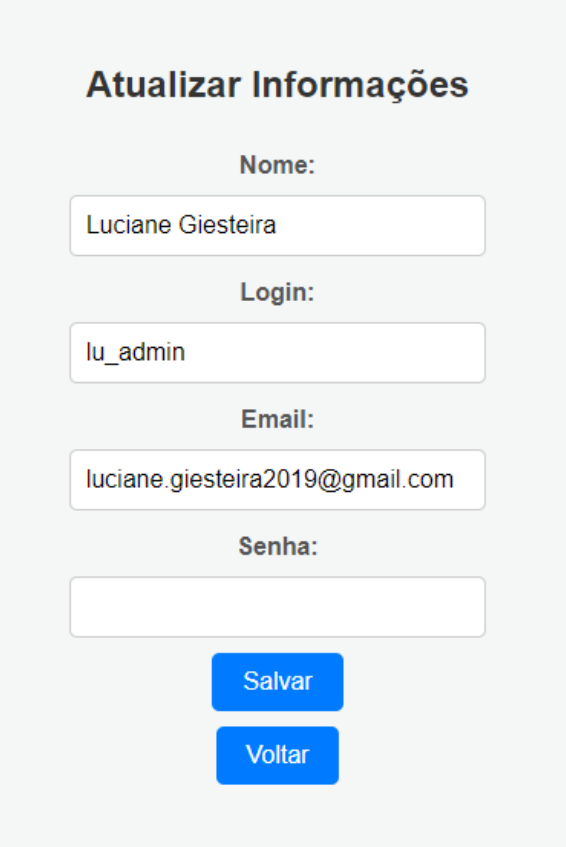


Figure 21. Edição de Informações do Usuário

6. Conclusão e Trabalhos Futuros

A assessoria de Visto Americano analisada durante o projeto possuía diversos desafios na organização de seu negócio e, por esse motivo, buscou uma solução digital personalizada. Atualmente, pequenas empresas estão buscando esse tipo de solução para potencializar o seu negócio, aumentando a eficácia dos processos.

Durante o desenvolvimento deste trabalho, a possibilidade de estar presente diariamente no ambiente de funcionamento do negócio facilitou a compreensão dos problemas e a identificação das necessidades para a aplicação do sistema, visando o aprimoramento das tarefas da empresa.

Os objetivos gerais e específicos do trabalho foram alcançados, pois os requisitos especificados foram atendidos durante o desenvolvimento. O controle sobre os dados dos clientes ficou mais simples, a atualização dos dados foi automatizada, permitindo que os usuários não percam tempo valioso com essa tarefa, e a comunicação entre os clientes e os assessores foi simplificada. Além disso, agora é possível ter um certo controle financeiro.

Por fim, para trabalhos futuros, seria ideal expandir o sistema para outros tipos de vistos, verificar a possibilidade de disponibilizar um calendário que gere avisos sobre o andamento dos processos dos clientes e, finalmente, tornar o sistema disponível para acesso web, criando também um perfil para o cliente, de modo que ele possa acompanhar o andamento de seu processo de maneira clara.

References

Assessoria visto americano: entenda como funciona! https://www4.mundodosvistos.com.br/p/quando-contratar-assessoria-para-visto-americano/. (Accessed on 01/07/2024).

Eua registram recorde de mais de 1,1 milhão de vistos emitidos para brasileiros em 2023. https://oglobo.globo.com/mundo/noticia/2024/01/05/eua-registram-recorde-de-mais-de-11-milhao-de-vistos.ghtml. (Acessado em 16/03/2024).

Procura por consultoria de vistos e cidadania dispara após vitória de lula. https://www.correiobraziliense.com.br/brasil/2022/11/5050079-procura-por-consultoria-de-vistos-e-cidadania-cresce.html. (Acessado em 16/03/2024).

Vistos - embaixada e consulados dos eua no brasil. https://br.usembassy.gov/pt/visas-pt/. (Accessed on 03/17/2024).

Barbosa, S. and Silva, B. (2010). *Interação Humano-Computador*. Elsevier Brasil.

Courage, C. and Baxter, K. (2005). *Understanding Your Users: A Practical Guide to User Requirements Methods, Tools, and Techniques*. Interactive Technologies. Elsevier Science.

de Castro, A. H. C. and Kurtz, G. C. Sistema de informação para controle de agendamento de tarefas para escritórios de advocacia.

Pressman, R. and Maxim, B. (2016). *Engenharia De Software: UMA ABORDAGEM PROFISSIONAL*. MCGRAW HILL - ARTMED.

Santos, L. H. F. d. (2016). Desenvolvimento de um sistema de informação web para gerenciar os processos de um buffet de doces e bombons.

Sommerville, I. (2011). *Engenharia de Software*. Pearson.

Zimmermann, A. F. (2010). Pila fácil : sistema de gerenciamento de pedidos. https://lume.ufrgs.br/handle/10183/28336?show=full. (Accessed on 04/06/2024).

ANEXO B – CÓDIGO FONTE

Nessa sessão serão apresentaods os códigos fontes desenvolvidos durante o projeto.

