Capítulo IV – Melhorando a comunicação com a imobiliária por meio de um sistema web

Silvio Aparecido Capelo[12]
Cadmiel Matioli Donato[13]
Rafael Augusto Balabem[14]
Anderson Paulo Ávila Santos[15]
Fabio Takeshi Matsunaga[16]
Lucas Busatta Galhardi[17]

RESUMO

Com o advento da tecnologia, o uso de serviços online tem crescido nos últimos anos, especialmente com a recente pandemia da Covid-19, em que o *lockdown* é exigido. Um dos setores que tem mais sofrido demanda é o de mercado imobiliário. Existem problemas como falta de comunicação direta com a imobiliária, falta de eficiência nas buscas e localizção e baixa qualidade das imagens. Considerando-se o exposto, o objetivo deste projeto é melhorar a comuniação entre os consumidores e a imobiliária por meio de UX. Para isso, será desenvolvido um sistema *web* com arquitetura MVC. O front-end foi desenvolvido em HTML, CSS e Javascript e o back-end em framework Laravel, que foi responsável por gerenciar os dados dos usuários, imóveis e as imagens. O sistema web torna o acesso ao serviço mais fácil, podendo ser realizado por qualquer dispositivo que tenha acesso à Internet.

Palavras-chave: API, *back-end*, *front-end*, Inteligência Artificial, *Machine Learning.*

Improving communication with the real estate company through a web system

ABSTRACT

[12] Graduando em Engenharia de Software.
[13] Graduando em Engenharia de Software.
[14] Graduando em Engenharia de Software.
[15] MSc. Anderson Paulo Ávila Santos. E-mail: anderson.avila@sistemafiep.org.br
[16] MSc. Fabio Takeshi Matsunaga. E-mail: fabio.matsunaga@sistemafiep.org.br
[17] MSc. Lucas Busatta Galhardi. E-mail: lucas.galhardi@sistemafiep.org.br

With the advent of technology, the use of online services has grown in recent years, especially with the recent Covid-19 pandemic, in which the lockdown is required. One of the sectors that has suffered the most demand is the real estate market. There are problems such as lack of direct communication with the real estate company, lack of efficiency in searches and location and low quality of images. Considering the above, the objective of this project is to improve communication between consumers and real estate through UX. For this, a web system with MVC architecture will be developed. The front-end was developed in HTML, CSS and Javascript and the back-end in Laravel framework, which was responsible for managing user data, properties and images. The web system makes access to the service easier and can be performed by any device that has access to the Internet.

Key-words: API, Artificial Intelligente, back-end, front-end, Machine Learning.

1. INTRODUÇÃO

Com o crescimento da quantidade de sites imobiliários, a tendência é que as pessoas deixem de se deslocar até uma imobiliária procurar um imóvel e passem a acessar estes serviços de forma *online*. O surgimento da pandemia do Covid-19 acelerou esse processo, uma vez que para conter a doença em vários estados foram tomadas medidas de segurança, tais como o *lockdown* (confinamento). Fato este que gerou um aumento no número de pessoas que buscam sites imobiliários ou realizam transações comerciais na *web*. (BALENA e PEGORINI, 2019)

De acordo com Giglio *et al*. (2006) o uso da internet pode representar um grande diferencial, pois ela proporciona maior agilidade na prestação de serviço. A plataforma Google em um de seus artigos afirma que houve um aumento de 34% na procura de imobiliárias no período de maio de 2021. Este quadro que se apresenta, sugere que as empresas que não se adequarem a esse novo formato de negócio perderão espaço no mercado.

Segundo Moro e Andrade (2020), em seu artigo Qualidade de Sites Imobiliários por meio de satisfação do cliente, a pesquisa dicômica, chega à conclusão de que 67% dos clientes que fizeram negociação com imobiliárias através da internet não estão satisfeitos com os serviços recebidos e tiveram um score abaixo de zero. Os pontos críticos apontados na pesquisa e que servem de base para a idealização desses projetos foram: falta de Informação dos produtos, buscas complexas, fotos

fornecidas dos imóveis fora de padronização, falta de comunicação direta com imobiliária, responsividade o site entre outros.

De acordo com os dados obtidos pela pesquisa acerca do mercado imobiliário *online*, que demonstra uma demanda crescente na procura de imóveis através da internet, a experiência do cliente tem se mostrado deficitária em alguns aspectos, tais como insatisfação com informações sobre os imóveis, falta de comunicação direta com a imobiliária, insatisfação com filtros de buscas, insatisfação com a geolocalização dos imóveis, baixa qualidade de fotos dos imóveis e a falta de responsividade para acesso em dispositivos móveis (MORO e ANDRADE, 2020).

