FACULDADE DE TECNOLOGIA DE AMERICANA "MINISTRO RALPH BIASI"

Curso Superior de Tecnologia em Jogos Digitais

Eduardo Henrique de Souza

DENTALCLOUD: Solução para laboratórios de prótese dentária e comunicação entre clínicas odontológicas

Americana, SP

2020

FACULDADE DE TECNOLOGIA DE AMERICANA "MINISTRO RALPH BIASI"

Curso Superior de Tecnologia em Jogos Digitais

Eduardo Henrique de Souza

DENTALCLOUD

Solução para laboratórios de prótese dentária e comunicação entre clínicas odontológicas

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Jogos Digitais, sob a orientação do (a) Prof.(a) Esp. Antonio Alfredo Lacerda.

Área de concentração: produção de *software web*.

Americana, SP

2020

Eduardo Henrique de Souza

DENTALCLOUD: Solução para laboratórios de prótese dentária e comunicação entre clínicas odontológicas

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Jogos Digitais pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.

Área de concentração: produção de *software web*.

Americana, 15 de dezembro de 2020.

Banca Examinadora:

_____
Antonio Alfredo Lacerda (Presidente)
Mestre
Fatec Americana

_____
Leonardo de Souza Lima (Membro)
Mestre
Fatec Americana

_____
José Mário Frasson Scafi (Membro)
Mestre
Fatec Americana

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus pelas oportunidades que tenho conquistado em minha vida e pela minha família que sempre me apoiou com os estudos. Agradeço também, todo apoio prestado por colegas de trabalho, professores e amigos que me auxiliaram em pontos em que faltava o meu conhecimento.

RESUMO

O presente trabalho tem por objetivo, demonstrar o processo de pesquisa e desenvolvimento de um *software.* O conteúdo deste projeto, permite a um desenvolvedor de *software* compreender todas as etapas necessárias, desde os conceitos básicos, com referências bastante didáticas, utilizando-as para iniciar o levantamento de requisitos, desenvolvimento e testes com o usuário final. A abordagem seguida, é bastante prática e com ênfase em experiências obtidas no dia a dia de um desenvolvedor de sistemas. Como objetivo inicial, foi proposta uma entrega de um produto mínimo viável, assunto que também é abordado no trabalho. Com todo esforço aplicado, buscando base nas referências que dão as respostas necessárias para as dúvidas em relação a conceitos, foi possível criar o *DentalCloud*, entregue como produto final para o gerenciamento de laboratório de prótese dentária.

Palavras Chave: Site; ERP; desenvolvimento

ABSTRACT

The present work aims to demonstrate the process of research and development of software. The content of the project allows a developer of this software to understand all the necessary steps, from the basic concepts, with very didactic references, using them to start the requirements survey, development and tests with the end user. The implemented approach is very practical and with an emphasis on basic services in the day to day of a system developer. As an initial objective, a delivery of a minimum viable product was proposed, a subject that is also addressed in the work. With all the effort applied, looking for the references that provide the necessary answers to the doubts in relation to the concepts, it was possible to create the DentalCloud, delivered as a final product for the management of the dental laboratory.

Keywords: *website;ERP;development*

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

O objetivo deste projeto é estabelecer um modelo padrão de gerenciamento de um laboratório de prótese dentária, onde observou-se pelo autor, que este processo não vem se adequando com as tecnologias da atualidade. Por este motivo se deu a necessidade de uma ferramenta de gerenciamento de laboratório de prótese dentária, pois analisando as ferramentas similares, identifica-se que são tecnologias arcaicas, na qual acabam dificultando o processo de gerenciamento.

É uma demanda bastante interessante, na qual é possível utilizar o maior recurso de um desenvolvedor, que é seu tempo, de forma muito eficiente. Este projeto poderá ser utilizado como base por outros desenvolvedores que desejam construir uma aplicação *web*, partindo do “zero” e em um curto prazo.

Com referências didáticas bastante práticas, o projeto real é relacionado ao embasamento teórico, que é extremamente necessário para suprir as informações necessárias ao desenvolvedor. O *framework* e a linguagem de programação utilizadas permitem criar aplicações robustas capazes de atender uma empresa grande, bastando que o projeto tenha sido realizado de forma detalhada, desde os requisitos até os diagramas e fluxos criados.

A metodologia utilizada foi definida com base em metodologias ágeis, que permitem ao desenvolvedor ou à organização gerenciar melhor o tempo e entregar um produto no mínimo viável ao cliente final com uma baixa curva de aprendizado. As seguintes ferramentas foram essenciais para o desenvolvimento deste projeto:

- *Astah Community*, para criação dos diagramas principais levantados nos requisitos, sendo eles, classes, casos de uso e sequência;
- *Visual Studio Code*, para o desenvolvimento de todo o código fonte*,* que garante bastante facilidade para utilizar os comandos de um ambiente *PHP*;
- *Azure DevOps* é o sistema utilizado para gerenciar as tarefas durante o desenvolvimento, com base nos conceitos do *Scrum;*
- *GitHub* é primordial durante o desenvolvimento de um *software* complexo, por este motivo é utilizado para versionamento do código;
- *Draw IO* foi utilizado para criar os fluxogramas para explicar os conceitos que serão aplicados no sistema para o usuário final;

- *Logo Maker* é uma ferramenta que pode ser utilizada para criar um logo para uma aplicação piloto;

No primeiro capítulo da estrutura do trabalho, é apresentado o embasamento teórico necessário para poder complementar com o conhecimento de um desenvolvedor neste segmento. O segundo capítulo apresenta ao leitor, as etapas necessárias durante o desenvolvimento deste trabalho, tais como o levantamento de requisitos e a metodologia ágil utilizada. No terceiro capítulo são apresentados os principais diagramas para este *software,* que são utilizados para ter uma visão geral desse processo. No quarto capítulo, é apresentado o histórico desse processo e sua evolução durante o desenvolvimento.

Com o estudo realizado, fundido com a prática de desenvolvimento de *software*, foi possível concluir a viabilidade do projeto em questão. Este fator, permitirá evoluir com a ideia, com o objetivo de transformá-la em uma plataforma global.

2 REFERENCIAL TEÓRICO

Neste capítulo, será apresentado o conteúdo bibliográfico necessário para o desenvolvimento do projeto como um todo, embasando o projeto na teoria. Etapa primordial para o projeto, em que conteúdo pesquisado é geralmente o que supre a falta de conhecimento do autor, caso ele não tenha experenciado determinado ponto abordado no trabalho. Serão tratados desde os conceitos de *software* e arquitetura, banco de dados, padrão de desenvolvimento utilizado, *template* de *Front-End* utilizado, linguagem de programação, *framework*s e meios de interface de comunicação de dados.

2.1 *Software* e Arquitetura

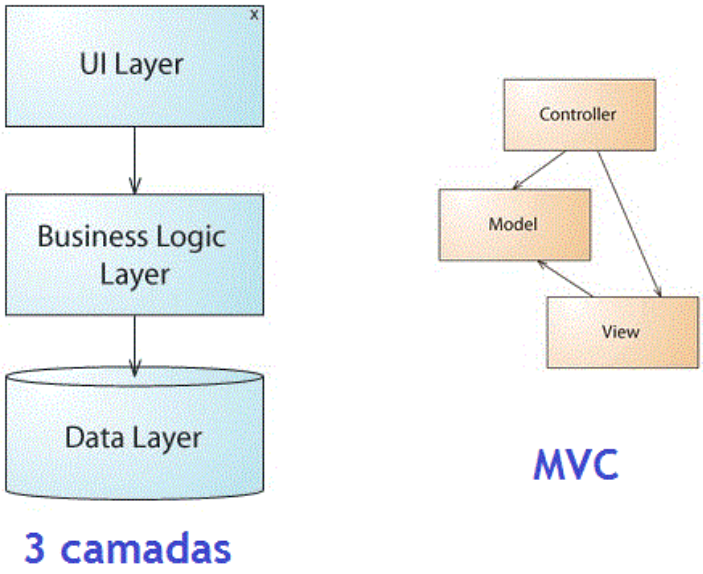
Segundo SOMMERVILLE (2011), todo *software*, independentemente de seu objetivo, necessita da engenharia de *software*. Isso se dá pelo fato de esta ser uma era em que a tecnologia só tem crescido e crescerá ainda mais. Como nesta engenharia não existe limites naturais, é necessário que um *software* tenha sua documentação e arquitetura bem executadas tanto quanto o funcionamento do sistema.

SOMMERVILLE (2011) expressa que um *software* profissional normalmente é criado por equipes ao invés de indivíduos, e é interessante deixar claro que isso ocorre normalmente. É possível que um projeto de *software* seja iniciado por somente um indivíduo, como no caso deste projeto. Porém é necessário que o desenvolvedor esteja ciente que a demanda pode aumentar drasticamente sem aviso prévio, então para isso é necessário que a documentação esteja bem estruturada, assim como sua arquitetura e todo o código fonte, pois poderá ser essencial a participação de novos integrantes no projeto.

BAPTISTELLA (2009) expressa, “É muito difícil construirmos um *software* livre de defeitos, mas podemos utilizar técnicas de engenharia de *software* que são fundamentais na organização para construção de aplicações(...)”, pois a ferramenta é manuseada por humanos que estão sempre suscetíveis a erros, o que pode impactar

em determinada regra implantada no sistema. Não é interessante focar nesse ponto, pois a tecnologia vem avançando com novas arquiteturas de sistema, tendo em comparação ao desenvolvimento em camadas com algum padrão de projeto, conforme exemplificado na figura 1.

Figura 1 - Arquitetura em camadas e *MVC*



Fonte: http://www.macoratti.net/vbn_mvc.htm

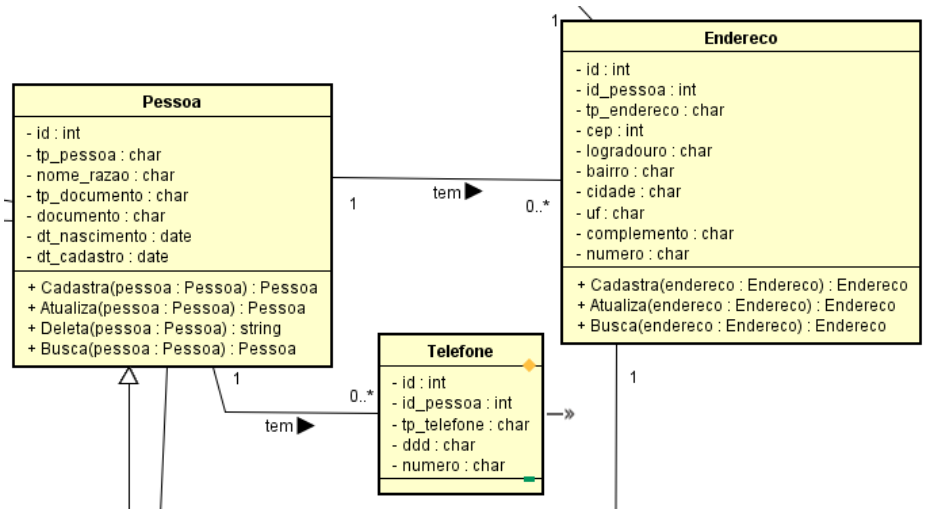
Com padrões de arquitetura para a criação de um *software*, é possível atingir um resultado de forma mais rápida, garantindo a fácil manutenção e permitindo o mais fácil entendimento de outros engenheiros de *software* SOMMERVILLE (2011). Como este projeto se trata de uma aplicação *web*, foi utilizada a arquitetura *MVC* (*Model*, *View*, *Controller*), que, mesmo que tenha sido criada na década de 1970, em pleno século XXI permite construir aplicações complexas mantendo-a estruturada.

2.2 Banco De Dados

“Ao conjunto de arquivos integrados que atendem a um conjunto de sistemas, dá-se o nome de banco de dados (BD).” HEUSER (1998). Isso significa que um banco de dados é onde disponibiliza-se cada informação de um processo que está sendo automatizado. Por exemplo, uma clínica odontológica pode solicitar realizações de

serviços de um laboratório. Neste cenário, é muito importante que o levantamento de requisitos tenha sido realizado da forma mais detalhada possível, pois são identificadas três entidades explícitas. São elas, clínica, laboratório e serviço. Porém, implicitamente existe uma matéria-prima que é consumida, um produto que é gerado e entregue à clínica, entre outros. Para isso utilizam-se tabelas de bancos de dados, que permitem armazenar e relacionar os dados de determinado processo, para utilizá-los no sistema em questão. Na figura 2, é possível compreender, de forma básica, o funcionamento de um banco de dados para armazenar dados básicos atrelados a uma pessoa, como endereço e telefone.

Figura 2 -Tabelas a serem criadas no banco de dados.



Fonte: Elaborado pelo autor.

Para um desenvolvedor de *software* trabalhar com um banco de dados, é necessário no mínimo um conhecimento básico da linguagem *SQL* (*Structured query language)*, que é utilizada para realizar ações de definições de entidades relacionadas com seus determinados atributos e além disso, manipular registros em massa desses dados.

Se tratando de banco de dados relacional, é necessário ter no mínimo os conceitos principais para trabalhar com um desenvolvimento de *software* completo. Conforme HEUSER (1998) explica em seu livro**,** “(...) a técnica de modelagem de dados mais difundida e utilizada é a abordagem entidade-relacionamento (ER)”, criada por Peter Chen em 1976. Essa abordagem se aproxima muito da vida real, em que

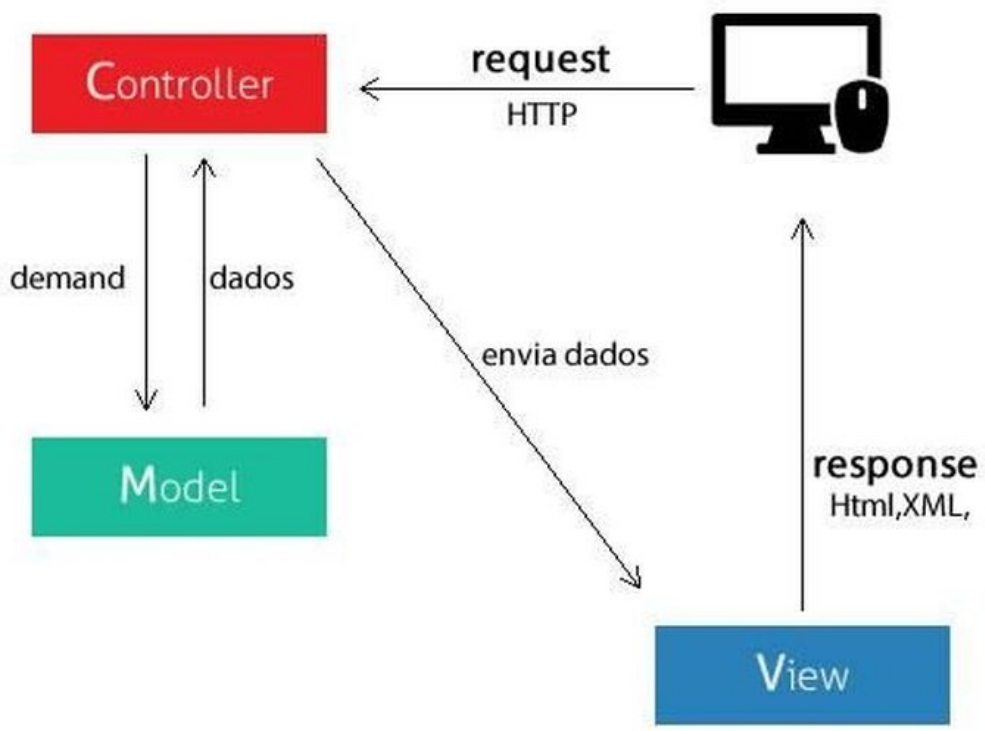
HEUSER (1998) define entidade como um conjunto de objetos que participam do processo do negócio no qual o sistema será implantado, ou seja, que serão primordiais para armazenar os dados que posteriormente serão utilizados para os devidos fins.

Entre as entidades de um processo, existe o relacionamento, ou seja, tudo que faz com que determinado objeto interaja com outro, como por exemplo as entidades Pessoa e Telefone. A entidade Pessoa, é um exemplo entidade que participará do processo e Telefone é como se fosse uma agenda telefônica de pessoas. Com essas duas entidades nascem de um relacionamento, que permite identificar quais são os telefones que determinada pessoa tem. Entidades contêm atributos que dão características a cada registro de objeto armazenado na tabela de pessoas. Por exemplo, esta entidade geralmente pode ter um ID (atributo identificador) e nome, podendo ser abstraída para duas outras entidades, como por exemplo Pessoa Física e Pessoa Jurídica, em que ambas são pessoas, porém cada uma contém dados específicos para seu segmento.

2.3 *MVC*

GAMMA et. al. (1994) específica como a arquitetura *MVC* é composta de maneira genérica. Começando pelo modelo (*model*), que pode-se também chamar de entidade, é o objeto que conterá as características, e ações ou métodos dessa classe, do processo em que o *software* será desenvolvido. Clínica, laboratório e paciente, são exemplos de modelos a serem utilizados para este *software*. A visão (*view*) é a interface para que o usuário utilize o sistema, acessando os dados das entidades envolvidas, porém nessa arquitetura não é realizada de forma direta. Para este fim, é utilizado o controlador (*controller*), que contém toda a regra de negócio e manipula os *models* que são apresentados ao usuário do sistema. Com essa separação, o *software* torna-se flexível e permite a reutilização para criar novas funcionalidades (GAMMA et. al. 1994). Portanto, se tratando de uma aplicação *web*, pode ser utilizado o padrão *MVC* para a construção de uma aplicação robusta, conforme apresentado na figura 3.

Figura 3 - Padrão *MVC*.



Fonte: https://www.portalgsti.com.br/2017/08/padrao-mvc-arquitetura-model-view-controller.html.

Uma de muitas vantagens de se utilizar a estrutura *MVC*, pelo menos no *Back-end*, é o fato de criar somente uma estrutura de manipulação dos dados, com seus atributos e métodos, mas que possa ser transmitido e alterado ao usuário por inúmeras interfaces, sendo elas *web*, aplicativos *mobile,* integração, entre outros.