B.1 APP.PY

```python
from asyncio.windows_events import NULL
from flask import Flask, render_template, request, redirect, url_for, flash,
session
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.exc import SQLAlchemyError
import pyodbc
from sqlalchemy.orm import aliased
from functools import wraps
import os
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import pyodbc
import time
from datetime import datetime
import re
from selenium.webdriver.chrome.options import Options
from flask_mail import Mail, Message
from werkzeug.utils import secure_filename


app = Flask(__name__)
app.secret_key = 'Ccgls1021#'
app.config['SQLALCHEMY_DATABASE_URI'] =
'mssql+pyodbc://sa:VisaHouse012345@LAPTOP-
8VC0L6D2/VISTOAMERICANO?driver=ODBC+Driver+17+for+SQL+Server'
db = SQLAlchemy(app)

# Configuração do Flask-Mail
app.config['MAIL_SERVER'] = 'smtp-mail.outlook.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USE_SSL'] = False
app.config['MAIL_USERNAME'] = 'tcc-exemplo-ufsc@outlook.com'
app.config['MAIL_PASSWORD'] = 'VH012345'
app.config['MAIL_DEFAULT_SENDER'] = 'tcc-exemplo-ufsc@outlook.comm'

mail = Mail(app)


# Pasta para salvar os arquivos enviados
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

# Configuração da conexão com o banco de dados
conn = pyodbc.connect(
```

```
    'DRIVER={ODBC Driver 17 for SQL Server};'
    'SERVER=LAPTOP-8VC0L6D2;'
    'DATABASE=VISTOAMERICANO;'
    'UID=sa;'
    'PWD=VisaHouse012345'
)

class Assessor(db.Model):
    __tablename__ = 'TB_ASSESSORES'
    ID_ASSESSOR = db.Column(db.Integer, primary_key=True)
    NOME = db.Column(db.String(40), nullable=False)
    LOGIN = db.Column(db.String(20), nullable=False)
    SENHA = db.Column(db.String(10), nullable=False)
    EMAIL = db.Column(db.String(40))
    ADMINISTRADOR = db.Column(db.Boolean, nullable=False)

class Cliente(db.Model):
    __tablename__ = 'TB_CLIENTES'
    ID_CLIENTE = db.Column(db.Integer, primary_key=True)
    NOME = db.Column(db.String(40), nullable=False)
    ID_ASSESSOR = db.Column(db.Integer,
db.ForeignKey('TB_ASSESSORES.ID_ASSESSOR'), nullable=False)
    ID_CONTATO = db.Column(db.Integer, nullable=True)
    EMAIL = db.Column(db.String(40), nullable=True)
    SENHA = db.Column(db.String(10), nullable=False)
    DS160 = db.Column(db.String(10), nullable=True)
    ID_AISVISA = db.Column(db.String(50), nullable=True)
    AGENDAMENTO_CASV = db.Column(db.DateTime, nullable=True)
    CIDADE_CASV = db.Column(db.Integer, nullable=True)
    AGENDAMENTO_CONSULAR = db.Column(db.DateTime, nullable=True)
    CIDADE_CONSULAR = db.Column(db.Integer, nullable=True)

class Contato(db.Model):
    __tablename__ = 'TB_CONTATO'
    ID_CONTATO = db.Column(db.Integer, primary_key=True)
    NOME_CONTATO = db.Column(db.String(20), nullable=True)
    TEL_CONTATO = db.Column(db.Integer, nullable=True)

@app.route('/')
def index():
    return render_template('login.html')

# Verifica se é o administrador
def admin_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'logged_in' not in session or not session.get('is_admin'):
            return redirect(url_for('login'))
```

```
        return f(*args, **kwargs)
    return decorated_function

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        login = request.form['login']
        senha = request.form['senha']

        cursor = conn.cursor()
        cursor.execute("SELECT * FROM TB_ASSESSORES WHERE LOGIN = ? AND SENHA
= ?", (login, senha))
        assessor = cursor.fetchone()

        if assessor:
            session['logged_in'] = True
            session['user_id'] = assessor.ID_ASSESSOR
            session['user_name'] = assessor.NOME
            session['is_admin'] = assessor.ADMINISTRADOR
            return redirect(url_for('dashboard'))
        else:
            flash('Login ou senha incorretos')
            return redirect(url_for('index'))


    return render_template('login.html')

@app.route('/dashboard')
def dashboard():
    if 'logged_in' not in session:
        return redirect(url_for('login'))

    cursor = conn.cursor()
    query = """
        SELECT
            C.ID_CLIENTE, C.NOME, A.NOME AS NOME_ASSESSOR, CO.NOME_CONTATO,
CO.TEL_CONTATO, C.EMAIL,
            C.SENHA, C.DS160, C.ID_AISVISA, C.AGENDAMENTO_CASV,
CIDADE_CASV.CIDADE AS CIDADE_CASV,
            C.AGENDAMENTO_CONSULAR, CIDADE_CONSULAR.CIDADE AS CIDADE_CONSULAR
        FROM
            TB_CLIENTES C
            JOIN TB_ASSESSORES A ON C.ID_ASSESSOR = A.ID_ASSESSOR
            LEFT JOIN TB_CONTATO CO ON C.ID_CONTATO = CO.ID_CONTATO
            LEFT JOIN TB_CIDADE CIDADE_CASV ON C.CIDADE_CASV =
CIDADE_CASV.ID_CIDADE
            LEFT JOIN TB_CIDADE CIDADE_CONSULAR ON C.CIDADE_CONSULAR =
CIDADE_CONSULAR.ID_CIDADE
    """
```

```
    cursor.execute(query)
    clientes = cursor.fetchall()

    return render_template('dashboard.html', clientes=clientes)

@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('index'))

@app.route('/refresh')
def refresh():
    return redirect(url_for('dashboard'))

@app.route('/sort/<criteria>')
def sort(criteria):
    # Ordena por nome
    if criteria == 'nome':
        cursor = conn.cursor()
        query = """
            SELECT
                C.ID_CLIENTE, C.NOME, A.NOME AS NOME_ASSESSOR,
CO.NOME_CONTATO, CO.TEL_CONTATO, C.EMAIL,
                C.SENHA, C.DS160, C.ID_AISVISA, C.AGENDAMENTO_CASV,
CIDADE_CASV.CIDADE AS CIDADE_CASV,
                C.AGENDAMENTO_CONSULAR, CIDADE_CONSULAR.CIDADE AS
CIDADE_CONSULAR
            FROM
                TB_CLIENTES C
                JOIN TB_ASSESSORES A ON C.ID_ASSESSOR = A.ID_ASSESSOR
                LEFT JOIN TB_CONTATO CO ON C.ID_CONTATO = CO.ID_CONTATO
                LEFT JOIN TB_CIDADE CIDADE_CASV ON C.CIDADE_CASV =
CIDADE_CASV.ID_CIDADE
                LEFT JOIN TB_CIDADE CIDADE_CONSULAR ON C.CIDADE_CONSULAR =
CIDADE_CONSULAR.ID_CIDADE ORDER BY C.NOME

        """
        cursor.execute(query)
        clientes = cursor.fetchall()
    elif criteria == 'casv':
        # Ordena por agendamento na CASV
        cursor = conn.cursor()
        query = """
            SELECT
                C.ID_CLIENTE, C.NOME, A.NOME AS NOME_ASSESSOR,
CO.NOME_CONTATO, CO.TEL_CONTATO, C.EMAIL,
                C.SENHA, C.DS160, C.ID_AISVISA, C.AGENDAMENTO_CASV,
CIDADE_CASV.CIDADE AS CIDADE_CASV,
```

```
                C.AGENDAMENTO_CONSULAR, CIDADE_CONSULAR.CIDADE AS
CIDADE_CONSULAR
            FROM
                TB_CLIENTES C
                JOIN TB_ASSESSORES A ON C.ID_ASSESSOR = A.ID_ASSESSOR
                LEFT JOIN TB_CONTATO CO ON C.ID_CONTATO = CO.ID_CONTATO
                LEFT JOIN TB_CIDADE CIDADE_CASV ON C.CIDADE_CASV =
CIDADE_CASV.ID_CIDADE
                LEFT JOIN TB_CIDADE CIDADE_CONSULAR ON C.CIDADE_CONSULAR =
CIDADE_CONSULAR.ID_CIDADE ORDER BY C.AGENDAMENTO_CASV

        """
        cursor.execute(query)
        clientes = cursor.fetchall()
    else:
        cursor = conn.cursor()
        # Ordena por agendamento no consulado
        query = """
            SELECT
                C.ID_CLIENTE, C.NOME, A.NOME AS NOME_ASSESSOR,
CO.NOME_CONTATO, CO.TEL_CONTATO, C.EMAIL,
                C.SENHA, C.DS160, C.ID_AISVISA, C.AGENDAMENTO_CASV,
CIDADE_CASV.CIDADE AS CIDADE_CASV,
                C.AGENDAMENTO_CONSULAR, CIDADE_CONSULAR.CIDADE AS
CIDADE_CONSULAR
            FROM
                TB_CLIENTES C
                JOIN TB_ASSESSORES A ON C.ID_ASSESSOR = A.ID_ASSESSOR
                LEFT JOIN TB_CONTATO CO ON C.ID_CONTATO = CO.ID_CONTATO
                LEFT JOIN TB_CIDADE CIDADE_CASV ON C.CIDADE_CASV =
CIDADE_CASV.ID_CIDADE
                LEFT JOIN TB_CIDADE CIDADE_CONSULAR ON C.CIDADE_CONSULAR =
CIDADE_CONSULAR.ID_CIDADE ORDER BY C.AGENDAMENTO_CONSULAR

        """
        cursor.execute(query)
        clientes = cursor.fetchall()
    return render_template('dashboard.html', clientes=clientes)

# Recuperação de Senha
@app.route('/recover_password', methods=['GET', 'POST'])
def recover_password():
    if request.method == 'POST':
        email = request.form['email']
        assessor = conn.execute('SELECT * FROM TB_ASSESSORES WHERE EMAIL = ?',
(email,)).fetchone()
        if assessor:
            return render_template('assessor_details.html', assessor=assessor)
```

```
        else:
            flash('Email não encontrado.')
    return render_template('recover_password.html')

# Atualização de Cliente
@app.route('/update-client', methods=['GET', 'POST'])
def update_client():