De encontro com a velocidade em que o mercado imobiliário se desenvolve na *web*, o objetivo deste presente projeto é desenvolver um sistema web que correspondam as expectativas dos usuários utilizando conceito de UX (*User Experience* ou Experiência do Usuário) e implementar as ferramentas necessárias para melhorar a comunicação entre consumidores e imobiliárias.

2. METODOLOGIA

2.1 Processo de *software* e gestão

Para este projeto, a metodologia aplicada para o desenvolvimento do *software* foi o Scrum, que se trata de uma metodologia ágil de desenvolvimento incremental e iterativo por meio da qual são produzidas e entregues partes dos softwares a cada iteração. De acordo com Cruz (2013), a ideia principal do Scrum é que um pequeno time de pessoas pode tratar e resolver problemas complexos e adaptativos enquanto entrega produtos de forma criativa.

Para gerenciar as tarefas, foi utilizada o Trello, que é uma ferramenta colaborativa que organiza projetos em quadros (*boards*), em que são inseridas listas de tarefas a serem seguidas individualmente ou em equipe.

Para o versionamento dos códigos-fontes do projeto, foi utilizado o GIT, que é um sistema de controle de versões distribuído, usado principalmente no desenvolvimento de *software*. Para o compartilhamento das versões do projeto, foi utilizado o GitHub, que é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o GIT.

2.2 Tecnologias

Para o desenvolvimento do sistema *web*, foram utilizados alguns *frameworks* para o desenvolvimento do *front-end* e *back-end*. Um *framework* é uma estrutura-base que contém um conjunto de funções e componentes pré-definidos, funções e

componentes estes que se relacionam para disponibilizar funcionalidades específicas ao desenvolvimento de *software*.

O Laravel foi utilizado para o desenvolvimento do *back-end*. Trata-se de um *framework* MVC de código aberto para PHP, robusto que fornece fácil desenvolvimento de aplicações web com recursos como um sistema de empacotamento modular com um gerenciador de dependências dedicado, acesso a bancos de dados relacionais e outros utilitários para implantação e manutenção de aplicações (STAUFFER, 2019).

Para o *front-end*, foi utilizado HTML para gerenciar o conteúdo e a estrutura de uma página na *web*, Javascript para a interação da página e o CSS (*Cascading Style Sheets*) para a estilização da página. Javascript é uma linguagem de script ou programação que permite implementar recursos complexos em páginas da web. Para o projeto, o Javascript foi utilizado em conjunto com o Ajax, que é o acrônimo para JavaScript assíncrono + XML, utilizado para tornar páginas Web mais interativas com o usuário. Além disso, em conjunto com o CSS, foi utilizado o Bootstrap, um *framework* CSS que organiza e gerencia o *layout* de um site.

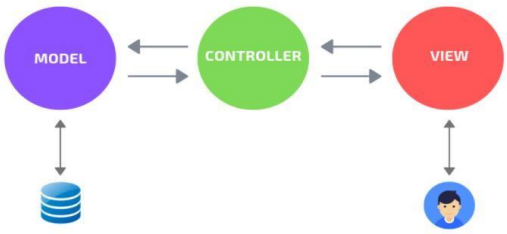
O sistema foi desenvolvido utilizando MVC, que é o acrônimo de *Model-View-Controller* (LUCIANO e ALVES, 2011). O MVC define um padrão arquitetural, que funciona da seguinte forma:

Model é a camada responsável pela parte lógica da aplicação, que gerencia todos os recursos (consultas ao BD, validações e notificações).

View é a camada responsável por exibir dados para o usuário, seja em páginas HTML, JSON, XML. A camada *View* não possui responsabilidade de saber quando vai exibir os dados, apenas como irá exibi-los.

Controller recebe todas as requisições do usuário. Seus métodos chamados *actions* são responsáveis por uma página, controlando qual model usar e qual *view* será mostrado ao usuário.

Figura 1 – Arquitetura MVC.



Fonte: os autores.