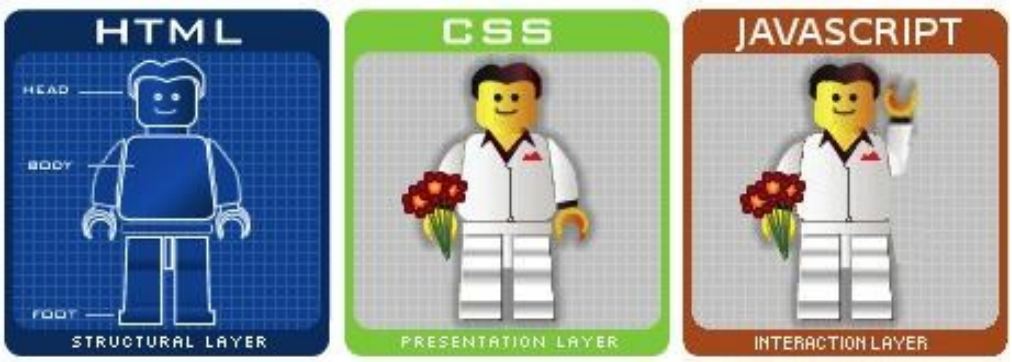
2.4 *Front-End*

"*HTML* é a sigla em inglês para *HyperText Markup Language*, que, em português, significa linguagem para marcação de hipertexto". SILVA (2008). Para ter-se o mínimo de conhecimento de hipertexto, SILVA (2008) mostra uma definição simples, que se aplica ao objetivo desse projeto. Portanto, se trata do conteúdo de um documento disponibilizado na web, que, por meio de *links* complementados com outras informações, permite navegar entre diversos documentos. Um *link* em um documento *HTML* redireciona a navegação do usuário para determinada *URL*

(*Uniform Resource Locator*, ou localizador uniforme de recursos), que poderá conter um outro documento que irá apresentar uma outra página.

Conforme SILVA (2008) mostra em seu livro, o *HTML* é dividido em três seções, sendo elas, o *DOCTYPE*, *head* e o *body*. O *DOCTYPE* é utilizado para determinar o tipo do presente documento, que será interpretado por um navegador *web*. No *head* (cabeçalho), ficam presentes informações essenciais como *title* que se trata do título daquela página *HTML* ao ser acessada pelo usuário. Finalmente o *body* (corpo), é onde o conteúdo da página será escrito para ser apresentado posteriormente.

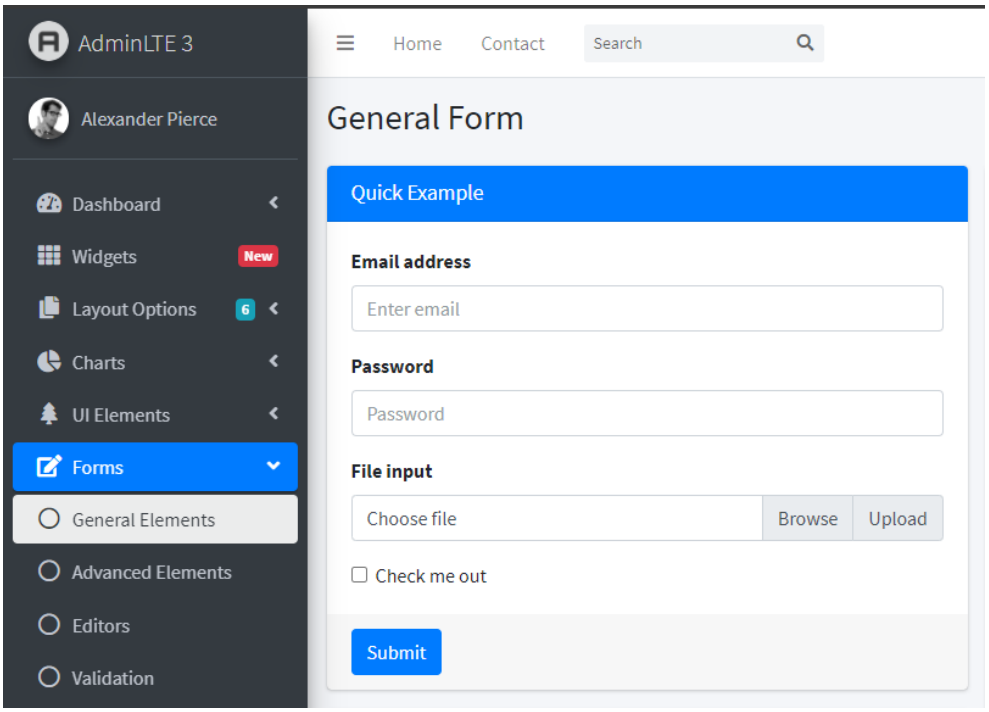
Figura 4 - Exemplificação *HTML, CSS* e *Javascript*.



Fonte: https://www.karasiak.net/html-and-css-lego-icon-have-a-new-family-member-javascript-animated-icon/.

Com *HTML, CSS* e *Javascript* (figura 4), é possível criar interfaces de apresentações de imagens, textos, formulários com campos para edição de dados, tabelas, entre outros. Neste projeto, foi definido como prioridade criar interfaces funcionais, alocando o tempo destinado ao funcionamento eficaz do sistema. Por este motivo, o projeto faz o uso de *template open-source* AdminLTE (figura 5). Este *template* contém modelos de formulários, componentes, botões, menus e tela de *login*.

Figura 5 – Exemplo de um formulário do *template* AdminLTE.



Fonte: Elaborado pelo próprio autor.

Esses recursos foram desenvolvidos em *HTML*, *CSS* com *framework Bootstrap* e linguagem de programação *Javascript*. "O *Bootstrap* é uma ferramenta gratuita para desenvolvimento *HTML*, *CSS* e *JS*", BOOTSTRAP (2020). "*JavaScript* é uma linguagem de programação que permite implementar funcionalidades mais complexas em páginas *web*" JAVASCRIPT (2020). Com isso, se torna mais prático desenvolver os formulários e outras interfaces com o usuário, requerendo somente utilizar os recursos disponibilizados pelo *template*, de acordo com cada projeto.

2.5 *API Web Service*

> O *HTTP* – Protocolo de transferência de Hipertexto (*Hypertext Transfer Protocol)* é um protocolo da camada de aplicação da *internet*, segundo (KUROSE, 2006) o *HTTP* está dividido em duas partes, o programa cliente e o programa servidor, os dois programas são executados em máquinas diferentes, conversam um com o outro por meio de mensagens *HTTP*. O *HTTP* define a estrutura destas mensagens e o modo como são trocadas entre o cliente e o servidor. (BEOCK et al)

No caso de uma *API* (*Application programming interface*) *web service*, geralmente é trabalhado a parte da aplicação do lado do servidor no *HTTP*. A todo

momento, requisições são disparadas paras rotas de uma aplicação *MVC* em *PHP* por meio da navegação de documentos ou páginas *web*, que por sua vez direcionam essa requisição para o controlador realizar alguma operação na camada de armazenamento. A Figura 14 representa de forma básica como os dados são enviados e recebidos de um servidor *web*, utilizado para hospedar, reter e disponibilizar dados ou documentos de uma aplicação *web* de um negócio.

Figura 6 - Contexto geral *HTTP*.

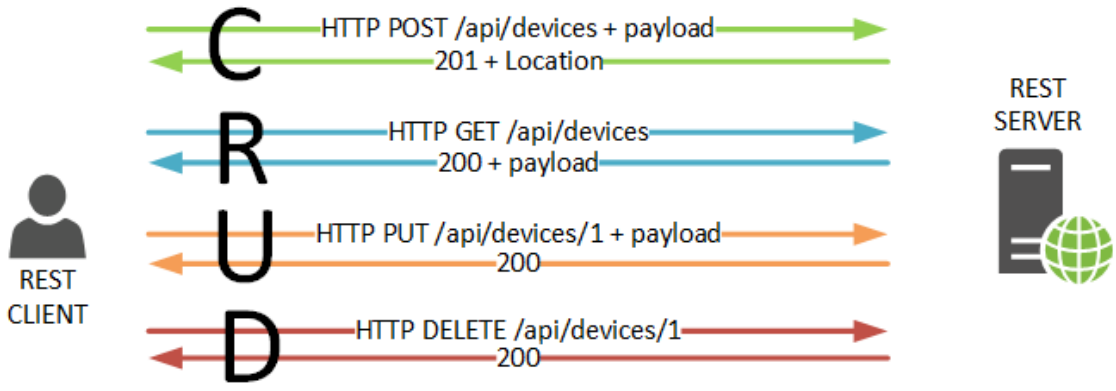
Fonte: KUROSE, 2006.

Para comunicação de dados e processamento de requisições pelo usuário do sistema, são utilizados w*eb services* que realizam a comunicação *HTTP* para transferir informações de determinando *software* da empresa X, para determinado *software* da empresa Y, portanto acaba se tornando uma linguagem de comunicação universal. “Existe alguma forma padrão que todos os diferentes programas podem usar para se comunicarem uns com os outros? (Assim como cada país tem suas próprias línguas nativas, mas eles podem usar o inglês para se comunicar com qualquer pessoa fora de seu país)” VIKMANI (2019).

Esta comunicação é realizada corretamente, graças a *URL* que, segundo TANEMBAUM (2003) explica, é uma forma de identificar unicamente aquela página ou resposta de *API*. Portanto, é bom que se conheça a estrutura da URL, para se trabalhar com aplicações *web,* pois é com ela que toda a navegação contendo requisições e diversos tipos de respostas é orquestrada. A *URL* (*Uniform Resource Locators*) é composta pelo protocolo dessa comunicação, no caso *HTTP*. Após isso é requisitado que o usuário, por meio de um navegador *web* ou um gerenciador de *API*, como o *Postman*, informe o *DNS* (*Domain Name Server*) ou endereço do IP público do servidor. Com isso, segundo a sequência de etapas da comunicação *HTTP* descritas por TANEMBAUM (2003), basta informar qual o caminho para o arquivo ou recurso que se deseja requisitar via *web service.*

Na figura 15, é realizada uma ligação do *CRUD* do sistema, com o funcionamento de *API RESTful*, na qual as *URL* são invocadas com seus respectivos métodos em busca de uma resposta, ou com a intenção de realizar determinada ação dentro do ambiente de uma aplicação.

Figura 7 - *CRUD* e *API RESTful*.



Fonte: https://www.edureka.co/blog/what-is-rest-api/.

- *GET*: utilizado para solicitar alguma informação que normalmente se encontra centralizada no servidor *web* ou em um servidor de banco de dados.

- *POST*: utilizado para enviar algum documento, como por exemplo, realizar um *upload* de um arquivo escaneado para futuras consultas. Relacionado ao banco de dados, é utilizado para criar novos registros relacionados a uma entidade ou um conjunto de entidades.
- *PUT*: geralmente, se utiliza este método ao ser necessária alguma atualização de dados que já se encontram no servidor, porém o usuário enviou dados que precisam ser alterados, e assim todos podem ter conhecimento.
- *DELETE*: como o próprio nome diz, é utilizado para apagar determinado recurso que consta no servidor, como também pode apagar o cadastro de algum usuário no banco de dados do sistema.

O método *GET*, geralmente é utilizado para realizar as navegações de um *site*, partindo da página inicial. Pois ao digitar determinada *URL* em um navegador *web*, aquele servidor alcançado pelo domínio ou *IP*, te redireciona para um documento ou página *HTML*. Assim o navegador processa o seu conteúdo, permitindo apresentá-los ao usuário de forma fácil.

Por exemplo, ao criar um usuário no sistema para um novo funcionário da empresa, ao consumir uma *API* que seguiu a estrutura e boas práticas, obtém-se como resultado uma *URL* http://dns.aplicacao.com/api/user, no qual por meio do corpo da requisição é composto por uma mensagem em *JSON* que segundo RAJEEVAN (2018) é um formato de mensagem utilizado na comunicação de fácil compreensão.

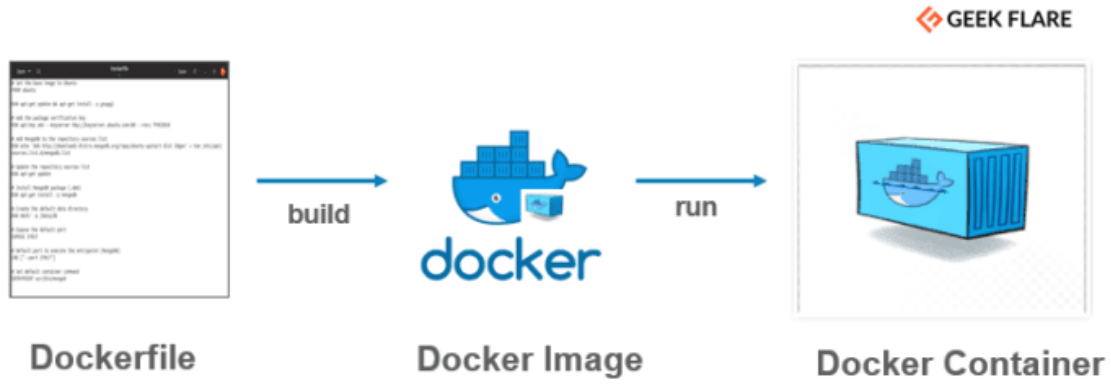
2.6 *Containers*

Ambientes de virtualização, segundo MERKEL D. começaram a se tornar muito utilizados por permitir um melhor gerenciamento de recursos, eficiência na instalação de dependências de aplicações. Enquanto *VM* (*Virtual machines*) funcionam na camada do *hardware*, em meados de 2013 começaram a surgir a utilização de *Containers* para o *deploy* de aplicações modernas. Segundo MERKEL D., *Docker* começa a dar fim no “inferno de dependências” ao disponibilizar aplicações que por requisitos, necessitam de ferramentas e outras aplicações. O uso de *Containers*

isolados uns dos outros, funcionam na camada de *software* de determinada distribuição, seja ela *Windows*, *Linux* ou *OS X*. Conforme explica MERKEL D., isso faz com que um *container* criado para aplicação X, não tenha conhecimento do recurso alocado ou outras informações do *container* para a aplicação Y.

"*Docker* é uma plataforma para desenvolvedores e administradores de sistemas criarem, executarem e compartilharem aplicativos com contêineres", DOCKER (2020). *Docker* é executado como uma aplicação aberta no SO (Sistema Operacional), portanto torna-se leve por esse quesito. Segundo a documentação do *Docker*, o processo dessa aplicação, utiliza imagens para compartilhar uma aplicação seja na própria estação do desenvolvedor ou em algum servidor. "Uma imagem inclui tudo o que é necessário para executar um aplicativo - o código ou binário, tempos de execução, dependências e quaisquer outros objetos do sistema de arquivos necessários". DOCKER (2020). Na figura 16 é possível compreender as etapas necessárias para se obter um container, iniciando pela criação do "*dockerfile",* que trata-se das especificações do que a imagem do container deve ter.

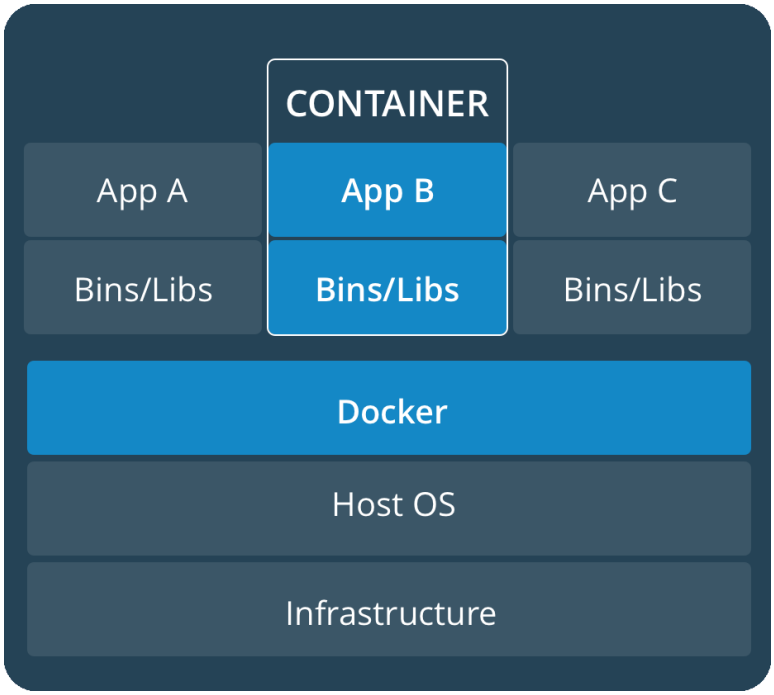
Figura 8 - Arquitetura *Docker*.



Fonte: https://geekflare.com/dockerfile-tutorial/.

Conforme mostra a Figura 17, vê-se como é a arquitetura de funcionamento do *Docker*, que independe do SO *host* utilizado.

Figura 9 - Arquitetura dos *Containers*.



Fonte: Docker.

Conforme o exemplo dado por MERKEL, uma aplicação em *Python* pode utilizar três *Containers* facilmente. O mesmo acontece para uma aplicação *web* em *PHP*, com banco de dados *PostgreSQL*, em que cada container um tem sua finalidade. Geralmente ao ser necessário um banco de dados, uma aplicação web e um SGBD (Sistema Gerenciador de Banco de Dados), utilizam-se três *Containers* facilmente, pois na prática não é viável ter o banco de dados juntamente com aplicações que sofrem atualizações. Portanto, criando um *container* com o *PostgreSQL*, um com as dependências para hospedar uma aplicação *PHP* e um outro com o *pgAdmin*, é possível obter um ótimo resultado, pois cada alteração que um container sofrer não afetará o funcionamento do outro.

3 DESENVOLVIMENTO DE PROJETO

Para o desenvolvimento de um *software*, existem cinco fases de alto nível que um bom desenvolvedor deve seguir. Que conforme demonstra MOREIRA (2000) são elas, análise de requisitos, análise, projeto, desenvolvimento e testes.

Neste projeto, foi realizado primeiramente a análise de requisitos, pois é nesta etapa que o usuário o sistema deve ser na visão do usuário. Com isso, o desenvolvedor interpreta os requisitos do usuário, combinando com sua habilidade, resultando nos requisitos do sistema. Os requisitos de sistema tratam as especificações do *software* de forma mais técnica e objetiva, facilitando a próxima etapa de análise a ser realizada. Na etapa de análise, foram criadas as entidades e campos principais a serem utilizados no sistema. Utilizando diagramas, foi possível absorver a complexidade das diversas entidades e os relacionamentos presente neste processo. Trabalhando na etapa do projeto em si, é possível criar o funcionamento genérico do sistema, em que é possível apresentar ao usuário e receber os primeiros *feedbacks* antes de iniciar a programação. A etapa mais gratificante é a de programação, principalmente se as outras etapas foram detalhadas da melhor forma, fazendo com que se torne prático para desenvolver o sistema. Os requisitos do sistema e os diagramas são muito importantes nessa etapa, pois serve como um manual de desenvolvimento, bastando ir seguindo todo o conceito levantado.

