Desenvolvimento de uma aplicação web para inquilinos da empresa Riskivector

Guilherme Ianhez Pereira dos Santos

Relatório Final de Estágio apresentado à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná.

Trabalho realizado sob a orientação de:

Prof. Doutor Paulo Alexandre Vara Alves

Prof. Doutor José Eduardo Moreira Fernandes

Marcin Włodarczyk

Este relatório não inclui as críticas e sugestões feitas pelo Júri.

Bragança

Fevereiro de 2020

Desenvolvimento de uma aplicação web para inquilinos da empresa Riskivector

Guilherme Ianhez Pereira dos Santos

Relatório Final de Estágio apresentado à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná.

Trabalho realizado sob a orientação de:

Prof. Doutor Paulo Alexandre Vara Alves

Prof. Doutor José Eduardo Moreira Fernandes

Marcin Włodarczyk

Este relatório não inclui as críticas e sugestões feitas pelo Júri.

Bragança

Fevereiro de 2020

Agradecimentos

Agradeço primeiramente a minha família e amigos que me motivaram, me deram apoio e força não só no desenvolvimento desse trabalho, mas em todos os momentos da minha vida independentemente das circunstâncias.

Aos meus orientadores, Professor Doutor José Eduardo Moreira Fernandes e Professor Doutor Paulo Alexandre Vara Alves, não só por me auxiliarem em todo o desenvolvimento do projeto, mas também por torná-lo possível através da ponte que fizeram entre o Instituto Politécnico de Bragança e a empresa Riskivector que nos acolheu.

Um agradecimento especial a Universidade Tecnológica Federal do Paraná por todo o suporte e os conhecimentos adquiridos no decorrer do curso. A empresa Riskivector que nos apoiou, auxiliou e deu a oportunidade de realizar esse projeto que agregou muito em minha vida acadêmica e também em meu crescimento profissional.

Resumo

A automatização de processos facilita o acesso à informação, reduz custos e tempo, além de possibilitar uma gestão segura e eficaz, acarretando em uma maior comodidade para os clientes que utilizam o sistema.

Este trabalho tem como objetivo melhorar a interação entre o cliente e a empresa *Riskivector*. Até então, todos os processos relacionados aos clientes da empresa eram realizados de forma manual, ou feita por ligações telefônicas e e-mail. Nessa perspectiva, a proposta foi criar uma aplicação web para que os clientes tenham acesso aos serviços de um modo mais simples, rápido e eficaz.

Dessa forma, durante o estágio foi elaborado um sistema web utilizando o framework Angular que provê diferentes funcionalidades para os inquilinos, tais como: realizar pedido de gás, gerenciar reservas, realizar pagamentos on-line, consultar movimentações e saldo de sua conta, entre outros serviços. Além disso, foi reestruturado o servidor da empresa, adotando duas novas arquiteturas.

Este documento especifica as etapas para o desenvolvimento da aplicação, abordando desde a fase do planejamento do software no qual recolheu-se os requisitos do sistema e escolheu-se as ferramentas utilizadas, até a fase de implementação, pontuando os problemas e soluções durante todo percurso.

Assim sendo, esse projeto atingiu seu objetivo em desenvolver uma aplicação simples e robusta para os clientes. Além disso, poupou tempo e esforço dos colaboradores e acrescentou diversos conhecimentos pessoais, profissionais e intelectuais.

Palavras-chave: Gerenciamento de recursos dos clientes, Desenvolvimento web, Angular.

Abstract

Process automation facilitates access to information, reduces costs and time, and enables safe and effective management, resulting in greater convenience for customers using the system.

This work aims to improve the interaction between the client and the company *Riskivector*. Until then, all processes related to the company's clients were carried out manually, or by phone calls and e-mail. From this perspective, the proposal was to create a web application so that clients have access to services in a simpler, faster and more effective way.

Thus, during the internship, a web system was developed using the Angular framework that provides different functionalities for the tenants, such as: making a gas order, managing reservations, making online payments, account movement reports and account balance, among other services. In addition, the company's server was restructured, adopting two new architectures.

This document specifies the steps for the development of the application, addressing from the software planning phase which the system requirements were collected and the tools used were chosen, to the implementation phase, punctuating the problems and solutions throughout the way.

Thus, this project achieved its goal in developing a simple and robust application for customers. Moreover, it saved time and effort of the employees and added several personal, professional and intellectual knowledge.

Keywords: Customer resource management, Web development, Angular.

Conteúdo

Lista de Tabelas

Lista de Figuras

Siglas

AJAX Asynchronous Javascript and XML. 25, 28

API Application Programming Interface. xiii, xiv, 3, 9, 18, 21–23, 27, 29, 49, 53, 54, 62, 87, 88, 91

CD Change Detector. 26, 27

CSS Cascading Style Sheets. xiii, 14–17, 23

DOM Document Object Model. 26–28

DTO Data Transfer Object. 46

ECMA European Computer Manufactures Association. 17

ER Entidade Relacionamento. 41, 47

FTP File Transfer Protocol. 8

HTML Hypertext Markup Language. xiii, 12–15, 17, 23, 25, 27

HTTP Hypertext Transfer Protocol. xii, xiii, 5–12, 18, 19, 22, 29, 30, 32, 46, 49, 53, 57

IDE Integrated Development Environment. 19

IPB Instituto Politécnico de Bragança. 2

JSON JavaScript Object Notation. 12, 25, 41, 45

JWT JSON Web Token. 61

MER Modelo Entidade Relacionamento. 33

MIME Multipurpose Internet Mail Extensions. xii, 12

MPA Multiple Page Application. xiii, 24–26

ORM Object-relational Mapping. xiii, xiv, 3, 23, 24, 46, 50, 51

OSI Open System Interconnection. 7

REST Representational State Transfer. 22, 29–32

RFC Request for Comments. 7

SEO Search Engine Optimization. 26

SOAP Simple Object Access Protocol. xiii, 29, 31, 32

SPA Single Page Application. xiii, 24–26

SQL Structured Query Language. 23, 57

URI Uniform Resource Identifier. ix, 6, 8, 9, 29–31

URL Uniform Resource Locator. 8

VDOM Virtual DOM. 26, 27

W3C World Wide Web Consortium. 13

WWW *World Wide Web*. 5–7

XML Extensible Markup Language. 25, 31, 32

Capítulo 1

Introdução

1.1 Enquadramento

A otimização em termos matemáticos consiste em encontrar os mínimos ou máximos de uma função com diversas variáveis em um certo conjunto (Martínez, 1998). Nos mais variados campos da atividade humana encontra-se esse tipo de necessidade. O meio empresarial é um dos exemplos onde a busca de otimizações implica em diversos ganhos, tanto para a empresa como também, na maioria das vezes, para seus clientes. Otimizar processos e sistemas é demasiadamente importante, visto que uma empresa busca minimizar seus custos e maximizar seus ganhos.

No momento, os processos relacionados com clientes são tratados e geridos no escritório da empresa. Assim sendo, para efetuar, por exemplo, pagamentos e tirar dúvidas é necessária a presença do inquilino. Dado que a empresa possui milhares de inquilinos e tendo em vista que todos os semestres recebem centenas de novos clientes, é imprescindível um sistema capaz de aproximar os clientes da empresa, gerenciando e automatizando seus serviços.

Este trabalho consiste em desenvolver um sistema web para os clientes da empresa Riskivector que visa entregar uma maior comodidade para o cliente, otimizando o seu tempo. Na perspectiva da empresa, tal aplicação implica na redução do tempo para

acessar seus serviços e também oferece uma maior flexibilidade para seus colaboradores, visto que poderão focar em outras atividades.

1.2 A Empresa

A empresa Riskivector está localizada estrategicamente na zona de Bragança que possui um grande potencial de crescimento, pois além de ser uma cidade capital do distrito de Bragança e possuir uma instituição de ensino superior de referência, o Instituto Politécnico de Bragança (IPB), ainda conta com um custo de vida relativamente baixo se comparado com outras regiões do país. Todas essas características geram um grande interesse nessa zona, tanto na parte empresarial, por possuir mão de obra qualificada, como também social.

Nesse contexto, uma vez que aumenta o interesse pela cidade de Bragança quer para estudar quer para trabalhar, consequentemente há também um aumento populacional e é justamente nesse ponto em que a empresa Riskivector criada em 23 de Fevereiro de 2009 e incubada pelo IPB em 2010, atua. Possuindo mais de 10 anos de experiência no ramo imobiliário, a empresa entrega serviços de qualidade planejado para o orçamento de seus clientes. Ela age como um intermediário entre o locador e o locatário, alugando imóveis ou uma fração destes. Além disso, a empresa também desenvolve atividades de manutenções gerais e serviços de gestão e apoio em reservas de alojamento, bem como viagens e eventos.

Todas essas características impulsionam o crescimento da empresa, começando apenas com um escritório, localizado no IPB, hoje encontra-se em diferentes sítios, contando com escritórios em Mirandela e Vila Real.

1.3 Objetivos

Desenvolver um sistema web para que os inquilinos realizem processos que atualmente são realizados de forma manual, ou demandam algum tipo de esforço. Visando atingir

uma maior quantidade de dispositivos e também evitar a reescrita de código, o objetivo é desenvolver um sistema web, juntamente com uma Application Programming Interface (API), onde o cliente poderá consultar seu saldo e movimentações financeiras, pagar contas, gerenciar reservas de alojamento, realizar pedido de gás, consultar endereços e dados pessoais, entre outros serviços. Tal sistema pretende não só otimizar e automatizar os processos, como também promover novos serviços, facilitando a interação entre o cliente e a empresa, provendo transparência e simplicidade, buscando melhorar a experiência do cliente ao utilizar seus serviços.

1.4 Estrutura do Documento

Após uma introdução do que foi realizado no projeto, este documento está estruturado da seguinte forma: o Capítulo 2 apresenta os conceitos necessários para entender o projeto em sua totalidade. Primeiramente é desenvolvido os fundamentos sobre a web, abordando as tecnologias ao seu redor que orquestram seu funcionamento, tais como: protocolos, arquitetura cliente/servidor, o que é um cliente, etc. Também são apresentados conceitos que são utilizados neste documento, como front-end/back-end, Object-relational Mapping (ORM), entre outros.

Após o estudo sobre a web e suas tecnologias, o Capítulo 3 diz respeito ao planejamento da construção do software. Esse capítulo envolve os processos de modelação, englobando desde a coleta de requisitos, até a escolha das tecnologias para sua implementação.

O Capítulo 4 enfatiza o caminho percorrido durante todo o desenvolvimento do sistema. Inicialmente são apresentados os problemas e dificuldades encontradas durante a implementação bem como os métodos adotados para suas soluções. Em seguida são apresentadas duas arquiteturas. A primeira ilustra a rede, demonstrando a disposição e os elementos que fazem parte da comunicação entre a aplicação web e o servidor. A segunda visa solucionar antigas dívidas técnicas adquiridas ao longo do tempo. O restante do capítulo aborda a implementação dos requisitos do sistema.

Com o sistema finalizado, o Capítulo 5 discorre sobre as conclusões finais deste trabalho, assim como, os trabalhos futuros a serem realizados nesse projeto.

Capítulo 2

Revisão da literatura

Esta seção apresentará os conceitos pertinentes para compreensão do enquadramento do projeto. Serão abordados conceitos básicos e necessários no ambiente da web, bem como as ferramentas disponíveis que auxiliam em seu desenvolvimento.

2.1 World Wide Web

Entre 1989 a 1990 o cientista inglês Tim Berners-Lee desenvolveu os conceitos necessários para a criação da primeira versão da *World Wide Web* (WWW), ou apenas web. Segundo Tim, a web surgiu para padronizar e facilitar o acesso a informação, uma vez que antes da web seu acesso muitas vezes envolvia aceder à diferentes computadores e aprender os programas que estes utilizavam (Foundation, 2000).

Até então, a Internet era utilizada em sua maioria, apenas na área acadêmica para efetuar login em hospedeiros remotos e transferir arquivos (Kurose, 2013). A WWW popularizou e revolucionou o modo de vida dentro e fora do trabalho de cada indivíduo. A popularidade da web foi tanta, que até mesmos nos dias de hoje, as pessoas confundem o conceito da WWW com a Internet.

A WWW segue a arquitetura cliente/servidor, onde a comunicação é estabelecida através do protocolo Hypertext Transfer Protocol (HTTP) que define a estrutura das mensagens trocadas, onde o cliente envia requisições geralmente através do navegador

(Internet Explorer, Google Chrome, Safari, Firefox, etc), para o servidor e este responde cada requisição individualmente através de seu Uniform Resource Identifier (URI).

Dessa forma pode-se dizer que a WWW é a coleção dos recursos e documentos web que são disponibilizados e trafegados na Internet, portanto não se deve confundir a WWW com a rede mundial de computadores (Internet), uma vez que a Internet é apenas o meio de transporte que a web utiliza para estar presente em qualquer dispositivo que esteja conectado a Internet. Nas próximas seções serão abordados os conceitos e tecnologias necessárias que tornam possível a navegação na web.

2.1.1 Web Site

Um web site, ou simplesmente site, é composto por um conjunto de páginas web (ver Seção 2.3). Cada página web dentro do site possui um identificador, para localizá-lo unicamente, chamado também de URI (ver Seção 2.2.1), ou também Uniform Resource Locator (URL) W3C (2001), e é hospedado em um servidor conhecido como servidor web, ou servidor HTTP.

2.1.2 Navegador (Browser)

Um navegador é um software capaz de traduzir as páginas web e exibi-las adequadamente para o usuário final. O navegador se encaixa na parte do cliente do modelo arquitetural cliente/servidor e quando o navegador busca dados de um servidor na web utilizando o protocolo HTTP. Ele utiliza um software chamado motor de renderização que é capaz de exibir na tela conteúdos de acordo com o HTML, CSS e JavaScript das páginas web. Mas não apenas isso, ele sabe como salvar e obter os cookies nos dispositivos, permitem a instalação de plugins e extensões que podem ser adicionadas para personalizá-lo ou prover alguma funcionalidade.

Existem diversos navegadores hoje no mercado como por exemplo: Google Chrome, FireFox, Safari, Internet Explorer, Edge e muitos outros. Cada um com seu estilo e

complexidade. Hoje um navegador possui diversas funcionalidades não servindo apenas para exibir uma página web.

2.2 Protocolo HTTP

O HTTP, ou Protocolo de Transferência de Hipertexto é um protocolo de rede da camada de aplicação do modelo Open System Interconnection (OSI) e é a base para a comunicação na WWW. Os padrões deste protocolo são estabelecidos pelas Request for Comments (RFC). A primeira RFC que introduziu o HTTP/1.0 foi a RFC 1945, porém atualmente já é possível utilizar o HTTP/3 que está em versão beta.

O HTTP é um protocolo sem estado e baseado na arquitetura cliente/servidor, onde o cliente inicia uma conexão requisitando algum recurso do servidor e este responde com o recurso ou mensagens sobre o processamento da requisição, como mensagens de erro, redirecionamentos, etc. A Figura 2.1 ilustra como essa arquitetura funciona.

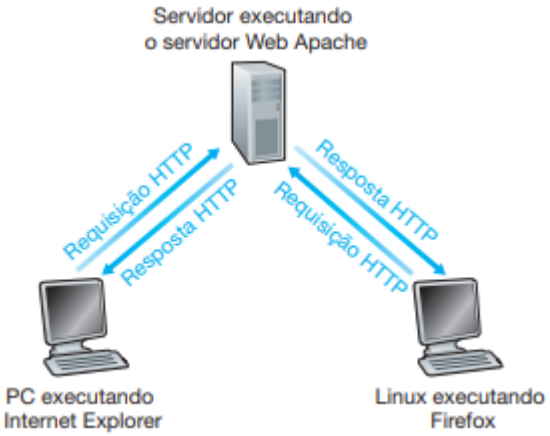


Figura 2.1: Arquitetura cliente/servidor aplicado ao HTTP (Kurose, 2013).

As requisições e respostas trocadas entre o cliente e o servidor são chamadas de mensagens HTTP e possuem algumas características em comum, tais como:

- Uma linha inicial informando a versão do protocolo e informações pertinentes as requisições/respostas, exemplificadas nas seções 2.2.2 e 2.2.3.

- Um conjunto opcional de cabeçalhos que são enviados logo após a primeira quebra de linha.
- Um corpo que pode conter dados que serão enviados para o servidor, ou dados recebidos do servidor.

A seguir serão abordados os demais elementos das mensagens HTTP. No primeiro momento serão discutidos as diferenças entre uma requisição e resposta através das seções 2.2.2 e 2.2.3, e posteriormente suas características em comum através das seções 2.2.4 e 2.2.5.

2.2.1 Uniform Resource Identifier

O URI ou também Uniform Resource Locator (URL) (W3C, 2001), é um identificador único para um recurso na Internet. Exemplo de um URI:

Protocolo : //**dominio/host** : **porta/caminho/recurso**?**query-params**#**fragmento**

Sendo que cada elemento especifica:

- ***Protocolo***: Especifica qual protocolo de comunicação está sendo utilizando (HTTP, File Transfer Protocol (FTP), etc).
- ***Dominio/Host***: O endereço do detentor do recurso.
- ***Porta***: Qual porta o detentor do recurso está utilizando. No caso do HTTP a porta padrão é a 80
- ***Caminho***: Todo o caminho até chegar ao recurso.
- ***Recurso***: O arquivo ou recurso sendo requisitado.
- ***Query-Params***: Os query params assim como o fragmento são opcionais e adicionam informações que são enviadas para o servidor.
- ***Fragmento***: É uma parte ou posição específica dentro do recurso

Dessa forma o cliente consegue buscar o recurso através da Internet. Mais especificamente na web, para um cliente encontrar um recurso desejado, é utilizado o protocolo HTTP. Um URI completo para um recurso na web é dado a seguir:

$http://www.exemplo.com/index.html?q = textoDeBusca\#rodape$

2.2.2 Requisições - Cliente

A arquitetura cliente/servidor inicia com a abertura de conexão vinda do cliente. Ele envia um pedido ao servidor requisitando um recurso ou algum tipo de tratamento do mesmo. A requisição dificilmente é criada textualmente, geralmente uma requisição é oriunda da interação com o navegador ou ainda interações com APIs. A Figura 2.2 ilustra um cliente requisitando uma página web a um servidor Apache.

```
GET https://tools.ietf.org/html/rfc2616 HTTP/1.1
Host: tools.ietf.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:71.0) Gecko/20100101 Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://www.google.com.br/
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Figura 2.2: Exemplo de requisição HTTP.

A primeira linha de cada requisição deste protocolo é composta de três partes (Fielding et al., 1999). Um método HTTP, que indica a ação desejada a ser executada para um determinado recurso, o caminho para esse recurso ou também chamado de URI e por fim a versão do protocolo que está sendo utilizado. A Tabela 2.1 descreve os métodos e quais são suas funcionalidades.

Método	Descrição
GET	Recebe informações ou objetos da web
HEAD	Solicita os cabeçalhos retornados de um recurso específico que foi requisitado por um método GET
POST	Envia informações para o servidor
PUT	Envia informações que podem ser alteradas no servidor
TRACE	Realiza um teste de loopback enviando uma mensagem por todo o caminho até o recurso alvo no qual foi destinado
CONNECT	Começa a comunicação bidirecional com o recurso solicitado
DELETE	Solicita a remoção de um recurso no servidor
OPTIONS	Verifica quais métodos estão disponíveis para determinado recurso no servidor

Tabela 2.1: Métodos HTTP.

2.2.3 Respostas - Servidor

Enquanto o cliente inicia a comunicação, o servidor é o responsável por processar o recurso requisitado e enviar uma resposta para o cliente, fechando o ciclo da arquitetura cliente/servidor. A Figura 2.3 ilustra uma mensagem de resposta HTTP.

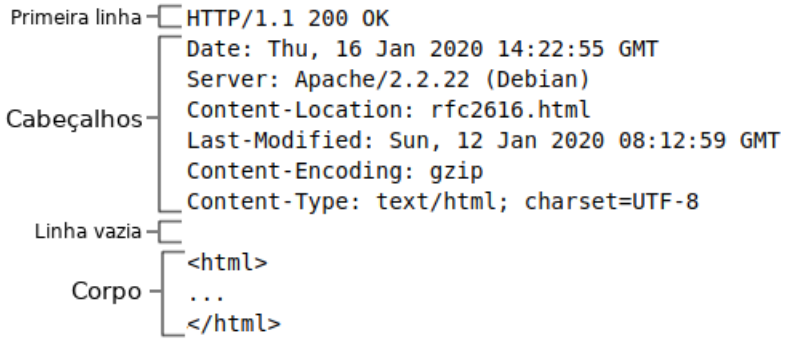


Figura 2.3: Exemplo de resposta HTTP.

Diferente de uma mensagem de requisição, a primeira linha de uma mensagem de resposta contém a versão utilizada do protocolo, seguido de um código de status HTTP resultante do processamento de cada pedido e uma breve descrição puramente informativa e textual do código de status. A Tabela 2.2 exemplifica os diferentes tipos de códigos que existem numa mensagem de resposta HTTP.

Código	**Descrição**
200+	Códigos 200 significam que o processamento do recurso foi concluído com sucesso
300+	Mensagens de redirecionamento
400+	Mensagens relacionada a erros do cliente durante o processamento do recurso.
500+	Mensagens relacionada a erros do servidor durante o processamento do recurso.

Tabela 2.2: Códigos HTTP.

2.2.4 Cabeçalho da mensagem

Os cabeçalhos do HTTP são enviados após a primeira quebra de linha do protocolo (ver Figura 2.3). São campos compostos por chave/valor case-insensitive (não diferencia letras maiúsculas e minúsculas) e são utilizados para enviar informações adicionais ou essenciais para o tratamento de uma mensagem entre o cliente e o servidor. Dessa forma, é comum enviar nos cabeçalhos por exemplo, tokens de autenticação do usuário, informar o tamanho e tipo do corpo da mensagem, etc. Os cabeçalhos seguem a seguinte estrutura:

Nome do cabeçalho : **Valor do cabeçalho**

2.2.5 Corpo da mensagem

Uma mensagem HTTP pode ou não conter um corpo que é enviado abaixo do cabeçalho, após uma linha vazia (ver Figura 2.3). Em uma resposta o corpo contém os recursos que o cliente solicitou em uma requisição, ou ainda uma mensagem de erro. Já em uma requisição, o corpo da mensagem pode conter dados que serão enviados para o servidor, de acordo com seu método HTTP. Quando a requisição ou a resposta possuem um corpo, pode ser incluído no cabeçalho entidades que especificam o tipo Multipurpose Internet Mail Extensions (MIME) do arquivo, bem como o tamanho do corpo em bytes. A Tabela 2.3 exemplifica alguns tipos possíveis do corpo das mensagens HTTP.

MIME	**Descrição**
application/json	Arquivo no formato JavaScript Object Notation (JSON).
text/html	Arquivo Hypertext Markup Language (HTML).
image/png	Imagem no formato PNG.
application/zip	Arquivo compactado.

Tabela 2.3: Exemplos de MIMEs possíveis do corpo das mensagens.

2.3 Página Web

Uma página web é um documento que pode ser mostrado em um navegador web. Esses documentos podem ser ligados uns com os outros, permitindo de uma maneira fácil e rápida a navegação. (Foundation, 2019b)

Essas páginas são compostas por três elementos, *HTML*, *CSS* e *JavaScript* que serão abordados a seguir.

2.3.1 HTML

O HTML, ou Linguagem de Marcação de Hipertexto, como o próprio nome sugere, é uma linguagem de marcação. Portanto não se deve confundir com linguagem de programação, uma vez que são utilizadas para propósitos diferentes. As linguagens de marcação, ou também, linguagens de apresentação, são utilizadas para preparar uma estrutura dos dados ou preparar o design da página. Elas definem um conjunto de regras para codificar documentos em um formato legível por humanos e pela máquina, e diferindo das linguagens de programação, não possuem nenhuma lógica ou algorítimo.

Um elemento HTML possui uma tag de abertura e na maioria dos casos uma de fechamento, podendo conter diferentes tipos de atributos. A Figura 2.4 exemplifica como um elemento "p" deve ser declarado.

Figura 2.4: Criando um paragrafo em HTML (Duckett, 2016).

A World Wide Web Consortium (W3C) diz que o HTML define a estrutura das páginas web e fornece os meios para:

- Publicar documentos online com títulos, texto, tabelas, listas, fotos, etc.
- Recuperar informações online através de links de hipertexto, com o clique de um botão.
- Criar formulários para a realização de transações com serviços remotos, para uso na busca de informações, reservas, pedidos de produtos, etc.
- Incluir planilhas, videoclipes, clipes de som e outros aplicativos diretamente em seus documentos.

O Código 2.1 exemplifica um arquivo básico em HTML.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Titulo da pagina</title>
5     </head>
6     <body>
7     </body>
8  </html>
```

Código 2.1: Documento HTML básico

2.3.2 CSS

Enquanto o HTML lida com a estrutura da página, o Cascading Style Sheets (CSS) trata de seu estilo, especificando regras de como um elemento deve aparecer (Duckett, 2016). Segundo a W3C, o CSS é a linguagem para descrever a apresentação de páginas da web, incluindo cores, layout e fontes (W3C, 2016a).