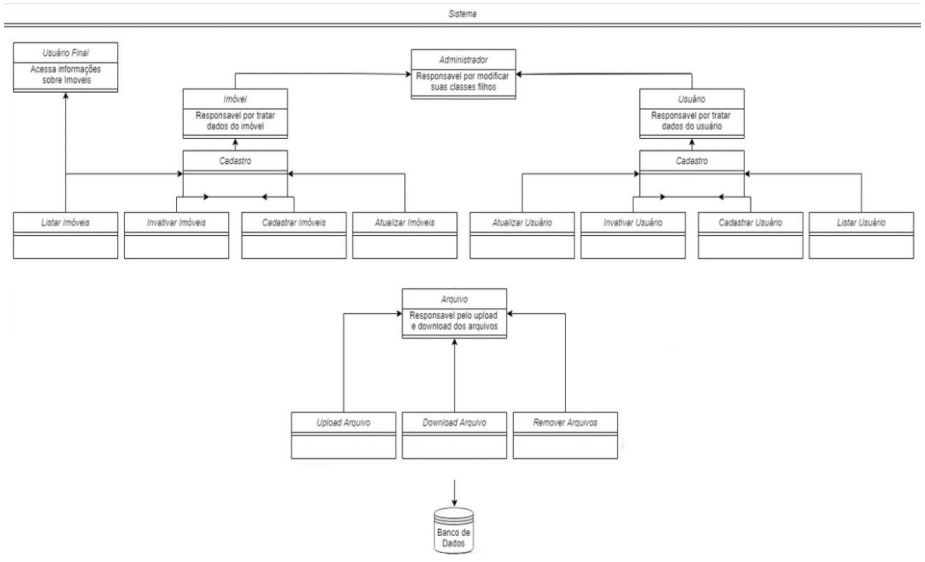
Ao receber uma requisição, o *Controller* solicita ao Model as informações necessárias (que virão do banco de dados), que as obtém e retorna ao *Controller*. De posse dessas informações, o *Controller* as envia para a *View* que irá renderizá-las, como mostra a Figura 1.

3. APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

3.1 Diagramas de modelagem

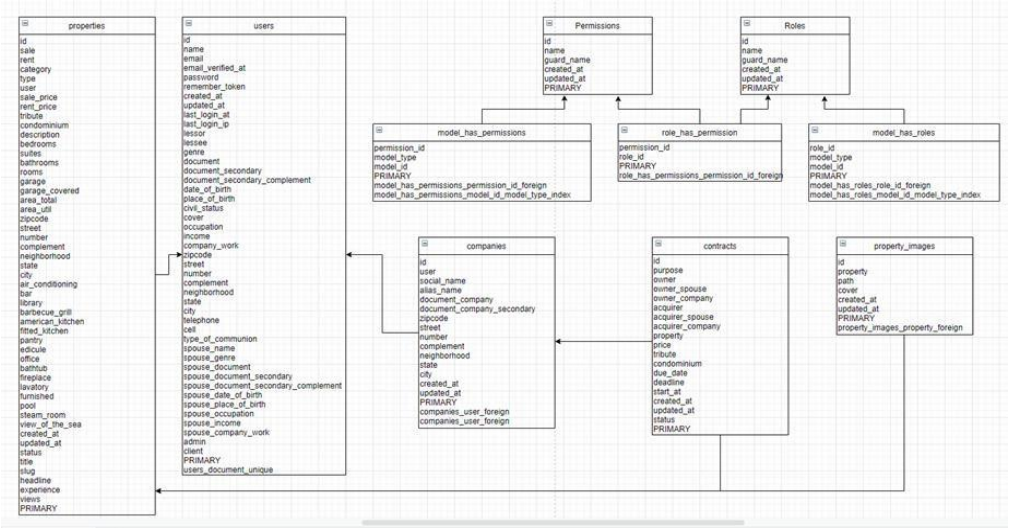
Os diagramas foram utilizados para facilitar o entendimento da solução arquitetônica adotada. É um recurso bastante didático, em que vários aspectos do projeto são explicados de maneira direta e visual. O diagrama de arquétipos auxilia na visualização da arquitetura do projeto como um contexto geral, possibilitando entender quais os pontos principais de acordo com a Figura 2.

Figura 2 - Arquétipos arquiteturais.



Fonte: os autores.

Figura 3 - Diagrama de entidade e relacionamento (DER).



Fonte: os autores.

3.2 Desenvolvimento do sistema

Nesta sessão serão apresentados os principais códigos para exemplificar o desenvolvimento do sistema. A partir deste ponto será explicado como foi desenvolvido a classe de Usuário, assim como seus métodos que servirão de base para implementação das demais classes. O início da implementação da classe se dá a partir das importações de bibliotecas e componentes necessários para desenvolvimento de todas as funcionalidades.