    if 'logged_in' not in session:
        return redirect(url_for('login'))

    cursor = conn.cursor()

    if request.method == 'POST':
        selected_name = request.form.get('selected_name')
        cursor.execute("SELECT * FROM TB_CLIENTES WHERE NOME = ?",
selected_name)
        cliente = cursor.fetchone()

        cursor.execute("SELECT * FROM TB_CONTATO WHERE ID_CONTATO = ?",
cliente.ID_CONTATO)
        contato = cursor.fetchone()

        return render_template('update_client.html', cliente=cliente,
contato=contato)

    cursor.execute("SELECT NOME FROM TB_CLIENTES ORDER BY NOME")
    clientes = cursor.fetchall()
    return render_template('select_client.html', clientes=clientes)

# Salva a atualização do cliente
@app.route('/save-client', methods=['POST'])
def save_client():
    if 'logged_in' not in session:
        return redirect(url_for('login'))

    cliente_id = request.form.get('ID_CLIENTE')
    nome = request.form.get('NOME')
    email = request.form.get('EMAIL')
    senha = request.form.get('SENHA')
    nome_contato = request.form.get('NOME_CONTATO')
    tel_contato = request.form.get('TEL_CONTATO')
    ds160 = request.form['DS160'] if request.form['DS160'] else None

    cursor = conn.cursor()
    cursor.execute("""
        UPDATE TB_CLIENTES
        SET NOME = ?, EMAIL = ?, SENHA = ?, DS160 = ?
        WHERE ID_CLIENTE = ?
```

```
    """, (nome, email, senha, ds160, cliente_id))

    cursor.execute("""
        UPDATE TB_CONTATO
        SET NOME_CONTATO = ?, TEL_CONTATO = ?
        WHERE ID_CONTATO = (SELECT ID_CONTATO FROM TB_CLIENTES WHERE
ID_CLIENTE = ?)
    """, (nome_contato, tel_contato, cliente_id))

    conn.commit()

    return redirect(url_for('dashboard'))

# Criação de um novo cliente
@app.route('/add-client', methods=['GET', 'POST'])
def add_client():
    if 'user_name' not in session:
        return redirect(url_for('login'))

    if request.method == 'POST':
        nome = request.form['NOME']
        email = request.form['EMAIL']
        senha = request.form['SENHA']
        nome_contato = request.form['NOME_CONTATO']
        tel_contato = request.form['TEL_CONTATO']
        ds160 = request.form['DS160'] if request.form['DS160'] else NULL

        # Obtém o ID do assessor logado
        id_assessor = session['user_id']

        cursor = conn.cursor()

        # Verifica se o contato já existe
        cursor.execute("SELECT ID_CONTATO FROM TB_CONTATO WHERE TEL_CONTATO =
?", tel_contato)
        row = cursor.fetchone()

        if row:
            id_contato = row[0]
        else:
            # Insere um novo contato se não existir
            cursor.execute("INSERT INTO TB_CONTATO (NOME_CONTATO, TEL_CONTATO)
OUTPUT INSERTED.ID_CONTATO VALUES (?, ?)", (nome_contato, tel_contato))
            id_contato = cursor.fetchone()[0]
            conn.commit()

        # Insere um novo cliente
```

```
        cursor.execute("INSERT INTO TB_CLIENTES (NOME, EMAIL, SENHA,
ID_ASSESSOR, ID_CONTATO, DS160) VALUES (?, ?, ?, ?, ?, ?)",
                       (nome, email, senha, id_assessor, id_contato, ds160))
        conn.commit()

        return redirect(url_for('dashboard'))

    return render_template('add_client.html')

# Deleta um cliente
@app.route('/delete_client', methods=['POST'])
def delete_client():
    if 'user_name' not in session:
        return redirect(url_for('login'))
    client_id = request.form.get('selected_name')
    client = Cliente.query.get(client_id)

    cursor = conn.cursor()

    if client:
        # Excluir o pagamento do cliente na TB_PAGAMENTOS
        cursor.execute("DELETE FROM TB_PAGAMENTOS WHERE ID_CLIENTE = ?",
(client_id))

        # Excluir o cliente na TB_CLIENTES
        cursor.execute("DELETE FROM TB_CLIENTES WHERE ID_CLIENTE = ?",
(client_id))
        db.session.commit()
        conn.commit()
    else:
        flash('Cliente não encontrado')

    return redirect(url_for('dashboard'))

# Atualizações da informação de um Assessor
@app.route('/update_assessor')
def update_assessor():
    if 'user_name' not in session:
        return redirect(url_for('login'))

    assessor = Assessor.query.get(session['user_id'])
    return render_template('update_assessor.html', assessor=assessor)

# Salva atualizações do assessor
@app.route('/save_assessor', methods=['POST'])
def save_assessor():
    if 'user_name' not in session:
        return redirect(url_for('login'))
```

```
    assessor = Assessor.query.get(session['user_id'])
    assessor.NOME = request.form['NOME']
    assessor.LOGIN = request.form['LOGIN']
    assessor.EMAIL = request.form['EMAIL']
    assessor.SENHA = request.form['SENHA']

    db.session.commit()
    flash('Informações atualizadas com sucesso')
    return redirect(url_for('dashboard'))

# Cria um novo assessor (Apenas o administrador pode realizar)
@app.route('/add-assessor', methods=['GET', 'POST'])
@admin_required
def add_assessor():
    if request.method == 'POST':
        nome = request.form['NOME']
        login = request.form['LOGIN']
        senha = request.form['SENHA']
        email = request.form['EMAIL']
        administrador = request.form.get('ADMINISTRADOR') == 'on'

        cursor = conn.cursor()

        # Insere novo assessor
        cursor.execute("INSERT INTO TB_ASSESSORES (NOME, LOGIN, SENHA, EMAIL,
ADMINISTRADOR) VALUES (?, ?, ?, ?, ?)",
                       (nome, login, senha, email, administrador))
        conn.commit()

        return redirect(url_for('dashboard'))

    return render_template('add_assessor.html')

# Deleta um assessor (Apenas o administrador pode realizar)
@app.route('/delete-assessor', methods=['GET', 'POST'])
@admin_required
def delete_assessor():
    if 'user_name' not in session:
        return redirect(url_for('login'))

    cursor = conn.cursor()

    if request.method == 'POST':
        assessor_id_now = session['user_id']

        assessor_id = request.form['assessor_id']
```

```
        cursor.execute("UPDATE TB_PAGAMENTOS SET ID_ASSESSOR = ? WHERE
ID_ASSESSOR = ?", (assessor_id_now, assessor_id))

        cursor.execute("DELETE FROM TB_ASSESSORES WHERE ID_ASSESSOR = ?",
(assessor_id))
        conn.commit()

        return redirect(url_for('dashboard'))

    cursor.execute("SELECT ID_ASSESSOR, NOME FROM TB_ASSESSORES WHERE LOGIN !=
? AND ADMINISTRADOR = 0", (session['user_name'],))
    assessores = cursor.fetchall()

    return render_template('delete_assessor.html', assessores=assessores)

# Gerenciamento de Pagamentos
@app.route('/manage-payments', methods=['GET', 'POST'])
def manage_payments():
    if 'user_name' not in session:
        return redirect(url_for('login'))

    cursor = conn.cursor()

    if request.method == 'POST':
        action = request.form['action']
        client_id = request.form['client_id']

        # Obter o ID do Assessor logado
        assessor_id = session['user_id']

        valor_pago = float(request.form['valor_pago'])
        valor_cobrado = float(request.form['valor_cobrado'])

        if valor_pago > valor_cobrado:
                flash('Valor pago não pode ser maior que o valor cobrado.',
'danger')
        else:
            if action == 'create':
                valor_pago = request.form['valor_pago']
                valor_cobrado = request.form['valor_cobrado']

                # Verifica se uma pagamento já existe para o cliente
                cursor.execute("SELECT COUNT(*) FROM TB_PAGAMENTOS WHERE
ID_CLIENTE = ?", (client_id,))
                if cursor.fetchone()[0] > 0:
                    flash('Já existe um pagamento para este cliente.',
'danger')
                else:
```

```
                    # Insere novo Pagamento
                    cursor.execute("""
                        INSERT INTO TB_PAGAMENTOS (ID_CLIENTE, ID_ASSESSOR,
VALOR_PAGO, VALOR_COBRADO)
                        VALUES (?, ?, ?, ?)
                    """, (client_id, assessor_id, valor_pago, valor_cobrado))
                    conn.commit()
                    flash('Pagamento criado com sucesso.', 'success')
            elif action == 'update':
                cursor.execute("SELECT * FROM TB_PAGAMENTOS WHERE ID_CLIENTE =
?", (client_id))
                payment = cursor.fetchone()

                if not payment:
                    flash('Não existe pagamento para este cliente.', 'danger')
                else:
                    valor_pago = request.form['valor_pago']