Para que todo esse trabalho seja válido, é necessário que sejam realizados os testes do sistema. O diagrama de casos de uso é muito importante para realizar essa validação, sendo bem mais eficiente se realizada com usuário presente dentro do cenário em que o sistema será utilizado. Caso o cliente não utilize um sistema no processo a ser automatizado, os testes são realizados passando as informações do modelo legado de gerenciamento, para dentro do sistema. Lembrando que na maioria das vezes, alguns ajustes surgirão, pois não foram previstos pelo usuário na análise de requisitos.

Portanto, para uma melhor organização, esse novo ajuste deverá ser analisado como um requisito de sistema, seja ele, uma alteração de um recurso existente ou até mesmo um novo recurso que será necessário implementar. Isso acaba formando um

ciclo de desenvolvimento, pois após ser analisado, deverá estar presente no projeto de sistema, se relacionando com as outras entidades do relacionamento.

3.1 Levantamento De Requisitos

De início, foi disponibilizado um dia intenso de entrevista com o usuário, a fim de levantar as especificações do sistema a ser desenvolvido. Nesta entrevista em um sábado, no qual não havia trabalho a ser realizado pelo usuário, iniciou-se com um café e um pequeno bate-papo para entender a visão geral do negócio. Nesta conversa discutiu-se como o funciona o fluxo do processo em geral, pessoas envolvidas no negócio, que provavelmente utilizarão o sistema e objetos que serão armazenados e gerados no sistema.

Foram apresentadas pelo usuário, algumas comandas de trabalho utilizadas no laboratório. Essas comandas contêm a maioria dos dados necessários para que o trabalho possa ser organizado e executado dentro do laboratório. Dados, como clínica, dentista responsável e paciente, constam nessa comanda para fins de identificação. Após isso é especificado o que deve ser realizado no trabalho, ou seja, suas especificações para chegar em um produto final após análise. Outro fator de complicação para o laboratório é em relação à comunicação com as clínicas, ao surgir necessidade se solicitar dados complementares para dar andamento no serviço em questão.

Esse processo se torna complicado, pois é todo efetuado de forma manual, utilizando papel. Portanto, foi solicitado que houvesse uma forma de ter todos esses dados cadastrados em sistema, e assim, todos os complementos seriam acrescentados por meio de uma interface em que o próprio solicitante possa utilizar. Conforme os serviços são realizados, é necessário dar movimentar estoque e faturamento do laboratório, portanto foi solicitado que fosse atrelado a realização de etapas no serviço, em que essas etapas precisam gerar produtos e consumir matérias-primas.

As etapas dos serviços serão controladas pelos próprios técnicos do laboratório, que utilizarão o computador ou o dispositivo móvel pessoal se preferir. O

próprio técnico registrará o consumo e produção em cada etapa realizada, pois o controle deverá ser feito por técnico, pois cada pessoa gasta mais material que a outra e também pode produzir um produto em mais tempo do que outro técnico. Esse controle poderá ser necessário futuramente, portanto foi solicitado que o contexto do sistema estivesse estrutura para novas demandas.

Como requisito final, é necessário que a ferramenta apresente relatórios do que foi consumido e o que foi produzido. Levando em consideração, que futuramente poderá ser liberado um relatório para o cliente, a fim de contribuir na melhoria da comunicação. Além de relatório para os clientes, o usuário informou que é interessante que os técnicos possam analisar seu desempenho e controlar seus trabalhos executados ou os que ainda estão pendentes.

3.1.1 Requisitos Funcionais

Foram definidos como requisitos funcionais do sistema, manter clínicas odontológicas, laboratórios de prótese, materiais para consumo, produtos que serão fabricados e pessoas, que podem ser, clientes, entregadores, técnicos dentistas ou protéticos, fornecedores e pacientes. Conforme o usuário solicita, uma pessoa pode conter mais de um endereço em seu cadastro, porém a clínica e o laboratório contêm somente um. Uma pessoa pode conter diversos contatos de e-mail e telefone, portanto deverá permitir o cadastro desses dados.

Tanto para os materiais e para os produtos, deverão ser cadastradas as unidades e seus respectivos código de barras. Por exemplo, ao analisar um detalhe apresentado pelo usuário, no laboratório é utilizado um material chamado “gesso pedra”, porém são comprados pacotes de 1 kg ou de 2 kg. Para conseguir gerenciar o estoque deverá ser cadastrada uma unidade de 1 kg e outra de 2 kg e associar os respectivos códigos de barras de cada embalagem. Tendo essa dinâmica no cadastro, deverão conter as listas de preço dos produtos, pois é essencial que possam ser cobrados diversos preços com acréscimos e descontos dependendo de cada cliente do laboratório. Poderão ser criadas listas de preço, contendo um desconto ou acréscimo geral para todos os itens, ou o valor de cada produto ou material poderá ser personalizado.

Com os dados cadastrados, vários serviços poderão ser criados pelo laboratório ou pelas clínicas, sendo que será necessário ter o acompanhamento desse serviço durante as diferentes etapas dentro do laboratório. Um serviço sempre se inicia e finaliza em um laboratório, mesmo havendo etapas que serão realizadas por terceiros. Portanto, é bom que seja armazenada a informação sobre quem realizou determinada etapa, em qual laboratório foi realizada, que horas iniciou, em que horário terminou, a que horas finalizou o serviço todo e o momento do despacho para a entrega. Como são serviços que custam caro para ambas as partes, clínica e laboratório, deverá conter o cadastro da pessoa entregadora, seja ela uma empresa transportadora ou uma pessoa física que realiza entregas, permitindo assim rastrear responsáveis de possíveis entregas extraviadas.

Esses dados, ao serem movimentados pelos usuários do sistema, darão origem a extratos que poderão ser gerados a qualquer momento de acordo com o tipo de relatório, período e filtros adicionais, permitindo por exemplo, gerar um extrato de serviços entregues a determinada clínica, em que o filtro adicional se encaixa a extrair um relatório para a clínica X e não para a Y. Neste tipo de relatório, deverá constar todo o gasto com cada serviço realizado para aquela clínica, assim como os produtos finais entregues.

Pelo módulo que a clínica utiliza, ela conseguirá extrair um relatório de si mesma para agilizar no processo de pagamento e economizar o tempo do laboratório, recurso que é utilizado ao enviar os extratos para as diferentes clínicas clientes. Para controle de estoque e controle do tempo de trabalho gasto pelos técnicos protéticos, o sistema deverá permitir gerar relatórios de tempo de trabalho por técnico nas etapas. Isso futuramente poderá permitir escolher o técnico menos atarefado para realizar determinado serviço, garantindo maior eficiência na entrega dos trabalhos no laboratório.

3.1.2 Requisitos Não Funcionais

As telas do sistema deverão ser de fácil compreensão, sendo bem objetivas e bem ligadas às suas dependências. Por exemplo, ao criar uma lista de preço, é

interessante haver um *link* para redirecionar o usuário para a tela de cadastro de um material ou produto.

Os técnicos utilizarão dispositivos móveis como *tablets* e celulares, portanto, o *design* da aplicação deve ser responsivo. Além de responsividade, as cores no sistema deverão representar as ações que serão realizadas. Por exemplo, para cancelar um serviço, geralmente se utiliza botões de cores vermelhas. Para iniciar uma nova etapa do serviço, utilizar um ícone no botão que represente um *play,* pois dessa forma, os recursos do sistema são memorizados, agilizando na utilização diária.

Os relatórios deverão conter as informações de forma bem objetiva para não confundir o cliente ou até mesmo o gestor do laboratório. Ao analisar um relatório é importante realizar a leitura e ver se deixa alguma dúvida que seria necessário questionar a alguém com mais experiência. Qualquer pessoa deve entender seu conteúdo, seja ele um extrato mensal de um cliente ou um relatório de serviços e etapas realizadas.

Todo esse contexto pertencerá a uma organização, que pode gerenciar mais de um laboratório. Portanto, o sistema deve ser pensado em atender desde o início de um laboratório até seu estágio mais avançado, visto que já se tornou uma grande empresa com várias filiais.

3.2 *Scrum*

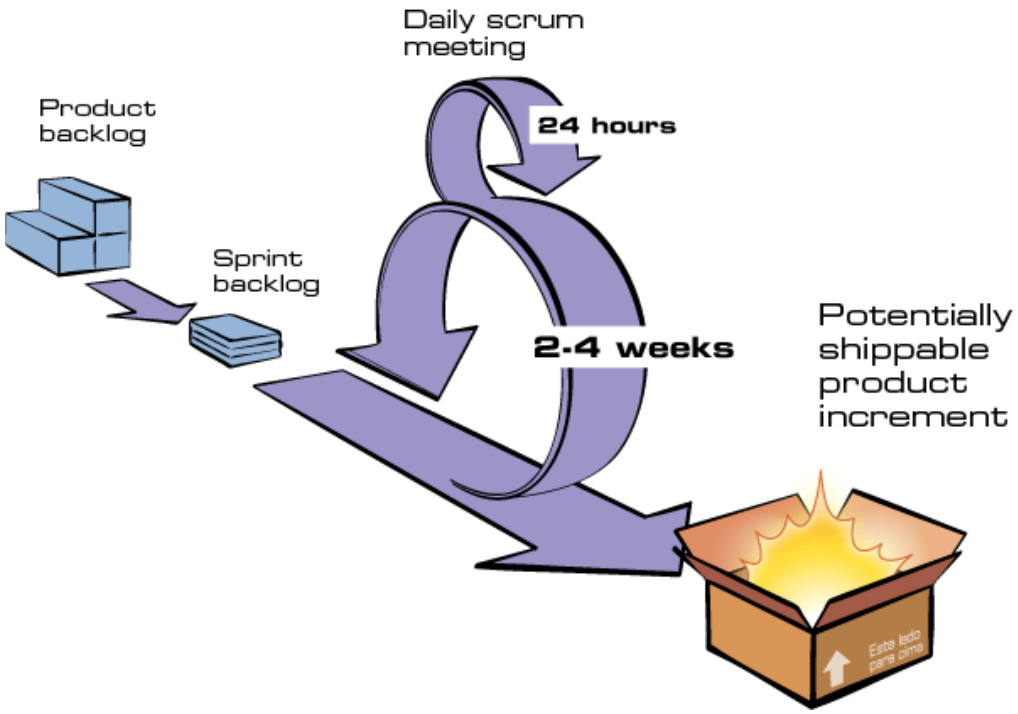
No início do projeto, foi definido que seria usado a metodologia ágil *Scrum,* voltado mais para o planejamento, mas contemplando os aspectos básicos de gestão. Este processo, auxilia na organização do participante do projeto e por conta de ser um projeto solo, é sempre interessante ter a possibilidade de contextualizar novos integrantes no que já foi desenvolvido, o que está sendo realizado e o que deverá ser entregue em breve. Foi utilizada a ferramenta *Azure DevOps*, com a ferramenta de *Boards*, que permite seguir os conceitos mais utilizados do *Scrum,* garantindo um bom planejamento do projeto e permitindo a colaboração com futuros integrantes do time de desenvolvimento.

O projeto, foi divido em 2 grandes ciclos, sendo o ciclo um de planejamento e desenvolvimento básico e o ciclo dois de desenvolvimento avançado com base nos últimos testes realizados. Este segundo ciclo, acaba se tornando um *loop* enquanto houver questões a serem desenvolvidas ou corrigidas no sistema. Para o primeiro ciclo, foi estabelecido um início em 01/08/2020 até 01/12/2020, totalizando em média, 120 dias. Com este período de entrega, foi possível dividir o desenvolvimento do projeto em 8 *sprints* de 15 dias cada. Neste ciclo, foi contemplado o conceito de *MVP*, que segundo SANTOS *et al* (2019), "O *MVP* é a entrega mínima do produto, no caso o aplicativo, de uma forma mais enxuta e ágil, porém com qualidade, levando-se em conta os requisitos propostos pelo cliente.".

Portanto, no caso do projeto *DentalCloud*, foi desenvolvida uma ferramenta *web*, com as funcionalidades básicas, para validar o resultado e perspectiva do projeto em relação aos requisitos levantados. Levando em consideração ter uma equipe de um integrante inicialmente, é de grande importância contextualizar esses conceitos de *MVP* juntamente com o *Scrum,* pois desde o planejamento até a entrega do mínimo produto viável, não foram necessárias alterações na metodologia de trabalho, o que garantiu maior eficiência.

Na figura 18, é possível compreender essa metodologia com uma visão geral, englobando os conceitos principais: *backlog* do produto, *backlog* da *sprint*, reunião diária da *sprint* e potencial entrega de recurso.

Figura 10 - Visão macro do *Scrum*.

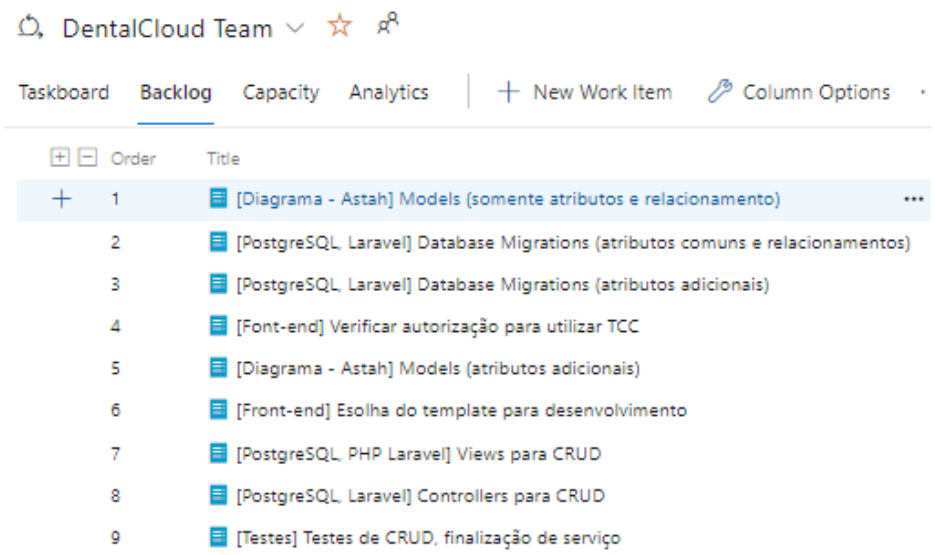


Fonte: https://www.desenvolvimentoagil.com.br/scrum/.

Conforme DESSOLDI (2019) apresenta, o *product owner* tem poder de definição de quais serão os itens no *backlog* do produto, atuando com papel de liderança sobre o mesmo. *Product backlog* é todo recurso necessário que o projeto a ser desenvolvido precisa ter, ou seja, suas funcionalidades. Após levantar os requisitos do usuário e transformá-los em requisitos de sistema, foi dado início na análise para determinar os itens do *backlog* do produto. Esses itens, foram divididos em X grandes grupos: monografia, diagramas,*back-end*, *front-end,* testes unitários, testes integrados e pequenas correções.

Desenvolvedores que vivenciam o dia a dia de projetos, iniciam com a distribuição dos itens do *backlog* do produto, em *backlog* de *sprints*. Um grande fator que auxilia em qualquer desenvolvimento ágil, é liberar uma funcionalidade e receber o *feedback* do usuário rapidamente, seja ele positivo ou negativo, mas o importante é que seja com maior quantidade de detalhes possível. Com duas *sprints* de quinze dias de duração cada uma, começaram a ser desenvolvidos os conceitos principais no sistema, como por exemplo, desde a criação de um modelo de entidade e relacionamento, passando pelo diagrama até chegar nas classes que fazem parte do processo e todo o *CRUD* da aplicação (figura 19).

Figura 11 - *Backlog* da *Sprint* 1.



Fonte: Elaborado pelo autor.

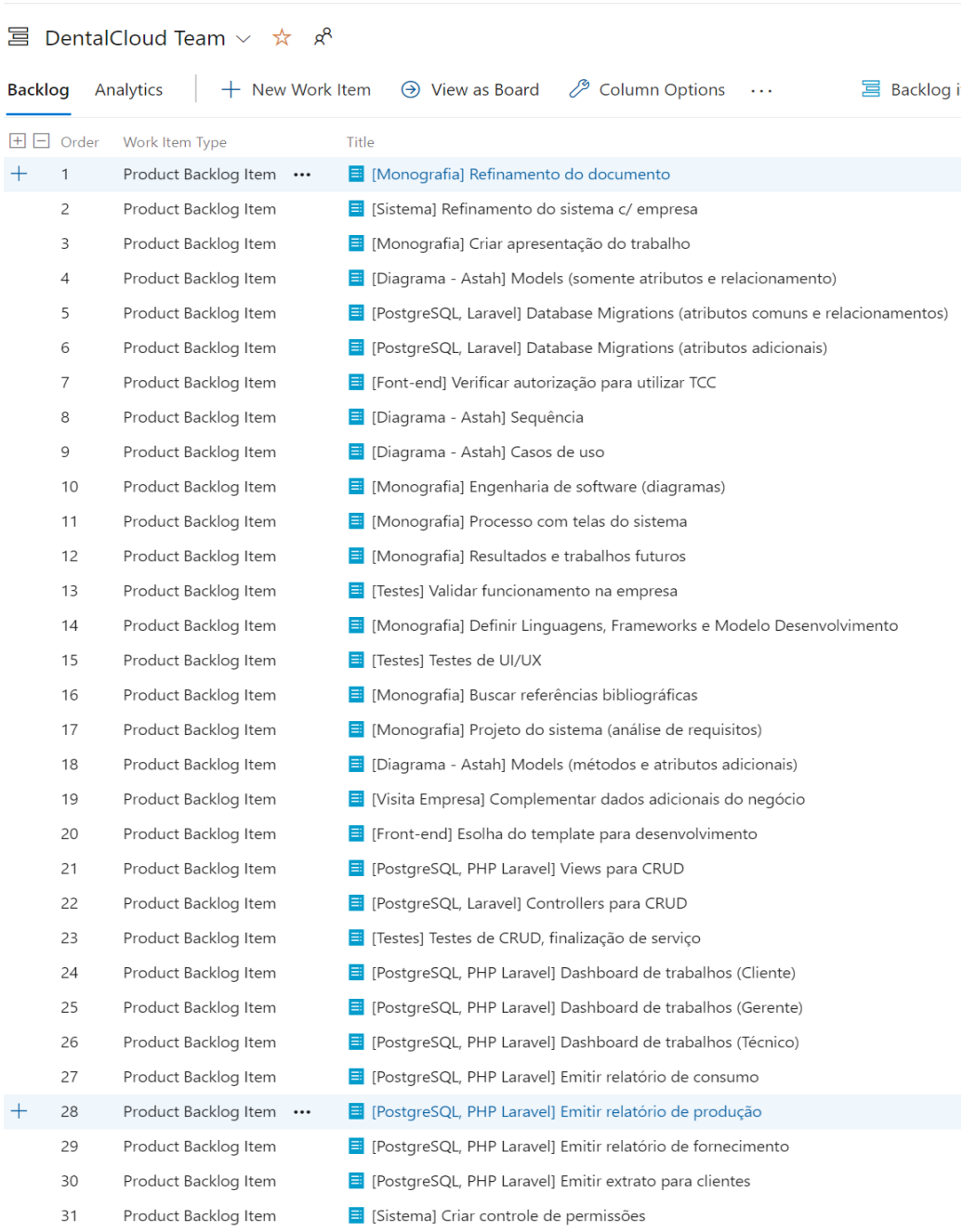
Nem sempre o *backlog* da *sprint* será cumprido, então se realiza a chamada *sprint review meeting,* na qual se discute quais objetivos foram concluídos e quais não foram. Após essa reunião realizada no fim de cada *sprint*, se faz necessário realizar a *sprint planning meeting*, para que seja criado o *backlog* da próxima *sprint* seja alinhado, podendo ser necessário incluir algumas tarefas da *sprint* anterior, caso ainda não tenham sido finalizadas. Após esse primeiro passo, bastante intenso, foi possível ter uma visão geral do sistema e do processo, facilitando o entendimento do usuário e garantindo um *feedback* mais preciso.