Ele permite adaptar todo conteúdo da página para diferentes dispositivos (responsividade), fazer operações 2D e 3D, animações, entre muitas outras coisas. Como toda linguagem, o CSS possui sua sintaxe. A Figura 2.5 exemplifica um código que altera a fonte de uma tag **p** do HTML.

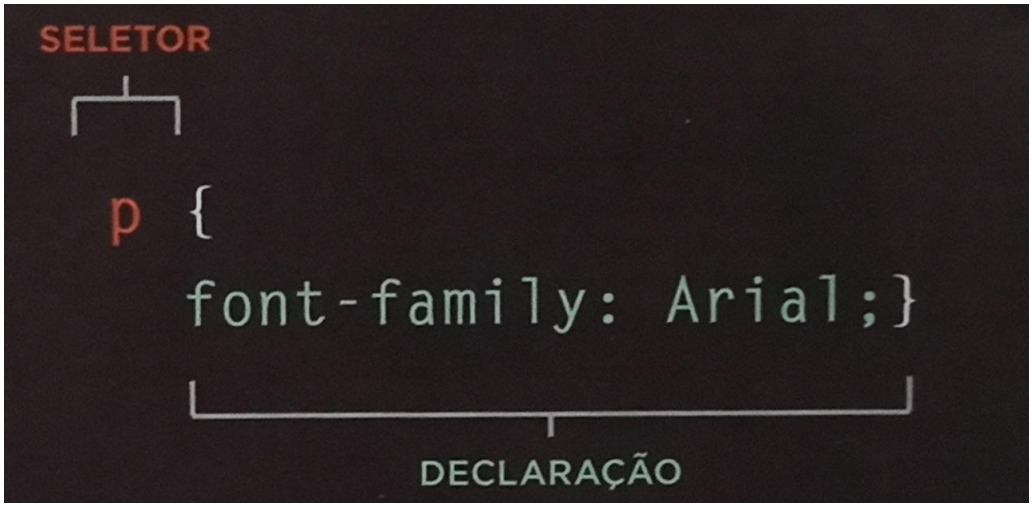


Figura 2.5: Exemplo de regra CSS aplicada a um paragrafo. (Duckett, 2016)

Todo código CSS possui um *seletor* e uma *declaração*. O seletor indica o(s) elemento(s) ao(s) qual(is) a(s) regra(s) se aplica(m). Existem basicamente cinco tipos de seletores:

- **Seletores de elemento HTML**: Especifica regras para todos os elementos HTML que estiverem nesse seletor.
- **Seletores com ID**: Especifica regras para um elemento específico que foi atribuído um ID. O id é escolhido pelo desenvolvedor.
- **Seletores com classe**: Esse seletor é utilizado juntamente com o atributo *class* em um elemento. Diferente do seletor com ID, muitos elementos em um documento podem pertencer a mesma classe.
- **Seletores de atributo**: Especifica outros atributos usando colchetes. Dentro dos colchetes é inserido o nome do atributo, opcionalmente seguido por um operador e valor.
- **Seletores de pseudo-classe**: Adiciona regras CSS aos elementos que possuem um estado especial, como :hover, :selected, etc.

O Código 2.2 exemplifica a utilização de cada seletor apresentado.

```
1   // Seletor HTML
2   p {
3       color: red;
4   }
5   // Seletor de pseudo-classe
6   p:hover {
7       color: black;
8   }
9   // Seletor de classe
10  .click {
11      cursor: pointer;
12  }
13  // Seletor de ID
14  #header {
15      border: 1px solid black;
16  }
17  // Seletor de Atributo
18  [disabled] {
19      color: white;
20  }
```

Código 2.2: Exemplificação de seletores e declaração de CSS.

Por fim, cada regra CSS possui uma *declaração*. A declaração especifica como os elementos referidos no seletor devem ser estilizados. Cada declaração possui duas partes e é dividida por campos propriedade e valor, separado por dois pontos. As propriedades indicam os aspectos que devem ser alterados, como por exemplo, *cor*, *borda*, etc., sendo que os valores especificam as configurações que serão utilizadas para uma propriedade específica.

2.3.3 JavaScript

Até então foi definida a linguagem de apresentação, que lida com a estrutura e estilos, descrevendo como uma página deve ser estruturada e exibida nas telas dos dispositivos. Porém uma página web faz mais do que apenas exibir dados estáticos de forma estruturada e elegante. Um script é uma série de instruções que um computador pode seguir para alcançar um objetivo (Duckett, 2014). Desse modo, quando uma página recebe atualizações oportunas, adiciona elementos, como um item de uma lista, com certeza o JavaScript estará atuando em segundo plano (W3C, 2016b).

O nome JavaScript é licenciado e registrado pela Oracle. Desde 1996 a linguagem segue padrões e normas estabelecidas pela European Computer Manufactures Association (ECMA) e teve seu nome alterado para ECMAScript desde então. Para tornar a leitura mais clara e evitar possíveis confusões, irá ser utilizado JavaScript e ECMAScript como sinônimos neste documento.

Dessa forma o ECMAScript ou JavaScript é uma linguagem de script que permite criar conteúdo com atualização dinâmica, controlar multimídias, manipular o HTML e CSS, validar dados, entre muita outras coisas. O Código 2.3, exemplifica um script em ECMAScript.

```
1        function calc(v1, v2) {
2            return function(callback) {
3                return callback(v1,v2);
4            }
5        }
6
7        var sum = calc(1,2);
8        sum((v1, v2) => {v1+v2})
```

Código 2.3: Exemplo de código JavaScript

Node.js

Com a ascensão da web e sua popularização, o JavaScript se tornou uma das linguagens de programação mais utilizadas no mundo (Stack Exchange, 2019). Porém, a única utilização da linguagem se dava no desenvolvimento de páginas web utilizadas apenas pelos navegadores, dessa forma não era possível ter acesso ao sistema de arquivos, ou as API do sistema operacional no geral.

Foi justamente nesse cenário, de popularização e a necessidade de independência dos navegadores, que Ryan Dahl em 2009 criou o Node.js, um interpretador capaz de interagir com o sistema operacional.

Por ser uma ferramenta orientada a eventos assíncronos e utilizar o ECMAScript, o Node.js se tornou popular e poderoso (Foundation, 2020), contribuindo para o desenvolvimento de aplicações web escaláveis (Foundation, 2019c), sendo utilizado em muitos servidores e serviços web. O Código 2.4 exemplifica a criação de um servidor HTTP com Node.js.

```
1  const http = require('http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World\n');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log('Server is running at ${port}');
```

```
14          });
```

Código 2.4: Criação de um servidor HTTP em Node.js (Foundation, 2019c)

TypeScript

A linguagem JavaScript é uma linguagem interpretada e fracamente tipada, podendo assumir diferentes tipos de valores ao decorrer da execução do código. Dessa forma é difícil garantir que os parâmetros e retorno das funções sejam o esperado. Para contornar esse problema foram criados diversos mecanismos para realizar a verificação de tipos no JavaScript, sendo o TypeScript uma dessas abordagens.

Desenvolvido (2012) e mantido pela Microsoft, o TypeScript é um super conjunto (do inglês, superset) do JavaScript que provê funcionalidades que ainda não estão disponíveis nativamente ou requerem um grande esforço para sua utilização, como a tipagem dos dados e a orientação a objeto (Fenton, 2017).

Dessa forma além de permitir a tipagem estática, facilitando o auto complemento das Integrated Development Environments (IDEs), o TypeScript adiciona funcionalidades que ainda são candidatas a entrarem como uma especificação do JavaScript, ou seja, o desenvolvedor pode utilizar funcionalidades que ainda não estão disponíveis nativamente, gerando código nativo através de seu mecanismo de transpilação. O Código 2.5 exemplifica a transpilação de TypeScript para JavaScript.

```
1  class TypeScript {
2      set setter1(val: boolean) {
3          this.varBoolean = val;
4      }
5      constructor(
6          public var1: number,
7          private var2: string,
8          private varBoolean: boolean
```

```
9             ) { }
10
11            funcao() {
12                const test: number[] = [];
13                const test2 = test.map((value: number) => {
14                    return value * 2;
15                });
16            }
17        }
18        // codigo transpilado para ES5
19        "use strict";
20        var TypeScript = /** @class */ (function () {
21            function TypeScript(var1, var2, varBoolean) {
22                this.var1 = var1;
23                this.var2 = var2;
24                this.varBoolean = varBoolean;
25            }
26            Object.defineProperty(TypeScript.prototype, "
                  setter1", {
27                set: function (val) {
28                    this.varBoolean = val;
29                },
30                enumerable: true,
31                configurable: true
32            });
33            TypeScript.prototype.funcao = function () {
34                var test = [];
35                var test2 = test.map(function (value) {
36                    return value * 2;
37                });
```

```
38              };
39              return TypeScript;
40          }());
```

Código 2.5: Exemplo de código TypeScript transpilado para JavaScript (ES5)

2.4 Application Programming Interface - API

A API pode ser vista como um contrato simples (a interface) entre o aplicativo que a oferece e outros elementos, tais como software ou hardware de terceiros (Foundation, 2019a).

Enquanto a interface de um sistema é feita para que um usuário final possa utilizá-la, as APIs são criadas para que um sistema consiga utilizar funcionalidades de um outro sistema. Portanto, a API especifica um conjunto de funções ou serviços que estão disponíveis para que um outro sistema consiga acessá-los bastando conhecer sua interface para realizar a comunicação.

Sendo assim, elas proporcionam a integração entre diversos sistemas, totalmente desacopladas da lógica e também da linguagem de programação envolvida em seu desenvolvimento. A Figura 2.6 exemplifica o uso de diversos sistemas utilizando serviços de uma API.

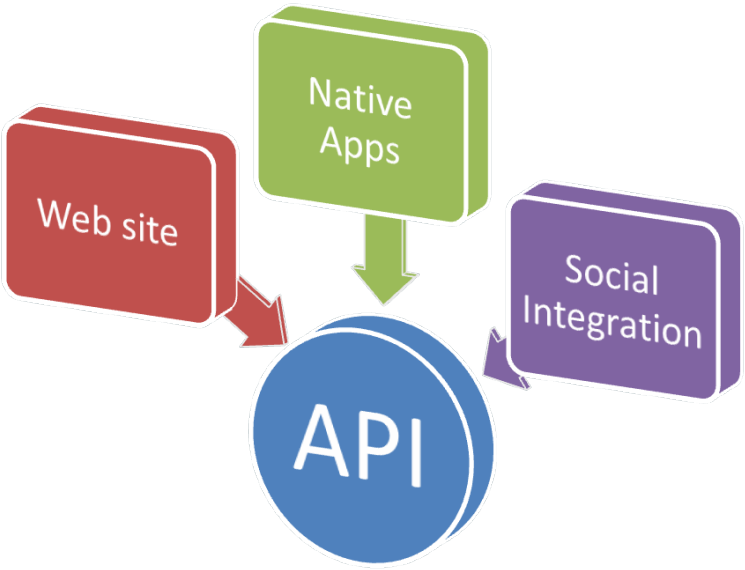


Figura 2.6: Exemplo de utilização entre diversos sistemas a uma API. (Pires, 2016)

Um exemplo de padrão arquitetural no desenvolvimento de API e muito utilizado em serviços web é o padrão de Transferência Representacional de Estado, ou Representational State Transfer (REST). As API RESTful permitem que os sistemas solicitantes acessem e manipulem os dados e recursos da web usando um conjunto uniforme e predefinido de operações sem estado.

Porém não existem apenas APIs RESTful que utilizam o protocolo HTTP. As APIs estão presentes no sistema operacional, disponibilizando funcionalidades para controlar o sistemas de arquivos, permissões, rede, etc. Até mesmo o navegador provê algumas APIs, como por exemplo a geolocalização, que pode ser usada para recuperar informações de localização de qualquer serviço que o usuário tenha disponível em seu dispositivo, API de armazenamento (local storage), entre muitos outros exemplos.

2.5 Front-end e Back-end

Front-end e back-end são termos utilizados por profissionais da área da informática que descrevem camadas que compõe um software, site e até mesmo o hardware.

O front-end, lida com a parte em que há interação com o usuário, ou seja, um desenvolvedor front-end se preocupa com a experiência do usuário. Na web um desenvolvedor front-end geralmente utiliza HTML, CSS, JavaScript, ou alguma ferramenta/framework visando otimizar seu desenvolvimento, como o bootstrap, Angular, React, Vue, etc.

Já o back-end é a parte responsável pela regra de negócio, segurança, banco de dados, entre outras atividades do servidor. Um back-end na web geralmente está envolvido com o desenvolvimento de uma API, que responderá para uma ou mais aplicações, adicionando, removendo, atualizando, testando, entre outras coisas, mas de uma maneira geral servindo os dados e provendo também sua segurança. Como no front-end, também existem tecnologias que agilizam o seu desenvolvimento. Algumas delas são: Node.js, Ruby, PHP, etc.

Dessa forma, como cada camada descreve e exige diferentes habilidades, muitos desenvolvedores focam em áreas específicas, das quais possuem maior afinidade. Porém existem profissionais que atuam em ambas as partes sendo denominados desenvolvedores full-stack.

2.6 Object-relational mapping - ORM

Um ORM é uma técnica de desenvolvimento que permite estabelecer uma relação entre os objetos de uma linguagem de programação orientada a objetos - (OO), com os dados que os representam no banco de dados. Geralmente as tabelas da base de dados são representadas por classes e os objetos representam instâncias do modelo. Com ele é possível realizar consultas em Structured Query Language (SQL) sem necessariamente escrever nenhuma query, acelerando a produtividade e mantendo a persistência dos dados. Os ORM também auxiliam na segurança ao banco de dados, impedindo falhas e brechas já conhecidas.

A Figura 2.7 exemplifica a interação entre a linguagem OO e o banco relacional.

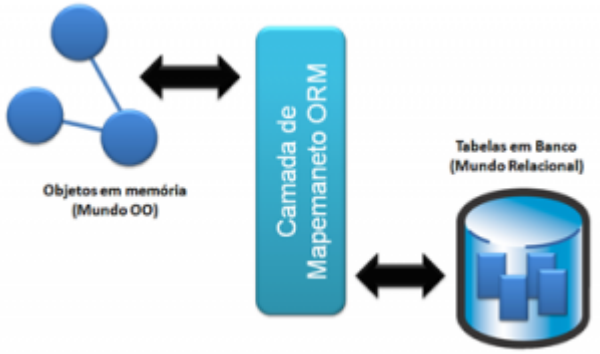


Figura 2.7: Exemplo do funcionamento básico de um ORM. (DevMedia, 2011)

Existem diversos ORM para as mais variadas linguagens de programação. Alguns deles são: Sequelize (JavaScript), Eloquent (PHP), Hibernate (Java), SQLAlchemy (Python), entre outros.

2.7 Single Page Application - SPA vs Multi Page Application - MPA

Uma aplicação web geralmente segue duas principais estratégias para sua criação, sendo elas: Multiple Page Application (MPA) e Single Page Application (SPA). Elas buscam solucionar diferentes tipos de problemas e cabe ao desenvolvedor analisar o escopo do projeto e aplicar a melhor estratégia em seu caso de uso. A Figura 2.8 ilustra a diferença entre essas duas abordagens.

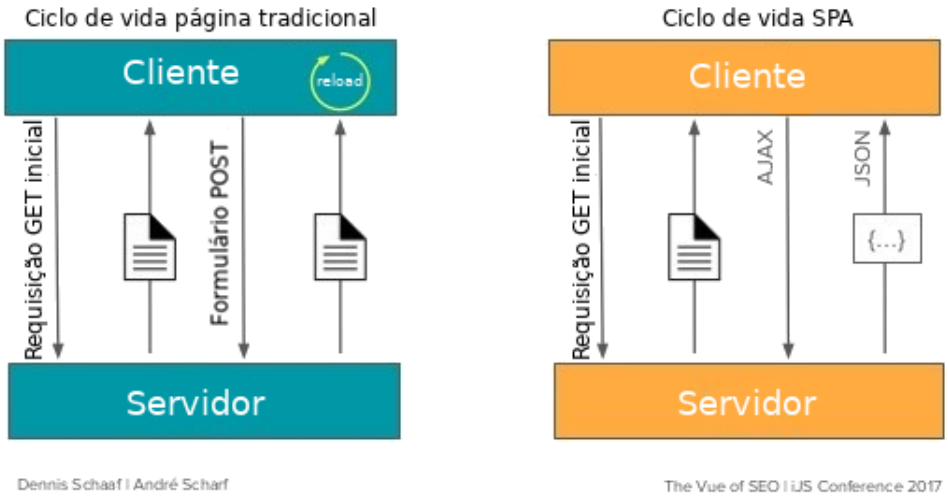


Figura 2.8: Diferença entre SPA e MPA.

As MPAs, como o nome sugere, são aplicações que possuem diferentes páginas para os recursos do site. As páginas são geradas dentro do servidor e à medida que o usuário interage com a aplicação, é enviado uma nova página para o cliente, sendo necessário o recarregamento da mesma.

Os recarregamentos constantes dentro de uma MPA podem comprometer a experiência do usuário. Para diminuir esse problema, elas utilizam o mecanismo de requisições via Asynchronous Javascript and XML (AJAX), que permite uma página enviar uma requisição ao servidor, requerendo um dado específico, tais como: imagens, arquivos JSON, Extensible Markup Language (XML), etc. Porém as MPAs utilizam o AJAX na maioria das vezes para renderizar alguns componentes dentro da aplicação, mas certamente em algum momento, ou interação com o usuário, será necessário o envio de uma nova página, seja para renderizar um formulário ou até na listagem de elementos.

Já nas SPAs toda a aplicação possui apenas uma única página web e à medida que o usuário interage com a aplicação são atualizados os dados da tela com a manipulação dos elementos da página pelo próprio navegador, evitando o recarregamento da página. Dessa forma, a primeira requisição retorna um único HTML e o servidor age como um serviço fornecendo dados e também scripts para a aplicação.

Empresas como a Google, Facebook, Microsoft criaram e utilizam mecanismos para construir SPAs. As principais bibliotecas e frameworks dessas empresas e que também se destacam hoje são: AngularJS, Angular, React e Vue. Eles se diferenciam no mecanismo responsável por detectar as mudanças no Document Object Model (DOM). Enquanto o React e Vue utilizam o Virtual DOM (VDOM) o Angular utiliza o Change Detector (CD), ambos serão discutidos na Seção 2.7.1.

Em resumo, as SPAs se tornam mais atrativas quando o assunto é experiência do usuário. Porém um dos grandes problemas com essa abordagem é a dificuldade que os mecanismos de busca possuem ao detectar suas páginas. Dessa forma para sites que dependem de uma otimização nos motores de busca (do inglês, Search Engine Optimization (SEO)), como comércios eletrônicos, sites de notícias a escolha de uma MPA pode ser a melhor opção.

2.7.1 Algoritmos de detecção de mudanças

O DOM é uma representação em árvore do modelo de objetos que está sendo utilizado pela página web. A medida que uma página cresce, seu DOM também aumenta. Visto que uma página pode ficar muito grande, tendo diversas dependências entre seus filhos, uma mudança realizada nessa árvore pode fazer com que toda a página seja re-renderizada. Isso significa que uma simples mudança de cor, ou ainda esconder um elemento, obriga toda árvore do DOM alterar e reposicionar seus elementos. Esse processo pode demandar muito processamento e tempo. Para controlar esse problema, as SPAs implementam diferentes tipos de mecanismos, que serão abordadas a seguir.

Virtual DOM - VDOM

O VDOM é uma cópia do DOM original, guardada em memória. Sua utilização visa otimizar o tempo de renderização das páginas web em uma SPA, melhorando a performance da manipulação do DOM real, alterando apenas elementos que sofreram alterações.

Segundo Minnick (2016), o mecanismo de verificação de mudança utilizando o VDOM segue basicamente três passos:

- Sempre que algo possa ter sido alterado, a interface do usuário inteira será renderizada novamente em uma representação do DOM virtual.
- A diferença entre a representação anterior do Virtual DOM e a nova será calculada.
- O DOM real será atualizado com o que realmente mudou.

Dessa forma o objetivo do VDOM é realizar todas as alterações em um DOM virtual, que é armazenado na memoria, e em seguida aplicar no DOM original da página, ganhando performance, visto que somente os elementos que realmente sofreram mudanças são alterados.

Change Detector - CD

O CD é o mecanismo que o Angular utiliza para lidar com a problemática da manipulação do DOM original. Ele age de forma bidirecional, detectando mudanças tanto no modelo (template, ou modelo HTML), como no componente (classe que manipula dados e controla o modelo). Para conseguir detectar as mudanças, o Angular sobrescreve algumas funções da API nativa do navegador, como por exemplo o addEventListner, O Código 2.6 fornecido pelo Angular Core, exemplifica tal mudança (Core, 2019).

```
1  // nova versao do addEventListener
2  function addEventListener(eventName, callback) {
3    // chama o addEventListener real
4    callRealAddEventListener(eventName, function
       () {
5    primeiro chama o callback original
6      callback(...);
7    // executa funcionalidades especificas do
       Angular
```

```
8                   var changed = angular2.runChangeDetection
                        ();
9                    if (changed) {
10                       angular2.reRenderUIPart();
11                   }
12               });
13           }
```

Código 2.6: Sobrescrita do método addEventListner Angular

Além disso é mudado também todos os eventos do navegador (clique, mouseover, keyup etc.), as funções setTimeout/setInterval e por fim as solicitações AJAX.

Cada componente Angular possui um detector de alterações associado, criado no momento da inicialização do aplicativo. Esse detector realiza o seguinte trabalho: para cada expressão usada no modelo, ele compara o valor atual da propriedade usada na expressão com o valor anterior dessa propriedade. Caso o valor da propriedade antes e depois for diferente, essa alteração é computada e realizada no DOM original.

Por norma, o Angular não verifica valores aninhados, visando otimizar o desempenho, ele monitora somente as variáveis que estão sendo utilizadas no modelo. O Código 2.7 ilustra a ligação bidirecional entre o modelo e sua classe de componente.

```
1            @Component({
2                selector: 'app-root',
3                template: `
4                // O proximo numero de 3 sera
5                <h1>{{titulo}} {{num1}} sera:</h1>
6                <div>{{num1 + 1}}</div> // 4
7            })
8            export class AppComponent {
9              titulo = 'O proximo numero de ';
10             num1 = 3;
```

```
11                 constructor(){}
12             }
```

Código 2.7: Ligação bidirecional através de interpolação em angular

Seguindo as recomendações do Angular Core, nesse exemplo foi criado um componente simples, dentro da classe do component AppComponent. Toda vez em que *title*, ou *num1*, ou algum tipo de interação acontecer com essas variáveis, será acionado o detector de mudanças, disparando as verificações e métodos de lifecycle (ciclo de vida) do Angular, que possibilitam adicionar alguma lógica sempre que ocorrer uma alteração no componente.

2.8 Serviços Web

Um serviço web é uma API envolvida (do inglês, wrapped) pelo HTTP. Dessa forma ele também é isolado e independente de linguagem de programação e sistemas operacionais, que visa realizar a comunicação de máquina-para-máquina (Richardson and Ruby, 2007). A principal diferença entre uma API e um serviço web é que ele precisa estar conectado a Internet (web) para realizar a comunicação, porém ambos provêm a interoperabilidade entre diferentes sistemas.