                    valor_cobrado = request.form['valor_cobrado']

                    # Atualliza Pagamento
                    cursor.execute("""
                        UPDATE TB_PAGAMENTOS
                        SET VALOR_PAGO = ?, VALOR_COBRADO = ?, ID_ASSESSOR = ?
                        WHERE ID_CLIENTE = ?
                    """, (valor_pago, valor_cobrado, assessor_id, client_id))
                    conn.commit()
                    flash('Pagamento atualizado com sucesso.', 'success')

    cursor.execute("SELECT ID_CLIENTE, NOME FROM TB_CLIENTES")
    clients = cursor.fetchall()

    return render_template('manage_payments.html', clients=clients)

# Obtem pagamentos existentes
@app.route('/get-payment-info/<int:client_id>', methods=['GET'])
def get_payment_info(client_id):
    if 'user_name' not in session:
        return redirect(url_for('login'))

    cursor = conn.cursor()

    cursor.execute("SELECT VALOR_PAGO, VALOR_COBRADO, ID_ASSESSOR FROM
TB_PAGAMENTOS WHERE ID_CLIENTE = ?", (client_id,))
    payment = cursor.fetchone()

    if payment:
        assessor_saved = payment.ID_ASSESSOR
```

```
        cursor.execute("SELECT NOME FROM TB_ASSESSORES WHERE ID_ASSESSOR = ?",
(assessor_saved,))
        assessor_details = cursor.fetchone()
        return {
            'exists': True,
            'valor_pago': payment.VALOR_PAGO,
            'valor_cobrado': payment.VALOR_COBRADO,
            'nome_assessor': assessor_details.NOME
        }
    else:
        return {'exists': False}

 # Atualiza os agendamentos
def login_and_capture_info():
    cursor = conn.cursor()

    # Recupera todos os clientes e suas credenciais de login
    cursor.execute("SELECT EMAIL, SENHA FROM TB_CLIENTES")
    clientes = cursor.fetchall()

    # Configura o navegador (nesse exemplo, usando Chrome)
    chrome_options = Options()
    chrome_options.add_argument("--headless")
    driver = webdriver.Chrome(options=chrome_options)


    # código para capturar as informações
    for cliente in clientes:
        email = cliente.EMAIL.strip()
        senha = cliente.SENHA.strip()

        # Abre o site
        driver.get("https://ais.usvisa-info.com/en-br/niv/users/sign_in")

        # Aguarda até que os campos de login estejam presentes
        time.sleep(5)

        # Preenche o campo de email
        email_field = driver.find_element(By.ID, "user_email")
        email_field.clear()
        email_field.send_keys(email)

        # Preenche o campo de senha
        password_field = driver.find_element(By.ID, "user_password")
        password_field.clear()
        password_field.send_keys(senha)

        # Clica no checkbox
```

```
        driver.find_element(By.XPATH,
'//*[@id="sign_in_form"]/div[3]/label/div').click()

        # Aguarda processamento
        time.sleep(2)

        # Submete o formulário
        login_button = driver.find_element(By.NAME, "commit")
        login_button.click()

        # Aguarda até que a página de login seja processada
        time.sleep(5)

        # Captura as informações de agendamento
        try:
            consular_appt = driver.find_element(By.CSS_SELECTOR, ".card
.consular-appt").text
            casv_appt = driver.find_element(By.CSS_SELECTOR, ".card .asc-
appt").text

            print(f"{consular_appt}")
            print(f"{casv_appt}")

            # Expressão regular para extrair data, hora e cidade do
agendamento consular
            consular_pattern = re.compile(r"(\d{2} \w+, \d{4}), (\d{2}:\d{2})
(.+?) -")
            consular_match = consular_pattern.search(consular_appt)
            if consular_match:
                consular_date = consular_match.group(1)
                consular_time = consular_match.group(2)
                consular_city = consular_match.group(3)

                # Converte a data e hora para datetime SQL
                consular_datetime = datetime.strptime(f"{consular_date}
{consular_time}", "%d %B, %Y %H:%M")

                consular_city_id = get_city_id(cursor, consular_city)

                print(consular_datetime)

                print(f"Agendamento Consular: Data - {consular_date}, Hora -
{consular_time}, Cidade - {consular_city}")


            # Expressão regular para extrair data, hora e cidade do
agendamento CASV
            casv_pattern = re.compile(r"(\d{2} \w+, \d{4}), (\d{2}:\d{2})
(.+?) local time")
```

```
            casv_match = casv_pattern.search(casv_appt)
            if casv_match:
                casv_date = casv_match.group(1)
                casv_time = casv_match.group(2)
                casv_city = casv_match.group(3)

                # Converte a data e hora para datetime SQL
                casv_datetime = datetime.strptime(f"{casv_date} {casv_time}",
"%d %B, %Y %H:%M")

                if casv_city == "Sao Paulo" :
                    casv_city = "São Paulo"

                casv_city_id = get_city_id(cursor, casv_city)

                print(casv_datetime)

                print(f"Agendamento CASV: Data - {casv_date}, Hora -
{casv_time}, Cidade - {casv_city}")

            # Atualiza a tabela no banco de dados com as informações
capturadas
            cursor.execute("""
                UPDATE TB_CLIENTES
                SET AGENDAMENTO_CONSULAR = ?, CIDADE_CONSULAR = ?,
AGENDAMENTO_CASV = ?, CIDADE_CASV = ?
                WHERE EMAIL = ?
            """, (consular_datetime, consular_city_id, casv_datetime,
casv_city_id, email))
            conn.commit()

        except Exception as e:
            print(f"Erro ao capturar informações: {e}")

        # Volta para a página de login para o próximo cliente
        driver.get("https://ais.usvisa-info.com/en-br/niv/users/sign_in")
        time.sleep(5)

    # Fecha o navegador
    driver.quit()

# Função Flask para o botão atualizar
@app.route('/update', methods=['GET'])
def update():
    success = login_and_capture_info()
    if success:
        flash('Informações atualizadas com sucesso.', 'success')
    else:
```

```
        flash('Erro ao atualizar informações.', 'danger')
    return redirect(url_for('dashboard'))


# Função para obter o ID da cidade
def get_city_id(cursor, city_name):
    cursor.execute("SELECT ID_CIDADE FROM TB_CIDADE WHERE CIDADE = ?",
(city_name,))
    result = cursor.fetchone()
    if result:
        return result[0]
    return None

# Funcao para enviar um email
@app.route('/send_email', methods=['GET', 'POST'])
def send_email():
    if 'user_name' not in session:
        return redirect(url_for('login'))

    cursor = conn.cursor()

    # Recupera todos os clientes
    cursor.execute("SELECT ID_CLIENTE, NOME, EMAIL FROM TB_CLIENTES")
    clients = cursor.fetchall()

    # Converte objetos Row para dicionários
    clients_list = []
    for client in clients:
        clients_list.append({
            'ID_CLIENTE': client.ID_CLIENTE,
            'NOME': client.NOME,
            'EMAIL': client.EMAIL
        })

    if request.method == 'POST':
        client_id = request.form['client_id']
        client_email = request.form['client_email']
        assessor_email = request.form['assessor_email']
        email_subject = request.form['email_subject']
        email_body = request.form['email_body']

        # Processa os arquivos enviados
        files = request.files.getlist('files')
        file_paths = []
        for file in files[:2]:  # Limite de 2 arquivos
            if file:
                filename = secure_filename(file.filename)
                file_path = os.path.join(app.config['UPLOAD_FOLDER'],
filename)
```

```
                file.save(file_path)
                file_paths.append(file_path)

        # Envia o e-mail
        try:
            msg = Message(subject=email_subject, recipients=[client_email])
            msg.body = email_body
            msg.sender = assessor_email

            # Adiciona os anexos
            for file_path in file_paths:
                with app.open_resource(file_path) as fp:
                    msg.attach(filename=file_path.split('/')[-1],
content_type='application/octet-stream', data=fp.read())

            mail.send(msg)
            flash('E-mail enviado com sucesso.', 'success')
        except Exception as e:
            flash(f'Erro ao enviar o e-mail: {e}', 'danger')

        return redirect(url_for('send_email'))

    return render_template('send_email.html', clients=clients_list)

if __name__ == '__main__':
    app.run(debug=True)
    #app.run(debug=True, host='0.0.0.0')
```