Um grande esforço é necessário para criar uma documentação detalhada e por este motivo, foram disponibilizadas quatro *sprints* para que todo o trabalho a ser aplicado na prática, tivesse um embasamento teórico objetivo e direto. Nessas *sprints*, foi necessário quebrar em pequenas tarefas, com o objetivo de um dia de trabalho ser focado em determinada parte do projeto. Por exemplo, o objetivo da *sprint* era concluir o tópico um da monografia, porém era tomado como objetivo diário, concluir somente um assunto abordado desse tópico, permitindo que o trabalho não parecesse tão extenso e assim garantindo maior eficiência.

Restando três *sprints* para a conclusão do projeto, o *backlog* mostrado na figura 20, planejado no início do projeto foi concluído, restando essa fase final para testes,

refinamento do sistema e documentação. Realizando entregas, o *MVP* com desenvolvimento durante as *sprints*, identifica-se que sempre há bastante trabalho pela frente até que o *software* seja definitivamente implantado na empresa, porém também é possível concluir que o caminho seguido e o *software* são no mínimo efetivos para o processo na empresa.

Figura 12 - *Product Backlog* criado no *Boards* do *Azure DevOps*.



Order	Work Item Type	Title
1	Product Backlog Item	[Monografia] Refinamento do documento
2	Product Backlog Item	[Sistema] Refinamento do sistema c/ empresa
3	Product Backlog Item	[Monografia] Criar apresentação do trabalho
4	Product Backlog Item	[Diagrama - Astah] Models (somente atributos e relacionamento)
5	Product Backlog Item	[PostgreSQL, Laravel] Database Migrations (atributos comuns e relacionamentos)
6	Product Backlog Item	[PostgreSQL, Laravel] Database Migrations (atributos adicionais)
7	Product Backlog Item	[Font-end] Verificar autorização para utilizar TCC
8	Product Backlog Item	[Diagrama - Astah] Sequência
9	Product Backlog Item	[Diagrama - Astah] Casos de uso
10	Product Backlog Item	[Monografia] Engenharia de software (diagramas)
11	Product Backlog Item	[Monografia] Processo com telas do sistema
12	Product Backlog Item	[Monografia] Resultados e trabalhos futuros
13	Product Backlog Item	[Testes] Validar funcionamento na empresa
14	Product Backlog Item	[Monografia] Definir Linguagens, Frameworks e Modelo Desenvolvimento
15	Product Backlog Item	[Testes] Testes de UI/UX
16	Product Backlog Item	[Monografia] Buscar referências bibliográficas
17	Product Backlog Item	[Monografia] Projeto do sistema (análise de requisitos)
18	Product Backlog Item	[Diagrama - Astah] Models (métodos e atributos adicionais)
19	Product Backlog Item	[Visita Empresa] Complementar dados adicionais do negócio
20	Product Backlog Item	[Front-end] Esolha do template para desenvolvimento
21	Product Backlog Item	[PostgreSQL, PHP Laravel] Views para CRUD
22	Product Backlog Item	[PostgreSQL, Laravel] Controllers para CRUD
23	Product Backlog Item	[Testes] Testes de CRUD, finalização de serviço
24	Product Backlog Item	[PostgreSQL, PHP Laravel] Dashboard de trabalhos (Cliente)
25	Product Backlog Item	[PostgreSQL, PHP Laravel] Dashboard de trabalhos (Gerente)
26	Product Backlog Item	[PostgreSQL, PHP Laravel] Dashboard de trabalhos (Técnico)
27	Product Backlog Item	[PostgreSQL, PHP Laravel] Emitir relatório de consumo
28	Product Backlog Item	[PostgreSQL, PHP Laravel] Emitir relatório de produção
29	Product Backlog Item	[PostgreSQL, PHP Laravel] Emitir relatório de fornecimento
30	Product Backlog Item	[PostgreSQL, PHP Laravel] Emitir extrato para clientes
31	Product Backlog Item	[Sistema] Criar controle de permissões

Fonte: Elaborado pelo autor.

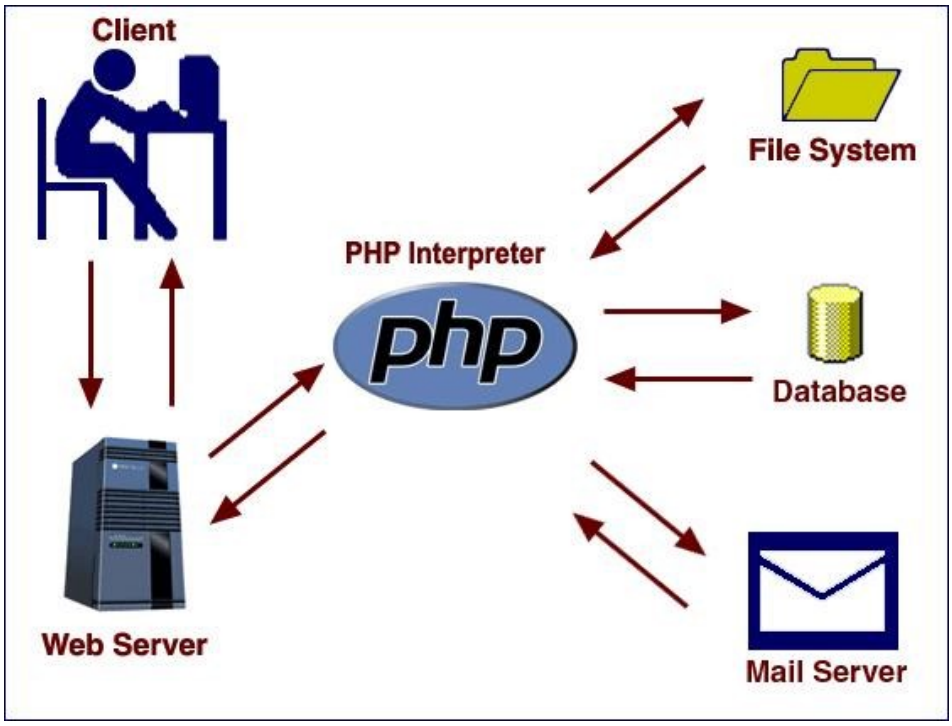
3.3 *PHP*

Um desenvolvedor que lida diariamente com a criação de novos *softwares* ou que está sempre a criar recursos de um sistema mantido pela equipe, parte em busca de uma linguagem de programação que permita seguir padrões de projeto de forma eficiente. O *PHP* é uma dessas linguagens que foi adotada para o desenvolvimento deste projeto, pois permite utilizar a arquitetura *MVC*, acessando bancos de dados de forma fácil, fazendo com que o desenvolvedor se preocupe realmente com processo do sistema aplicado ao código, do que ficar em recriando a roda em cada novo sistema que é desenvolvido, permitindo assim um melhor gerenciamento de tempo para reais necessidades (SUPAARTAGORN, 2011).

Considerando isso, essa linguagem permite reaproveitar grandes sucessos que já foram utilizados por pesquisadores, que, segundo GAMMA (1994), é uma prática muito realizada por projetistas experientes.

Conforme SUPAARTAGORN (2011) resume de forma objetiva, *PHP* é uma linguagem de *script* que executa suas *performances* do lado do servidor, designada especificamente para aplicações *web* (figura 6). Essa linguagem tem uma rápida curva de aprendizagem para desenvolvedores que têm o conceito de programação orientada a objetos, então permite que o desenvolvedor se preocupe em criar o sistema de forma focada.

Figura 13 - Fluxo de uma aplicação *PHP*.



Fonte: http://www.w3programmers.com/overview-of-web-development-with-php-and-mysql/.

Entendendo o fluxo de uma aplicação em *PHP*, conforme ilustrado na figura, o desenvolvedor *PHP* trabalhando em um projeto com a estrutura *MVC*, deve se preocupar em três diretórios do projeto:

- */app/Models*: diretório onde são criados os arquivos "*.php*" referentes às entidades levantadas com a engenharia de *software* e que, posteriormente, é criada toda estrutura no banco de dados, com a criação de tabelas e seus atributos. Então, um arquivo "*User.php*" neste diretório, pode ser referente à tabela "*users*" do banco de dados que escolheu utilizar no desenvolvimento do sistema. Este arquivo conterá a definição dos campos criados para essa tabela "*users*", como por exemplo, id, nome, *email* e data cadastro. Além disso, podem ser criados métodos que retornam uma outra classe relacionada a esta, por exemplo, pode-se retornar uma lista de telefones com o método *public function phones(),* fazendo uma busca no banco de dados na tabela "*phones*", porém informando de qual usuário deseja retornar. Assim é possível que seja criada facilmente uma interface para apresentação desses dados, bastando somente acessar esses atributos ou métodos da *model.*

- */resources/views*: a *view,* se trata do *front-end* de um sistema completo em *PHP*, na qual cria-se um formulário para cadastro de um registro para persistência em banco de dados. O *Blade* é formato de arquivo para criação dessas interfaces de apresentação e manipulação de dados, utilizando o *HTML* e, com a sintaxe *Blade*, podem ser criadas rapidamente páginas *web*, por exemplo, que apresentam uma lista de usuários com seus respectivos telefones, conforme exemplificado no item anterior, bastando que o controlador da *view*, "*index*.*Blade*.*php*", referente ao usuário, receba em seu contexto de apresentação uma lista de usuários, sendo que cada usuário contém uma lista de telefones. Para exemplificar de forma simples, na figura 8 há a tabela apresentada pela *view* index, criada na sintaxe do *Blade,* que lista todas as pessoas cadastradas na tabela "*people"* de determinado sistema. Tendo uma coleção de pessoas no contexto do documento *Blade*, conforme documentação LARAVEL (2020), é possível que se utilize a sintaxe correta para acessar a variável "$*people*" como um *array* de objetos. Conforme é possível ver pela figura 8, o código da figura 7 criou a apresentação de uma tabela os atributos da *model Person.php,* utilizando o conhecido *loop* para cada item do *array people*, acessando-o assim como um objeto.

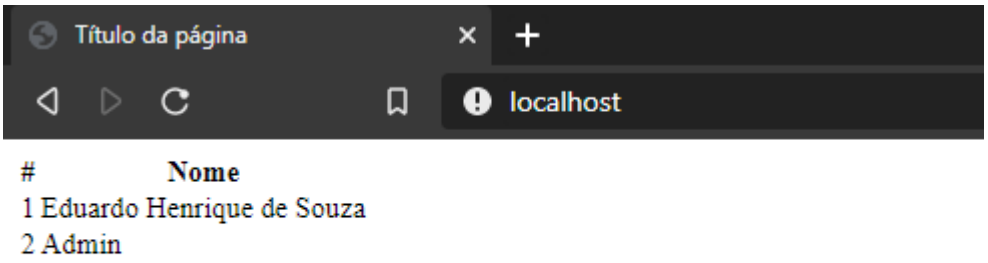
Figura 14 – Exemplo *table HTML* em *Blade PHP*.

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Título da página</title>
    </head>
    <body>
        <div>
            <table>
                <thead>
                    <tr>
                        <th>#</th>
                        <th>Nome</th>
                    </tr>
                </thead>
                <tbody>
                    @foreach ($people as $person)
                    <tr>
                        <td>
                            {{$person->id}}
                        </td>
                        <td>
                            {{$person->name}}
                        </td>
                    </tr>
                @endforeach
                </tbody>
            </table>
        </div>
    </body>
</html>
```

Fonte: Elaborado pelo autor.

A figura 8 lista todas as pessoas cadastradas na tabela “*people*”.

Figura 15 - Resultado *view Blade PHP*.



Fonte: Elaborado pelo autor.

- */app/Http/Controllers*: os controladores, que são disparados pelo acesso em determinada rota de *api* ou *web,* permitem combinar a programação da interface com o os modelos de dados do sistema. Os controladores podem simplesmente retornar uma *view* com determinado layout em *HTML*. Para este exemplo utilizou-se o controlador "*HomeController.php*" para levar o usuário que está navegando no sistema para a página inicial da aplicação. Com a Figura 9, observa-se que foram necessárias duas linhas de códigos. Uma buscando todas as pessoas cadastradas na tabela "*people"* do banco de dados e outra com a linha de retorno no controlador, no qual o documento *HTML* interpretado pelo navegador *web*, apresenta aquela *view* e todos seus componentes com base no caminho em que ela se encontra. O parâmetro após o nome da *view,* indica qual o conteúdo será acessível pelo *Blade*, em que neste caso, a variável *"people"* contém um *array* de objetos pessoa que foi utilizado no *Blade* e *HTML* para criar as contendo as informações do banco de dados. Por exemplo, utilizando-se o arquivo "index.*Blade*.php" no diretório "*/resources/views*/versao2/", bastaria alterar o retorno para "*return view(*"versao2.*index")*", no qual o caractere ponto é utilizado para navegar nas pastas criadas para organizar o projeto.

Figura 16 - *Laravel HomeController*.