Dentre o ecossistema dos serviços web duas principais tecnologias se destacam, sendo elas: O *SOAP* e *REST* que serão discutidos nas próximas seções.

2.8.1 REST

O REST é um padrão arquitetural que define um conjunto de restrições na criação de um serviço web. O REST foi baseada no protocolo HTTP/1.0 e é produto da tese de Phd de Roy Fielding defendida em 2000. Os serviços da web que respeitam as condições do REST são chamados de RESTful e permite que os solicitantes acessem os recursos desejados através de um URI (W3C, 2001).

Diferentemente do Simple Object Access Protocol (SOAP) que expõem um conjunto arbitrário de operações, o REST utiliza um conjunto predefinido e uniforme de operações

sem estado. Para que um serviço seja RESTful, ele precisa seguir seis restrições (Erl, 2012). Sendo elas:

- **Arquitetura cliente/servidor**: Isso significa que o aplicativo do cliente deve evoluir separadamente sem depender do servidor, ou seja, há a separação de preocupações, dessa forma o cliente deve apenas conhecer o URI dos recursos. Portanto cliente e servidor podem ser substituídos e desenvolvidos independentemente.

- **Sistema em camadas**: O REST permite que seja utilizado diversos serviços intermediários, dessa forma, um cliente não pode dizer se está conectado ao servidor final ou a um intermédio ao longo de toda requisição. Mesmo que um proxy seja colocado entre o cliente e o servidor, o cliente não sofrerá alterações em sua comunicação com o servidor.

- **Sem estado**: Essa restrição é reflexo do embasamento de Roy sobre o protocolo HTTP, fazendo com que em todas as interações entre cliente e servidor não seja guardado nada sobre as mesmas. Dessa forma todas solicitações devem contém as informações necessárias para serem efetuadas com sucesso.

- **Armazenamento em cache**: No REST, o armazenamento em cache deve ser aplicado aos recursos, sendo que, esses recursos devem se declarar armazenados em cache. O armazenamento em cache é de extrema importância trazendo um aprimoramento do desempenho para o cliente e melhor escalabilidade para o servidor, visto que a carga é reduzida.

- **Código sob demanda (opcional)**: Esta é uma restrição opcional e permite retornar um código executável do servidor, podendo ser um widget para a interface do usuário, um script, etc.

- **Interface uniforme**: A restrição da interface uniforme possui quatro restrições que estão relacionadas a interface para os recursos dentro do sistema, tais restrições fazem com que solicitações diferentes possuam a mesma aparência. São elas:

- Recursos individuais são identificados a cada solicitação. (Em serviços da web essa identificação é feita utilizando o URI).
- As informações que o servidor retorna inclui todas as informações necessárias para que um cliente consiga alterá-las.
- Cada mensagem inclui informações suficientes para serem processadas, tanto do lado do cliente enviando ao servidor, quanto do servidor enviando para o cliente.
- A restrição de hipermídia como mecanismo do estado faz com que um cliente possa navegar dinamicamente entre links fornecidos pelo servidor, conseguindo acessar diferentes ações e a medida em que ele navega, o servidor envia outros hiperlinks com as ações que estão disponíveis para o cliente.

2.8.2 SOAP

Sendo mais antigo que o REST o SOAP surgiu justamente com objetivo de prover a interoperabilidade entre diferentes softwares. Diferentemente do REST, essa tecnologia não é um padrão arquitetural, mas sim um protocolo baseado em XML que estabelece uma interface para que os demais sistemas consigam acessar e utilizar seus serviços. Essa interface é também conhecida como Web Services Description Language (WSDL), e define o contrato do serviço, ou seja, descreve quais tipos de serviços existem, e como acessá-los.

Após definido a WSDL, os clientes podem acessar tais funcionalidades enviando uma mensagem SOAP, que é composta pelos seguintes campos:

- *Envelope*: Campo requerido. Trata do elemento raiz de uma mensagem SOAP. Este elemento define o documento XML como uma mensagem SOAP.
- *Header (cabeçalho)*: Campo opcional. Contém informações específicas do aplicativo sobre a mensagem SOAP, podendo conter metadados. Os atributos definidos no cabeçalho SOAP definem como um destinatário deve processar a mensagem SOAP.

- *Body (corpo)*: Campo necessário, contém o conteúdo real da solicitação ou da resposta.

Dessa forma, conhecendo a interface do serviço web, é possível realizar chamadas para esse serviço e assim utilizá-lo. A Figura 2.9 ilustra uma mensagem SOAP, sendo enviada utilizando o protocolo HTTP.

Figura 2.9: Mensagem SOAP utilizando HTTP.

Segundo Mulligan and Gračanin (2009), o REST se sobressai em termos da largura de banda da rede utilizada na transmissão de solicitações de serviço pela Internet e também em sua latência. Além disso a arquitetura REST quando atrelada com o protocolo HTTP e comparada com o SOAP é consideravelmente menos complexa e verbosa, não sendo necessário o uso do XML para a transmissão dos dados e nem a criação de uma WSDL.

Capítulo 3

Projetando a aplicação

Antes da fase de desenvolvimento é essencial realizar a coleta de requisitos, escolher as ferramentas que auxiliam na produção do software e também realizar sua modelagem. Este capítulo decorre sobre tais processos seguindo a estrutura: primeiramente na Seção 3.1 é apresentado os requisitos funcionais e não funcionais do sistema e sua representação com o diagrama de casos de uso, depois na Seção 3.2 é apresentado uma visão geral dos sistemas e módulos da empresa, e quais são seus papéis no desenvolvimento do sistema proposto por esse trabalho. Em sequência, a Seção 3.3 apresentada a modelagem dos dados através do Modelo Entidade Relacionamento (MER). A Seção 3.4 discorre sobre como foi implementado a metodologia *SCRUM* ao projeto e na Seção 3.5 é apresentado algumas ferramentas utilizadas durante a fase de desenvolvimento. Por fim as conclusões finais referentes a esse capítulo são abordadas na Seção 3.6. A Figura 3.1 ilustra os passos seguidos para o planejamento do sistema.

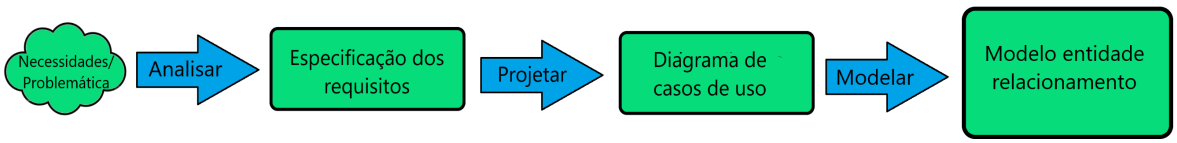


Figura 3.1: Etapas do planejamento do software.

3.1 Requisitos

A primeira etapa da modelagem do projeto começa com a coleta dos requisitos do sistema. Tais requisitos são divididos em requisitos funcionais, ou ainda o que o sistema deve fazer, e requisitos não funcionais, que definem como sistema irá realizar tais funcionalidades, bem como descrever as suas características.

3.1.1 Requisitos funcionais

Os requisitos funcionais foram coletados através de reuniões com os colaboradores da empresa, elencando as principais funcionalidades que um inquilino utilizaria em seu cotidiano. Ao total foram levantados 12 casos de usos, e três tipos de usuários, como detalha a Tabela 3.1: cliente, colaborador e o módulo de gestão financeira.

Ator	**Descrição**
Cliente	Usuário principal da aplicação, refere-se a um inquilino da empresa *Riskivector*
Colaborador	Funcionários da empresa que fornecem suporte aos clientes, bem como validações e gerência de recursos.
Módulo de gestão financeira	Todas as operações financeiras da aplicação comunicam-se com o módulo financeiro da empresa.

Tabela 3.1: Atores do sistema.

Cada requisito do sistema foi mapeado para um caso de uso, gerando assim o diagrama de caso de uso exemplificado através da Figura 3.2. Cada caso de uso é nomeado com "UC" seguido de um identificador. A seguir será apresentado uma breve descrição e os respectivos atores de cada caso de uso.

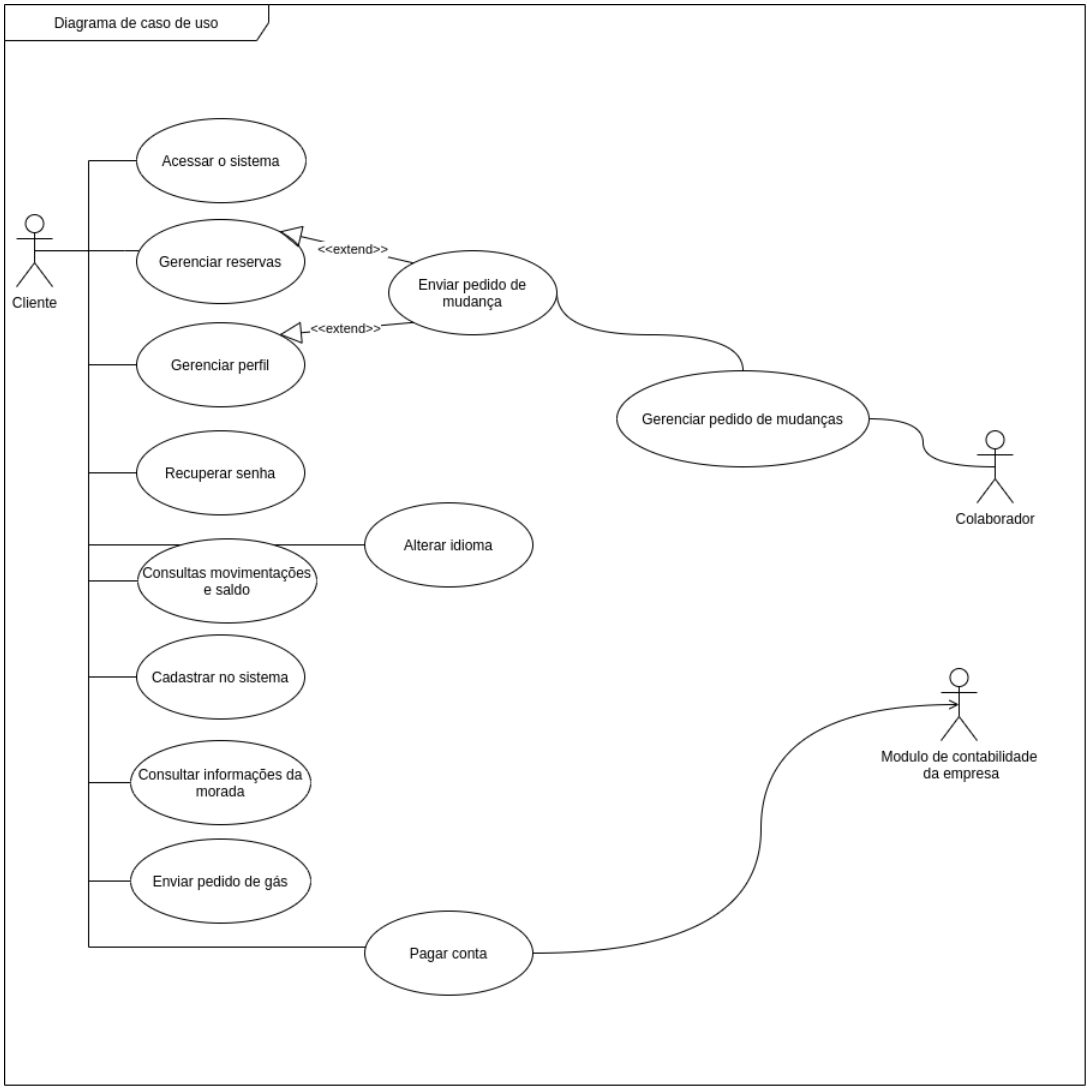


Figura 3.2: Diagrama de casos de uso do sistema.

- **[UC01] Cadastrar no sistema**
  - **Ator(es):** Cliente
  - **Descrição:** O cliente deve ser capaz de se cadastrar no novo sistema, adquirindo uma senha e e-mail. O cadastro deve ser gerado a partir do momento em que um cliente requisita uma reserva.
- **[UC02] Acessar sistema**

  - **Ator(es):** Cliente
  - **Descrição:** O cliente deve ser capaz de entrar no novo sistema. Ele poderá efetivar o login caso não esteja bloqueado por qualquer motivo na aplicação.

- **[UC03] Recuperar senha**
  - **Ator(es):** Cliente
  - **Descrição:** O cliente deve ser capaz de recuperar sua senha. Assim como no login do sistema, ele poderá efetivar a recuperação apenas se não estiver bloqueado por qualquer motivo na aplicação. O cliente têm 15 minutos para efetuar a mudança após sua solicitação.

- **[UC04] Gerenciar reservas**
  - **Ator(es):** Cliente
  - **Descrição:** Um inquilino pode requisitar, editar e consultar uma reserva. Ele pode requisitar quantas reservas forem necessárias e editar apenas reservas que ainda não foram confirmadas. A edição de uma reserva deve passar pela aprovação de um colaborador. Após requisitada a reserva ele deve ser capaz de acessar suas informações na aplicação, podendo filtra-las de acordo com seu status.
  - **Ações**
    - **[UC04.1] Requisitar uma nova reserva de quarto.**
    - **[UC04.2] Editar reservas de status não confirmado.**
    - **[UC04.3] Consultar reservas.**

- **[UC05] Gerenciar perfil**
  - **Ator(es):** Cliente
  - **Descrição:** Um cliente deve ser capaz de consultar e editar as informações do seu perfil. Entretanto nem todas as informações do inquilino podem ser

editadas, pois uma alteração pode implicar em inconsistência com os dados do escritório. Dessa forma, assim como as reservas, algumas informações do perfil do inquilino devem ser autorizadas pela empresa.

  - **Ações**
    - **[UC05.1] Consultar informações pessoais.**
    - **[UC05.2] Editar informações pessoais.**

- **[UC06] Consultar movimentações e saldo**
  - **Ator(es):** Cliente
  - **Descrição:** Exibe o saldo e histórico de cada movimentação do cliente. Uma movimentação pode ser um pagamento de aluguel, contas do apartamento (água, luz, etc.) e contas diversas, como cobrança de taxas, descontos, etc. O cliente deve ser capaz de filtrar as movimentações pelo período que desejar e também pelo seu tipo.
  - **Ações**
    - **[UC06.1] Consultar movimentações.**
    - **[UC06.2] Consultar saldo.**

- **[UC07] Consultar informações da morada**
  - **Ator(es):** Cliente
  - **Descrição:** Um inquilino pode alugar diversos apartamentos. Dessa forma ele deve ser capaz de acessar informações de suas moradas, tais como endereço, histórico de contas (água, gás, etc).

- **[UC08] Enviar pedido de gás**
  - **Ator(es):** Cliente
  - **Descrição:** O inquilino deve ser capaz de realizar pedidos de gás, informando a morada e quantidade de botijas.

- **[UC09] Pagar conta**
  - **Ator(es):** Cliente e módulo de gestão financeira.
  - **Descrição:** Um inquilino pode pagar suas contas através do sistema. O pagamento é feito inserindo os dados de seu cartão e informando o valor que está pretendendo pagar. Após efetuado com sucesso, ele pode acessar o recibo e verificar seu novo saldo.
- **[UC10] Gerenciar pedido de mudanças**
  - **Ator(es):** Colaborador
  - **Descrição:** Os inquilinos não podem realizar algumas mudanças sem a verificação de um colaborador. Portanto, sempre que um inquilino requisitar tais mudanças será enviado uma solicitação de alteração à empresa. Um colaborador deve aceitar ou rejeitar essa alteração, bem como ser capaz de reverter a ação realizada. Tais mudanças podem se originar de diferentes elementos, podendo ser uma reserva, mudança no perfil, entre outros.
  - **Ações**
    - **[UC10.1] Consultar pedidos de mudanças.**
    - **[UC10.2] Validar/invalidar pedido de mudança.**
    - **[UC10.3] Adicionar observações a respeito da solicitação.**
    - **[UC10.4] Reverter alterações realizadas.**
- **[UC11] Alterar idioma**
  - **Ator(es):** Cliente
  - **Descrição:** O inquilino pode escolher dentre os idiomas disponíveis na aplicação, tais como inglês, português, etc.
- **[UC12] Enviar pedido de mudança**
  - **Ator(es):** Cliente

– **Descrição:** Os recursos que um inquilino não pode alterar diretamente é enviado um pedido de mudança para a empresa. Cada pedido deve informar o solicitante e qual(is) dado(s) deseja alterar.

3.1.2 Requisitos não funcionais

- **Interface gráfica**: O sistema deve ser simples e objetivo para o cliente.
- **Banco de dados**: Deve ser utilizado MySQL.
- **Metodologia**: Deve-se utilizar uma metodologia ágil para o desenvolvimento da aplicação web.
- **Plataforma**: O sistema deve ser multi-plataforma.
- **Linguagem de programação**: Deve ser utilizado a linguagem TypeScript.

3.2 Integração entre sistemas - visão geral

O desenvolvimento desse projeto envolve a interoperabilidade entre os sistemas existentes na empresa. A Figura 3.3 exibe uma visão geral sobre as aplicações e módulos da empresa *Riskivector*.

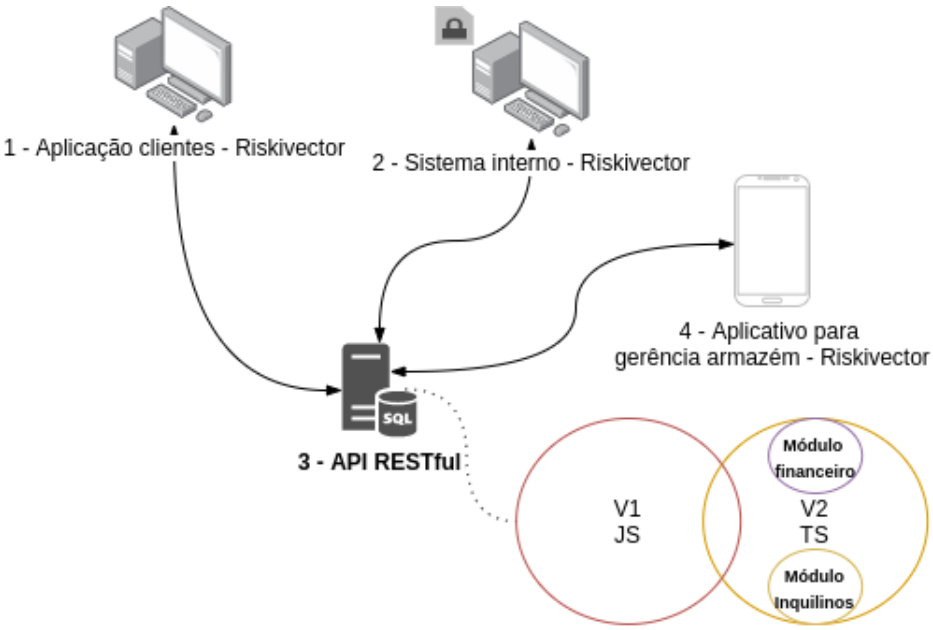


Figura 3.3: Sistemas e servidores da empresa *Riskivector*.

Antes do desenvolvimento desse projeto, a empresa mantinha dois projetos. O primeiro envolve a aplicação interna da empresa, representado na Figura 3.3 pelo número (2), onde apenas os colaboradores possuem acesso. O segundo, representado pelo número (3), é o servidor ou serviço que provê a gerencia dos dados da empresa.

O desenvolvimento desse projeto acrescentou mais uma aplicação nesse ecossistema, representada pelo número (1). Tal aplicação realiza a comunicação com o serviço de dados (3), porém para as novas funcionalidades propostas para esse sistema, foi criada uma nova versão do servidor (3). A primeira versão foi mantida visando a compatibilidade com os sistemas que as utilizavam. Já a nova versão foi reservada para os sistemas emergentes, como o proposto nesse trabalho.

Dessa forma, para o funcionamento correto da aplicação dos clientes (1), foi necessário o desenvolvimento em três sistemas da empresa, sendo o primeiro a aplicação web desenvolvida em Angular (1), a implementação do módulo dos inquilinos na nova versão do servidor, bem como a consultar o módulo financeiro (3), e também a implementação de algumas funcionalidades no sistema interno da empresa (2), como representa o caso de uso **[UC10]**, ilustrado na Seção 3.1.

3.3 Modelo de Entidade Relacionamento - ER

Após a etapa de coleta de requisitos, iniciou-se a etapa da modelagem de dados. A modelação foi fundamentada na base de dados já existente da empresa.

Dessa forma, para adicionar as funcionalidades deste trabalho, foram alterados atributos e adicionadas novas entidades. A modelagem do sistema de contabilidade é resumida, visto que se trata de um módulo externo ao sistema proposto. A Figura 3.4 ilustra o modelo de Entidade Relacionamento (ER).

Um dos desafios da modelagem e que demonstra a importância da mesma, foi projetar o requisito encapsulado nos casos de uso **Gerenciar pedidos de mudança** e **Enviar pedido de mudança**. Visto que um inquilino pode pedir uma mudança para diferentes recursos, o banco de dados precisa armazenar diferentes tipos de dados (varchar, date, integer, tinyint, etc.) em um único local. Como não é possível ter um atributo com seu tipo variável, a primeira modelação tinha o objetivo de salvar os dados em JSON.

Porém essa abordagem limita a capacidade de realização de consultas no banco, uma vez que os dados estão no formato JSON, verificações como, maior, menor, etc. implicaria em sua decodificação. Dessa forma a solução para esta problemática foi modelar o banco de dados para armazenar os dados em texto puro e posteriormente na fase de implementação, seria adicionado gatilhos (do inglês, triggers) para realizar o trabalho da conversão de texto para os demais tipos (boolean, int, etc.). Toda implementação desse caso de uso é descrita no Capítulo 4 e é representada no modelo de ER pelas entidades tenant_change_request e tenant_change_item.

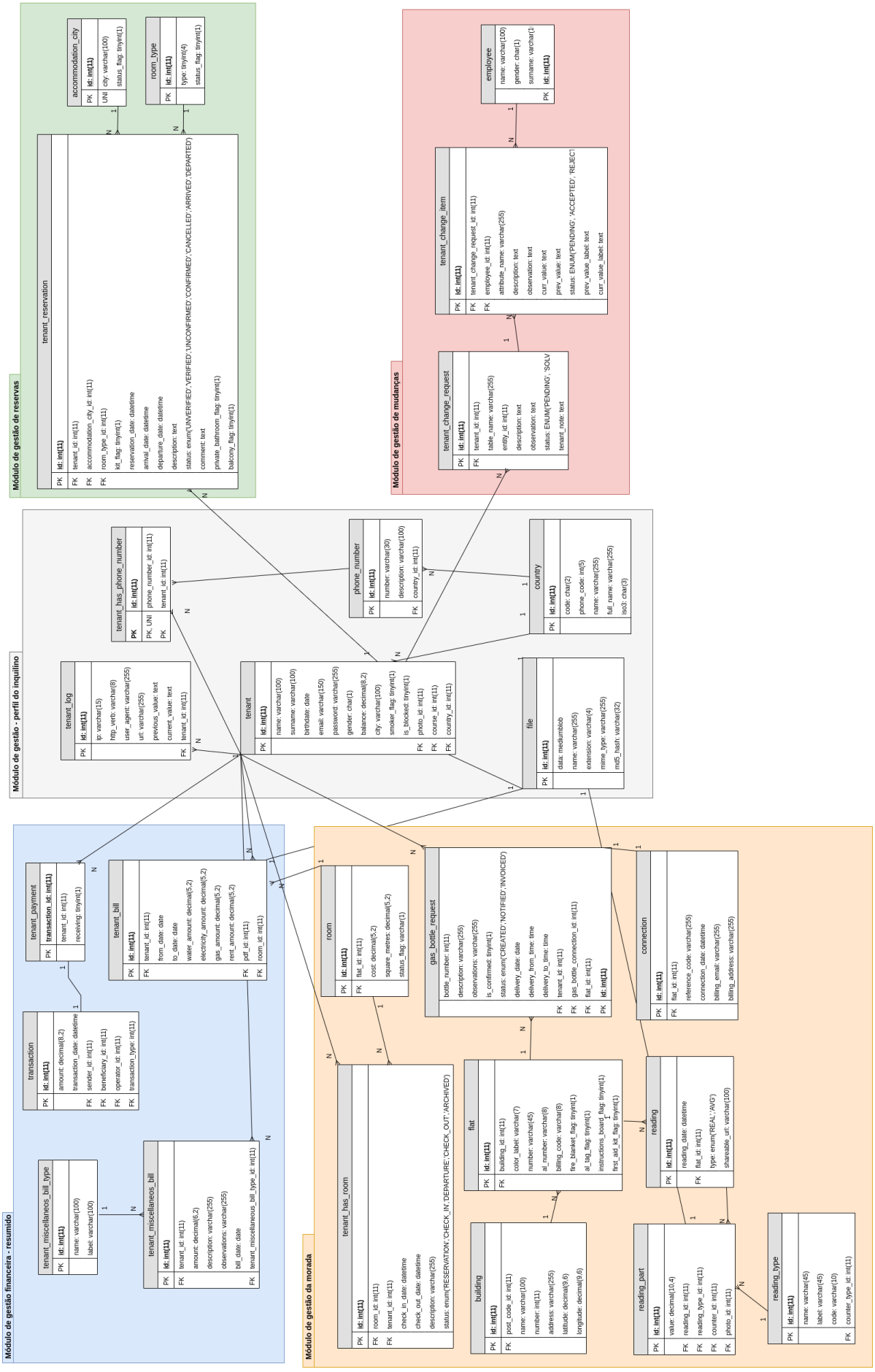


Figura 3.4: Modelo entidade e relacionamento.