B.2 TEMPLATES/ADD_ASSESSOR.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Adicionar Assessor</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
    <div class="dashboard-container">
        <h2>Adicionar Assessor</h2>
        <form method="POST" action="{{ url_for('add_assessor') }}">
            <label for="NOME">Nome:</label>
            <input type="text" name="NOME" id="NOME" required>

            <label for="LOGIN">Login:</label>
            <input type="text" name="LOGIN" id="LOGIN" required>

            <label for="SENHA">Senha:</label>
            <input type="password" name="SENHA" id="SENHA" required>

            <label for="EMAIL">Email:</label>
            <input type="email" name="EMAIL" id="EMAIL" required>

            <label for="ADMINISTRADOR">Administrador:</label>
            <input type="checkbox" name="ADMINISTRADOR" id="ADMINISTRADOR">

            <button type="submit">Adicionar Assessor</button>
            <button onclick="location.href='{{ url_for('dashboard')
}}'">Voltar</button>
        </form>
    </div>
</body>
</html>
```

B.3 TEMPLATES/ADD_CLIENT.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Adicionar Cliente</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
    <div class="dashboard-container">
        <h2>Adicionar Cliente</h2>
        <form method="POST" action="{{ url_for('add_client') }}">
            <label for="NOME">Nome:</label>
            <input type="text" name="NOME" id="NOME" required>

            <label for="EMAIL">Email:</label>
            <input type="email" name="EMAIL" id="EMAIL" required>

            <label for="SENHA">Senha:</label>
            <input type="password" name="SENHA" id="SENHA" required>

            <label for="DS160">DS160:</label>
            <input type="text" name="DS160" id="DS160" placeholder="Deixe em
branco para NULL">

            <label for="NOME_CONTATO">Nome do Contato:</label>
            <input type="text" name="NOME_CONTATO" id="NOME_CONTATO" required>

            <label for="TEL_CONTATO">Telefone do Contato:</label>
            <input type="tel" name="TEL_CONTATO" id="TEL_CONTATO" required>

            <button type="submit">Adicionar Cliente</button>
            <button onclick="location.href='{{ url_for('dashboard')
}}'">Voltar</button>
        </form>
    </div>
</body>
</html>
```

B.4 TEMPLATES/ASSESSOR_DETAILS.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dados do Assessor</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
    <div class="assessor-details-container">
        <h2>Dados do Assessor</h2>
        <p><strong>Nome:</strong> {{ assessor.NOME }}</p>
        <p><strong>Login:</strong> {{ assessor.LOGIN }}</p>
        <p><strong>Email:</strong> {{ assessor.EMAIL }}</p>
        <p><strong>Senha:</strong> {{ assessor.SENHA }}</p>
        <p><strong>Administrador:</strong> {{ 'Sim' if assessor.ADMINISTRADOR
else 'Não' }}</p>
        <button onclick="location.href='{{ url_for('index')
}}'">Voltar</button>
    </div>
</body>
</html>
```

B.5 TEMPLATES/DASHBOARD.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
    <div class="dashboard-container">
        <h2>Bem-vindo, {{ session['user_name'] }}!</h2>
        <div class="button-container">
            <div class="left-buttons">
                <a href="{{ url_for('add_client') }}"><button>Adicionar
Cliente</button></a>
                <a href="{{ url_for('update_client') }}"><button>Atualizar
Cliente</button></a>
                <button id="delete-button">Excluir Cliente</button>
                <a href="{{ url_for('send_email') }}"><button>Enviar
Email</button></a>
                <a href="{{ url_for('manage_payments') }}"><button>Gerenciar
Pagamentos</button></a>
            </div>
            <div class="right-buttons">
                <div class="dropdown">
                    <button class="dropbtn">Ordenar</button>
                    <div class="dropdown-content">
                        <a href="{{ url_for('sort', criteria='nome') }}">Por
Nome</a>
                        <a href="{{ url_for('sort', criteria='casv') }}">Por
Agendamento CASV</a>
                        <a href="{{ url_for('sort', criteria='consular')
}}">Por Agendamento Consular</a>
                    </div>
                </div>
                <a href="{{ url_for('update')
}}"><button>Atualizar</button></a>
                <a href="{{ url_for('update_assessor') }}"><button>Atualizar
Informações</button></a>
                <a href="{{ url_for('logout') }}"><button>Sair</button></a>
            </div>
        </div>
        <div class="table-container">
            <table>
                <thead>
                    <tr>
                        <th>Nome</th>
```

```
                        <th>Assessor</th>
                        <th>Contato</th>
                        <th>Email</th>
                        <th>Senha</th>
                        <th>DS160</th>
                        <th>Agendamento CASV</th>
                        <th>Cidade CASV</th>
                        <th>Agendamento Consular</th>
                        <th>Cidade Consular</th>
                    </tr>
                </thead>
                <tbody>
                    {% for cliente in clientes %}
                    <tr>
                        <td>{{ cliente.NOME }}</td>
                        <td>{{ cliente.NOME_ASSESSOR }}</td>
                        <td>{{ cliente.NOME_CONTATO }} - {{
cliente.TEL_CONTATO }}</td>
                        <td>{{ cliente.EMAIL }}</td>
                        <td>{{ cliente.SENHA }}</td>
                        <td>{{ cliente.DS160 }}</td>
                        <td>{{ cliente.AGENDAMENTO_CASV }}</td>
                        <td>{{ cliente.CIDADE_CASV }}</td>
                        <td>{{ cliente.AGENDAMENTO_CONSULAR }}</td>
                        <td>{{ cliente.CIDADE_CONSULAR }}</td>
                    </tr>
                    {% endfor %}
                </tbody>
            </table>
        </div>

        {% if session['is_admin'] %}
        <div class="admin-button-container">
            <a href="{{ url_for('add_assessor') }}"><button>Adicionar
Assessor</button></a>
            <a href="{{ url_for('delete_assessor') }}"><button>Excluir
Assessor</button></a>
        </div>
        {% endif %}

        <!-- Modal for Delete Client -->
        <div id="delete-modal" class="modal">
            <div class="delete-assessor-container">
                <span class="close">&times;</span>
                <h2>Excluir Cliente</h2>
                <form method="POST" action="{{ url_for('delete_client') }}">
                    <label for="selected_name">Nome do Cliente:</label>
                    <select name="selected_name" id="selected_name">
```

```
                        {% for cliente in clientes %}
                        <option value="{{ cliente.ID_CLIENTE }}">{{
cliente.NOME }}</option>
                        {% endfor %}
                    </select>
                    <button type="submit">Excluir</button>
                </form>
            </div>
        </div>
    </div>

    <script>
        // Modal script
        var modal = document.getElementById("delete-modal");
        var btn = document.getElementById("delete-button");
        var span = document.getElementsByClassName("close")[0];

        btn.onclick = function() {
            modal.style.display = "block";
        }

        span.onclick = function() {
            modal.style.display = "none";
        }

        window.onclick = function(event) {
            if (event.target == modal) {
                modal.style.display = "none";
            }
        }
    </script>
</body>
</html>
```