```
26      public function index()
27      {
28          $people = Person::all();
29          return view('index', compact(['people']));
30      }
31  }
```

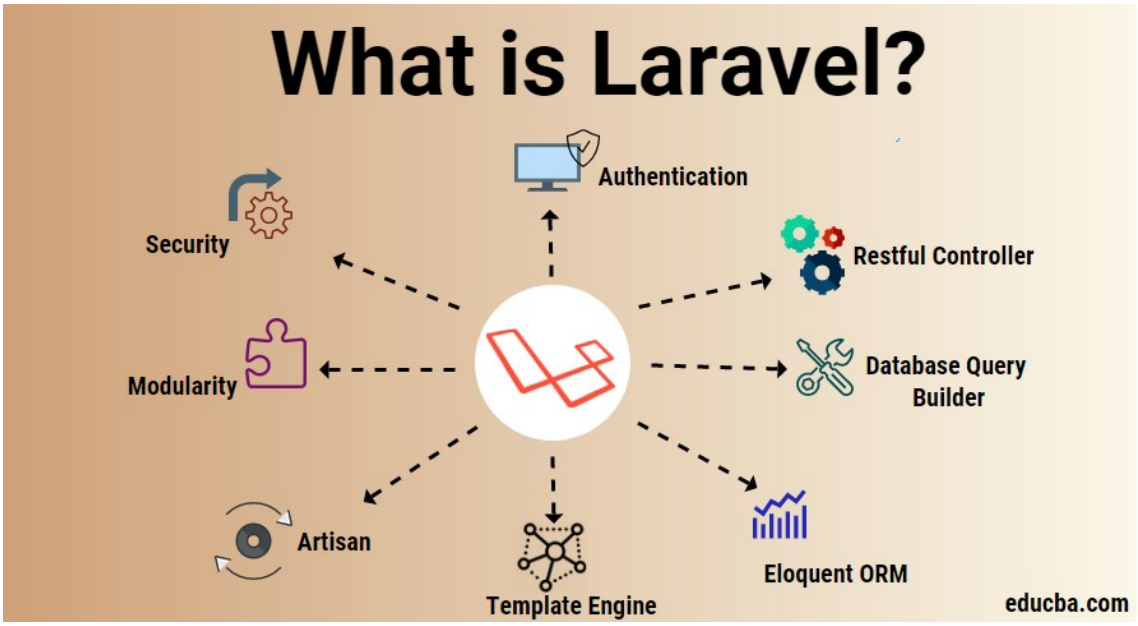
Fonte: Elaborado pelo autor.

3.4 *Framework Laravel* Para *PHP*

Tendo uma base de como se comporta um banco de dados, desenvolvedores que estão sempre antenados a *podcasts*, artigos e publicações, buscam deixar o seu trabalho menos complicado para os dias e as demandas de atualmente, então é aí que se dá grande importância aos chamados *frameworks*. Um *framework* aplicado a

linguagem *PHP*, engloba basicamente todo o processo básico de recursos bases contidos em *softwares,* conforme observa-se na Figura 10.

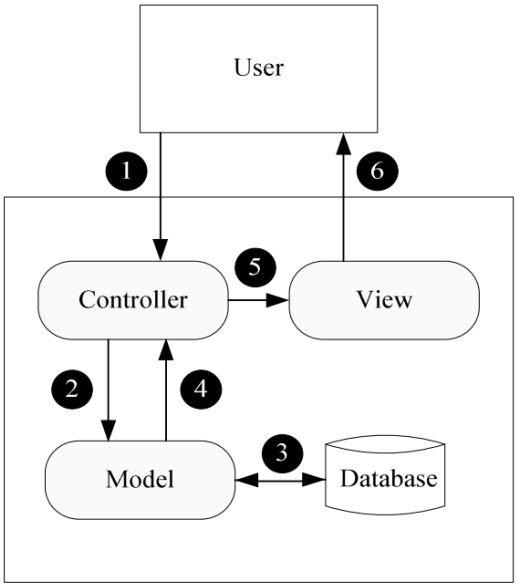
Figura 17 - Abrangência do *framework Laravel*.



Fonte: https://cdn.educba.com/academy/wp-content/uploads/2019/09/What-is-Laravel.png.

O *framework Laravel* para *PHP*, com base no padrão de desenvolvimento *MVC*, engloba desde a interface de apresentação com o usuário até a manipulação com o banco dados, conforme ilustram as ligações na Figura 11.

Figura 18 - Arquitetura framework.



Fonte: Chanchai Supaartagorn (2011)

Como DOCKINS (2016) exemplifica em seu livro, o *framework Laravel* auxilia o desenvolvedor a criar desde a mais simples até a mais complexa manipulação de um objeto no *software*. Para cada tabela criada no banco de dados, este *framework* espera sempre uma classe (*model*) com o mesmo nome para que possam conter as definições de colunas, começando pelo nome e se estendendo ao relacionamento com outras *models*. Para as manipulações desses modelos, utilizam-se os controladores que estão em contato com a *view* e também com a *model,* permitindo gerenciar navegação de telas do *software* e manipular a persistência de dados.

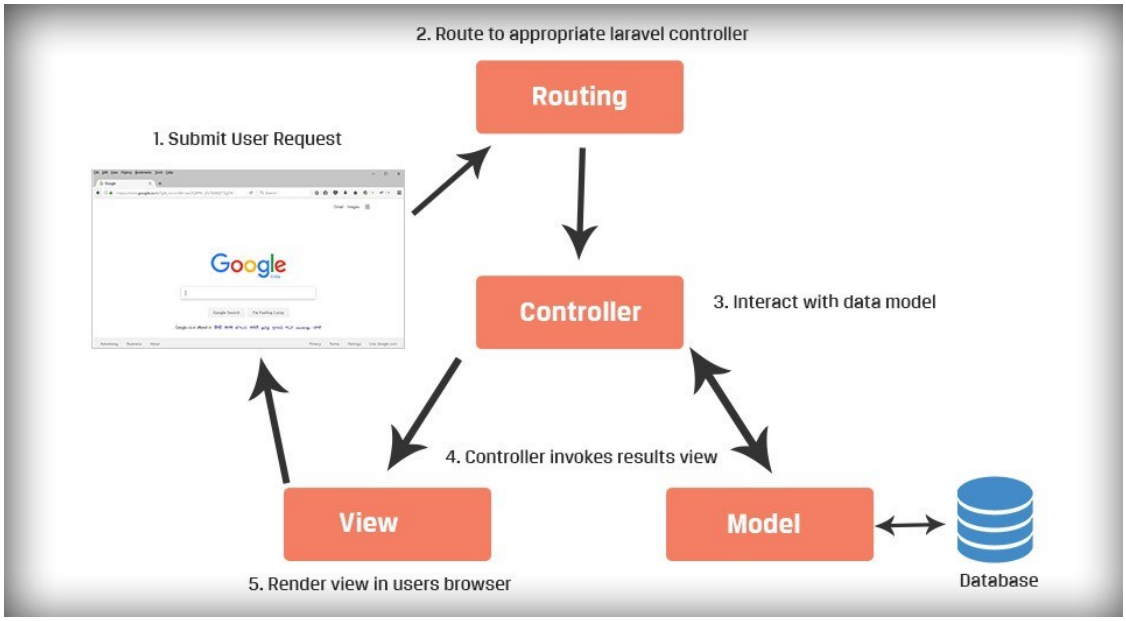
Por convenção, um *controller* no *Laravel* tem o nome da *model*, seguido da palavra *Controller*, tornando-se mais prático encontrar onde aquela entidade que é manipulada. Ao criar um *controller* com os recursos do *framework*, o desenvolvedor tem à sua disposição seis métodos muito importantes, e na maioria das vezes, os essenciais para trabalhar com determinada entidade. Esses métodos permitem realizar praticamente toda operação *CRUD* (*Create, Read, Update* e Delete) do sistema (figura 12), que, para facilitar, pode-se dividi-los em dois grupos: visualização de telas para leitura, e manipulação de dados para criar, atualizar e remover.

Figura 19 - Objetivos do *CRUD* de um sistema.

Fonte: https://i1.wp.com/stefanomanfredini.info/wp-content/uploads/2017/06/crudblog.png?resize=566%2C299&ssl=1.

As rotas de navegação de uma aplicação *Laravel*, são divididas em 2 arquivos, que são compostos basicamente por uma ação, nome e um *path*. A ação se encontra definida dentro do controlador, com as regras negócio e validações necessárias. Pode-se entender o *path* como um caminho para alcançar e executar ação, que complementa a *URL* de uma requisição *HTTP* de uma aplicação, por exemplo http://dentalcloud.com.br/user/create em que o *path* "*/api/user/create*" direciona para determinado método do controlador da classe *user*, que no caso por convenção do *framework* e boas práticas, redireciona para o método *create* do controlador do usuário.

Figura 20 - Roteamento no *framework Laravel* com padrão *MVC*.



Fonte: https://medium.com/@dannyhuang_75970/learning-laravel-controllers-101-ad28d2bb5569.

Levando em consideração que o *Laravel* foi projetado para se desenvolver uma aplicação completa, são separados os arquivos de direcionamento das rotas ilustradas na figura acima, sendo eles "*web.php*" no qual são criadas rotas de navegação de telas ou recursos do *software*, que por padrão direciona para os três métodos:

- *Index*: neste método retorna-se a listagem de registros armazenados no banco de dados referente aquela, por exemplo, ao "chamar" o método é acionado pela rota "*/user/index*", pode-se primeiramente buscar todos os usuários cadastrados no sistema e apresentá-los em uma tabela com o *HTML*.

- *Create*: como o próprio nome diz, trata-se de processo de criação de um novo registro para a entidade, que como no exemplo da *URL* completa citada, com "*/user/create*", apresenta-se uma *view* contendo um formulário de cadastro do usuário e que posteriormente será chamado um outro método que persistirá esses dados.

- *Edit*: após ter criado e listado os registros armazenados de uma classe, quase certo que algum momento deverá ser alterado algum dado que foi persistido.

Para isso utiliza-se o método *edit* que redireciona para um formulário bem parecido da criação, porém com algumas particularidades que se inicia com o *path,* no qual "*/user/edit*" direcionaria o usuário da aplicação para esta tela.

- *Show*: ao ser necessária uma apresentação de detalhes completos de determinada classe, utiliza-se o método *show,* que é utilizado para apresentar uma *view* de detalhes com base. Lembrando que ao editar ou mostrar algo, se trata de um único registro daquela entidade, que foi filtrado pelo ID único no banco de dados.

Os métodos apresentados acima, não estão relacionados a persistência de dados da classe respectiva. Para isso existem outros três métodos, que são acessados por rotas da mesma forma que a navegação entre telas, porém em um arquivo "*api.php*" e com finalidades um pouco diferentes.

- *Store*: é utilizado para realizar o procedimento de inserção no banco de dados, que geralmente é definido na ação de método *POST* em um *form* do *frontend* em *HTML*, permitindo referenciar as colunas a serem inseridas com base nos campos do formulário da *view.* Um exemplo do mesmo contexto de usuários, ao utilizar uma rota *POST* em "/*api/user*/" é realizada essa operação de alimentar a tabela *users* do banco de dados.

- *Update:* da mesma forma que inserimos os dados, é possível realizar as alterações permitidas pela regra de negócio implantada no *software*, como por exemplo, alterar o status de um usuário para bloqueado. Então, com um formulário bem parecido com o de criação, pode-se ao invés de implementar a ação chamando uma rota de *POST*, direcionar-se para a rota referente a um *PUT* ou *PATCH* no mesmo *path "/api/user*". Lembrando que para atualizar um registro no banco de dados, é de grande importância que seja referenciado pelo ID único daquele registro na tabela, que por padrão é um parâmetro do método *update* dos controladores do *Laravel*. Com este ID, é possível que seja buscado o registro atual armazenado e com base neles também sejam realizadas validações antes de efetivar qualquer alteração para aquele registro.

- *Destroy:* que pelo próprio nome indica, utilizado para realizar operações de deletar registro do banco de dados referente à determinada entidade. O objetivo desse método é bem simples, pois com base no ID informado como um parâmetro, seja realizada a validação e posteriormente a exclusão do registro no banco de dados.

Basicamente com esses seis métodos, forçam a não violar o princípio da responsabilidade única, que conforme PIRES (2013) explica que *SOLID*, “é um acrônimo dos cinco primeiros princípios da programação orientada a objetos e design de código identificados por Rober C. Martin (ou Uncle Bob) por volta do ano 2000”. Este é um dos cinco princípios propostos, que reforça que uma classe deve ter somente um motivo para ser alterada, pois colocando um exemplo simples na qual a classe que cadastra um usuário, envia um e-mail para o mesmo. Isso impacta ao realizar alguma alteração na regra de negócio referente ao cadastro, forçando com que sejam testadas as duas ou mais responsabilidades dessa classe. Portanto, com um diagrama de classes bem planejado, torna-se menos complicado aplicar este conceito em qualquer projeto de desenvolvimento de *software*, pois torna-se mais fácil enxergar as todas as entidades necessárias no processo e deixando a preocupação de analisar com mais cautela, somente os métodos que foram criados além dos padrões propostos pelo *framework*.

4 ENGENHARIA DE *SOFTWARE*

> “Engenharia de *software* é uma abordagem sistemática para a produção de *software*; ela analisa questões práticas de custo, prazo e confiança, assim como as necessidades dos clientes e produtores do *software*. A forma como essa abordagem sistemática é realmente implementada varia dramaticamente de acordo com a organização que esteja desenvolvendo o *software*, o tipo de *software* e as pessoas envolvidas no processo de desenvolvimento.” (SOMMERVILLE, 2011).

Na engenharia de *software* para este projeto, foi priorizada a questão das necessidades do negócio, por se tratar de uma relação próxima entre o desenvolvedor e o cliente final. O projeto agregará para ambos os lados, permitindo que o laboratório seja mais bem gerenciado, e permitindo o nascimento de uma nova ferramenta para esta finalidade. Posteriormente, serão apresentados os diagramas desenvolvidos, após ser realizada toda análise dos requisitos levantados.

4.1 Diagramas

Conforme explica MOREIRA (2000), “A idéia central da UML é fazer com que os arquitetos de sistemas trabalhem com Análise e Projeto Orientados a Objetos a partir de uma linguagem consistente para especificação, visualização, construção e documentação dos aspectos do *software*, bem como para a modelagem do negócio”.