3.4 Metodologia de desenvolvimento

É essencial a adoção de uma metodologia ou padrão ao se elaborar um software. Ao longo dos anos o processo de desenvolvimento de software sofreu grandes mudanças. Diversos métodos e metodologias, foram criadas e aperfeiçoadas visando orientar e consequentemente entregar um produto final de melhor qualidade, respeitando os prazos estipulados. Dentro de todo o ecossistema de padrões de desenvolvimento e no escopo deste projeto, as metodologias ágeis se destacam, pois além de serem de simples implementação, o seu desenvolvimento é incremental, ou seja, entrega pequenas funcionalidades ao decorrer da implementação do software.

Para elaboração deste trabalho foi utilizado a metodologia ágil *SCRUM* que implementa métodos científicos do empirismo que substitui uma abordagem algorítmica programada por uma abordagem heurística (Org, 2019).

Dessa forma, a implementação do SCRUM no projeto seguiu os seguintes passos:

- Depois de coletados todos os requisitos do sistema, foi gerado o Product backlog (ou lista de pendências do produto)
- Reuniões diárias de aproximadamente 15 minutos, apontando o andamento do projeto, desafios e o estado atual do mesmo.
- Foi estipulado sprints (entregas de funcionalidades) de aproximadamente 15 dias.
- Depois de transcorrido esse período e realizado os devidos testes, a funcionalidade era acoplada ao software.
- Cada funcionalidade pode sofrer mudanças, voltando novamente para o backlog, recomeçando o processo.

Essa sequência de ações foi utilizada para desenvolver todas as funcionalidades. A Figura 3.5 ilustra o fluxograma do *SCRUM* aplicada ao trabalho.

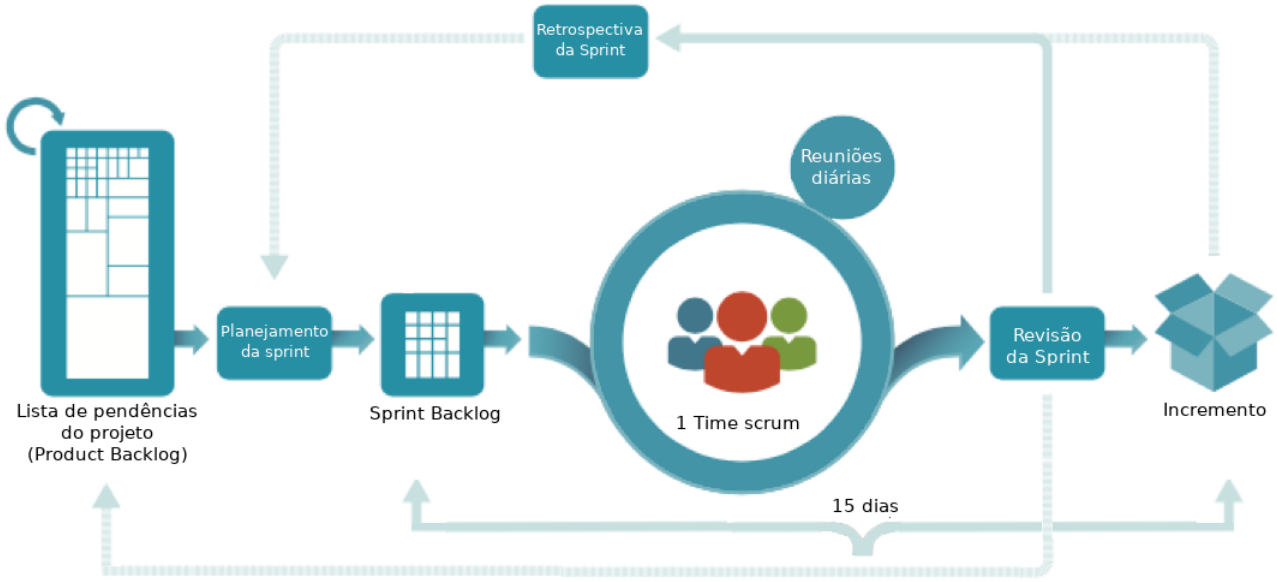


Figura 3.5: Fluxograma do *SCRUM* aplicado ao projeto. (Schwaber and Sutherland, 2012) - adaptada.

3.5 Workspace - Tecnologias e ferramentas

Para o desenvolvimento desse projeto foram aprimoradas e aproveitadas tecnologias que já eram utilizadas pela empresa. Dessa forma obtém-se um melhor aproveitamento dos conhecimentos e facilita a manutenibilidade e interoperabilidade entre colaboradores.

A seguir serão elencados as ferramentas utilizadas no desenvolvimento full-stack da aplicação.

3.5.1 Versionamento

O versionamento de todos os projetos desenvolvidos pela *Riskivector* é feita pelo *Git*, um sistema distribuído multiplataforma de controle de versões de código aberto (Chacon, 2009). Para fins de segurança e manter o sistema colaborativo, o projeto também é hospedado na Internet pela aplicação *Gitlab* que além de possuir uma integração com o software de versionamento *Git*, também oferece funcionalidades extras.

O *Gitlab* permite a gerencia de tarefas, sendo possível a atribuição de issues (tarefas) para diferentes desenvolvedores, possui também mecanismos de integração contínua e entrega contínua, criação de milestones que são marcos (cronograma) na linha de desenvolvimento do projeto, entre muito outros recursos.

No escopo desse projeto, para cada sprint foi gerada uma milestone, sendo possível acompanhar o avanço do desenvolvedor através de gráficos, como por exemplo Burndown chart e também sua porcentagem de progresso. A Figura 3.6 demonstra a evolução de uma milestone.

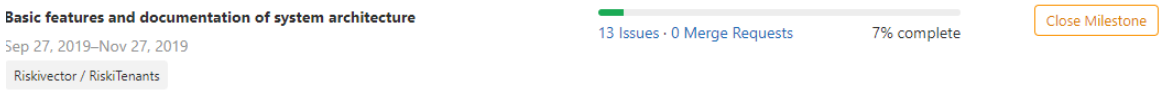


Figura 3.6: Sprint representada como Milestone no *Gitlab*.

3.5.2 Front-end

O desenvolvimento front-end foi realizado utilizando o framework Angular.

Para auxiliar nos estilos das páginas, utilizou-se o template **Fuse**, que é compatível com o Angular Material, um módulo que implementa estilos baseado no material design do *Google*. Esse padrão de design segundo a *Google* é um padrão visual que sintetiza os princípios clássicos do bom design, unificando a experiência do usuário independente de plataforma.

A tradução da aplicação também foi realizada ao lado do cliente (client-side), utilizando a biblioteca *ngx-translate*, que salva a tradução em arquivos de cada idioma em TypeScript ou JSON. A Tabela 3.2 pontua tais ferramentas e suas respectivas versões.

Ferramenta	Versão
Angular	8.1.2
Ngx-Translate	11.0.1
Fuse template	8.1.2
TypeScript	3.4.5

Tabela 3.2: Algumas ferramentas utilizadas no front-end e suas versões.

3.5.3 Back-end

Na parte do servidor back-end, foi utilizado o Node.js adicionando uma configuração para trabalhar com o TypeScript.

O Node.js possui nativamente módulos que permitem a criação de um servidor HTTP. Porém, algumas tarefas comuns para o desenvolvimento na web, como a manipulação específica para diferentes requisições HTTP (por exemplo GET, POST, DELETE, etc.) ou até servir arquivos estáticos, precisam ser criadas manualmente.

Portanto visando otimizar a produtividade foi utilizado o framework Express, que fornece diversos mecanismos para a gestão de cada rota, podendo ser adicionado middlewares para interceptar, processar ou pré-processar cada requisição antes de ser tratada pela rota em questão.

O projeto ainda conta com outras bibliotecas que visam otimizar o desempenho de desenvolvimento, algumas delas são:

- *Sequelize*: ORM que provê toda a comunicação com o banco de dados.
- *Moment.js*: Uma biblioteca robusta para a manipulação de datas.
- *express-validator*: Uma biblioteca que trabalha em conjunto com o *express* executando validações ou limpando os dados (sanitizer) através de um middleware, respeitando seu respectivo Data Transfer Object (DTO), ou ainda, sua formatação esperada.

- *accesscontrol*: Cada recurso é associado a uma permissão. Essa biblioteca provê a administração e gerência do controle de acesso no servidor.

A Tabela 3.3 ilustra tais ferramentas e suas versões.

Ferramenta	**Versão**
Node.js	8.1.2
Express	11.0.1
Express-Validator	8.1.2
Sequelize	4.44.3
Moment	2.24.0
TypeScript	3.7.2
Mysql	5.7
Accesscontrol	2.2.1

Tabela 3.3: Algumas ferramentas utilizadas no back-end e suas versões.

3.6 Considerações finais

Neste capítulo foi apresentado as etapas do planejamento do software, bem como a escolha de suas ferramentas para produção. Após a coleta dos requisitos, foi gerado o diagrama de casos de uso. Essa abordagem além de permitir estruturar cada requisito em um formato visual de fácil compreensão, com o diagrama de casos de uso foi possível gerar uma documentação do software, onde estão concentradas todas as suas funcionalidades.

Com um esboço estruturado produzido pelo diagrama, a próxima etapa foi realizar a modelagem dos dados. Para isso foi utilizado o modelo ER, que além de permitir uma visualização de como os dados serão dispostos no banco de dados, foi possível depurar sua modelagem, encontrando falhas e corrigindo-as antes de sua implementação [3.3].

Dessa forma, o planejamento do software foi de suma importância não só para compreender melhor sua estrutura, conseguindo um melhor controlo do mesmo, mas também como apresentado, resolver eventuais problemas antes da fase de desenvolvimento.

Capítulo 4

Desenvolvimento

Após serem apresentadas as ferramentas e projetado o software no Capítulo 3, neste capítulo será apresentado o processo de desenvolvimento da aplicação web para os clientes da empresa *Riskivector*.

O restante desse capítulo está organizado da seguinte forma: na Seção 4.1 são apresentadas as dificuldades e os problemas encontrados durante a fase de desenvolvimento, bem como suas soluções. A Seção 4.2 apresenta duas arquiteturas, sendo que uma retrata a arquitetura no âmbito de rede, representando como a aplicação web se comunica com o servidor HTTP, enquanto a outra representa a abordagem de uma nova versão da API. Tal versão visa solucionar antigas dívidas técnicas acumuladas ao longo do tempo através da implementação de uma nova arquitetura. A Seção 4.3 descreve as implementações dos requisitos apresentados no Capítulo 3, e por fim a Seção 4.4 é apresentado as considerações finais referentes a esse capítulo.

4.1 Problemas e dificuldades

Diversos problemas surgiram no decorrer do desenvolvimento. Alguns desses problemas serão elencados nesta seção, bem como as estratégias adotadas para solucioná-los.

4.1.1 Limitações e dependência de bibliotecas

Um dos grandes problemas no desenvolvimento deste trabalho, foi o forte acoplamento com bibliotecas de terceiros, sendo que uma das principais dependências estava com a biblioteca do Sequelize. Embora esse ORM seja um dos mais utilizados pela comunidade, como demonstra a Figura 4.1, ele apresenta alguns problemas de consistência em seu funcionamento.

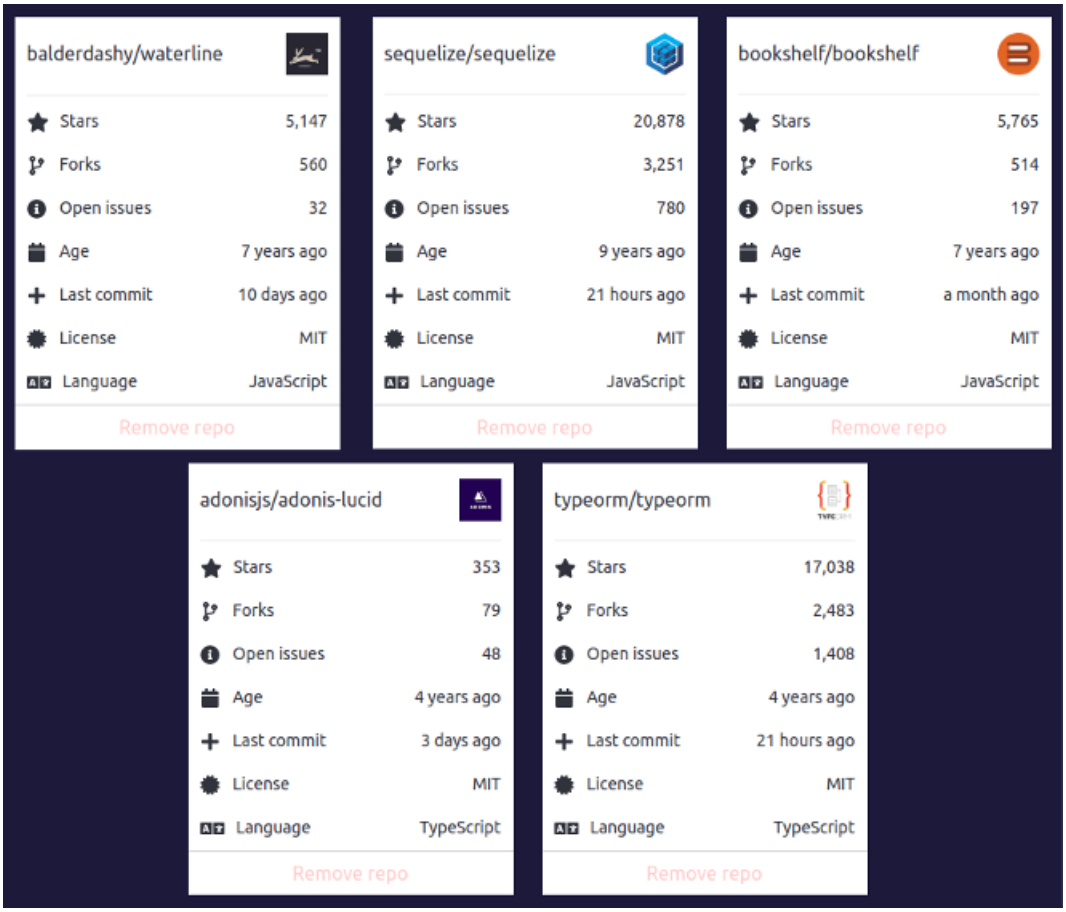


Figura 4.1: Comparação de ORM JavaScript/TypeScript baseadas em estrelas do repositório do Github, utilizando a ferramenta *GithubCompare*[2].

Um problema que ilustra essa inconsistência ocorreu quando foi atualizada sua versão. A biblioteca na atual versão do projeto (versão 4) não possui a tipagem nativa, sendo

[2] `https://www.githubcompare.com/sequelize/sequelize+balderdashy/waterline+bookshelf/bookshelf+adonisjs/adonis-lucid+typeorm/typeorm`

necessária a utilização de bibliotecas de terceiros para realizá-la. Dessa forma, para o desenvolvimento da nova versão da API (ver Seção 4.2.2) foi decidido atualizá-la para a última versão estável, isto é, a versão 5, a qual conta com suporte nativo de tipagem. Além disso, a atualização foi feita em virtude do lançamento recente de uma versão beta, levantando a hipótese de que a ferramenta poderia deixar de dar suporte para versões mais antigas.

Porém, após realizado todos os procedimentos para a atualização de sua versão, notou-se o mal funcionamento da mesma, e apesar de estar prestes a lançar a versão 6, existe um problema que até então não foi solucionado e que afetava o funcionamento do servidor, forçando o retorno para a versão antiga.

Outro problema da biblioteca está em sua documentação, que é demasiadamente limitada, visto que nem todas as configurações e funcionalidades estão devidamente expostas, sendo necessário em alguns casos, recorrer a issues da plataforma GitHub para encontrar tais informações.

Dessa forma, devido ao projeto ser altamente acoplado e dependente desse ORM, essa dívida técnica ocupou muito tempo e esforço, que poderiam ser evitados se uma arquitetura melhor tivesse sido adotada. Por esse motivo foi desenvolvido uma nova arquitetura, descrita na Seção 4.2.2, que visa através da divisão de responsabilidades, reduzir o acoplamento com bibliotecas externas e aprimorar a manutenibilidade do servidor.

4.1.2 Detecção de erros ao lado do cliente

Idealmente os desenvolvedores desejam detectar erros antes que os usuários os encontrem. Porém nem sempre essa abordagem é possível.

Uma das dificuldades da detecção de erros é realizá-la de forma escalável e eficiente. O tratamento comum das exceções de erros utilizando blocos *try/catch* para casos específicos, muitas vezes leva a duplicação de código, tornando a aplicação insustentável em pouco tempo. Deste modo, a solução para essa problemática foi centralizar toda a manipulação

de erros em um único local. A Figura 4.2, ilustra a reutilização e centralização do código com um manipulador global de erros.

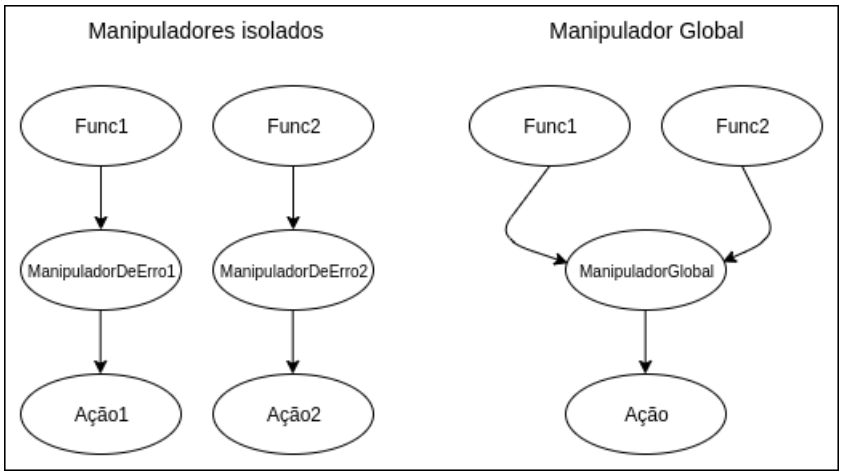


Figura 4.2: Manipulador isolado de erros vs Manipulador global de erros.

No contexto desse projeto a centralização de erros foi feita com o auxilio do gatilho (do inglês, hook) ErrorHandler disponível no Angular, que é disparado sempre que ocorre qualquer tipo de erro na aplicação. Dessa forma, cada novo erro gerado pela aplicação web é despachado para o manipulador global de erros, que além de prover a consistência e reutilização de código, facilitou solucionar uma outra problemática, o rastreamento de erros da aplicação web.

Uma vez que a aplicação é acessada por diferentes navegadores/dispositivos, é necessário notificar o servidor sobre a ocorrência dos erros, caso contrário eles seriam disparados em cada dispositivo e não seria possível acessá-los. Sendo assim, o manipulador global de erros processa e envia as informações para o servidor da empresa, adicionando uma redundância, enviando-os para nuvem da plataforma *Slack* utilizada pela companhia. A Figura 4.3 exemplifica esse processo.

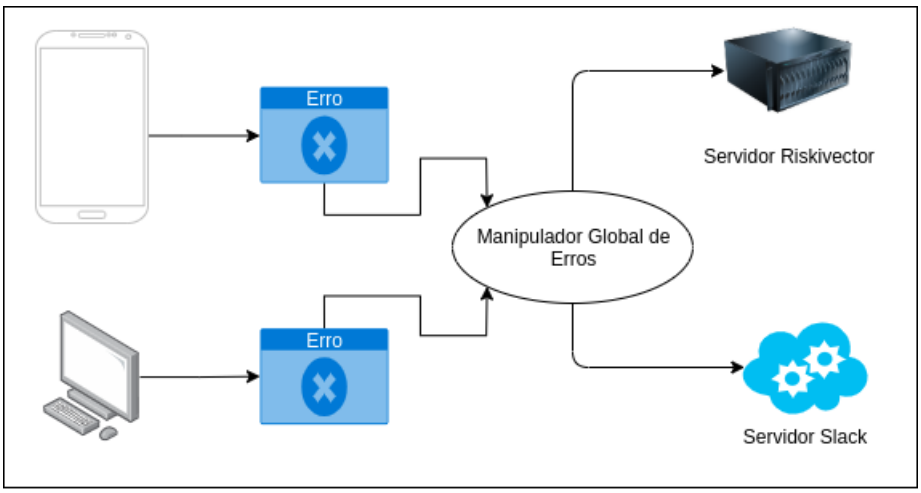


Figura 4.3: Armazenando erros no servidor acarretados pela aplicação dos clientes.

4.2 Arquitetura

Para a realização desse projeto foram criadas duas arquiteturas. Uma referente a arquitetura de rede, que aborda os processos de comunicação entre front-end e back-end, e outra referente a arquitetura em camadas que foi adicionada ao servidor HTTP. As próximas seções abordaram a disposição e funcionamento de cada elemento das arquiteturas desenvolvidas.

4.2.1 Arquitetura de rede

Como a aplicação será acessada por agentes externos à empresa, é fundamental utilizar mecanismos de segurança que visam proteger todo o servidor de possíveis ataques. Para isso, uma das medidas tomadas foi a utilização de um proxy entre a aplicação Angular e o web service RESTful da empresa.

Todas as rotas que estão relacionadas à aplicação dos clientes, possuem o prefixo *api/v2/tenants*, e à medida que as requisições são encaminhadas para a API, têm seu caminho (do inglês, path) alterado, adicionando o prefixo ao caminho principal. A Figura 4.4 exemplifica como essa mudança é realizada.

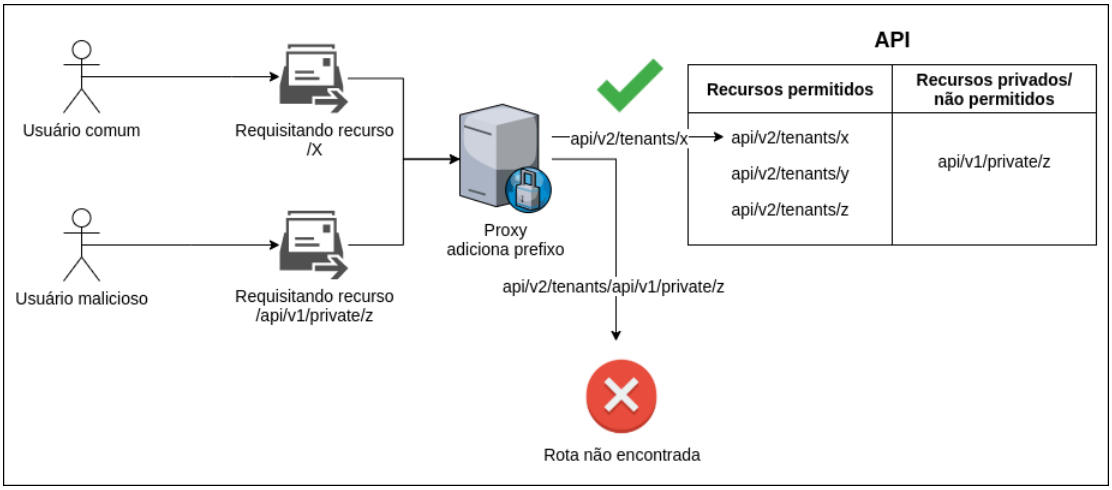


Figura 4.4: Redirecionamento realizado pelo Proxy para a API.