B.6 TEMPLATES/DELETE_ASSESSOR.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Excluir Assessor</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
    <div class="delete-assessor-container">
        <h2>Excluir Assessor</h2>
        <form method="POST" action="{{ url_for('delete_assessor') }}">
            <label for="assessor_id">Selecione o Assessor:</label>
            <select name="assessor_id" id="assessor_id" required>
                {% for assessor in assessores %}
                <option value="{{ assessor.ID_ASSESSOR }}">{{ assessor.NOME
}}</option>
                {% endfor %}
            </select>
            <button type="submit">Excluir Assessor</button>
            <button onclick="location.href='{{ url_for('dashboard')
}}'">Voltar</button>
        </form>
    </div>
</body>
</html>
```

B.7 TEMPLATES/LOGIN.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='login.css')
}}">
</head>
<body>
    <div class="login-container">
        <img src="{{ url_for('static', filename='logo.png') }}" alt="Company
Logo" class="logo">
        <h2>Login</h2>
        <form action="/login" method="post">
            <label for="login">Login:</label>
            <input type="text" id="login" name="login" required>

            <label for="senha">Senha:</label>
            <input type="password" id="senha" name="senha" required>

            <button type="submit">Entrar</button>
        </form>
        <a href="/recover_password" class="recover-password">Esqueceu sua
senha?</a>
        {% with messages = get_flashed_messages() %}
          {% if messages %}
            <ul class="messages">
              {% for message in messages %}
                <li>{{ message }}</li>
              {% endfor %}
            </ul>
          {% endif %}
        {% endwith %}
    </div>
</body>
</html>
```

B.8 TEMPLATES/MANAGE_PAYMENTS.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Gerenciar Pagamentos</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='manage_payments.css') }}">
</head>
<body>
    <div class="dashboard-container">
        <h2>Gerenciar Pagamentos</h2>
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                {% for category, message in messages %}
                    <div class="alert {{ category }}">{{ message }}</div>
                {% endfor %}
            {% endif %}
        {% endwith %}
        <label for="client_id">Selecione o Cliente:</label>
        <select name="client_id" id="client_id" required>
            {% for client in clients %}
            <option value="{{ client.ID_CLIENTE }}">{{ client.NOME }}</option>
            {% endfor %}
        </select>

        <div class="button-container">
            <button id="create-button">Criar Pagamento</button>
            <button id="update-button">Atualizar Pagamento</button>
            <button onclick="location.href='{{ url_for('dashboard')
}}'">Voltar</button>
        </div>

        <form method="POST" action="{{ url_for('manage_payments') }}"
id="create-form" style="display: none;">
            <input type="hidden" name="action" value="create">
            <input type="hidden" name="client_id" id="create-client-id">
            <label for="valor_pago">Valor Pago:</label>
            <input type="number" name="valor_pago" id="valor_pago"
required><br>
            <label for="valor_cobrado">Valor Cobrado:</label>
            <input type="number" name="valor_cobrado" id="valor_cobrado"
required><br>
            <button type="submit">Salvar</button>
        </form>

        <form method="POST" action="{{ url_for('manage_payments') }}"
id="update-form" style="display: none;">
```

```
            <input type="hidden" name="action" value="update">
            <input type="hidden" name="client_id" id="update-client-id">
            <label for="">Valor Pago:</label>
            <input type="number" name="valor_pago" id="update-valor_pago"
required><br>
            <label for="valor_cobrado">Valor Cobrado:</label>
            <input type="number" name="valor_cobrado" id="update-
valor_cobrado" required><br>
            <label for="assessor">Assessor:</label>
            <input type="text" name="nome_assessor" id="update-nome_assessor"
readonly><br>
            <button type="submit">Atualizar</button>
        </form>
        <div>
        </div>
    </div>

    <script>
        document.getElementById('create-button').addEventListener('click',
function() {
            document.getElementById('create-client-id').value =
document.getElementById('client_id').value;
            document.getElementById('create-form').style.display = 'block';
            document.getElementById('update-form').style.display = 'none';
        });

        document.getElementById('update-button').addEventListener('click',
function() {
            var clientId = document.getElementById('client_id').value;
            document.getElementById('update-client-id').value = clientId;
            fetch('/get-payment-info/' + clientId)
                .then(response => response.json())
                .then(data => {
                    if (data.exists) {
                        document.getElementById('update-valor_pago').value =
data.valor_pago;
                        document.getElementById('update-valor_cobrado').value
= data.valor_cobrado;
                        document.getElementById('update-nome_assessor').value
= data.nome_assessor;
                        document.getElementById('update-form').style.display =
'block';
                        document.getElementById('create-form').style.display =
'none';
                    } else {
                        alert('Não existe pagamento para este cliente.');
                        document.getElementById('update-form').style.display =
'none';
```

```
                    }
                });
        });
    </script>
</body>
</html>
```

B.9 TEMPLATES/RECOVER_PASSWORD.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Recuperar Senha</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
    <div class="recover-password-container">
        <h2>Recuperar Senha</h2>
        <form action="/recover_password" method="post">
            <label for="email">Email cadastrado:</label>
            <input type="email" id="email" name="email" required>

            <button type="submit">Recuperar Senha</button>
            <button onclick="location.href='{{ url_for('index')
}}'">Voltar</button>
        </form>
        {% with messages = get_flashed_messages() %}
          {% if messages %}
            <ul class="messages">
              {% for message in messages %}
                <li>{{ message }}</li>
              {% endfor %}
            </ul>
          {% endif %}
        {% endwith %}
    </div>
</body>
</html>
```

B.10 TEMPLATES/SELECT_CLIENT.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Selecionar Cliente</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
    <div class="delete-assessor-container">
        <h2>Selecionar Cliente para Atualização</h2>
        <form method="POST" action="{{ url_for('update_client') }}">
            <label for="selected_name">Nome do Cliente:</label>
            <select name="selected_name" id="selected_name">
                {% for cliente in clientes %}
                <option value="{{ cliente.NOME }}">{{ cliente.NOME }}</option>
                {% endfor %}
            </select>
            <button type="submit">Selecionar</button>
        </form>
        <button onclick="location.href='{{ url_for('dashboard')
}}'">Voltar</button>
    </div>
</body>
</html>
```

B.11 TEMPLATES/SEND_EMAIL.HTML

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Enviar E-mail</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='send_email.css') }}">
    <script>
        function updateClientEmail() {
            var clients = {{ clients | tojson }};
            var client_id = document.getElementById('client_id').value;
            var client = clients.find(client => client.ID_CLIENTE ==
client_id);
            document.getElementById('client_email').value = client ?
client.EMAIL : '';
        }
    </script>
</head>
<body>
    <div class="send-email-container">
        <h2>Enviar E-mail</h2>
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                {% for category, message in messages %}
                    <div class="alert {{ category }}">{{ message }}</div>
                {% endfor %}
            {% endif %}
        {% endwith %}
        <form method="POST" action="{{ url_for('send_email') }}"
enctype="multipart/form-data">
            <label for="client_id">Selecione o Cliente:</label>
            <select name="client_id" id="client_id"
onchange="updateClientEmail()" required>
                <option value="" disabled selected>Selecione um
cliente</option>
                {% for client in clients %}
                    <option value="{{ client.ID_CLIENTE }}">{{ client.NOME
}}</option>
                {% endfor %}
            </select>
            <label for="client_email">E-mail do Cliente:</label>
            <input type="email" name="client_email" id="client_email"
required>
            <label for="assessor_email">Seu E-mail:</label>
            <input type="email" name="assessor_email" id="assessor_email"
required>
            <label for="email_subject">Assunto:</label>
```

```
            <input type="text" name="email_subject" id="email_subject"
required>
            <label for="email_body">Mensagem:</label>
            <textarea name="email_body" id="email_body" rows="4"
required></textarea>
            <label for="files">Anexar Arquivos (máximo 2):</label>
            <input type="file" name="files" id="files" multiple>
            <button type="submit">Enviar E-mail</button>
            <button onclick="location.href='{{ url_for('dashboard')
}}'">Voltar</button>
        </form>
    </div>
</body>
</html>
```

B.12 TEMPLATES/UPDATE_ASSESSOR.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Atualizar Informações</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
    <div class="update-assessor-container">
        <h2>Atualizar Informações</h2>
        <form method="POST" action="{{ url_for('save_assessor') }}">
            <label for="NOME">Nome:</label>
            <input type="text" name="NOME" id="NOME" value="{{ assessor.NOME
}}" required>