LOBO (2009) mostra que um desenvolvedor que planeja seguir no mercado de *software*, deve conhecer os conceitos de modelagem de um processo antes de iniciar a codificação do sistema. Com o desenvolvimento de diagramas antes do código do programa, é possível obter uma visão geral do projeto, facilitando o trabalho em equipe e mantendo o projeto organizado. Segundo LOBO (2009), os diversos modelos de diagramas, representam as diferentes perspectivas do *software*. “O diagrama de Casos de Uso auxilia no levantamento dos requisitos funcionais do sistema, descrevendo um conjunto de funcionalidades do sistema e suas interações com elementos externos e entre si.”. VIEIRA (2015). O diagrama de casos de uso, auxilia na identificação das entidades, o que facilita a criação do diagrama de classe, que apresenta os objetos que farão parte do sistema, entidades que serão manipuladas dentro do *software* e quais atributos e métodos deverá conter. O diagrama de

sequência é criado após esse dois anteriores, pois ele detalha os casos de uso, mostrando por quais métodos a execução de determinado processo passa.

4.1.1 Diagrama de Casos de Uso

"O diagrama de Casos de Uso auxilia no levantamento dos requisitos funcionais do sistema, descrevendo um conjunto de funcionalidades do sistema e suas interações com elementos externos e entre si.". VIEIRA (2015).

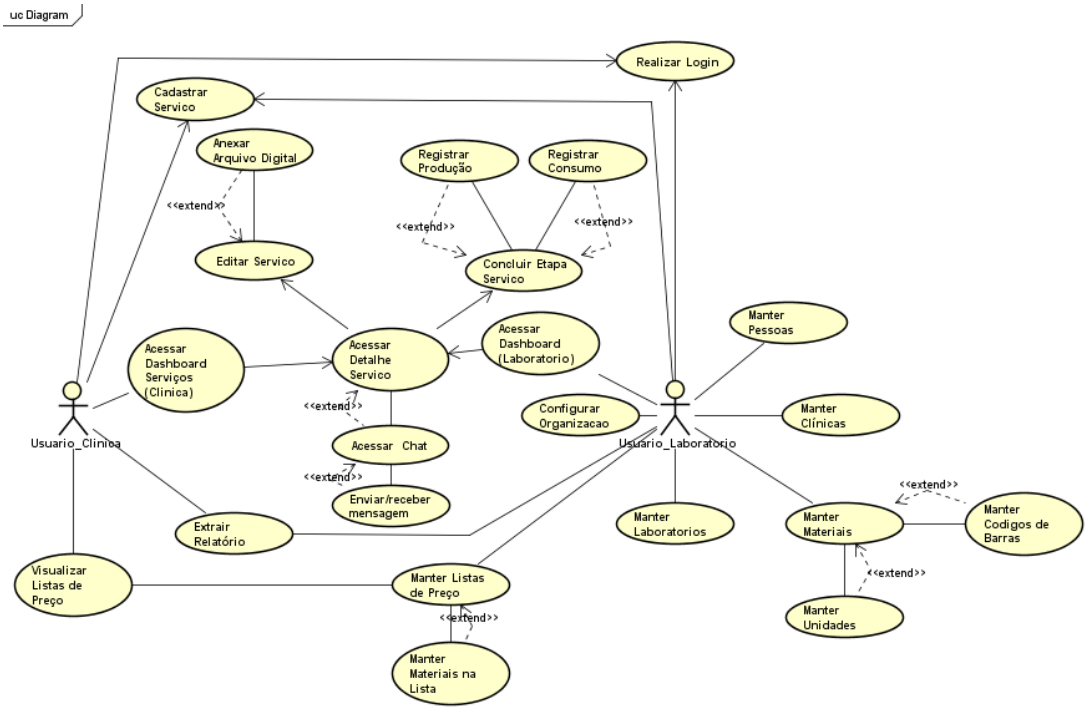
A entrevista com o cliente levanta um diálogo que deverá ser interpretado pelo analista de *software*. Essa interpretação, acabará gerando os requisitos funcionais do sistema que deverão ser validados com o cliente, pois deverá com base nos requisitos do usuário. O diagrama de casos de uso é utilizado para apresentar ao usuário, facilitando a comunicação com o analista.

Conforme VIEIRA (2015) explica, os principais elementos desse diagrama são:

- Atores: se refere a quem tem a intenção de executar determinada ação no sistema. Exemplos de atores para este projeto, seriam o usuário do laboratório e o usuário da clínica. Para as funcionalidades automáticas do processo, também é possível utilizar o sistema como um ator do diagrama.
- Casos de uso: este elemento, representa uma funcionalidade do sistema que foi interpretada dos requisitos de usuário, que acabou se tornando um requisito de sistema. Neste projeto foram utilizados verbos como, manter, acessar, anexar, configurar, entre outros que são relacionados.

  Relacionamentos: um relacionamento é a ligação entre a intenção de um ator, entre um ator e um caso de uso, ou entre atores e entre casos de uso.

Figura 21 - Diagrama de casos de uso.



Fonte: Elaborado pelo autor.

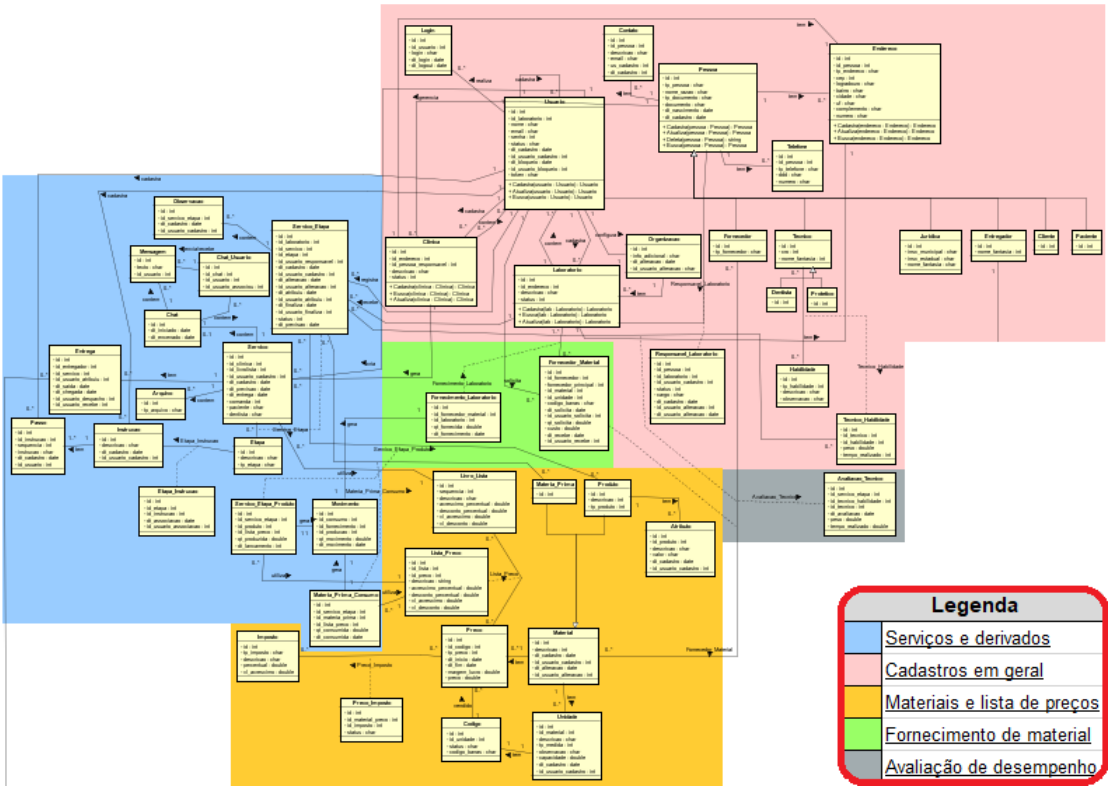
4.1.2 Diagrama de Classe

O diagrama de classe de um projeto apresenta de forma gráfica, as classes/modelos de objetos que farão parte do sistema e os relacionamentos entre elas. "Um objeto é um termo que usamos para representar uma entidade do mundo real” (MACORATTI). Por exemplo, ao surgir a necessidade de persistir dados de um sistema orientado a objetos, é necessário se utilizar modelos pré-definidos que surgirão após a análise de requisitos. Conforme SALMA (2017) mostra em seu artigo, uma classe contém um nome, seus atributos e métodos de determinado objeto. Por exemplo, para manter um cadastro de pessoas pode-se utilizar uma classe com nome “pessoa” com os atributos, nome, tipo do documento identificador, número do documento identificador e data de nascimento.

Esses modelos de objetos, geralmente são relacionados uns com outros, permitindo que o contexto do sistema esteja interligado, deixando a liberdade de operações sistêmicas cada vez mais robustas e automatizadas, caso haja novas

demandas ou integrações. Na figura 21, observa-se o diagrama completo do sistema, divido por seções de diferentes cores.

Figura 22 - Diagrama de classes das entidades e relacionamentos do sistema *DentalCloud*, dividido em seções.



Fonte: Elaborado pelo autor.

- Serviços e derivados: as chamadas operações do sistema e suas classes de dados complementares. Por exemplo, registro de um novo serviço, que poderá conter muitas etapas, em que cada etapa poderá gerar e consumir materiais, que por sua vez, são matérias-primas e produtos.

- Cadastros em geral: cadastros que poderão ser utilizados em alguma operação dentro do sistema. Por exemplo, cadastro pessoas, laboratórios, fornecedores e seus respectivos dados complementares, endereços da pessoa, contatos do laboratório ou do fornecedor.

- Materiais e lista de preços: cadastro de materiais produzidos e consumidos ao realizar determinada etapa do serviço. Por exemplo, para entregar o produto “modelo de gesso”, é necessário utilizar a matéria-prima “gesso”.

- Fornecimento de material: entidades utilizadas para representar a entrada de uma matéria-prima entregue pelo fornecedor. Por exemplo, para registrar uma movimentação no estoque, é necessário registrar um fornecimento de material, que relaciona um fornecedor cadastrado no sistema, com um material. O fornecimento é realizado em uma data e com determinada quantidade, que se encaixam como atributos dessa entidade de relacionamento, além de ser recepcionado por outra pessoa cadastrada no sistema, podendo ser cliente, fornecedor ou colaborador do laboratório.

- Avaliação de desempenho: tem por objetivo armazenar dados simples, para avaliação do desempenho de um técnico, no momento em que o mesmo realiza uma etapa do serviço. Desta forma, o sistema pode conter análises automáticas de tomada de decisão, escolhendo o técnico com mais disponibilidade. É possível escolher o técnico que realiza determinada etapa mais rapidamente, para que assim possa ser entregue um produto com o menor tempo possível.

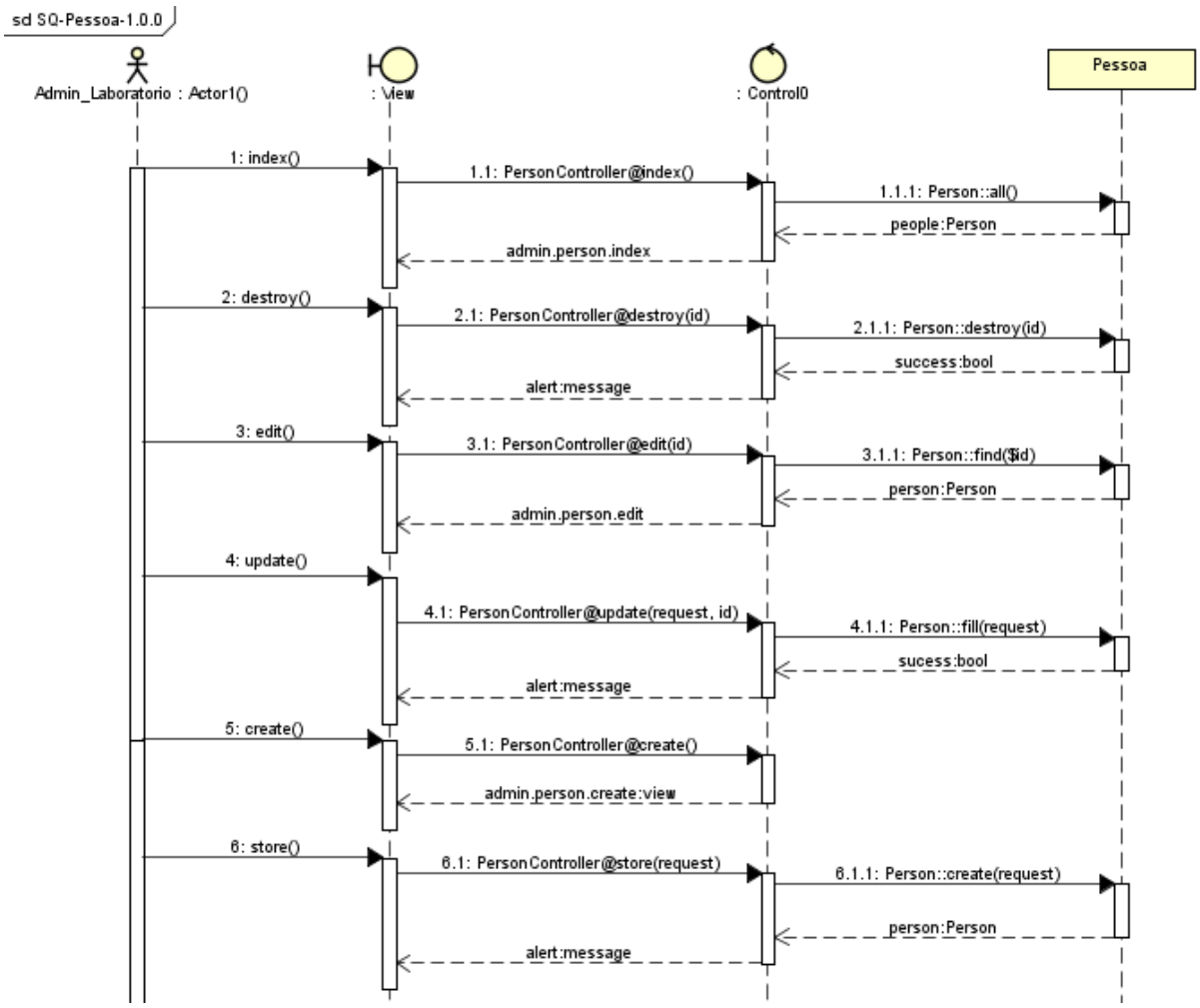
4.1.3 Diagrama de Sequência

Com o diagrama de casos de uso construído, é possível criar o diagrama de sequência para detalhar a sequência de passos que o sistema vai seguir. Utilizando o *framework Laravel*, é possível que todos os controladores funcionem sempre com os mesmos métodos, *index, create, store, edit, update* e *destroy.* Esse fator permite que os diagramas de sequência do projeto, sejam muito semelhantes.

Este diagrama é composto por atores, que são os responsáveis por solicitar um serviço no sistema, acontecendo por meio de uma interface com o usuário. Esta interface por sua vez, envia um comando para o controlador da ação em questão, podendo ser o envio de um formulário de dados para guardar dados de uma pessoa

em banco. Por exemplo, na Figura 23, é apresentado o diagrama de sequência para que seja possível manter a entidade “Pessoa” dentro do sistema *DentalCloud.* Observe que o padrão *MVC* e toda sua comunicação durante as etapas, ficam bem visíveis ao serem representadas por este diagrama.

Figura 23 - Diagrama de sequência (Pessoa).



Fonte: Elaborado pelo autor.

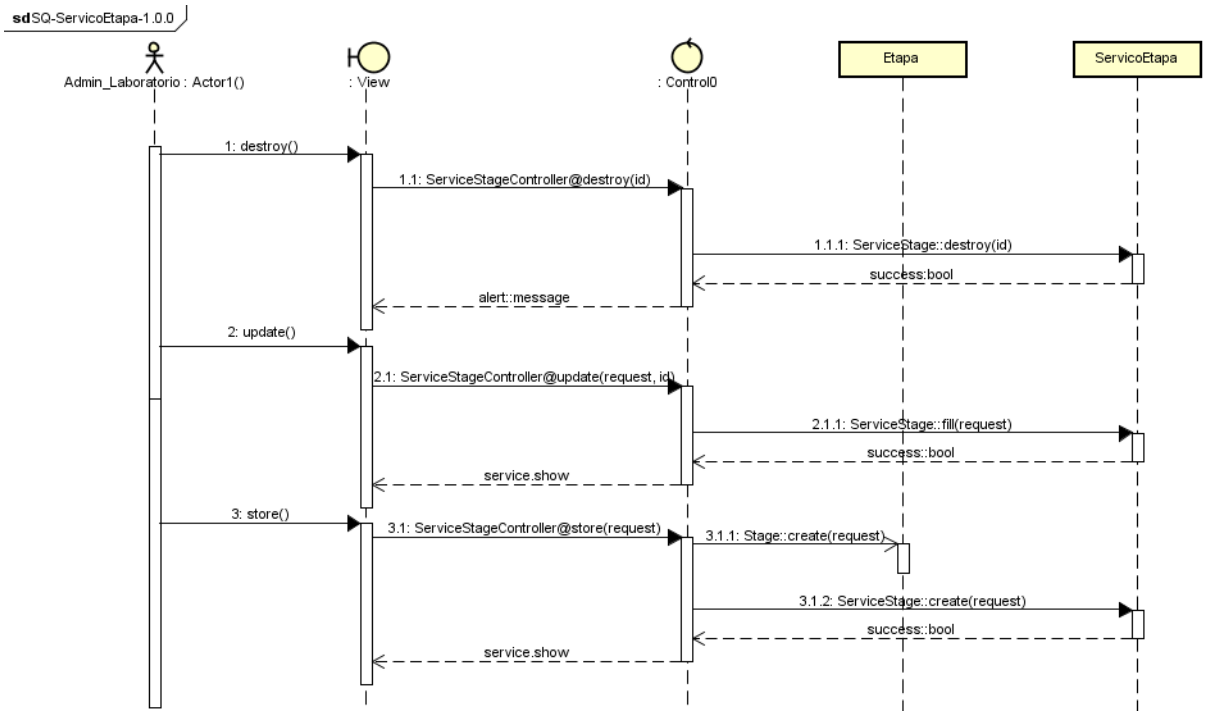
Na esquerda do diagrama, geralmente é posicionado o ator do sistema, que no caso é o administrador do laboratório. Por meio de diversas telas do sistema ou *views,* será possível requisitar determinada informação ou enviar algum comando para ser processado. Um exemplo para armazenar dados de uma pessoa, seria por meio da rota *store*, que ao ser chamada, envia a requisição para o controlador da pessoa. No controlador, podem ser realizadas diversas validações e tratamentos nos dados

recebidos pela interface, até que esteja conforme a estrutura do modelo “Pessoa”, para que finalmente possa ser persistido em banco de dados.

É possível arquitetar um diagrama idêntico para manter um serviço no sistema, basta que se alterem os controladores que estão sendo chamados. Por exemplo, *PersonController* se torna *ServiceController*, e a tabela que está sendo alimentada, Serviço. Determinados controladores do sistema, poderão trabalhar com mais de uma entidade na mesma operação realizada pelo usuário.

Observa-se na Figura 24, que, ao criar uma etapa para o serviço, é criada também uma etapa genérica que poderá ser utilizada para alguma análise de dados. Além de permitir que outros técnicos conheçam e utilizem as etapas mais realizadas naquele laboratório, é possível identificar qual técnico do laboratório que realiza maior quantidades de etapas do mesmo tipo, ou o mais disponível para realizá-la.

Figura 24 - Diagrama de sequência (Etapa do Serviço).



Fonte: Elaborado pelo autor.

5 HISTÓRICO DO PROCESSO

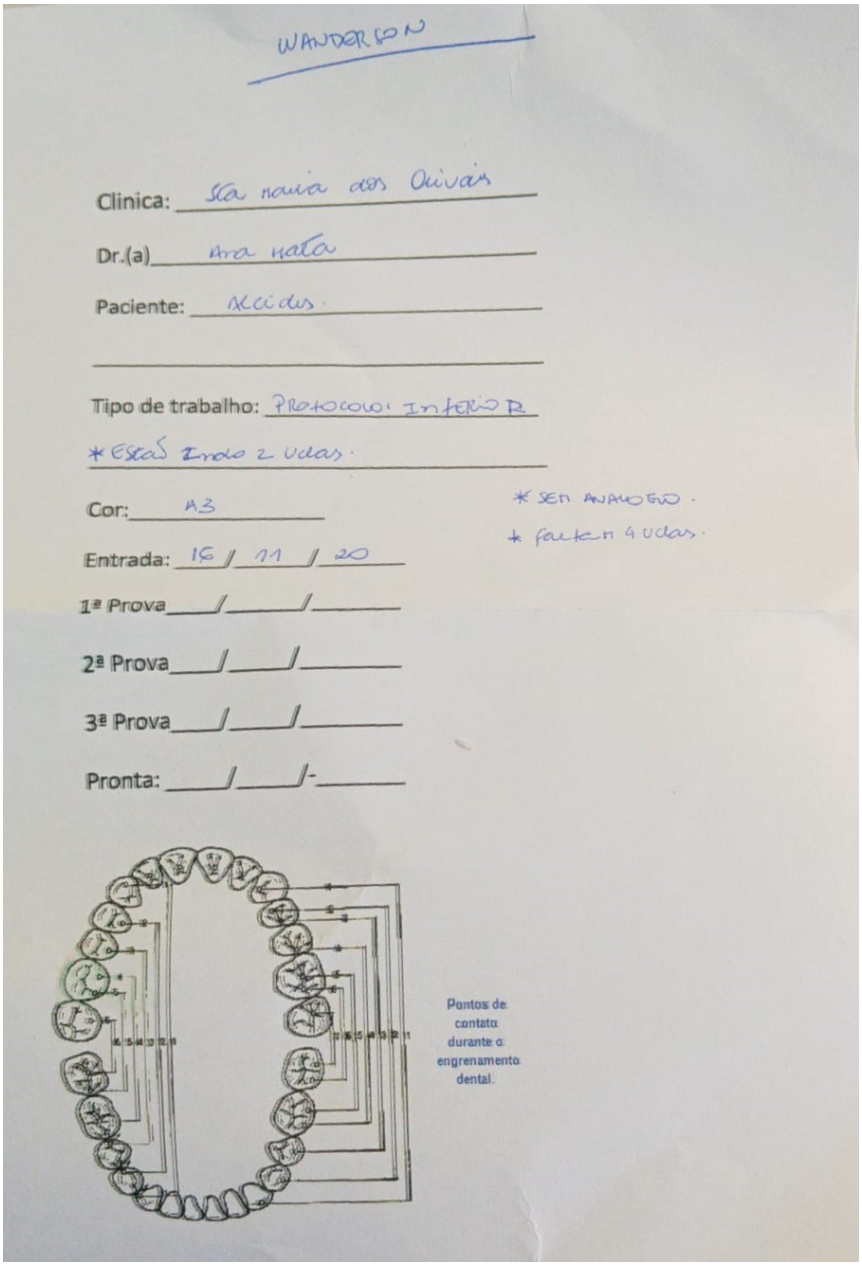
Com um projeto do trabalho bem definido e detalhado, é possível que o desenvolvedor utilize a maior parte do tempo trabalhando no sistema em si. Portanto, se o analista entende o processo do negócio, elabora uma solução sistémica compreendida pelo usuário e esta solução é validada com a entrega de um produto mínimo viável conforme abordado, basta que o processo de melhoria continua do sistema seja continuado. Posteriormente, é possível acompanhar o que o sistema irá agregar ao laboratório de prótese, comparado com a dinâmica atual do negócio.

5.1 Processo Legado