Dessa forma, apenas o proxy sabe como chegar à API, conhecendo seu IP e expondo para a aplicação web apenas o endereço do proxy. Além disso, a adição do prefixo ao endereço delimita o escopo que um agente externo consegue acessar. O exemplo mostrado na Figura 4.4 ilustra um caso normal realizado por um usuário comum, onde o caminho requisitado representa um recurso que pode ser acessado por inquilinos. O recurso alvo é x e quando passado pelo proxy, foi rescrito para */api/v2/tenants/x*. O servidor consegue reconhecer esse caminho, visto que existe uma rota correspondente dentro da API.

Porém caso um usuário malicioso tente acessar uma rota que não é permitida, como por exemplo, as rotas presentes da versão 1.0 (*V1*), a rescrita do endereço ocasiona em um erro 404, pois a adição do prefixo dos clientes na requisição leva a um recurso que não existe dentro do servidor. A Figura 4.5 ilustra a arquitetura de rede completa.

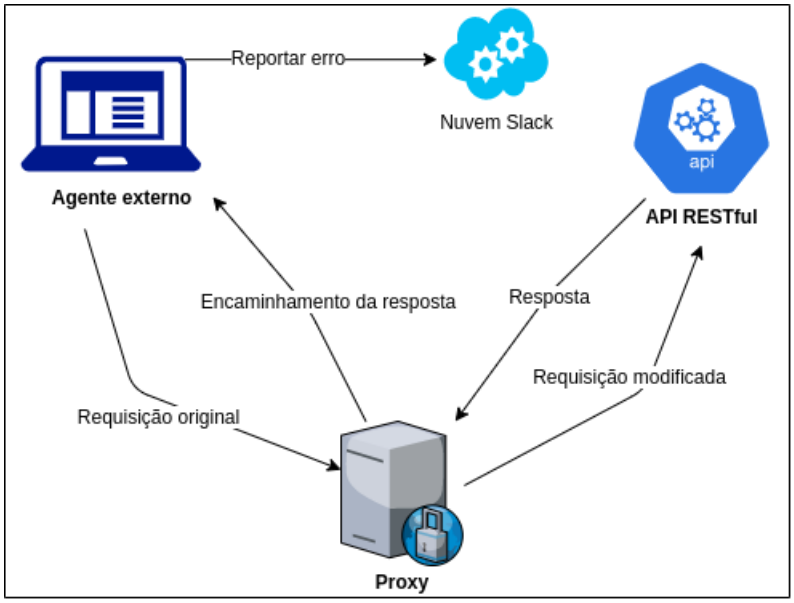


Figura 4.5: Arquitetura da rede adotada.

4.2.2 Arquitetura em camadas - Servidor

Para o desenvolvimento desse projeto foi criado uma nova versão do servidor, dividindo-o em duas versões. A primeira versão contém as rotas antigas, com a finalidade de manter a compatibilidade entre os sistemas que as utilizavam. Já a segunda versão introduz uma nova arquitetura em camadas, contando ainda com a utilização do TypeScript em seu desenvolvimento. A Figura 4.6 ilustra a estrutura do diretório que suporta ambas versões.

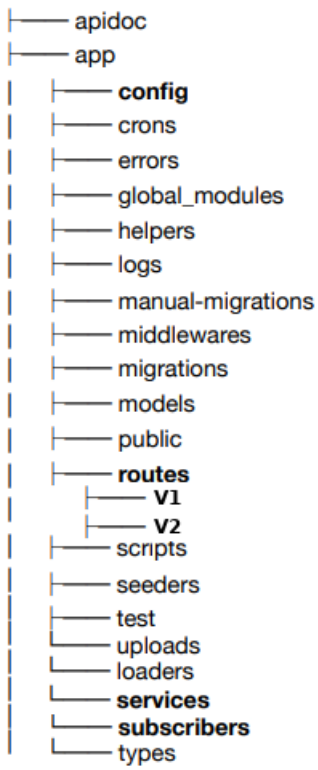


Figura 4.6: Nova estrutura do diretório do servidor.

A arquitetura em camadas consiste na divisão de responsabilidades dentro de um software. Os componentes dentro dessa arquitetura são organizados em camadas horizontais, dessa forma, cada camada desempenha uma função especifica e bem definida no aplicativo. Cada camada da arquitetura é uma abstração do trabalho que precisa ser feito para satisfazer uma solicitação específica.

Segundo Richards (2015), esse padrão não especifica um número e os tipos de camadas que devem existir, porém a maioria das arquiteturas em camadas consiste em quatro camadas padrões, sendo elas elas: apresentação, negócio, persistência e banco de dados. Em alguns casos a camada de negócio e persistência são combinadas em uma única camada, principalmente quando elementos da camada de persistência são utilizados nos componentes da camada de negócio.

No âmbito desse projeto, optou-se pela adoção de três camadas. Tais camadas foram dívidas da seguinte forma: o controlador (controller), camada de negócio + persistência

denominada serviços (services) e por fim a camada do banco de dados. A Figura 4.7 ilustra a arquitetura utilizada e nas próximas seções será detalhado o funcionamento e objetivo de cada camada.

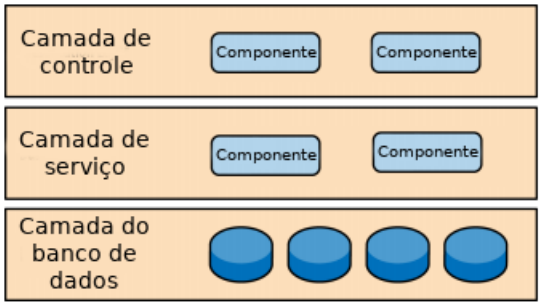


Figura 4.7: Arquitetura em camadas, (Richards, 2015) - adaptada.

Camada de controle

Essa camada é responsável pela lógica de comunicação com o navegador, fazendo a interface entre o cliente e o servidor. É nele que são feitas as validações de controle de acesso de recurso e também o tratamento das requisições bem como as respostas HTTP. Portanto o controlador é um orquestrador que delega tarefas para serviços, e não contém nenhuma lógica além de lidar com as solicitações e respostas do servidor.

Camada de serviço

Essa camada encapsula a lógica de negócio e também a persistência dos dados. Um serviço é uma classe feita em TypeScript que pode ser injetado em um controlador, através da injeção de dependência (ID), com a finalidade de executar alguma lógica de negócio ou prover meios de acesso a base de dados através de uma interface bem definida. Dessa forma o serviço não deve implementar nenhuma consulta SQL e sim delegar para camada do banco de dados a ação que deve ser tomada.

Camada da base de dados

Essa camada é a própria base de dados, sendo responsável por executar consultas, inserções, deleções e tudo que envolve a manipulação de dados.

4.3 Desenvolvimento da aplicação Web

Após estruturado e melhorado a arquitetura do servidor, foi possível começar o desenvolvimento da aplicação.

A seguir será demonstrado as telas e a lógica utilizada para conseguir implementar os requisitos elencados no Capítulo 3, tais como: permitir o cadastro de inquilinos para utilização do sistema, consultar do histórico de pagamentos e saldo, prover um serviço de gestão de reserva de alojamento, permitir editar seu perfil, resgate de senha, realizar pagamentos, alterar o idioma do sistema, permitir aos colaboradores gerenciar as mudanças que necessitam de validação, entre outros serviços.

4.3.1 Cadastrar inquilinos no sistema

O cadastro dos inquilinos foi desenvolvido em uma outra plataforma, uma vez que o projeto proposto neste trabalho apenas administra recursos para pessoas que são clientes da empresa. A Figura 4.8 exibe o formulário de reserva necessário para o cadastro no sistema proposto.

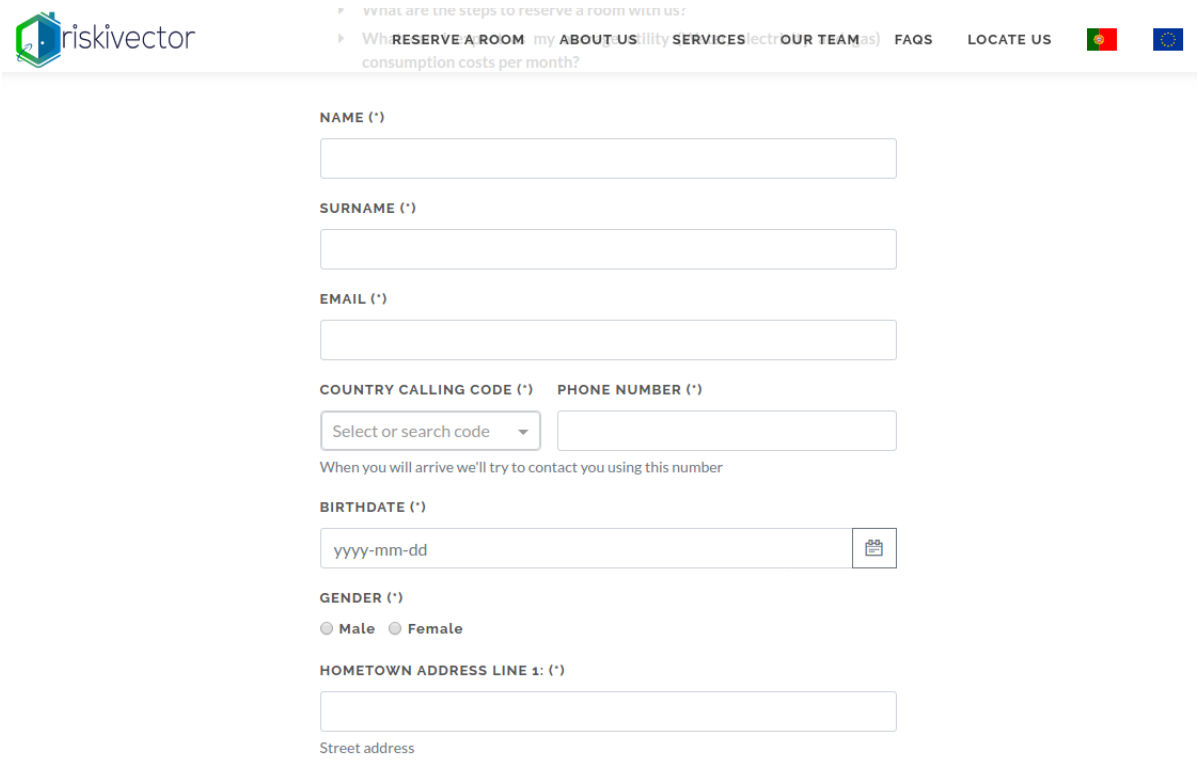
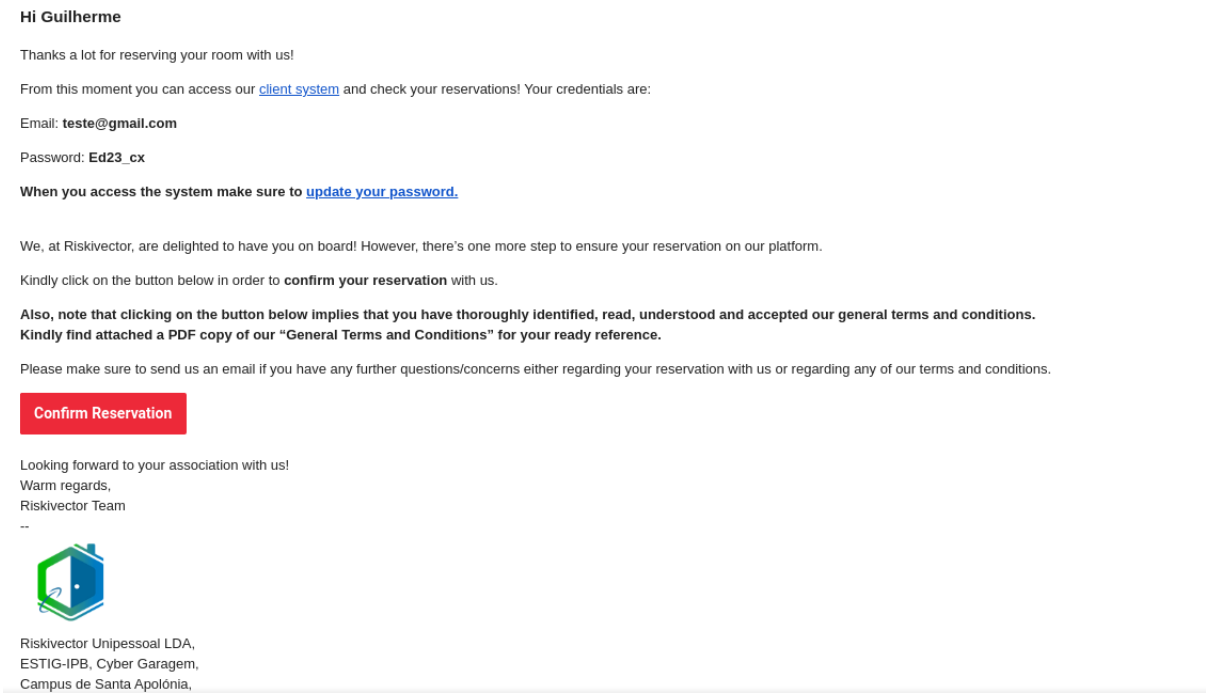


Figura 4.8: Parte do formulário necessário para o cadastro no sistema.

O cadastro é dividido em duas partes. A primeira consiste no envio do formulário de reserva demonstrado pela Figura 4.8. Nele são requisitados diversos dados, tais como: nome, sobrenome, número de telefone, passaporte, tipo de quarto, cidade que deseja a acomodação, e-mail, etc. Esses dados são de suma importância para a aplicação, uma vez que serão exibidos e manipulados pelo inquilino na aplicação web.

A segunda parte é feita após o envio do formulário, onde o servidor armazena o pedido e encaminha um e-mail para o cliente confirmar sua reserva, juntamente com as credencias para realizar o acesso ao sistema. A senha é gerada pelo servidor e atrelada ao inquilino no momento em que realiza o pedido de reserva. Dessa forma o inquilino deve acessar o sistema e realizar a mudança da mesma.

A partir desse momento é possível acessar e acompanhar as mudanças em sua reserva na aplicação. A Figura 4.9 exibe o e-mail encaminhado ao inquilino com suas credenciais.



Hi Guilherme

Thanks a lot for reserving your room with us!

From this moment you can access our client system and check your reservations! Your credentials are:

Email: **teste@gmail.com**

Password: **Ed23_cx**

When you access the system make sure to update your password.

We, at Riskivector, are delighted to have you on board! However, there's one more step to ensure your reservation on our platform.

Kindly click on the button below in order to **confirm your reservation** with us.

Also, note that clicking on the button below implies that you have thoroughly identified, read, understood and accepted our general terms and conditions. Kindly find attached a PDF copy of our "General Terms and Conditions" for your ready reference.

Please make sure to send us an email if you have any further questions/concerns either regarding your reservation with us or regarding any of our terms and conditions.

Confirm Reservation

Looking forward to your association with us!
Warm regards,
Riskivector Team
--

Riskivector Unipessoal LDA,
ESTIG-IPB, Cyber Garagem,
Campus de Santa Apolónia,

Figura 4.9: E-mail recebido pelo inquilino informando suas credenciais.

4.3.2 Acessar o sistema

Em posse das credenciais, o inquilino pode acessar a aplicação. A Figura 4.10 exibe a tela que possibilita seu acesso.

Figura 4.10: Tela de acesso ao sistema.

O processo de acesso a aplicação segue os seguintes passos:

1. A interface web realiza diferentes validações antes do envio das credenciais ao sistema. É verificado se o texto inserido no campo e-mail é um e-mail válido e se a senha possuí ao menos 6 caracteres, uma letra e um número. Após validado, a aplicação envia uma requisição para o back-end para realizar a autenticação.

2. O servidor checa as credenciais, e caso estejam corretas é enviado um JSON Web Token (JWT) contendo em seu payload o id do inquilino, seu nome e sobrenome, bem como um tempo de expiração de 9 horas. O campo id é necessário para que a aplicação consiga realizar requisições resgatando informações do inquilino. Já o nome e sobrenome é necessário para realizar o log das ações do inquilino no servidor.

    - Caso as credenciais estejam erradas ou o inquilino está bloqueado, o servidor retorna erros e a aplicação web exibe um componente compartilhado de notificação com uma mensagem informando o ocorrido.

3. Depois que é realizado o acesso à aplicação, o token é adicionado a todas as requisições feitas pelo inquilino, através dos interceptadores do Angular. Dessa forma

o Angular intercepta uma requisição, adiciona o token a esse pedido e encaminha para a API.

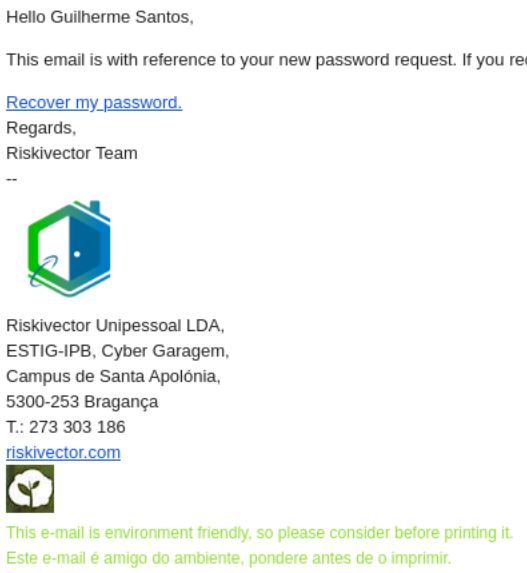
4. O servidor valida o token e caso não seja válido, é exibida a tela de acesso novamente para o inquilino, retornando para o primeiro passo.

4.3.3 Recuperar senha

A recuperação de senha possui duas partes. Primeiro o inquilino solicita a mudança de senha inserindo seu e-mail, conforme exibe a Figura 4.11.

Figura 4.11: Recuperação de senha - parte 1/2.

Após enviada a solicitação, é encaminhado um e-mail com um link para realizar a mudança de senha na plataforma web. O link contém um token que é válido por até 15 minutos, devendo ser realizada a mudança nesse período. A Figura 4.12 exibe o e-mail encaminhado para o inquilino.


Hello Guilherme Santos,

This email is with reference to your new password request. If you requested a new password, click the link below, or please ignore this email.

Recover my password.
Regards,
Riskivector Team
--

Riskivector Unipessoal LDA,
ESTIG-IPB, Cyber Garagem,
Campus de Santa Apolónia,
5300-253 Bragança
T.: 273 303 186
riskivector.com

This e-mail is environment friendly, so please consider before printing it.
Este e-mail é amigo do ambiente, pondere antes de o imprimir.

Figura 4.12: E-mail de recuperação de senha.

Quando o inquilino acessa o link encaminhado, pode então realizar a troca de senha. Nesse caso é necessário inserir duas vezes a nova senha. Em seguida é realizado a validação verificando se ambas as senhas são iguais e se estão no padrão estipulado. A Figura 4.13 exemplifica a segunda parte de mudança de senha.

Figura 4.13: Recuperação de senha - parte 2/2.

Após realizado a mudança com sucesso, o inquilino é redirecionado para página de acesso ao sistema, podendo aceder utilizando a nova senha.

4.3.4 Alterar o idioma

O Angular possui nativamente um módulo que cuida da internacionalização da aplicação. Porém o sistema de tradução adotado foi o mesmo que é utilizado pelo template **Fuse**, com a biblioteca *ngx-translate*. A Figura 4.14 exibe os passos para realizar a mudança de idioma na aplicação.

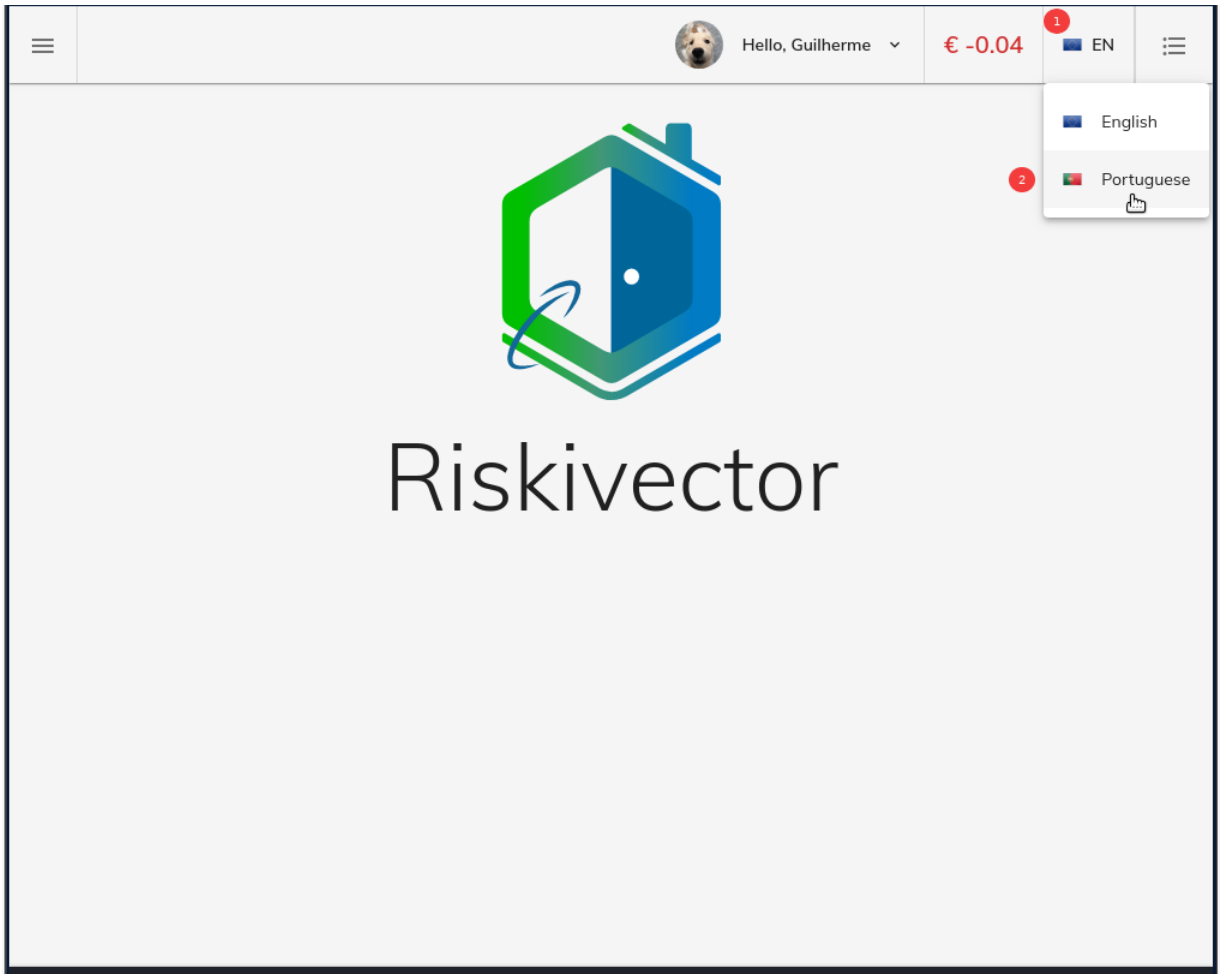


Figura 4.14: Alterando o idioma da aplicação.

O primeiro passo para alterar o idioma ilustrado pela Figura 4.14 é feito utilizando a barra de ferramentas, localizada no canto superior da tela. O usuário deve clicar sobre a bandeira representando sua tradução, como ilustra o passo "1" e depois selecionar o idioma de sua preferência no menu suspenso, ilustrado pelo passo "2".

A solução foi modelada de forma a permitir a expansão para novos idiomas. Tal ampliação é possível devido à separação das traduções por módulos. Cada módulo da aplicação contém seu respectivo arquivo de tradução para diferentes idiomas e estes são carregados de acordo com a interação do usuário (lazy load). Dessa forma, além de permitir uma melhor estruturação dos arquivos de traduções, também é reduzido o tráfego de rede, em razão de não ser feito o download de uma só vez de todos os componentes e suas respectivas traduções.