            <label for="LOGIN">Login:</label>
            <input type="text" name="LOGIN" id="LOGIN" value="{{
assessor.LOGIN }}" required>

            <label for="EMAIL">Email:</label>
            <input type="email" name="EMAIL" id="EMAIL" value="{{
assessor.EMAIL }}" required>

            <label for="SENHA">Senha:</label>
            <input type="password" name="SENHA" id="SENHA" required>

            <button type="submit">Salvar</button>
            <button onclick="location.href='{{ url_for('dashboard')
}}'">Voltar</button>
        </form>
    </div>
</body>
</html>
```

B.13 TEMPLATES/UPDATE_CLIENT.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Atualizar Cliente</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
    <div class="dashboard-container">
        <h2>Atualizar Cliente</h2>
        <form method="POST" action="{{ url_for('save_client') }}">
            <input type="hidden" name="ID_CLIENTE" value="{{
cliente.ID_CLIENTE }}">
            <label for="NOME">Nome:</label>
            <input type="text" name="NOME" value="{{ cliente.NOME }}">

            <label for="EMAIL">Email:</label>
            <input type="email" name="EMAIL" value="{{ cliente.EMAIL }}">

            <label for="SENHA">Senha:</label>
            <input type="text" name="SENHA" value="{{ cliente.SENHA }}">

            <label for="DS160">DS160:</label>
            <input type="text" name="DS160" id="DS160" value="{{ cliente.DS160
}}" placeholder="Deixe em branco para NULL">

            <label for="NOME_CONTATO">Nome do Contato:</label>
            <input type="text" name="NOME_CONTATO" value="{{
contato.NOME_CONTATO }}">

            <label for="TEL_CONTATO">Telefone do Contato:</label>
            <input type="text" name="TEL_CONTATO" value="{{
contato.TEL_CONTATO }}">

            <button type="submit">Salvar</button>
            <button onclick="location.href='{{ url_for('dashboard')
}}'">Voltar</button>
        </form>
    </div>
</body>
</html>
```

B.14 STATIC/LOGIN.CSS

```css
body {
    font-family: 'Arial', sans-serif;
    background-color: #f4f7f6;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.login-container {
    background-color: white;
    padding: 40px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    width: 90%;
    max-width: 400px;
    text-align: center;
}

.login-container img {
    width: 150px;
    margin-bottom: 20px;
}

.login-container h2 {
    margin-bottom: 20px;
    color: #333;
}

.login-container form {
    display: flex;
    flex-direction: column;
    align-items: center;
}

.login-container label {
    margin: 10px 0 5px;
    font-weight: bold;
    color: #555;
    width: 100%;
    text-align: left;
}

.login-container input {
    padding: 10px;
    margin: 5px 0;
```

```
    width: 100%;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
}

.login-container button {
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
    width: 100%;
    padding: 10px;
    margin-top: 20px;
    font-size: 16px;
    border-radius: 5px;
}

.login-container button:hover {
    background-color: #0056b3;
}

.login-container .forgot-password {
    margin-top: 20px;
    color: #007bff;
    text-decoration: none;
    font-size: 14px;
}

.login-container .forgot-password:hover {
    text-decoration: underline;
}

.alert {
    margin-top: 20px;
    padding: 20px;
    background-color: #f44336;
    color: white;
    border-radius: 5px;
    text-align: center;
}

.alert.success {
    background-color: #4caf50;
}
```

B.15 STATIC/MANAGE_PAMENT.CSS

```css
body {
    font-family: 'Arial', sans-serif;
    background-color: #f4f7f6;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.dashboard-container {
    background-color: white;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.1);
    width: 90%;
    max-width: 600px;
}

h2 {
    text-align: center;
    color: #333;
}

form {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-top: 20px;
}

label {
    margin: 10px 0 5px;
    font-weight: bold;
    color: #555;
}

select, input, button {
    padding: 10px;
    margin: 5px 0;
    width: 100%;
    max-width: 400px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
}
```

```css
button {
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
    width: auto;
    padding: 10px 20px;
}

button:hover {
    background-color: #0056b3;
}

.button-container {
    display: flex;
    justify-content: center;
    gap: 10px;
    margin-top: 10px;
}

.alert {
    margin-top: 20px;
    padding: 20px;
    background-color: #f44336;
    color: white;
    border-radius: 5px;
    text-align: center;
}

.alert.success {
    background-color: #4caf50;
}
```

B.16 STATIC/SEND_EMAIL.CSS

```css
body {
    font-family: Arial, sans-serif;
    background-color: #f7f7f7;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.send-email-container {
    background-color: #ffffff;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    padding: 20px;
    width: 400px;
    box-sizing: border-box;
}

.send-email-container h2 {
    text-align: center;
    color: #333333;
}

form {
    display: flex;
    flex-direction: column;
}

label {
    margin-bottom: 8px;
    color: #555555;
}

input[type="text"],
input[type="email"],
textarea,
select {
    margin-bottom: 16px;
    padding: 10px;
    border: 1px solid #dddddd;
    border-radius: 4px;
    font-size: 14px;
    width: calc(100% - 22px);  /* to account for padding and border */
}

textarea {
```

```
    resize: vertical;
}

input[type="file"] {
    margin-bottom: 16px;
}

select, input, button {
    padding: 10px;
    margin: 5px 0;
    width: 100%;
    max-width: 400px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
}

button {
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
    width: auto;
    padding: 10px 20px;
}

button:hover {
    background-color: #0056b3;
}

.button-container {
    display: flex;
    justify-content: center;
    gap: 10px;
    margin-top: 10px;
}

.alert {
    padding: 10px;
    border-radius: 4px;
    margin-bottom: 16px;
    font-size: 14px;
    text-align: center;
}

.alert.success {
    background-color: #d4edda;
    color: #155724;
```

```
    border: 1px solid #c3e6cb;
}

.alert.danger {
    background-color: #f8d7da;
    color: #721c24;
    border: 1px solid #f5c6cb;
}
```