Para otimizar o código foi utilizado o ORM (*Object Relational Mapping*), responsável por aprimorar as buscas de dados e pelo auxílio no envio de informações para o banco de dados através das classes. Com o objetivo de demonstrar com mais clareza temos um exemplo abaixo da implementação de um método que realiza buscas através da classe de Usuário com utilização do ORM (Figura 4).

Figura 4 - Criação da classe e método de listagem de usuários

```
class UserController extends Controller
{
    public function index(){
        $users = User::get();
        return view('admin.users.index', compact('users'));
    }
```

Fonte: os autores.

Figura 5 – Método de criação de usuários

```php
public function create(){
    return view('admin.users.create');
}

public function store(UserRequest $request){

    $userCreate = User::create($request->all());

    return redirect()->route('admin.users.edit', [
        'user' => $userCreate->id
    ])->with(['color' => 'green', 'message' => 'Cliente cadastrado com sucesso!']);
}
```

Fonte: os autores.

Seguindo modelo de implementação de classe fornecido pelo Framework Laravel, foram criados vários métodos com a finalidade de tramitar as informações entre o *back-end* e *front-end*. O método *create* tem a função de chamar o código que mostra a parte visual da aplicação, assim como passar dados recebidos do banco de dados para que seja mostrado em tela, enquanto o método *store* se encarrega de enviar de fato as informações para o banco de dados, conforme mostra a Figura 5.

Do mesmo modo se dá a implementação dos métodos de edição, onde a função *edit* se encarrega de mostrar o formulário, assim como servir de condutor para enviar as informações necessárias para a edição do usuário. O método *update* se encarrega de receber as informações vindas do formulário através da classe *Request* (requisição vida do cliente pelo formulário) e enviar essas informações para o banco de dados. Na figura 6, é apresentado o método *update* que recebe todos os dados de usuários provenientes da classe *Request* e posteriormente serão salvos no banco de dados.

Figura 6 – Método update, para edição de usuários.

```php
public function edit(User $user){
    return view('admin.users.edit', compact('user'));
}

public function update(Request $request, User $user){
    $user->update($request->all());
    return redirect()->route('admin.users.index')
            ->with(['color'
                => 'green', 'message'
                => __('messages.users.update.success')]);
}
```

Fonte: os autores.

Tendo como base a implementação da classe de usuário, a classe *properties* matém a mesma lógica de funcionamento. A figura 7 apresenta a criação e definição do nome da classe com seu primeiro método, a listagem de propriedades e retorno da interface. A lista é adquirida do banco de dados de forma decrescente.

Figura 7 - Criação da classe e método de listagem de propriedades.

```php
class PropertyController extends Controller
{
    public function index(){

        $properties = Property::orderBy('id', 'DESC')->get();

        return view('admin.properties.index', compact('properties'));
    }
```

Fonte: os autores.

Para criar uma propriedade e fazer o registro no banco de dados é necessário ter dois métodos: o *create* que fica responsável de retornar os devidos dados para comtemplar o relacionamento entre classes, possibilitando o registro que será feito através do método *store*, pelo qual serão enviados os dados a serem salvos, retornando para a tela de edição acordo com a Figura 8.

Para proceder na edição de propriedades são necessários dois métodos: o *edit* e o *update* já anteriormente citados. Através do método *imageSetCover* é possível definir qual será a imagem principal do anúncio da propriedade. O método *imageRemove* removerá uma imagem da propriedade de acordo com a Figura 9.

Figura 8 – Criação de propriedades.

```php
public function create(){

    $users = User::orderBy('name')->get();

    return view('admin.properties.create', compact('users'));
}

public function store(PropertyRequest $request){

    $createProperty = Property::create($request->all());

    $validator = Validator::make($request->only('files'), ['files.*' => 'image']);

    if ($validator->fails() === true) {
        return redirect()->back()->withInput()->with([
            'color' => 'orange',
            'message' => 'Todas as imagens devem ser do tipo jpg, jpeg ou png.',
        ]);
    }

    foreach ($request->allFiles()['files'] as $image) {
        $propertyImage = new PropertyImage();
        $propertyImage->property = $createProperty->id;
        $propertyImage->path = $image->store('properties/' . $createProperty->id);
        $propertyImage->save();
        unset($propertyImage);
    }

    return redirect()->route('admin.properties.edit', [
        'property' => $createProperty->id,
    ])->with(['color' => 'green', 'message' => 'Imóvel cadastrado com sucesso!']);
}
```

Fonte: os autores.