4.3.5 Gerenciar reservas

Após requisitada a reserva, o inquilino pode acompanhar o andamento da mesma, bem como realizar mudanças. A Figura 4.15 exibe a tela responsável por listar todas as reservas de um cliente.

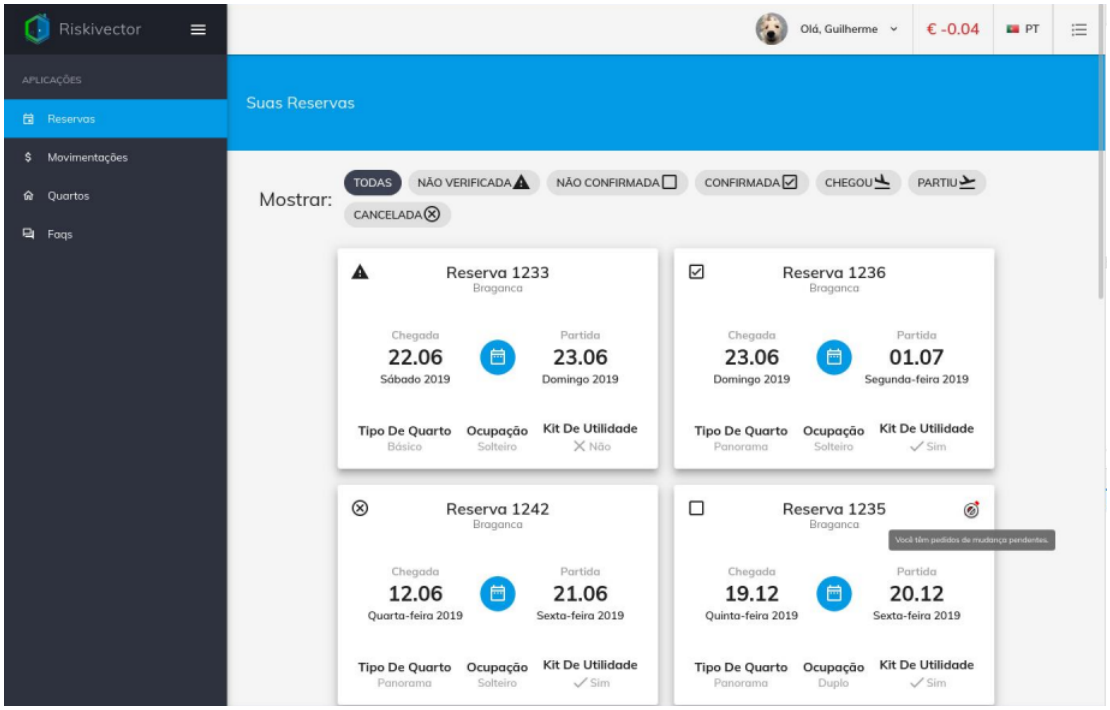


Figura 4.15: Reservas feitas pelo inquilino.

Cada reserva é exibida como cartões na aplicação. Nela é possível consultar de forma fácil e clara suas informações. O primeiro elemento da tela de reservas exibe os filtros que podem ser aplicados de acordo com o status da mesma. Uma reserva pode possuir diferentes status, sendo eles:

- **Não verificado:**

  Estado inicial de uma reserva. Significa que um inquilino enviou o formulário de reserva, mas ainda não confirmou em seu e-mail.

- **Não confirmado:**

  Este estado significa que um inquilino confirmou o pedido de reserva em seu e-mail e aguarda a confirmação de um quarto disponível de acordo com a especificação de sua reserva.

- **Confirmado:** Significa que um quarto foi atribuído ao inquilino.

- **Chegou:** Esse estado indica que o inquilino chegou na cidade requisitada, de acordo com a data especificada no momento da reserva.
- **Partiu:** Esse estado indica que a reserva já não pertence ao inquilino, respeita a data especificada de partida.
- **Cancelada:** Representa uma reserva cancelada.

Cada cartão tem como título e subtítulo o id e cidade que foi requisitado a reserva. Logo abaixo estão informações referentes as datas de chegada e partida. As datas são exibidas de formas diferentes, respeitando o padrão do idioma selecionado. No caso exemplificado pela Figura 4.15, demonstra o padrão de datas em português. Ao final de cada reserva é exibido informações especiais da reserva, sendo elas:

- **Tipo de quarto:**
  - **Básico:** Quartos sem varanda e sem banheiros exclusivos.
  - **Panorama:** Quartos que possuem varanda.
  - **Suíte:** Quartos com banheiro.
- **Ocupação:**
  - **Solteiro:** Quarto individual.
  - **Duplo:** Divisão entre dois clientes no quarto.
  - **Tripo:** Divisão entre três clientes no quarto.
- **Kit de utilidade:** Informa se o inquilino deseja o kit de utilidade, que possui produtos essenciais para uso do quotidiano.

As reservas que estão no estado de não confirmadas podem ser editadas pelos inquilinos. Porém a edição precisa ser validada ou invalidada pela empresa (ver Seção 4.3.7), não

sendo possível realizar novas mudanças enquanto houver pedidos pendentes (ver Figura 4.15). O formulário de edição de uma reserva é demonstrado na Figura 4.16.

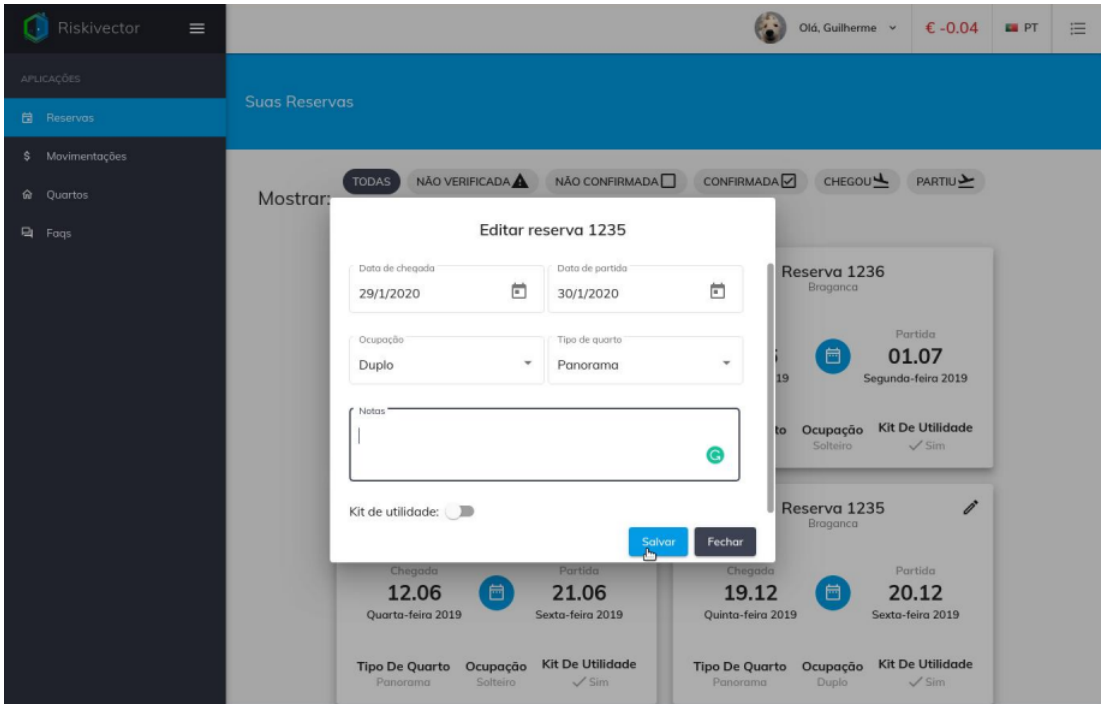


Figura 4.16: Formulário de edição de uma reserva.

Cada módulo da aplicação é responsivo, ou seja, as informações são adaptadas de forma a ocupar melhor seu espaço entre os diferentes tipos de telas. A Figura 4.17 exibe a tela de consulta de reservas em um dispositivo móvel, aplicando o tema escuro da aplicação.

Figura 4.17: Responsividade para dispositivos móveis para consultas de reservas - Tema escuro.

4.3.6 Enviar pedido de mudança

Como aludido, nem todas as mudanças dos inquilinos são diretamente atualizadas no banco de dados. Algumas mudanças necessitam de validações. Nesse caso toda vez que um inquilino deseja alterar tais elementos, é gerado um pedido de mudança na aplicação interna da empresa.

Essas mudanças são originadas de diferentes recursos, podendo ser uma mudança de reserva, de perfil, ou qualquer outro recurso. Dessa forma, para conseguir mapear todas as mudanças, oriundas de diferentes entidades do banco de dados foi gerado as tabelas *tenant_change_request* e *tenant_change_item*, como demonstra a Figura 4.18

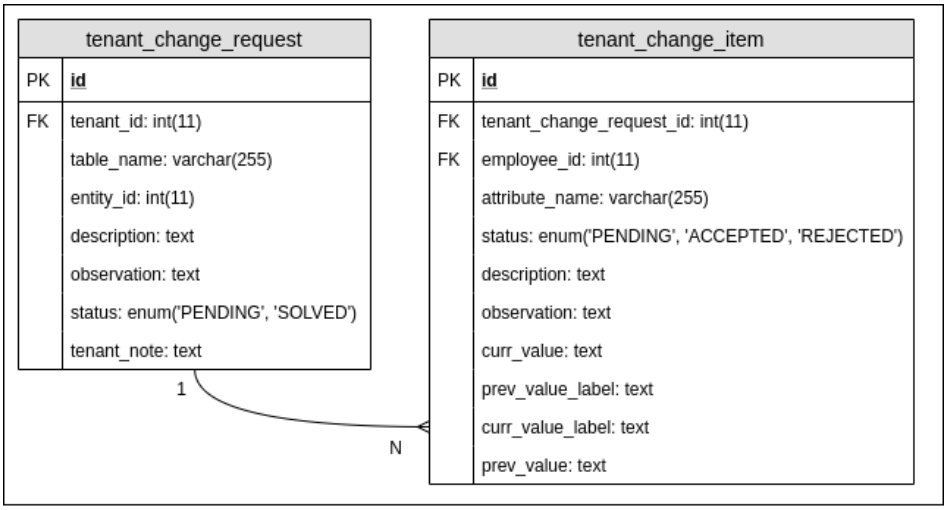


Figura 4.18: Tabelas responsáveis por armazenar pedidos de mudança do inquilino.

Um inquilino pode realizar o pedido de mudança para mais de um atributo de uma tabela específica. Dessa forma, a responsabilidade da tabela tenant_change_request é informar qual é a tabela alvo do pedido de mudança, e também agrupar todos os itens que o cliente deseja alterar. Assim sendo, os atributos das tabelas tenant_change_request e tanant_change_item, especificam:

- **tenant_change_request**:

  - **tenant_id**: Indica o inquilino que requisitou a mudança.
  - **table_name**: Nome da tabela alvo do pedido de mudança.
  - **entity_id**: O id do elemento alvo.
  - **description**: Esse campo é de utilização interna da empresa. Caso um funcionário queira deixar um comentário a respeito da mudança do inquilino, poderá fazê-lo.
  - **observation**: As observações, assim como as descrições são opcionais. As observações são destinadas para enviar um feedback ao inquilino. Quando essa mudança for aceita, será encaminhado um e-mail incluindo tais observações.

  - **status**: Informa se o pedido está pendente ou já foi resolvido.
  - **tenant_note**: Uma nota do inquilino a respeito da mudança.

- **tenant_change_item**:

  - **attribute_name**: Especifica o atributo alvo da mudança.
  - **status**: Indica se o pedido de mudança foi aceito/rejeitado ou está pendente.
  - **description**: Descrição interna para controle na empresa.
  - **observation**: Descrição enviada como feedback para o inquilino.
  - **curr_value**: Armazena o valor que o inquilino deseja alterar.
  - **prev_value**: Armazena o valor anterior ao pedido.
  - **prev_value_label / curr_value_label**: Armazena de uma forma compreensível o valor que o inquilino deseja alterar. Valores booleanos, por exemplo, serão exibidos para o colaborador de forma mais amigável.
  - **employee_id**: Indica o colaborador que aceitou/rejeitou a mudança.
  - **tenant_change_request_id**: Indica o pedido que esse item pertence.

Com isso, é possível armazenar cada pedido de mudança na base de dados. A Figura 4.19 ilustra alguns desses itens mapeados dentro da aplicação.

1 - tenant_change_request	2 - tenant_change_item
1.1 - entity_id: 88 1.2 - table_name: tenant_reservation 1.3 - tenant_note: teste status: 'PENDING'	2.1 - curr_value: '24/01/2020' prev_value: '15/11/2019' status: 'PENDING' attribute_name: 'arrival_date' prev_value_label: 'arrival date: 15/11/2019' curr_value_label: 'arrival date: 24/01/2020' 2.2 - curr_value: 1 prev_value: 2 status: 'PENDING' prev_value_label: 'ocupancy: single' prev_value_label: 'ocupancy: double' attribute_name: 'room_type_id'

Figura 4.19: Elementos do pedido de mudança mapeados na interface de edição de uma reserva.

Como mencionado as mudanças podem ser oriundas de diferentes recursos. Dessa forma é necessário armazenar diferentes tipos de dados em uma única tabela. No exemplo ilustrado pela Figura 4.19, o atributo “2.1” chega ao servidor no formato de texto, enquanto o atributo “2.2” está formatado numericamente. A solução encontrada para esse

problema foi armazenar todos os pedidos em formato de texto e realizar a conversão dos dados tanto para a inserção no banco, transformando-os em texto, quanto para obter as informações do banco, retornando seu valor original. A Figura 4.20 ilustra esse processo.

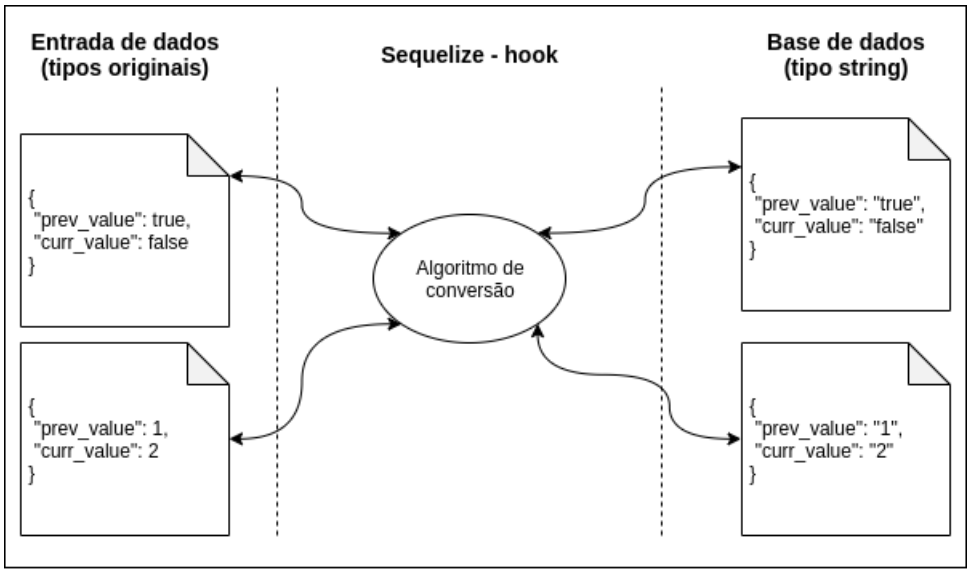


Figura 4.20: Conversão dos tipos de dados realizado na inserção e consulta dos pedidos de mudança.

Para realizar esse trabalho foi utilizado os gatilhos da biblioteca Sequelize, que funcionam similarmente aos triggers do banco de dados. Antes de inserir um elemento, o algoritmo realiza a conversão de diferentes tipos para texto. De forma análoga, quando são retornados esses elementos da base de dados, o gatilho é acionado novamente e realiza a conversão para seu formato original.

O estado de um *tenant_change_request* também é alterado utilizando um gatilho do Sequelize. Sempre em que tenha alterações no status dos *tenant_change_item* é verificado se não há nenhum item pendente relacionado ao seu respectivo *tenant_change_request*. Dessa forma, caso não haja mais itens pendentes sobre aquele pedido, o status do pedido é atualizado para resolvido, demonstrado no banco de dados como 'SOLVED' (ver Figura 4.18).

4.3.7 Gerenciar pedidos de mudança

Após enviado o pedido de mudança descrito previamente, um colaborador pode gerenciar tais pedidos, aceitando/rejeitando, adicionando descrições/observações e também desfazer suas ações. A Figura 4.21 exibe a tela responsável por exibir as mudanças pendentes.

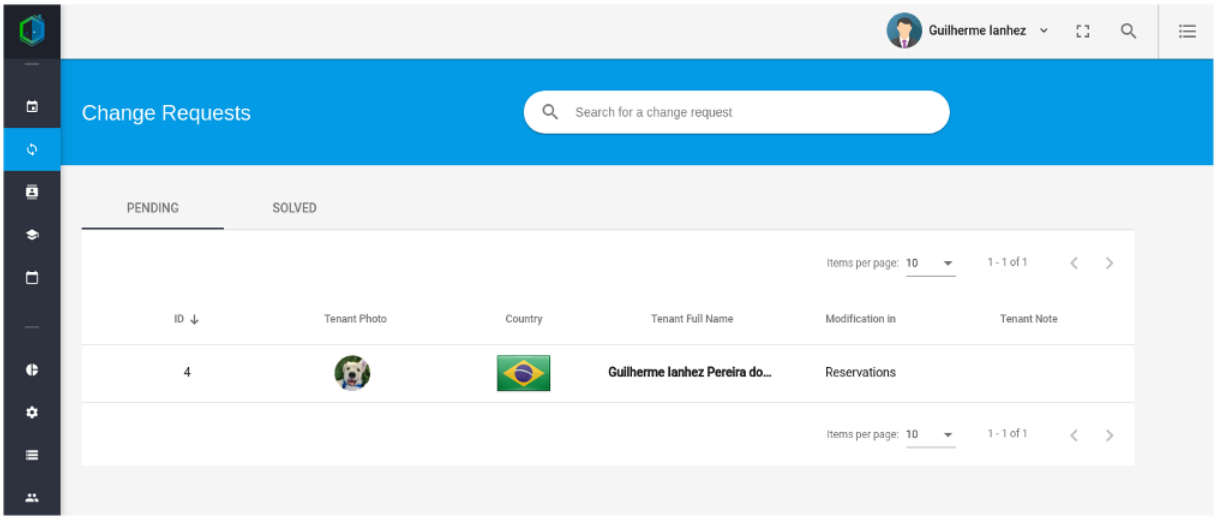


Figura 4.21: Exibição dos pedidos de mudanças pendentes.

Para exibir mais informações e consultar os itens referentes a esse pedido o colaborador deve clicar sobre uma linha da tabela. A Figura 4.22 exibe os detalhes de um pedido de mudança.

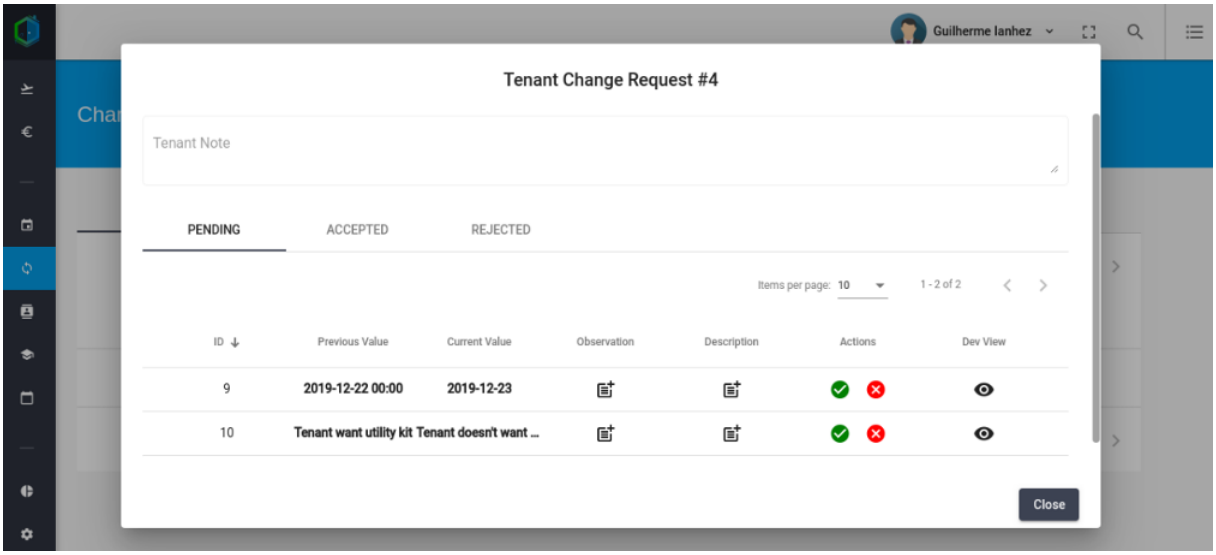


Figura 4.22: Informações sobre um pedido específico de mudança.

Através dessa tela é possível aceitar/rejeitar as mudanças, bem como exibir os itens que já foram validados. As observações/descrições podem ser adicionadas caso um pedido ainda esteja pendente. A Figura 4.23 exibe o formulário responsável por essa função.

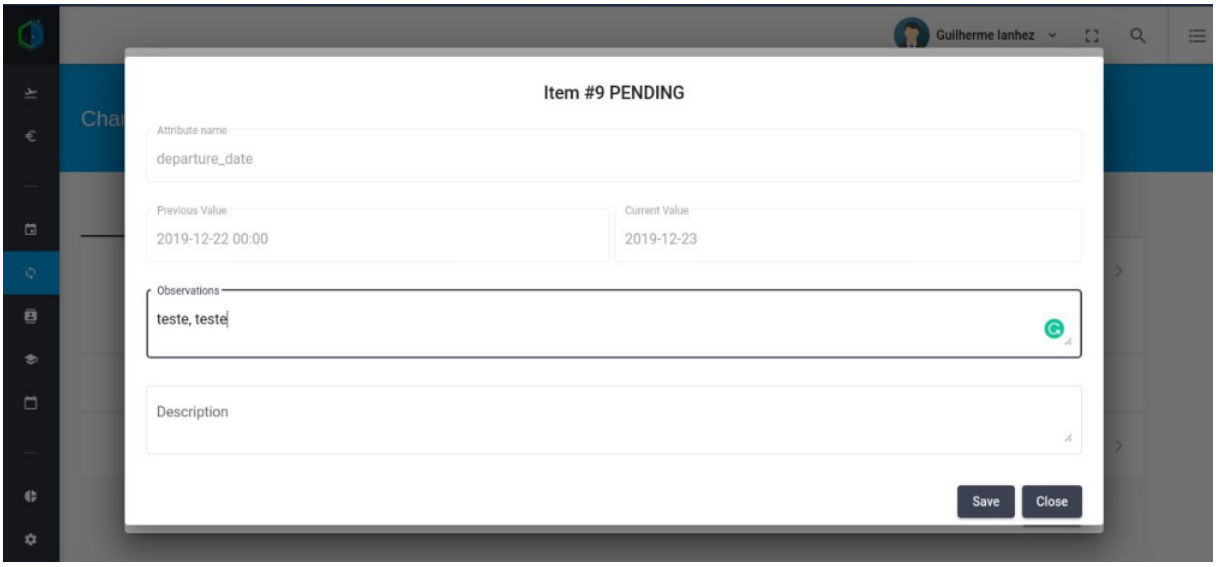


Figura 4.23: Formulário responsável por adicionar observações/descrições.

Para os pedidos que já foram validados é exibido a opção para reverter as mudanças efetuadas, bem como o colaborador que realizou tal ação. A Figura 4.24 exibe essa funcionalidade.