Relatos do usuário, mostraram ao analista que o processo no laboratório é bem manual, utilizando comandas de papel para acompanhar um serviço desde sua origem até a entrega do mesmo. A comunicação entre a clínica e o laboratório, acaba deixando uma especificação de cada serviço um pouco vaga, principalmente por tratar-se de uma grande movimentação mensal de trabalhos, começa a gerar impactos ao negócio. Isso ocorre porque o telefone é bastante utilizado para tirar definições para tomada de decisão, acabando por não documentar corretamente as alterações que surgem no meio do caminho. Por este motivo, era gerado muito retrabalho por conta de detalhes acertados por telefone, não baterem realmente com o que o cliente solicitou.

Com base em relatos de técnicos protéticos, responsáveis por realizar as etapas para a conclusão de um serviço, foi identificado que o controle solicitado pelo gestor da organização, acabava tornando o dia a dia complicado para se organizar, as comandas muitas vezes eram interpretadas com muita dificuldade, por ser escrita à mão. Em busca de um melhor entendimento desse relato, concluiu-se que o técnico tinha dificuldades em marcar a quantidade utilizada de cada material em determinada etapa, por conta de realizar esse controle manualmente, perdia-se um tempo considerável anotando todas essas informações, além de registrar a etapa concluída, gerando assim uma porcentagem de comissão no fechamento do mês. Além disso, observa-se na figura 25, um exemplo de comanda de serviço utilizada pelo laboratório.

Figura 25 - Exemplo de comanda de trabalho utilizada no laboratório.



Fonte: Elaborado pelo autor.

A lista de preços dos diferentes serviços que o laboratório disponibiliza era acessada por meio de livretos impressos em gráfica ou por meio de arquivo *PDF*, via *e-mail*, mas que também passavam pelas mãos da gráfica ao surgir uma necessidade de alteração. Esse processo causava uma grande trava para aplicar um preço dinâmico para cada cliente ou região, algo muito praticado nesta área, pois a dificuldade de realização de trabalhos entre uma clínica e outra é muito grande.

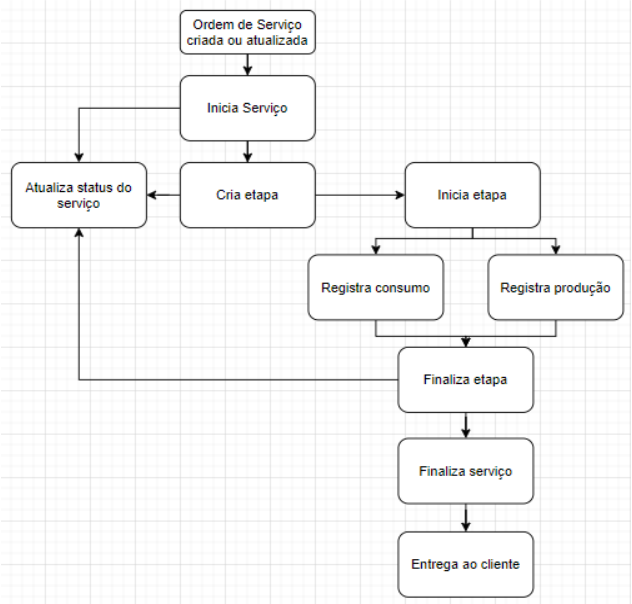
As entregas dos trabalhos são realizadas com baixa rastreabilidade, pois são em média cinco entregadores que prestam esse serviço, acabando por ser difícil gerenciar qual serviço está com determinado entregador. Sem um sistema contemplando esse fragmento do processo, todas as entregas são anotadas manualmente por entregador, para que no fechamento mensal seja possível saber quanto pagar pelas entregas.

O controle desse processo no laboratório, acabava gerando noites de trabalho, analisando cada papel anotado, somando todos os saídas e entradas no caixa, sem contar que, ao realizar esse processo manualmente, o negócio fica muito suscetível a erros, podendo até perder dinheiro com a falta de atenção nesse intenso trabalho.

5.2 Sistema Proposto

Pelo fato de o laboratório estar em constante crescimento e passando por apuros no ano de 2020, foi proposta a entrega de um *MVP* (produto mínimo viável), com base nas maiores dores relatadas. Portanto, o *software* a ser entregue contempla desde a origem, passando pela conclusão de um serviço no laboratório, até que seja possível extrair um relatório para tomada de decisão.

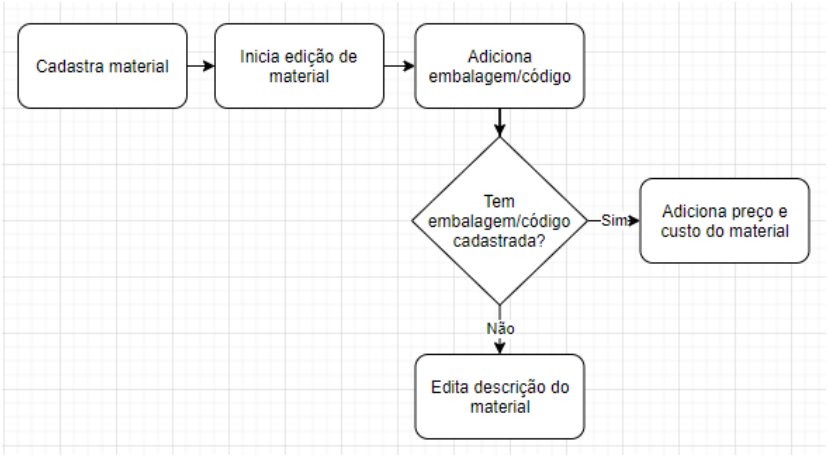
Figura 26 - Fluxograma do processo no laboratório.



Fonte: Elaborado pelo autor.

O sistema contempla o cadastro de materiais utilizados, isto é, toda matéria-prima utilizada para concluir determinada etapa pelo técnico, além de cadastrar o produto final que é gerado a cada etapa, ou entregue ao cliente final. Para seguir o processo dentro do sistema, é disponibilizado um cadastro de pessoas, que podem ser clientes, pacientes, técnicos, entregadores e fornecedores. Além de pessoas, serão cadastradas e mantidas as diferentes listas de preços do laboratório, permitindo um preço diferente por cada serviço, ou uma porcentagem de acréscimo ou desconto na lista inteira. Na figura 27, é apresentado o fluxograma de criação de um material. Percebe-se que o preço e custo dependem de uma parametrização, que no caso é o código ou embalagem. O mesmo conceito é utilizado nos outros tipos de cadastros. Por exemplo, para cadastrar um endereço é necessário que ocorra dentro de um cadastro de pessoa, laboratório ou clínica.

Figura 27 - Cadastro de um material no *DentalCloud*.



Fonte: Elaborado pelo autor.

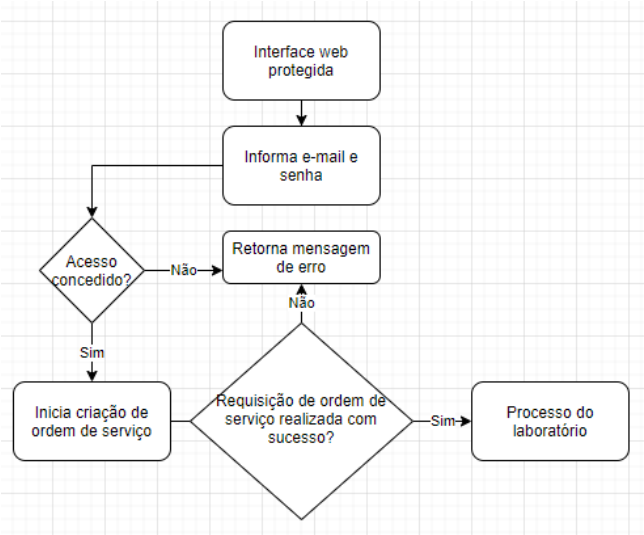
Tendo esses atores cadastrados no sistema, estes poderão utilizar o sistema para aquilo que lhe é proposto. Por exemplo, um cliente que solicita um serviço atribuindo a um paciente, um entregador que está com determinados serviços em processo de entrega e os técnicos que realizam as etapas para a conclusão de um trabalho. Um novo serviço passa pelo gestor do laboratório para ser aprovado, poderá ser atribuído a um técnico para que possa dar sequência nas etapas a serem realizadas, de acordo com cada tipo de serviço a ser entregue.

O técnico terá um acesso ao sistema com *e-mail* e senha, que permitirá que ele alimente o sistema com as observações em cada etapa, registrando os consumos de matérias-primas e produtos gerados em cada etapa. Após realizar todas as etapas previstas, o técnico pode finalizar o serviço e liberar para entrega ou poderá retornar a uma etapa anterior para realizar determinada correção. Todo esse procedimento, é gerenciado pelo administrador do laboratório, que poderá acompanhar o andamento de cada serviço, sabendo se está dentro do prazo ou não.

Com uma interface de criação de uma ordem de serviço, o cliente poderá acessar de onde estiver, bastando que tenha as credenciais para criar um novo serviço. Esta interface requere os dados principais do serviço, além de permitir que seja complementado com mais informações enquanto já estiver em andamento. O administrador do laboratório será o responsável por manter as engrenagens do processo sempre em movimento, pois é ele quem irá manter as listas de preços,

cadastros em geral e orquestrar os serviços que vão chegando no laboratório. Na figura 28, é apresentado o fluxograma do processo explicado.

Figura 28 - Fluxograma da interface cliente-laboratório.



Fonte: Elaborado pelo autor.

A etapa final de um serviço será a liberação para entrega, que incluirá o entregador no processo ou não, pois poderá haver casos em que o próprio cliente busca o trabalho, então, caso seja entregue diretamente, bastará dar baixa no sistema como um serviço finalizado. Haverá casos em que o serviço poderá voltar com defeito, seja por falha do cliente ou até mesmo do próprio laboratório. Então para isso, o sistema deve permitir a reabertura de uma ordem de trabalho e liberar um novo ciclo de etapas, que irá demandar que o cliente, pela mesma interface de serviço, solicite uma correção, podendo anexar imagens para comprovar algum problema. Assim, o sistema entenderá como um serviço normal e permitirá que os técnicos registrem o seu trabalho.

Como o processo de um serviço está totalmente no sistema, no fechamento mensal, o administrador pode extrair relatórios de gastos com materiais, ganhos com serviços entregues, entre outros. Para os clientes, também será possível extrair relatórios com a relação de trabalhos a serem pagos ao laboratório, deixando a operação mais autônoma para ambos os lados. Os técnicos terão acesso para extrair relatório de comissão, no qual constam todos os trabalhos ou etapas efetuadas por

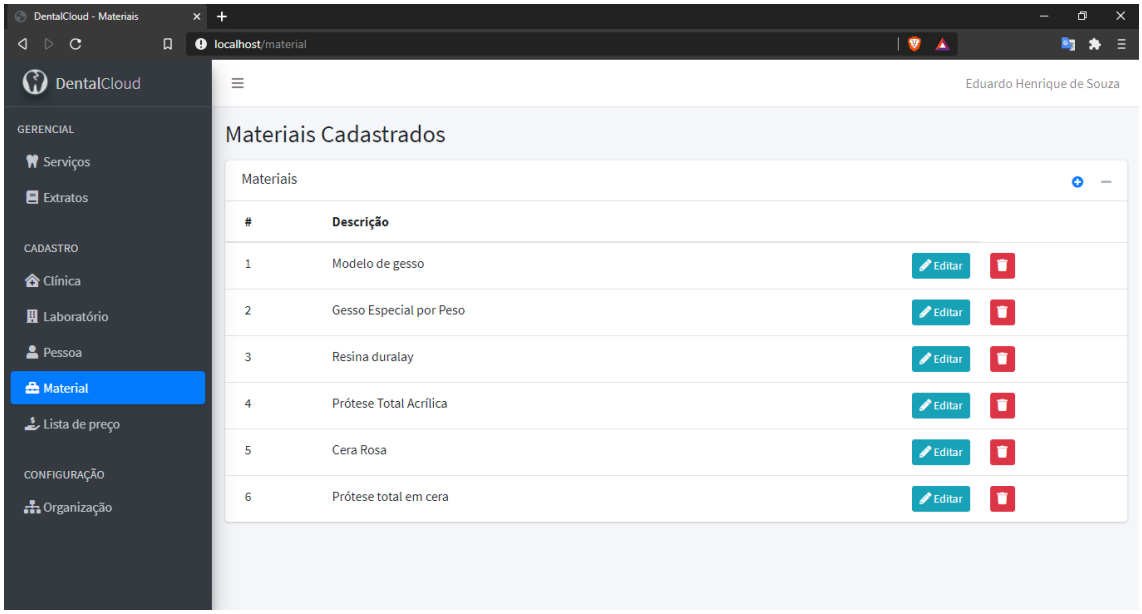
ele, assim será possível acompanhar o seu desempenho e ganhos de cada mês trabalhado.

Por se tratar de um *MVP*, na intenção de amenizar as “dores” atuais do negócio e, graças à tecnologia de um sistema *WEB,* foi possível criar um produto eficiente, trabalhado nos problemas prioritários para o negócio, garantindo uma entrega agilizada, e apresentando inúmeros benefícios para o laboratório.

5.3 Resultado

Como resultado do projeto realizado, foi obtido um sistema que atende as maiores necessidades do processo em questão. Portanto, foram realizadas primeiramente as validações dos cadastros de materiais, pessoas, clínicas, laboratórios e listas de preço. Esses cadastros são essenciais para iniciar as operações relacionadas às ordens de serviço e extrações de relatórios e por este motivo, foi essencial que essas operações fossem validadas como prioridade máxima. Na figura 29 pode-se ver o resultado do cadastro de materiais, que partiu de uma análise do processo legado da empresa, tendo sido elaborado um fluxo de trabalho facilitado, que se transformou em uma ferramenta sistêmica.

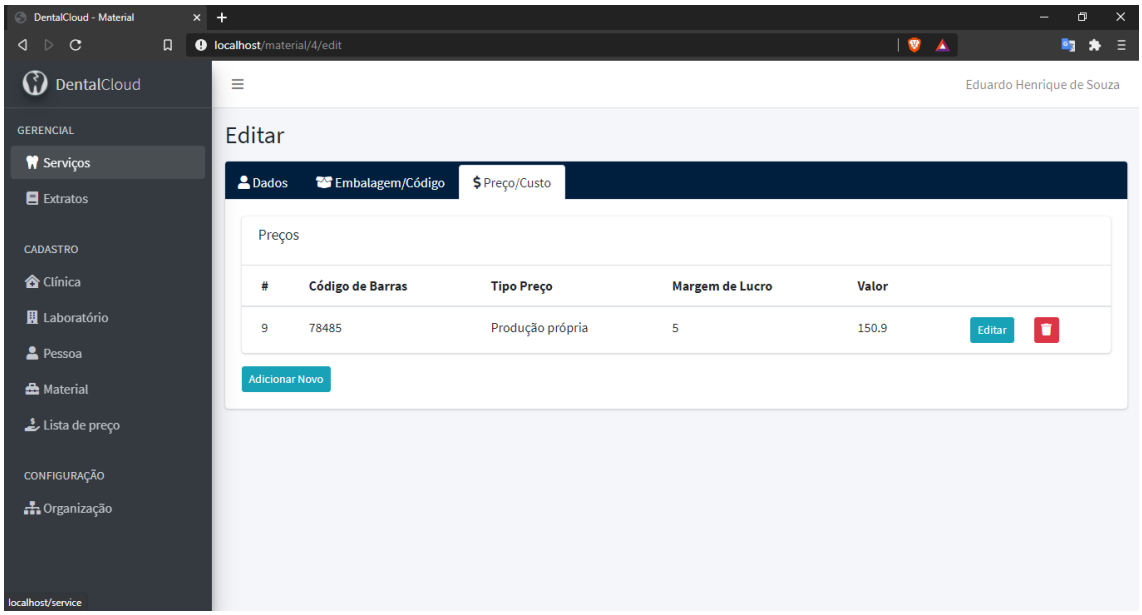
Figura 29 - Tela de listagem de material/produto.



Fonte: Elaborado pelo autor.

O cadastro inicial de um material é realizado somente pela sua descrição. Assim a liberação para criar embalagens, preços é liberada somente na edição do material, uma vez que já foi criado. Na figura 30, observa-se a tela de cadastro de preços de determinado material.

Figura 30 - Tela de edição de preço/custo do material/produto.

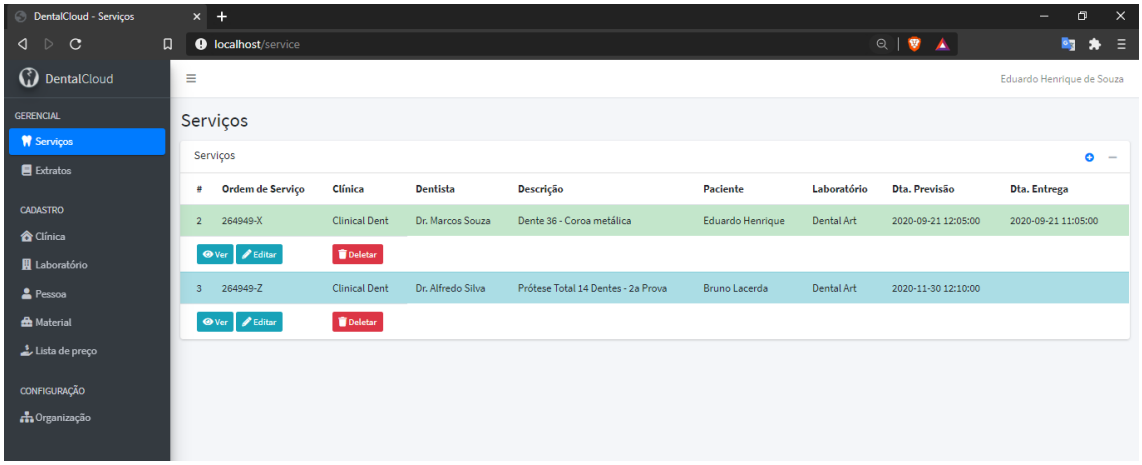


Fonte: Elaborado pelo autor.

O segundo processo mais importante no sistema tem relação com a produção de serviços do laboratório de prótese, em que cada serviço tem a sua importância relacionada ao prazo e também às etapas a serem realizadas. Foi criada uma tela principal que permite ver todos os serviços em andamento, a partir da qual é possível criar, atualizar e concluir etapas.