B.17 STATIC/STYLES.CSS

```css
body {
    font-family: 'Arial', sans-serif;
    background-color: #f4f7f6;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.dashboard-container {
    background-color: white;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.1);
    width: 90%;
    max-width: 1200px;
}

h1, h2 {
    text-align: center;
    color: #333;
}

form, .button-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-top: 20px;
}

label {
    margin: 10px 0 5px;
    font-weight: bold;
    color: #555;
}

select, input, button {
    padding: 10px;
    margin: 5px 0;
    width: 100%;
    max-width: 400px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
}
```

```css
button {
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
    width: auto;
    padding: 10px 20px;
}

button:hover {
    background-color: #0056b3;
}

.table-container {
    overflow-x: auto;
    margin-top: 20px;
}

table {
    width: 100%;
    border-collapse: collapse;
}

th, td {
    padding: 12px;
    border: 1px solid #ddd;
    text-align: left;
}

th {
    background-color: #007bff;
    color: white;
}

td {
    background-color: #f9f9f9;
}

.table-actions form {
    display: inline;
}

.table-actions button {
    background-color: #dc3545;
    margin: 0 5px;
}

.table-actions button.edit {
```

```
    background-color: #ffc107;
}

.table-actions button:hover {
    background-color: #c82333;
}

.table-actions button.edit:hover {
    background-color: #e0a800;
}

.button-container {
    display: flex;
    justify-content: space-between;
    flex-wrap: wrap;
    gap: 10px;
}

.left-buttons, .right-buttons {
    display: flex;
    gap: 10px;
}

.left-buttons a, .right-buttons a {
    text-decoration: none;
}

.alert {
    margin-top: 20px;
    padding: 20px;
    background-color: #f44336;
    color: white;
    border-radius: 5px;
    text-align: center;
}

.alert.success {
    background-color: #4caf50;
}

.dropdown {
    position: relative;
    display: inline-block;
}

.dropdown-content {
    display: none;
    position: absolute;
```

```
    background-color: #f1f1f1;
    min-width: 160px;
    box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
    z-index: 1;
}

.dropdown-content a {
    color: black;
    padding: 12px 16px;
    text-decoration: none;
    display: block;
}

.dropdown-content a:hover {
    background-color: #ddd;
}

.dropdown:hover .dropdown-content {
    display: block;
}

.dropdown:hover .dropbtn {
    background-color: #1100ff;
}

.recover-password-container, .assessor-details-container {
    background-color: white;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
    width: 80%;
    max-width: 500px;
    text-align: center;
}

/* Modal */
.modal {
    display: none;
    position: fixed;
    z-index: 1;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    overflow: auto;
    background-color: rgb(0,0,0);
    background-color: rgba(0,0,0,0.4);
    padding-top: 60px;
```

```css
}

.modal-content {
    background-color: #fefefe;
    margin: 5% auto;
    padding: 20px;
    border: 1px solid #888;
    width: 80%;
    max-width: 600px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
}

.close {
    color: #aaa;
    float: right;
    font-size: 28px;
    font-weight: bold;
}

.close:hover,
.close:focus {
    color: black;
    text-decoration: none;
    cursor: pointer;
}

/* CSS para a página de exclusão de assessor */
.delete-assessor-container {
    background-color: white;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
    width: 60%;
    max-width: 600px;
    margin: 50px auto;
    text-align: center;
}

.delete-assessor-container h2 {
    margin-bottom: 20px;
    font-size: 24px;
    color: #333;
}

.delete-assessor-container label {
    display: block;
    margin-bottom: 5px;
```

```
    font-weight: bold;
    color: #333;
}

.delete-assessor-container select {
    width: 100%;
    padding: 10px;
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
}

.delete-assessor-container button {
    padding: 10px 20px;
    background-color: #1100ff;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-size: 16px;
}

.delete-assessor-container button:hover {
    background-color: #1100ff;
}
```

B.18 DB/VISTOAMERICANO.SQL

```
USE [VISTOAMERICANO]
GO
/****** Object:  Table [dbo].[TB_ASSESSORES]    Script Date: 14/06/2024
22:57:30 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TB_ASSESSORES](
    [ID_ASSESSOR] [int] IDENTITY(1,1) NOT NULL,
    [NOME] [nchar](40) NOT NULL,
    [LOGIN] [nchar](20) NOT NULL,
    [SENHA] [nchar](10) NOT NULL,
    [EMAIL] [nchar](40) NULL,
    [ADMINISTRADOR] [bit] NOT NULL,
 CONSTRAINT [PK_TB_ASSESSORES] PRIMARY KEY CLUSTERED
(
    [ID_ASSESSOR] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[TB_CIDADE]    Script Date: 14/06/2024 22:57:30
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TB_CIDADE](
    [ID_CIDADE] [int] IDENTITY(1,1) NOT NULL,
    [CIDADE] [nchar](40) NOT NULL,
    [ESTADO] [nchar](40) NULL,
 CONSTRAINT [PK_TB_CIDADE] PRIMARY KEY CLUSTERED
(
    [ID_CIDADE] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[TB_CLIENTES]    Script Date: 14/06/2024 22:57:30
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TB_CLIENTES](
```

```
    [ID_CLIENTE] [int] IDENTITY(1,1) NOT NULL,
    [NOME] [nchar](40) NOT NULL,
    [ID_ASSESSOR] [int] NOT NULL,
    [ID_CONTATO] [int] NULL,
    [EMAIL] [nchar](40) NULL,
    [SENHA] [nchar](10) NOT NULL,
    [DS160] [nchar](10) NULL,
    [ID_AISVISA] [nchar](50) NULL,
    [AGENDAMENTO_CASV] [datetime] NULL,
    [CIDADE_CASV] [int] NULL,
    [AGENDAMENTO_CONSULAR] [datetime] NULL,
    [CIDADE_CONSULAR] [int] NULL,
 CONSTRAINT [PK_TB_CLIENTES] PRIMARY KEY CLUSTERED
(
    [ID_CLIENTE] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[TB_CONTATO]    Script Date: 14/06/2024 22:57:30
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TB_CONTATO](
    [ID_CONTATO] [int] IDENTITY(1,1) NOT NULL,
    [NOME_CONTATO] [nchar](20) NULL,
    [TEL_CONTATO] [nchar](20) NULL,
 CONSTRAINT [PK_TB_CONTATO] PRIMARY KEY CLUSTERED
(
    [ID_CONTATO] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[TB_PAGAMENTOS]    Script Date: 14/06/2024
22:57:30 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TB_PAGAMENTOS](
    [ID_PAGAMENTO] [int] IDENTITY(1,1) NOT NULL,
    [ID_CLIENTE] [int] NULL,
    [ID_ASSESSOR] [int] NULL,
```

```
    [VALOR_PAGO] [money] NULL,
    [VALOR_COBRADO] [money] NULL,
 CONSTRAINT [PK_TB_PAGAMENTOS] PRIMARY KEY CLUSTERED
(
    [ID_PAGAMENTO] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[TB_CLIENTES]  WITH CHECK ADD  CONSTRAINT
[FK_TB_CLIENTES_TB_ASSESSORES] FOREIGN KEY([ID_ASSESSOR])
REFERENCES [dbo].[TB_ASSESSORES] ([ID_ASSESSOR])
GO
ALTER TABLE [dbo].[TB_CLIENTES] CHECK CONSTRAINT
[FK_TB_CLIENTES_TB_ASSESSORES]
GO
ALTER TABLE [dbo].[TB_CLIENTES]  WITH CHECK ADD  CONSTRAINT
[FK_TB_CLIENTES_TB_CIDADE] FOREIGN KEY([CIDADE_CASV])
REFERENCES [dbo].[TB_CIDADE] ([ID_CIDADE])
GO
ALTER TABLE [dbo].[TB_CLIENTES] CHECK CONSTRAINT [FK_TB_CLIENTES_TB_CIDADE]
GO
ALTER TABLE [dbo].[TB_CLIENTES]  WITH CHECK ADD  CONSTRAINT
[FK_TB_CLIENTES_TB_CIDADE1] FOREIGN KEY([CIDADE_CONSULAR])
REFERENCES [dbo].[TB_CIDADE] ([ID_CIDADE])
GO
ALTER TABLE [dbo].[TB_CLIENTES] CHECK CONSTRAINT [FK_TB_CLIENTES_TB_CIDADE1]
GO
ALTER TABLE [dbo].[TB_CLIENTES]  WITH CHECK ADD  CONSTRAINT
[FK_TB_CLIENTES_TB_CONTATO] FOREIGN KEY([ID_CONTATO])
REFERENCES [dbo].[TB_CONTATO] ([ID_CONTATO])
GO
ALTER TABLE [dbo].[TB_CLIENTES] CHECK CONSTRAINT [FK_TB_CLIENTES_TB_CONTATO]
GO
ALTER TABLE [dbo].[TB_PAGAMENTOS]  WITH CHECK ADD  CONSTRAINT
[FK_TB_PAGAMENTOS_TB_ASSESSORES] FOREIGN KEY([ID_ASSESSOR])
REFERENCES [dbo].[TB_ASSESSORES] ([ID_ASSESSOR])
GO
ALTER TABLE [dbo].[TB_PAGAMENTOS] CHECK CONSTRAINT
[FK_TB_PAGAMENTOS_TB_ASSESSORES]
GO
ALTER TABLE [dbo].[TB_PAGAMENTOS]  WITH CHECK ADD  CONSTRAINT
[FK_TB_PAGAMENTOS_TB_CLIENTES] FOREIGN KEY([ID_CLIENTE])
REFERENCES [dbo].[TB_CLIENTES] ([ID_CLIENTE])
GO
ALTER TABLE [dbo].[TB_PAGAMENTOS] CHECK CONSTRAINT
[FK_TB_PAGAMENTOS_TB_CLIENTES]
```

```
GO
```