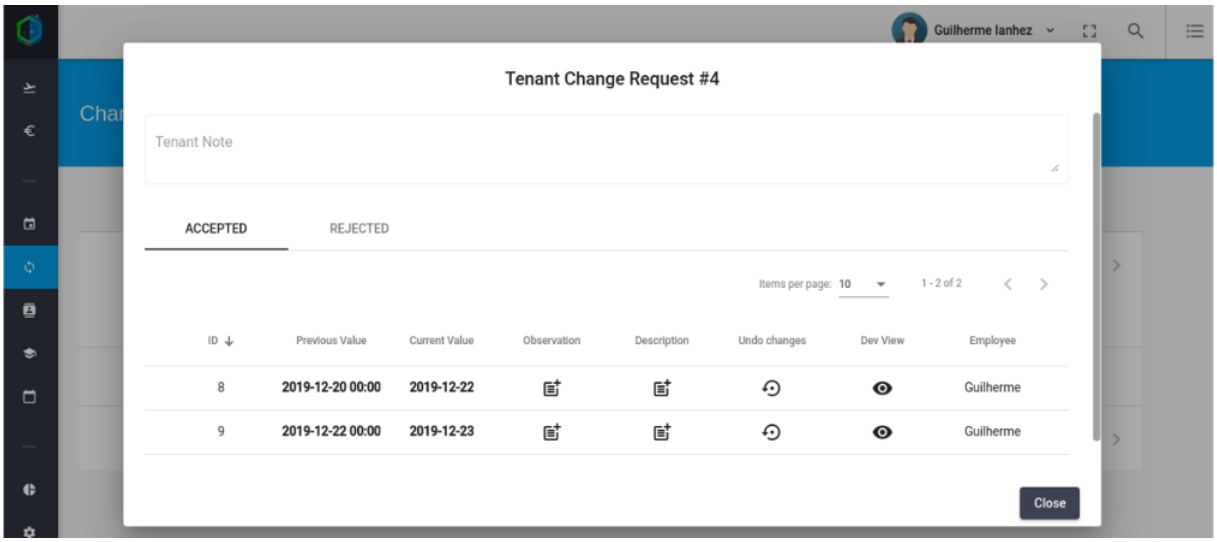


Figura 4.24: Tela de pedidos que já foram aceitos/rejeitados.

Dessa forma quando um colaborador desfaz a ação realizada, o pedido volta novamente para o status de pendente, sendo necessário uma nova validação. Vale ressaltar que não só o status é alterado mas também o valor da tabela alvo no banco de dados. Dessa forma, caso um colaborador aceitou uma mudança no kit de utilidades na tabela de reservas por exemplo, o kit voltará ao seu valor antigo antes da validação.

4.3.8 Gerenciar perfil

A edição do perfil, assim como a edição de reservas, necessita de autorização da empresa. Dessa forma, apenas alguns atributos podem ser alterados livremente, como a foto e sua senha. O nome, sobrenome, passaporte, entre outros precisam de avaliação prévia, pois a mudança desses dados podem levar a inconsistência com documentos armazenados e assinados pelo inquilino. A Figura 4.25 exibe a tela de perfil do usuário.

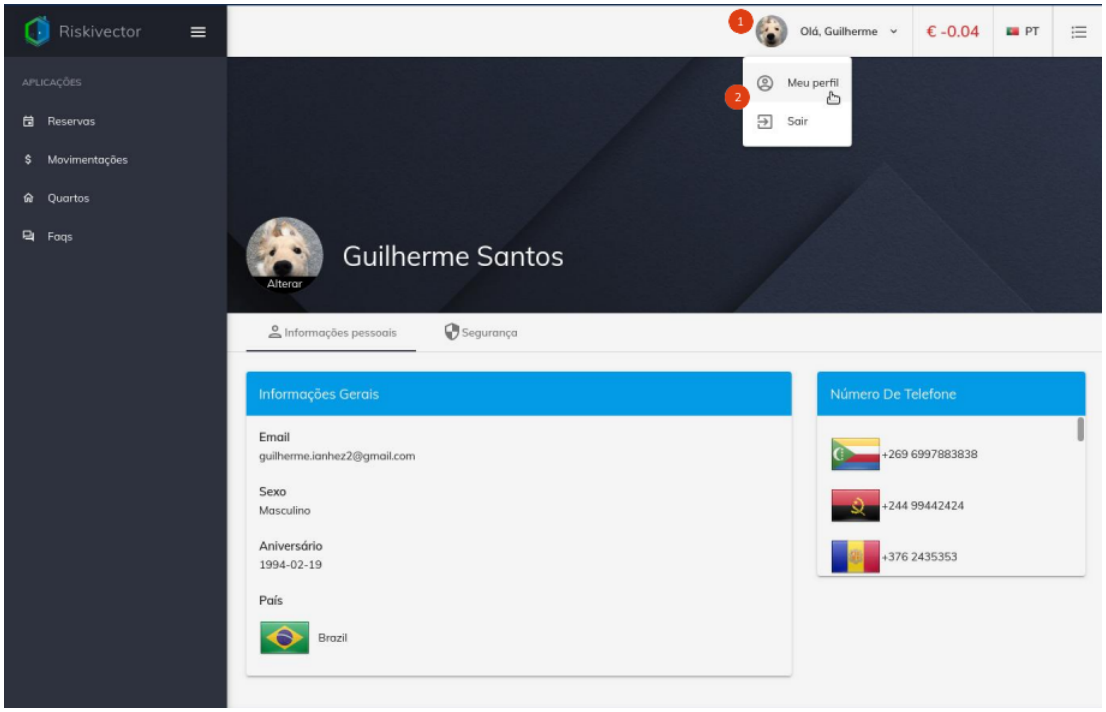


Figura 4.25: Informações do perfil - Inquilino.

O perfil é acessado clicando em sua foto ou nome na barra de ferramentas, localizada no canto superior da tela, como ilustra os números “1” e “2” da Figura 4.25. A troca da foto de perfil é feita seguindo os seguintes passos:

- O usuário clica em sua foto atual.
- O navegador exibe uma lista com as imagens.
- Após a escolha da imagem é aberto em uma janela de diálogo com sua foto.
- O usuário escolhe a área da foto que deseja, através de uma máscara de proporção 1 / 1. A proporção é necessária para a correta visualização da imagem dentro do círculo que ocupa na barra de ferramentas e em seu perfil. A Figura 4.25 exibe a janela de diálogo para seleção da foto.

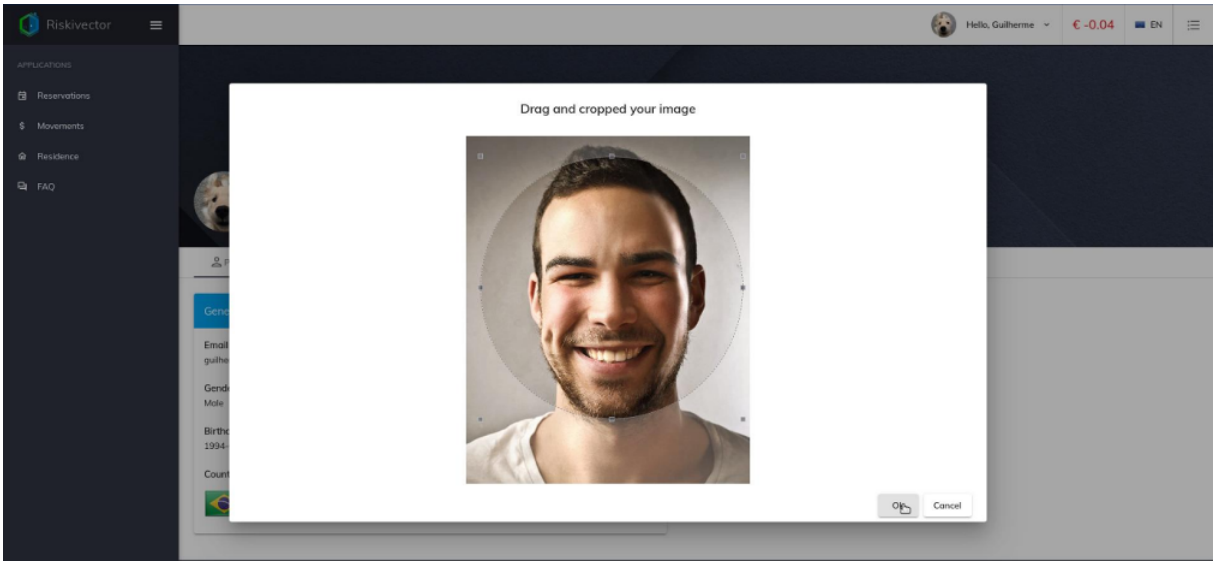


Figura 4.26: Janela para selecionar foto do inquilino.

Já para a troca de senha, o usuário deve seguir os seguintes passos:

- Acessar a aba segurança.
- Escolher entre as opções *mudar senha* ou *esqueceu a senha*. Caso o inquilino escolha a opção *esqueceu a senha*, é seguido o mesmo procedimento demonstrado na Seção 4.3.3.

A Figura 4.27 exibe a tela de mudança de senha.

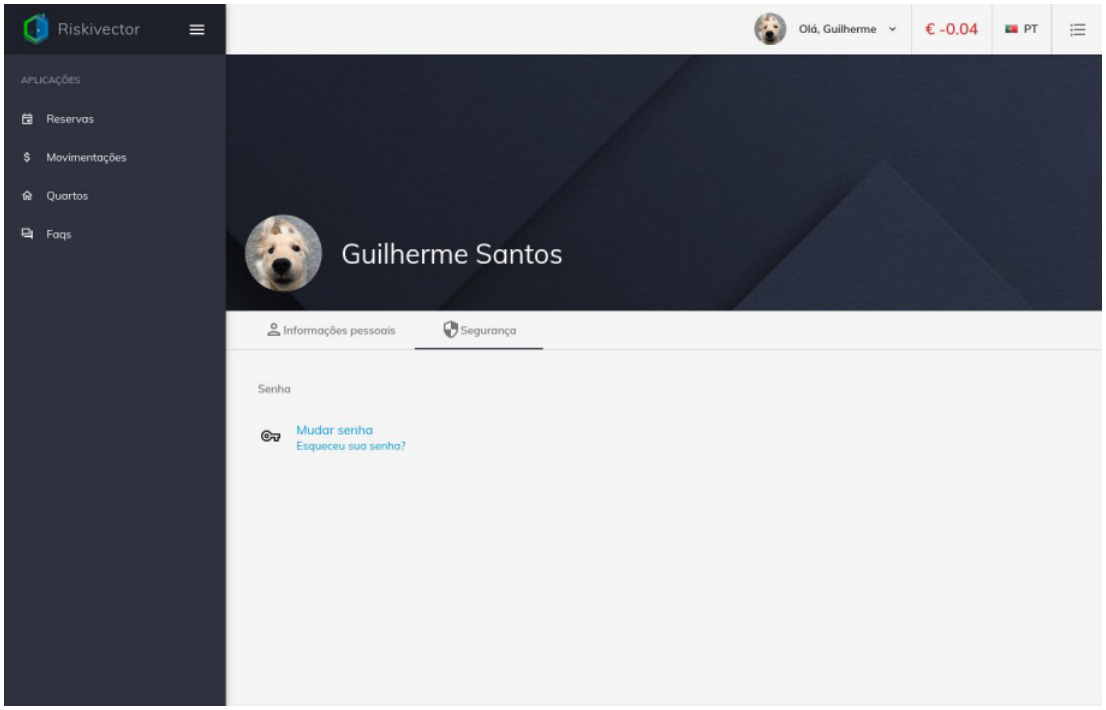


Figura 4.27: Tela responsável pelas configurações de segurança.

Caso o usuário selecione *Mudar senha*, é exibido uma janela de diálogo e o inquilino precisa inserir sua nova senha, como demonstra a Figura 4.28.

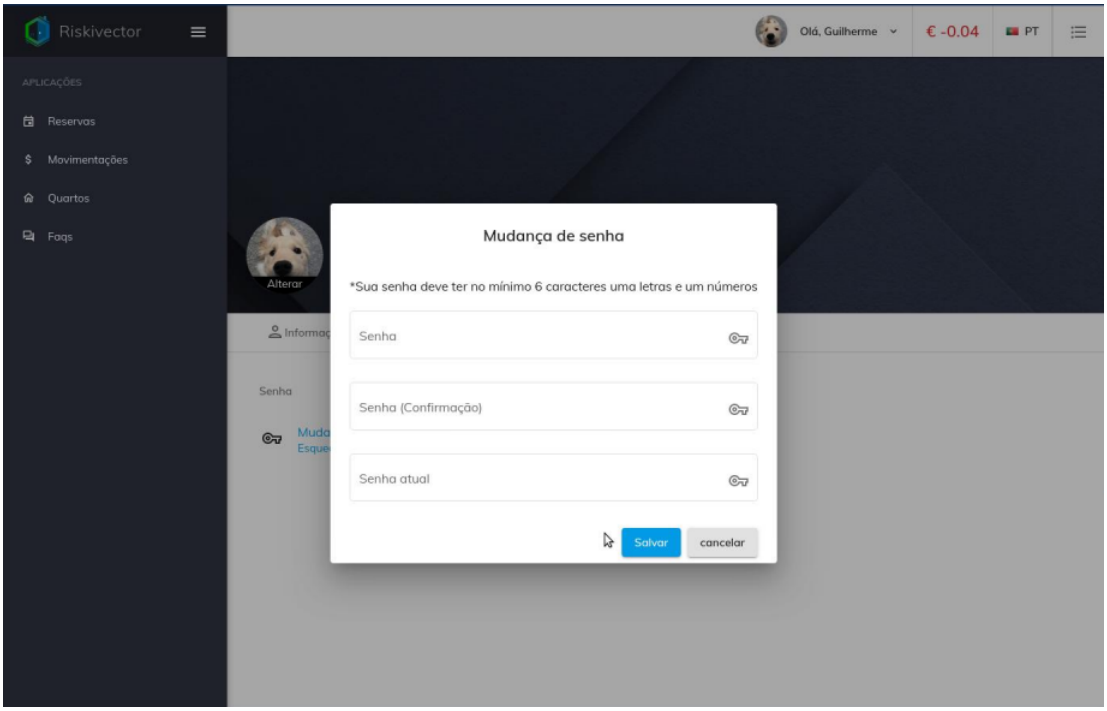


Figura 4.28: Janela de diálogo - mudança de senha.

4.3.9 Consultar movimentações e saldo

O saldo do inquilino leva em consideração o quanto ele deve ou seu crédito dentro da empresa. Ele é calculado baseado nas movimentações de contas e pagamentos efetuados. Existem três tipos possíveis de movimentações, elencados na Tabela 4.1.

Tipo	Descrição
Pagamentos	Armazena os pagamentos realizados pelo inquilino. Podendo ser pagamento de multa, renda, etc.
Contas	Armazena as contas da morada, água, luz, etc.
Diversos	Armazena outros tipos de contas, podendo ser multas, taxas, etc.

Tabela 4.1: Tipos de movimentações.

As movimentações podem ser filtradas baseadas em um intervalo de datas, ou pelo seu tipo. A Figura 4.29 exibe a tela responsável pela consulta e filtro das movimentações.

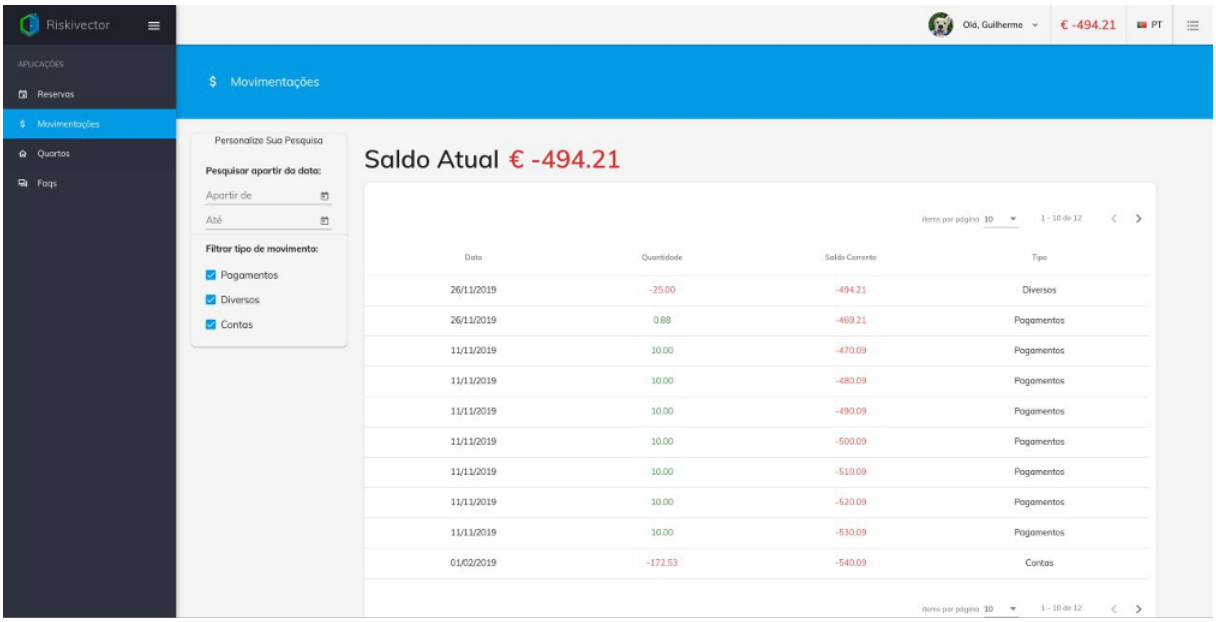


Figura 4.29: Tabela de movimentações e saldo.

Cada linha da tabela retrata uma movimentação, e as colunas especificam:

- **Data:** Especifica a data em que foi gerada.
- **Quantidade:** Valor referente à movimentação.
- **Saldo Corrente**: Saldo calculado após realizada a movimentação.
- **Tipo**: Pagamentos, contas ou diversos.

O saldo do inquilino é exibido em cima da tabela de movimentação e também na barra de ferramentas. Este é um componente compartilhado e que além de exibir o saldo, realiza o pagamento das contas, como descrito na Seção 4.3.11.

As informações referentes a cada tipo de movimentação, como recibos, faturas, etc. são exibidos ao clicar sobre uma movimentação. A Figura 4.30 exibe as informações específicas de uma movimentação.

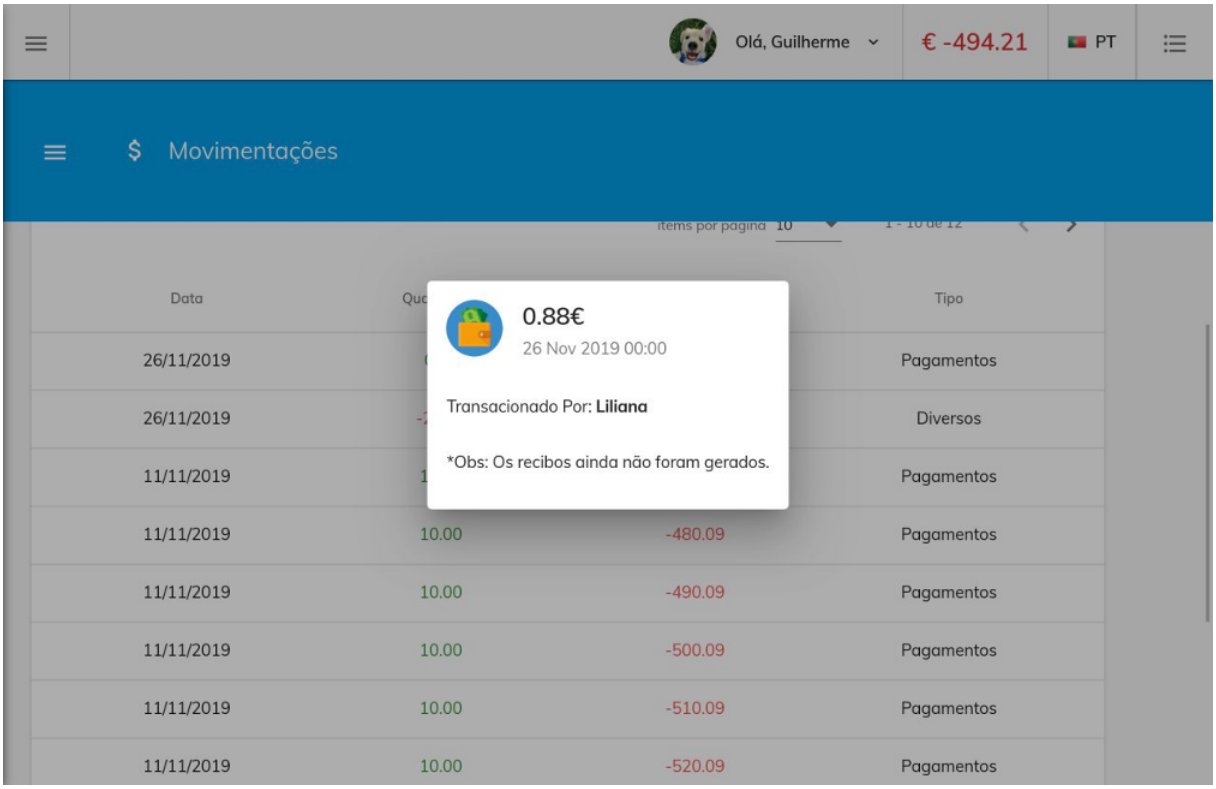


Figura 4.30: Janela de diálogo exibindo informações de um pagamento.

Esse exemplo ilustra uma movimentação do tipo pagamento de conta. Nele é demonstrado o modo de pagamento (dinheiro, cartão, etc.), a data, quantia e também um comprovante de pagamento se este já estiver sido emitido. Os demais tipos são exibidos analogamente, mostrando faturas e informações de acordo com sua especificidade.

4.3.10 Consultar informações da morada e pedir gás

Seguindo a modelação do banco de dados, um inquilino pode ter mais de uma morada. Dessa forma, caso o inquilino possua mais de uma morada é necessário especificar qual ele deseja gerenciar, como indicado na Figura 4.31. Caso ele possua apenas uma morada a aplicação redireciona-o para a página de gerenciamento de morada, indicado na Figura 4.32.

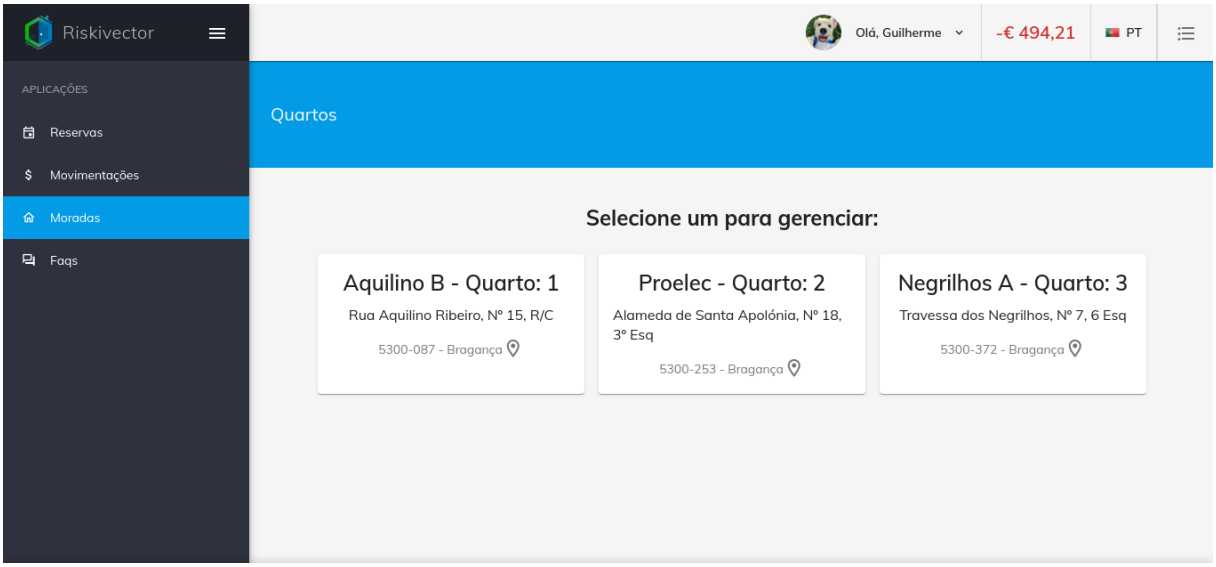


Figura 4.31: Tela responsável por exibir as moradas do inquilino.

Depois de escolhida a morada, o inquilino pode gerenciar os seguintes recursos: informações gerais, consultar leituras (água/gás) e realizar o pedido/consultar gás. A Figura 4.32 ilustra a tela que permite gerenciar uma morada.