Figura 9 – Definir e remover imagens de propriedades.

```php
public function imageSetCover(Request $request){
    $imageSetCover = PropertyImage::where('id', $request->image)->first();
    $allImage = PropertyImage::where('property', $imageSetCover->property)->get();

    foreach ($allImage as $image) {
        $image->cover = null;
        $image->save();
    }

    $imageSetCover->cover = true;
    $imageSetCover->save();

    $json = [
        'success' => true,
    ];

    return response()->json($json);
}

public function imageRemove(Request $request){

    $imageDelete = PropertyImage::where('id', $request->image)->first();

    Storage::delete($imageDelete->path);
    Cropper::flush($imageDelete->path);
    $imageDelete->delete();

    $json = [
        'success' => true,
    ];
    return response()->json($json);
}
```

Fonte: os autores.

Fundamentando-se na implementação descrita até o momento, pode-se observar que há um padrão que norteia o desenvolvimento do sistema. A criação de cada classe que o compõe segue um padrão que se repete durante todo o projeto. Portanto, o que foi abordado demonstra claramente como se desenvolverá a implementação das demais classes. O projeto do *front-end* é constituído de um *layout* que representa toda a parte visual do projeto. Os principais componentes gerados são:

Painel administrativo para a gestão de Funcionários, Cadastros e Permissões;

Cadastros de Contratos;

Painel Administrativos com fluxo de Caixa;

Implementações de Geolocalização e mapas com Street View;

Integração com Redes Sociais;

Filtros simples e avançados para buscas mais detalhadas;

Chat para facilitar a comunicação com a imobiliária;

Módulo de conexão com WhatsApp.

Deste modo, vale ressaltar que todas estas funcionalidades foram implementadas levando em consideração as pesquisas referentes a satisfação dos clientes, já mencionada nesta pesquisa e que serviu como razão primordial o modo o qual foi concebido.

4. CONCLUSÃO

O desenvolvimento deste projeto visa fornecer um software de gestão Imobiliária, melhorando a qualidade e agilidade dos processos administrativos. A linguagem de programação utilizada, o PHP, foi utilizada devido à grande familiaridade dos desenvolvedores com a linguagem e por ter uma vasta quantidade de biblioteca disponível para integração na aplicação, suprindo as demandas e requisitos. O próximo passo foi a escolha por aplicação *web* para que pudesse ser acessível de qualquer máquina com acesso à internet, seja de um computador, celular ou tablet, simplificando o acesso.

Para além de tudo que já foi implementado, mais recursos serão agregados futuramente para melhorar ainda mais a experiência do usuário tanto do ponto de vista do empresário, quanto do ponto de vista do cliente. Serão implementadas melhorias como: alertas de ajustes de contratos, notificações, notas fiscais, baixa automática de boletos, melhorias na área de cliente, melhorias no *dashboard*, entre outros.

O presente trabalho trouxe um grande aprendizado, ambos relacionados à modelagem, análise de demanda e implantação de sistema. Durante todo este processo, muitos testes e alterações foram feitos até que um resultado mais próximo do ideal fosse obtido, podendo superar as expectativas.

No processo de execução do trabalho, foram encontradas algumas dificuldades. Entre as que se destacam podemos citar os mapas e documentos. Através dos conceitos apresentados neste artigo, será possível desenvolver novas aplicações que buscam como principal objetivo a experiência do usuário com foco na resolução de problemas, proporcionando maiores benefícios e informações aos usuários.

REFERÊNCIAS

BALENA. V. M.; PEGORINI, V. Sistema web para gestão imobiliária. 2019. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná (UTFPR), Pato Branco, 2019.

CRUZ, Fabio. **Scrum e PMBOK unidos no Gerenciamento de Projetos**. Editora Brasport, 2013. 416 p.

GIGLIO, E. M.; PEREIRA, P. G.; RYNGELBLUM, A. Investigação sobre as relações entre a internet e as mudanças estratégicas, exemplificadas no mercado imobiliário. **Revista Brasileira de Gestão de Negócios**, v. 8, n. 21, p. 43-54, 2006.

LUCIANO J.; ALVES, W. J. Padrão de arquitetura MVC: Model-view-controller. **Revista EPeQ Fafibe 3ª. Ed.**, v. 1, 2011.

MORO, M. F.; ANDRADE, D. F. Avaliação da Qualidade de WebSites de Imobiliária por Meio da Avaliação dos Usuários, novembro de 2020. In. XL Encontro Nacional de Engenharia de Produção, 2020. Foz do Iguaçu. **Anais**... Foz do Iguaçu: S.N., 2020.

STAUFFER, M. **Laravel: Up & running**: A framework for building modern PHP apps. O'Reilly Media, 2019.