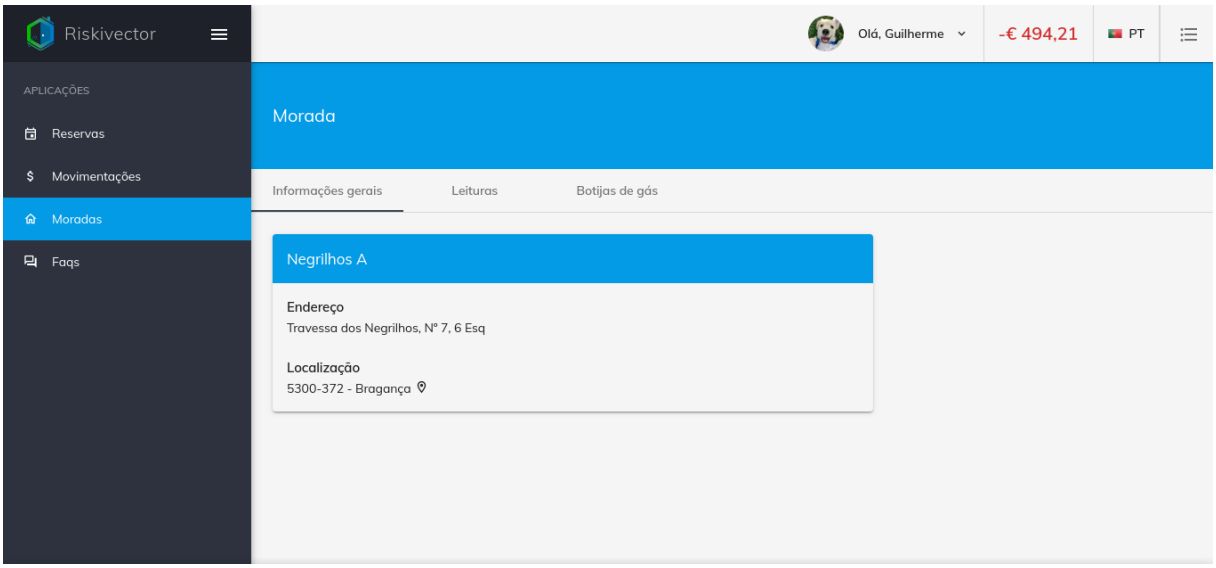


Figura 4.32: Informações gerais da morada e menu para gerenciar seus recursos.

As leituras são agrupadas pelo mês em que foram coletadas suas fotos. É possível verificar o valor da leitura na foto e também no rodapé da imagem. A Figura 4.33 ilustra essa funcionalidade.

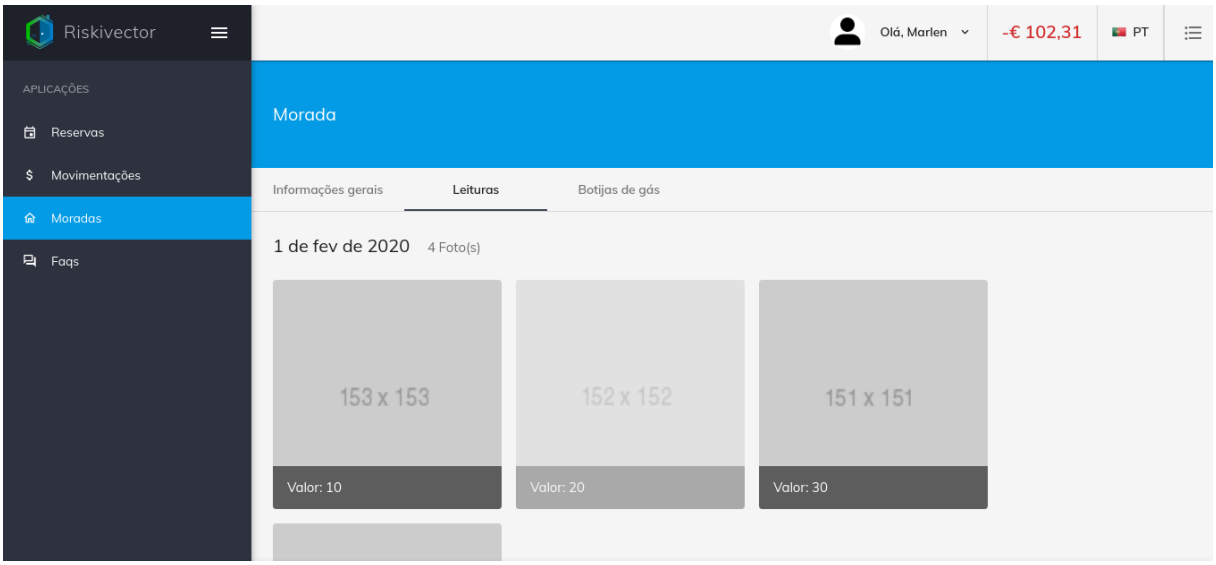


Figura 4.33: Tela responsável por exibir as leituras de água e gás da morada.

Por fim, as funcionalidades relacionadas ao gás são realizadas ao clicar sobre a aba "Botijas de gás". A Figura 4.34 ilustra a tabela que exibe as informações dos pedidos de gás que foram realizados.

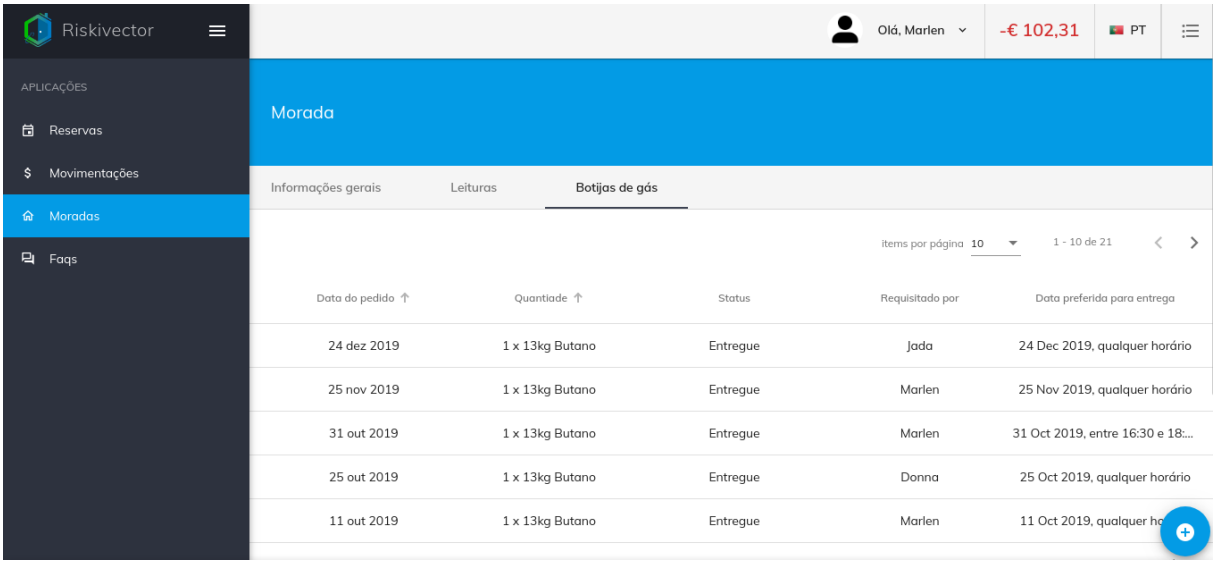


Figura 4.34: Tela responsável pela consulta de pedidos de gás de uma morada.

Para adicionar um novo pedido de gás o inquilino deve clicar sobre o botão "+" localizado no canto inferior direito, (ver Figura 4.34).

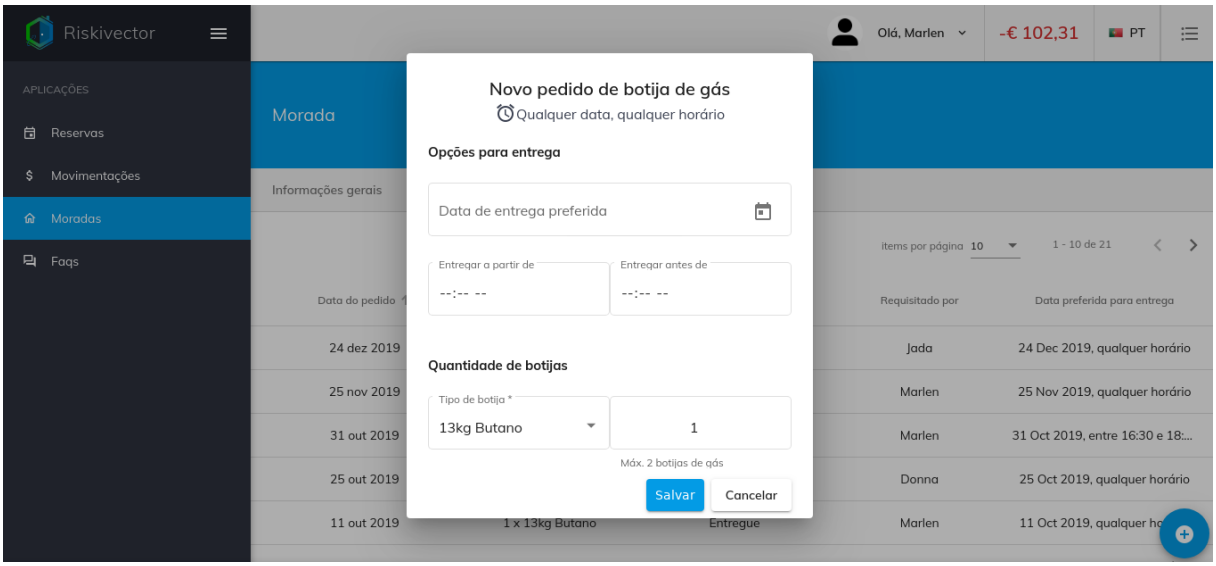


Figura 4.35: Formulário de requisição de gás.

Todos os campos em "Opções para entrega" ilustrados na Figura 4.36 são opcionais. Caso o usuário especifique uma data ou horário para a entrega o texto do subtítulo

também é alterado e atualizado de acordo com os dados coletados. Nesse caso, como não foi inserido nenhum dado a entrega será agendada para qualquer data e qualquer horário.

Em seguida o inquilino deve especificar qual botija de gás e a quantidade desejada.

Após enviar o formulário, o servidor encaminha um e-mail informando a data e horário (se houver), bem como a botija, quantidade e endereço da morada.

A Figura ilustra esse processo através de um diagrama de sequência.

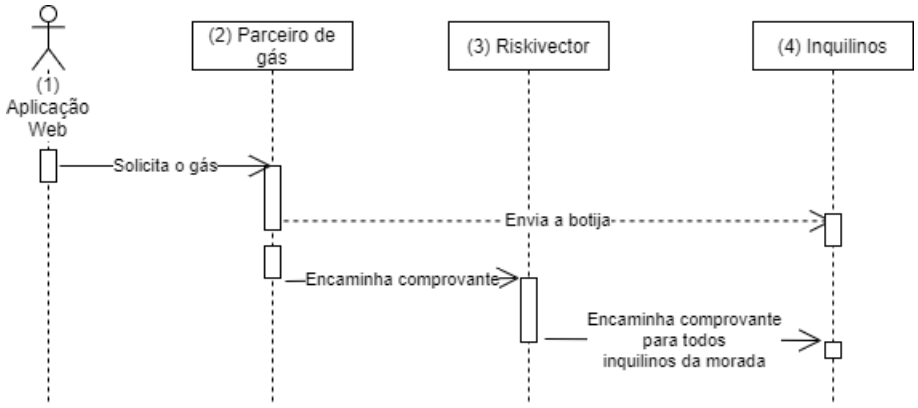


Figura 4.36: Diagrama de sequência - Requisição de gás.

4.3.11 Pagamento de contas

O pagamento de contas envolve diferentes processos na empresa. O primeiro passo para realizar o pagamento é feito pelo inquilino, através da aplicação web. O sistema de pagamentos é realizado por recarregamentos, ou seja, o inquilino não paga uma conta específica, como de aluguel, botija, etc. e sim escolhe uma quantia para creditar em sua conta. A Figura 4.37 demonstra os passos para exibir o formulário para envio de um pagamento.

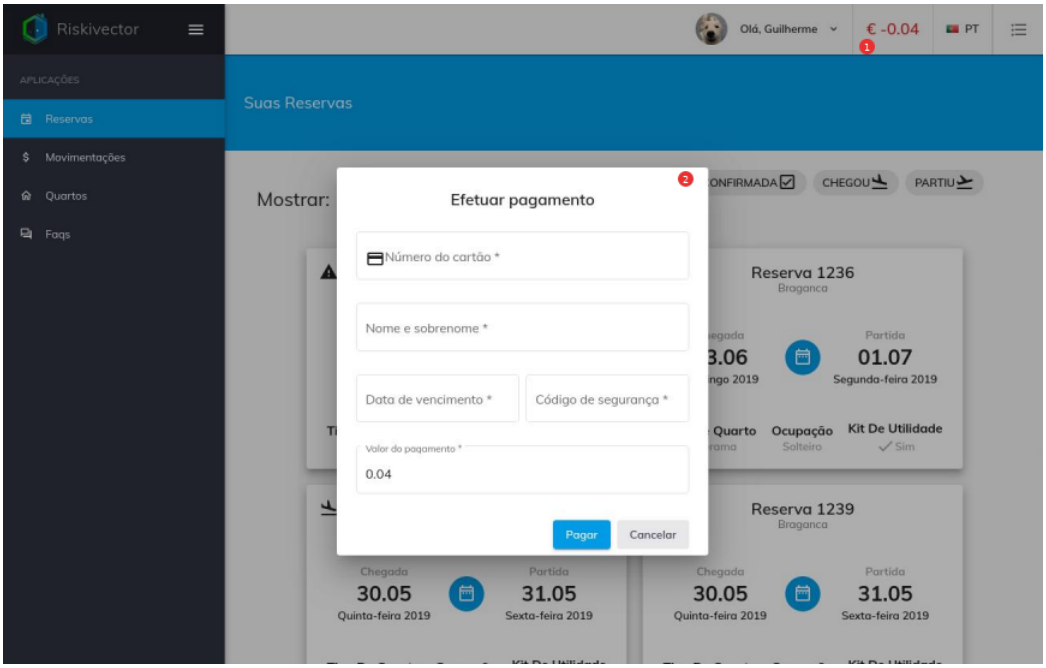


Figura 4.37: Formulário de pagamentos de contas.

Por padrão o componente de pagamento exibe no campo *Valor do pagamento*, a quantia total que o inquilino deve à empresa. Após o preenchimento do formulário a aplicação encaminha para o módulo financeiro da empresa. É nele que será processado e realizado as operações necessárias para efetuar o pagamento, contactando APIs da Caixa Geral de Depósitos e de faturas. A Figura 4.38 exibe o processo de pagamento de contas.

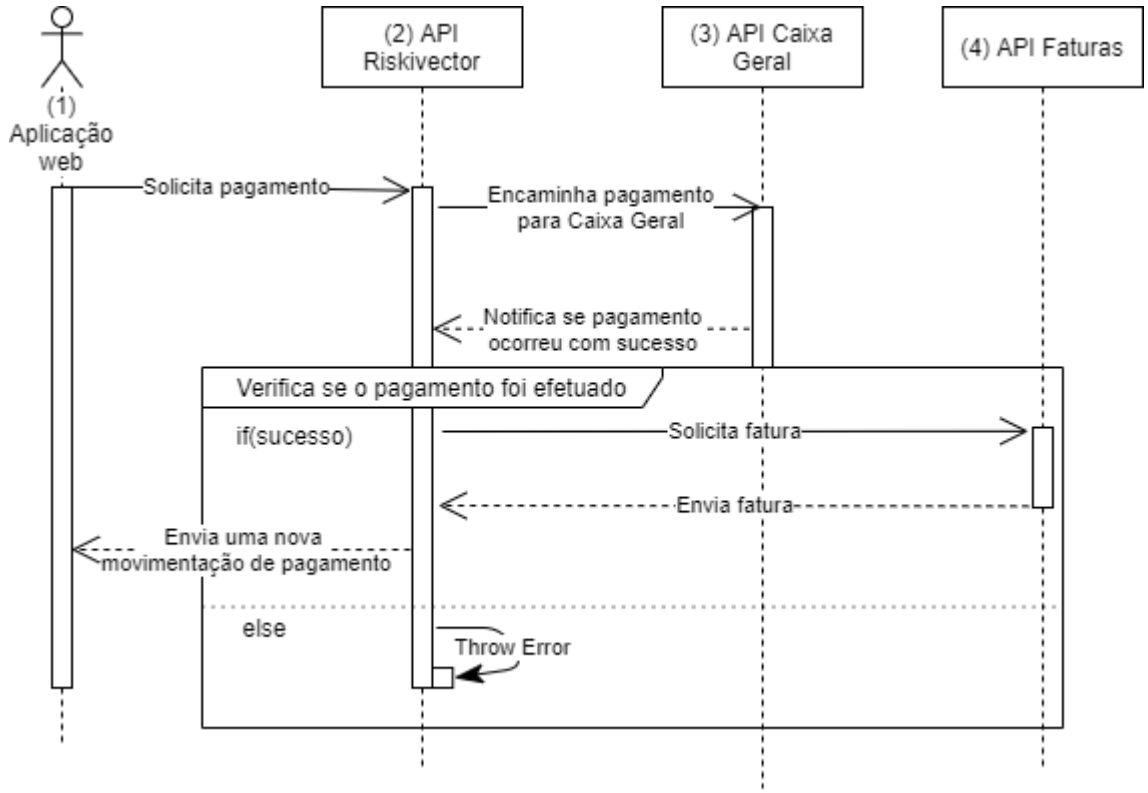


Figura 4.38: Diagrama de sequência - pagamento de contas.

Todo o processo para realização de pagamentos é feito pelo módulo financeiro da empresa. Esse módulo solicita o pagamento primeiramente a API da Caixa Geral, representado pelo número "2" do diagrama. Em seguida, caso o pagamento for efetuado com sucesso o módulo contacta uma segunda API para gerar o comprovante de pagamento. Se tudo ocorreu como o esperado o servidor cria uma nova movimentação de pagamento e o inquilino poderá acessar essas informações, assim como indicado pela Seção 4.3.9. Caso ocorra algum erro é lançado uma exceção e a aplicação web, indicada pelo "1", exibe uma notificação apontando o erro.

4.4 Considerações finais

Neste capítulo foi apresentado o desenvolvimento dos requisitos da aplicação web para clientes da empresa *Riskivector*, incluindo os problemas encontrados bem como as escolhas para solucioná-los.

Além do desenvolvimento da aplicação web, foi apresentado uma nova arquitetura para o servidor da empresa, que visa diminuir o acoplamento, aumentar a reusabilidade do código e provê uma melhor manutenibilidade. Essa arquitetura faz parte da nova versão do servidor, que opera através do TypeScript, possibilitando a tipagem e uso da orientação a objetos.

Capítulo 5

Conclusões e trabalhos futuros

Neste capítulo serão discutidas as conclusões referentes a esse trabalho, bem como os aspectos e funcionalidades a serem explorados em trabalhos futuros. O principal objetivo desse trabalho foi apresentar as diferentes atividades realizadas durante o estágio na empresa *Riskivector*. Por ser uma empresa que possui milhares de clientes, a demanda exige muito esforço por parte dos funcionários, tornando-se imprescindível a utilização de ferramentas tecnológicas como a desenvolvida/estudada neste trabalho.

Algumas funcionalidades dessa aplicação exigiram a integração com outros sistemas da empresa. Como mencionado no Capítulo 4, as validações realizadas por colaboradores *Riskivector* envolveram o sistema interno da empresa. Já os cadastros dos inquilinos é realizado em outra plataforma. Em vista disso, para o funcionamento correto da aplicação proposta nesse trabalho fez-se necessário agregar funcionalidades de outros três sistemas, sendo eles: aplicação interna dos colaboradores, o site oficial para realizar reservas da empresa e também o desenvolvimento da API que realiza a comunicação, provê a gerência e os dados para os demais sistemas.

Além do desenvolvimento da plataforma web, foi desenvolvida uma nova arquitetura no servidor da empresa, que busca solucionar antigas dívidas técnicas acumuladas ao decorrer do tempo, ocasionadas por não adotar um padrão estruturado. Assim sendo, foi criada uma nova versão do servidor que implementa uma arquitetura em camadas, buscando através da divisão lógica e objetiva de cada camada, organizar e prover maior

manutenibilidade para a empresa. Além da arquitetura, a nova versão conta com os benefícios de tipagem e orientação a objetos fornecidos pela linguagem TypeScript e também a injeção de dependência, provendo menor acoplamento e facilitando a criação e execução de testes.

Cabe ressaltar que esse trabalho implementa parte dos serviços que a empresa oferece para seus clientes. Diante disso, como trabalhos futuros, a empresa deseja ampliar suas funcionalidades e serviços ofertados pela aplicação, bem como melhorar os que já estão presentes no sistema, tais como:

- Nova funcionalidade de envio de pedidos de reparo na morada.
- Apresentar estatísticas de uso geral, como: água, gás, quantidade de botijas compradas no mês, etc.
- Melhorar o conteúdo da página principal.
- Realizar o agendamento para pedido de gás.
- Adicionar outras formas de pagamentos.
- Melhorar aspectos visuais na interface.
- Adicionar passos para pré configuração, caso seja o primeiro acesso do cliente.
- Solicitar pedido de mudança para outro apartamento.
- Aplicar questionários para avaliar a aplicação.

Desse modo, pode-se concluir que a implementação do site evidenciou as etapas e desafios que uma companhia enfrenta ao desenvolver um produto novo, bem como as estratégias para solucioná-los. Em suma, o estágio realizado na empresa proporcionou a aprendizagem em diferentes esferas do conhecimento. Conhecimentos estes que se aplicaram não só para o desenvolvimento profissional, adicionando novos conceitos e tecnologias como o framework Angular, mas também no convívio e a importância do trabalho em equipa dentro de uma empresa.

Bibliografia

Chacon, S. (2009). *Pro Git*. Books for professionals by professionals. Apress.

Core, A. (2019). Angular change detection - how does it really work?

DevMedia (2011). Orm : Object relational mapper.

Duckett, J. (2014). *JavaScript and JQuery: Interactive Front-End Web Development*. Wiley.

Duckett, J. (2016). *Html E Css Projete E Construa Websites*. ALTA BOOKS.

Erl, T. (2012). *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST*. Prentice Hall service-oriented computing series from Thomas Erl. Prentice Hall.

Fenton, S. (2017). *Pro TypeScript: Application-Scale JavaScript Development*. Apress.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Rfc 2616, hypertext transfer protocol – http/1.1.

Foundation, M. (2019a). Api.

Foundation, M. (2019b). Qual a diferença entre página web, site, servidor web e mecanismo de busca?

Foundation, M. (2020). Express/node introduction.

Foundation, O. (2019c). About node.js.

Foundation, W. W. W. (2000). History of the web.

Kurose, J.F. e Ross, K. (2013). *Redes de computadores e a internet: uma abordagem top-down*. ADDISON WESLEY BRA.

Martínez, J. M. e Santos, A. S. (1998). *METODOS COMPUTACIONAIS DE OTIMIZAÇÃO*.

Minnick, C. (2016). The real benefits of the virtual dom in react.js.

Mulligan, G. and Gračanin, D. (2009). A comparison of soap and rest implementations of a service based interaction independence middleware framework. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 1423–1432.

Org, S. (2019). What is scrum.

Pires, J. (2016). O que é api? rest e restful? conheça as definições e diferenças!

Richards, M. (2015). *Software Architecture Patterns*. O'Reilly Media, Incorporated.

Richardson, L. and Ruby, S. (2007). *RESTful Web Services*. O'Reilly, Beijing.

Schwaber, K. and Sutherland, J. (2012). *Software in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, and Leave Competitors in the Dust*. ITPro collection. Wiley.

Stack Exchange, I. (2019). Developer survey results 2019.

W3C (2001). Uris, urls e urns: Esclarecimentos e recomendações 1.0.

W3C (2016a). Html & css.

W3C (2016b). Javascript web apis.