Além da realização de etapas, é registrado todo consumo de matéria-prima do laboratório, permitindo que o estoque seja controlado e facilitado para gerenciar as compras. O registro de produção também é necessário pela necessidade de enviar um extrato para os clientes, para que seja possível ter a relação de trabalhos entregues, a entregar e o valor a receber. Na figura 31, é apresentada a tela principal de serviços, onde é possível ter uma visão macro do andamento dos trabalhos que estão no laboratório.

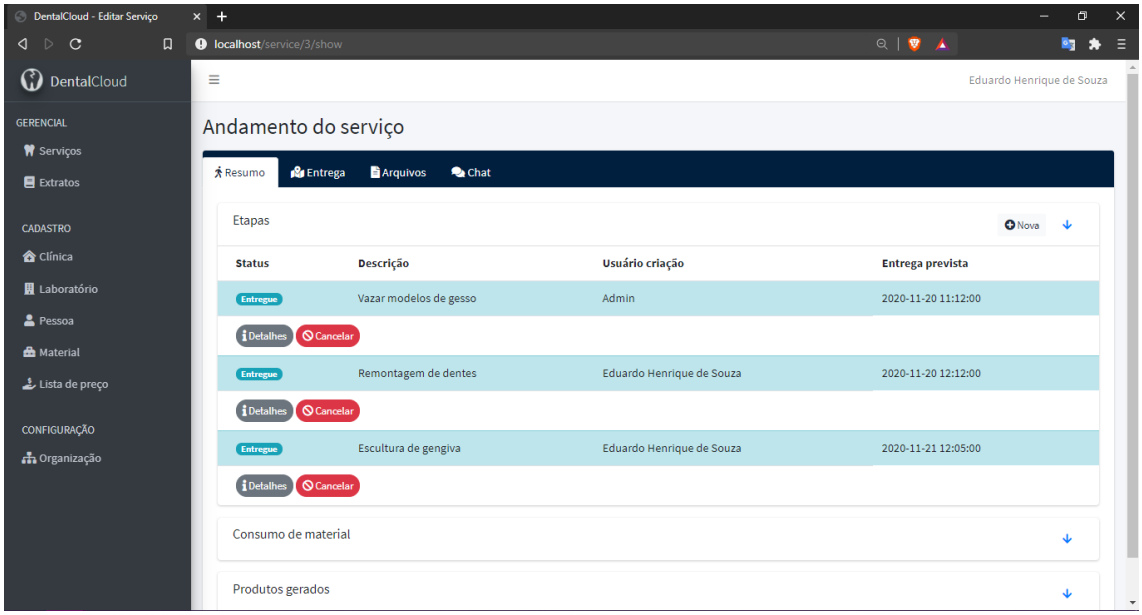
Figura 31 - Tela principal dos serviços.



Fonte: Elaborado pelo autor.

Para cada ação necessária no serviço, são criadas as etapas para serem realizadas pelos técnicos colaboradores do laboratório. Cada etapa pode ser atribuída a um técnico diferente, pois dependerá do tipo de serviço a ser entregue. O técnico que recebeu essa tarefa deverá registrar o início do seu trabalho, pois dessa forma será possível analisar o tempo que determinado técnico leva para finalizar uma determinada etapa de um serviço. Na figura 32, é possível ter uma visão geral das etapas concluídas ou que ainda estão em andamento, sempre associada ao serviço.

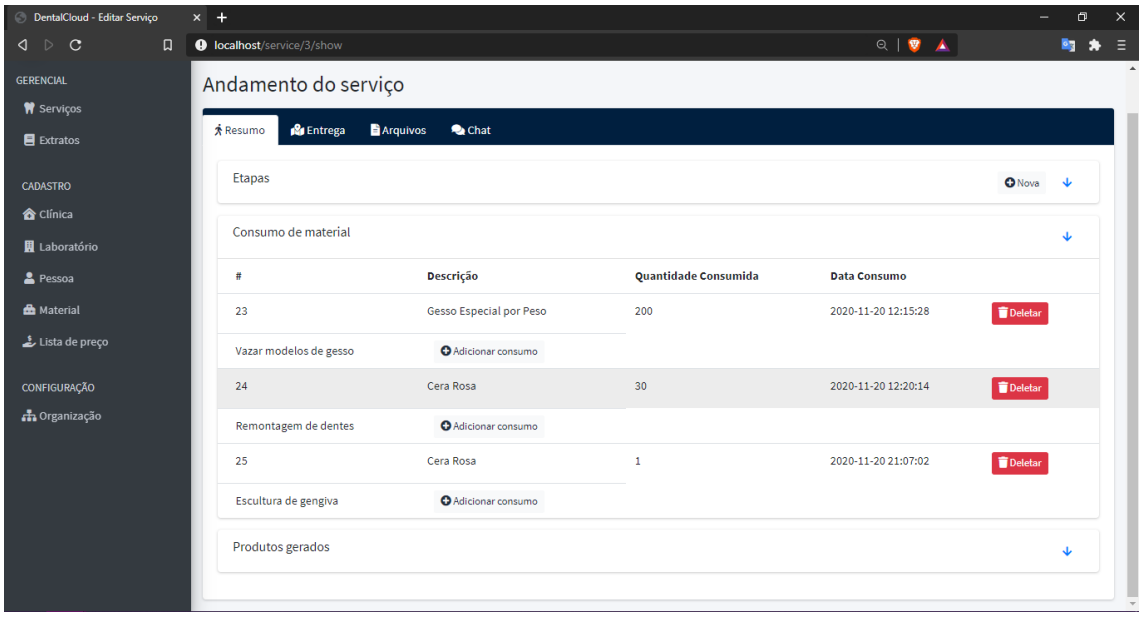
Figura 32 - Tela de andamento do serviço. Etapas do serviço.



Fonte: Elaborado pelo autor.

Após finalizar cada etapa do serviço, é necessário lançar o consumo de matéria-prima necessária. Da mesma forma que é lançado o consumo, é registrada também a produção em cada etapa, que poderá ou não existir. Na figura 33, é possível compreender como o técnico deverá realizar esse lançamento.

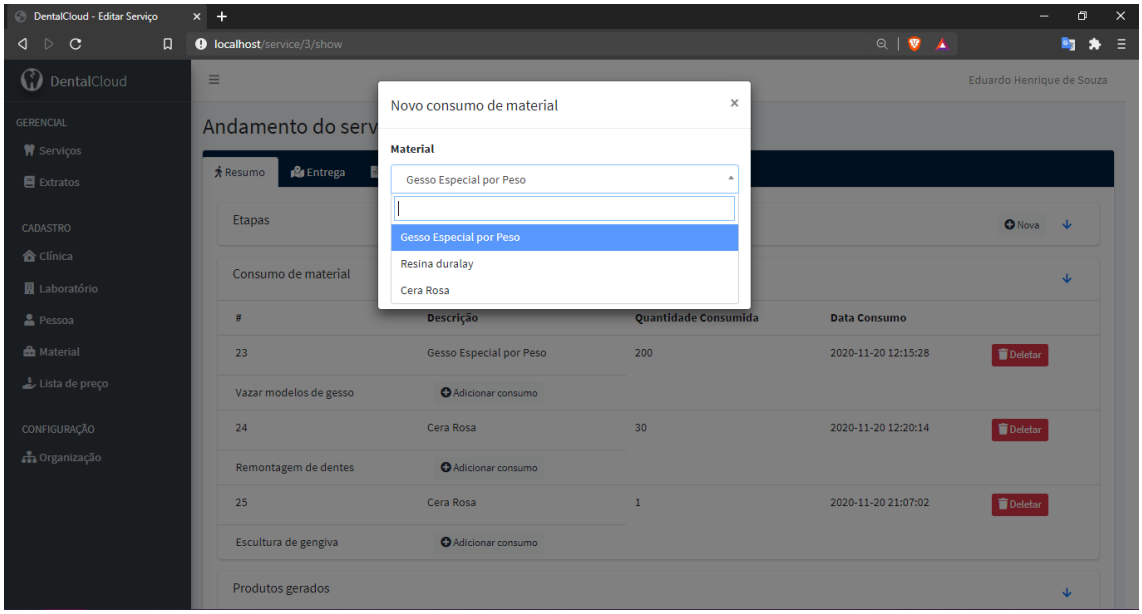
Figura 33 - Tela de andamento do serviço. Consumo de material das etapas.



Fonte: Elaborado pelo autor.

Para adicionar um novo consumo, o sistema permite filtrar os materiais cadastrados e registrar a quantidade consumida da matéria-prima utilizada (figura 34).

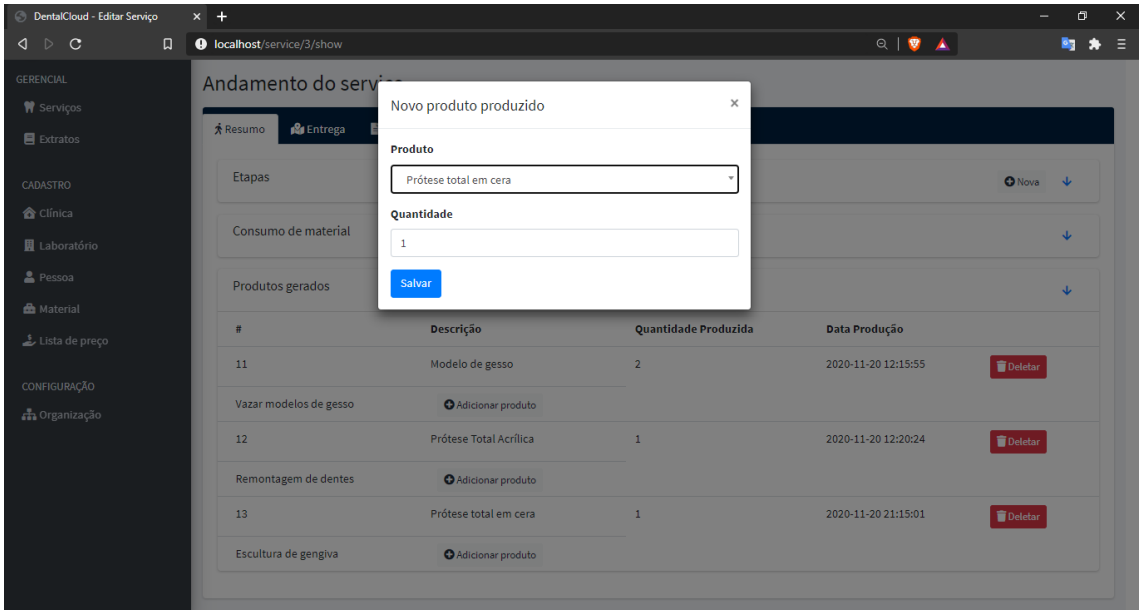
Figura 34 - Tela de andamento de serviço. Adicionar consumo de material.



Fonte: Elaborado pelo autor.

O mesmo acontece para adicionar um produto gerado na etapa, conforme é possível compreender na figura 35.

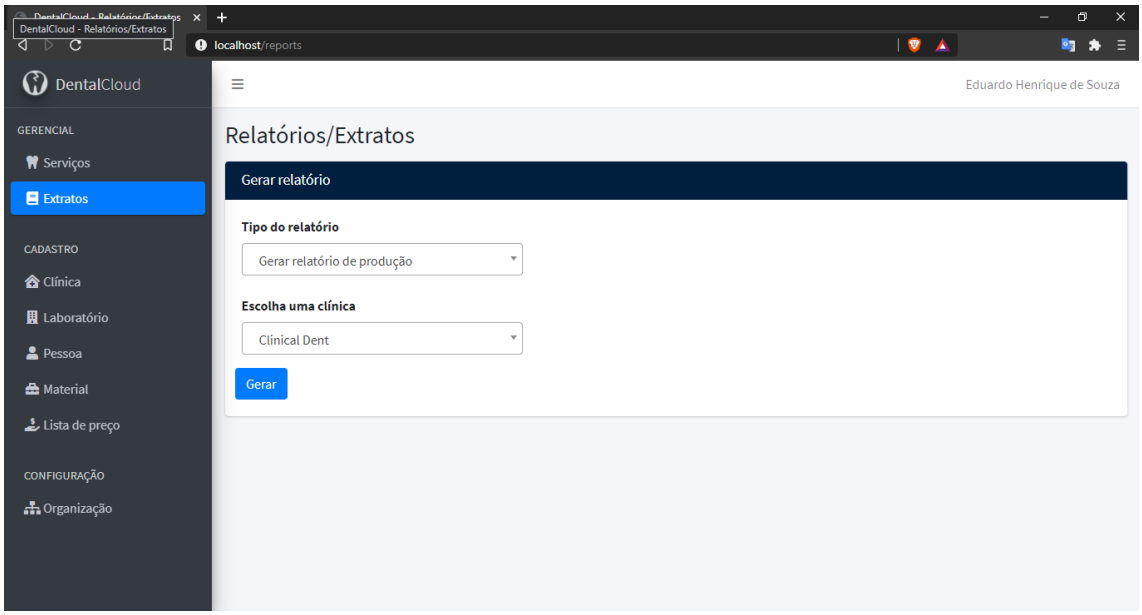
Figura 35 - Tela de andamento de serviço. Adicionar produto produzido.



Fonte: Elaborado pelo autor.

Persistindo as informações de cada etapa do serviço realizado, é possível gerar relatórios dos produtos entregues ao cliente. Portanto, o sistema proposto tem o recurso de impressão de relatórios, que foi julgado como prioritário somente o relatório de trabalhos entregues. Na figura 36 e 37, são apresentadas as telas referentes à extração de um relatório no *DentalCloud*.

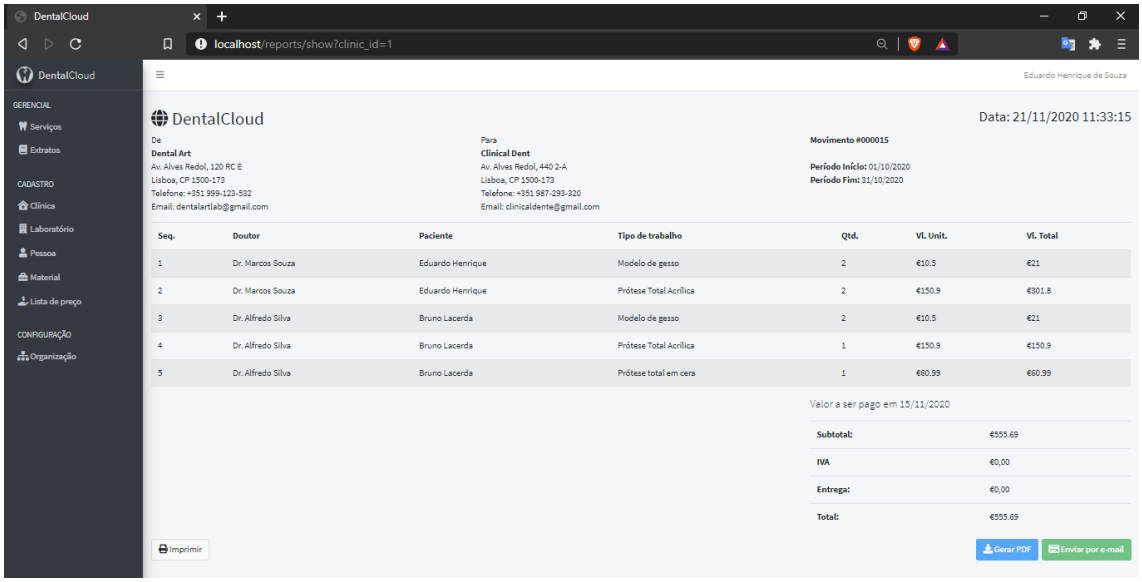
Figura 36 - Tela do menu de relatórios.



Fonte: Elaborado pelo autor.

Após selecionar o tipo de relatório e os filtros necessários, que serão dinâmicos para cada extração, é possível obter a página de visualização que permite imprimir. O *layout* do relatório, é desenvolvido em *HTML* que com o auxílio do *Blade* para *PHP,* torna possível criar qualquer tipo de relatório de dados armazenados no sistema. Na figura 37, observa-se um modelo de relatório de entregas de serviço para determinado cliente.

Figura 37 - Tela de impressão de relatório.



Fonte: Elaborado pelo autor.

Com estes recursos desenvolvidos no sistema, foi possível que a organização começasse seu trabalho utilizando o *DentalCloud*. Foram sugeridas algumas modificações e liberações de novas ferramentas com base no levantamento de requisitos, portanto o sistema entrará no ciclo de melhoria contínua, até que atenda completamente a empresa em questão e até mesmo podendo se tornar uma plataforma global para esta área de negócio.

6 CONCLUSÃO

Após concluir que a ideia de um sistema para laboratório foi válida na prática, será possível trabalhar nos demais requisitos levantados como necessidade do negócio. Os testes realizados dentro do laboratório, com dados de casos de serviços reais, porém censurando os nomes de pacientes e clientes, foram realizados pelo usuário final da ferramenta. Algumas sugestões foram levantadas durante o uso, como por exemplo, colocar máscaras em alguns campos, criar uma busca de endereço mais centralizada, com a possibilidade de apresentar um mapa, ao cadastrar item na lista de preço é interessante ter um botão que redireciona para o cadastro de material, evitando alguns cliques desnecessários que podem impactar no uso geral, entre outros relacionados que não impactam no uso em geral. Enfim, são pontos de melhorias previstos analisando que o objetivo de entrega é um produto mínimo viável, mas foi possível concluir que as expectativas foram atingidas.

A documentação do sistema é de extrema importância para referenciar teoricamente, seja na questão didática para contextualizar outros integrantes, mas também serve como estrutura da arquitetura do sistema que foi desenvolvido, para melhorias no futuro. Durante o desenvolvimento deste projeto, foi possível concluir que um trabalho em equipe será muito bem vindo para trabalhar nos próximos recursos e melhorias levantadas durante os testes finais.

Para os testes finais, foram enfrentadas diversas dificuldades, pelo fato de o usuário final do sistema viver em Portugal, dificultando o contato presencial. Porém, foi possível ensinar, via ferramenta de comunicação por vídeo, e, graças ao compartilhamento de tela, foi demonstrada toda a utilização do sistema, que foi gravada pelo usuário para futuras consultas. Além dessa demonstração, foi exemplificado como realizar o inventário de materiais utilizados, fazendo com que o usuário percebesse de forma clara a simplicidade de utilização do sistema.

Com um objetivo inicial de criar uma ferramenta global de gerenciamento de laboratórios de prótese, e uma oportunidade de validar a ideia em empresas parceiras, garantiu-se a eficiência de poder trabalhar os recursos e melhorias futuras de forma modular, permitindo que regras de negócios mais específicas possam ser

implementadas pela equipe de desenvolvimento e customizações, mas sempre mantendo sua base da maneira como o sistema foi criado inicialmente.

REFERÊNCIAS

ADMINLTE. **AdminLTE Bootstrap Admin Dashboard Template**. Disponível em: https://adminlte.io/. Acesso em: 29 de set. 2020.

BAPTISTELLA, A. J. **Abordando a arquitetura MVC, e Design Patterns: Observer, Composite, Strategy**. Set. 2009. Disponível em: http://www.linhadecodigo.com.br/artigo/2367/abordando-a-arquitetura-mvc-e-design-patterns-observer-composite-strategy.aspx. Acesso em: 23 de set. 2020.

BEOCK, et al. **Protocolo HTTP.** Disponível em: https://img.vivaolinux.com.br/imagens/artigos/comunidade/Protocolo%20HTTP.pdf. Acesso em: 21 de set. 2020.

BOOTSTRAP. Disponível em: https://getbootstrap.com.br/. Acesso em: 29 de set. 2020.

DOCKER. Disponível em: https://docs.docker.com/get-started/. Acesso em: 28 de set. 2020.

DOCKINS, K**. Design Patterns in PHP and Laravel**. Apres, 2017. Disponível em: file:///D:/Users/Eduardo%20Souza/Desktop/FATEC/Jogos%20-%20TCC2/Referencia/Design%20Patterns%20in%20PHP%20and%20Laravel%20(%20PDFDrive%20).pdf. Acesso em: 25 de set. 2020.

GAMMA E. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Bookman, Porto Alegre, 2007. Disponível em: file:///D:/Users/Eduardo%20Souza/Desktop/FATEC/Jogos%20-%20TCC2/Referencia/Padroes_de_Projetos_Solucoes_Reutilizave.pdf. Acesso em: 22 de set. 2020.

HEUSER, C. A. **Projeto de Banco de Dados**. Edição 4, 1998. Editora Sagra Luzzatto. Acesso em: 22 de set. 2020.

LARAVEL. Disponível em: https://laravel.com/. Acesso em: 29 de set. 2020.

LOBO, E. J. R. **Guia Prático de Engenharia de Software: Desenvolva softwares profissionais com o uso do UML e "*best pratices*" de gestão**. Editora Digerati Books, 2009. Acesso em: 14 de out. 2020.

MACORATTI, J. C. **UML - Diagrama de Classes e objetos.** Disponível em: http://www.macoratti.net/net_uml1.htm#:~:text=A%20melhor%20maneira%20de%20conceituar,de%20um%20exerc%C3%ADcio%20de%20abstra%C3%A7%C3%A3o.). Acesso em: 26 de nov. 2020.

MERKEL, D. **Docker: Lightweight Linux *Containers* for Consistent Development and Deployment**. Disponível em: https://www.seltzer.com/margo/teaching/CS508.19/papers/merkel14.pdf. Acesso em: 28 de set. 2020.

MOREIRA**,** L. B. **Análise e Projeto Orientado a Objeto Usando UML**. Disponível em: http://computacao.unitri.edu.br/downloads/monografia/49351143168297.pdf. Acesso em: 13 de out. 2020.

PIRES, E. **SOLID – Single Responsibility Principle – SRP**. 2013. Disponível em: https://www.eduardopires.net.br/2013/05/single-responsibility-principle-srp/. Acesso em: 24 de set. 2020.

RAJEEVAN, A. **Fundamentals of REST API Design**. Disponível em: https://medium.com/@arunrajeevan/fundamentals-of-rest-api-design-d9c425c1b0f6. Acesso em: 26 de set. 2020.

SALMA. **UML Class Diagrams Tutorial, Step by Step**. Disponível em: https://medium.com/@smagid_allThings/uml-class-diagrams-tutorial-step-by-step-520fd83b300b. Acesso em: 14 de nov. 2020.

SILVA, M. S. **Criando sites com HTML: Sites de alta qualidade com HTML e CSS**. Novatec Editora LTDA, 2008. Primeira edição. Acesso em: 28 de set. 2020.

SANTOS, O. G. F; TIOSSO, F; PETRUCELLI, E. E. **Demonstração Dos Benefícios Do Minimum Viable Product Na Criação De Um Novo Aplicativo Móvel**. Revista Interface Tecnológica, [S. l.], v. 16, n. 1, p. 124-135, 2019. Disponível em: https://revista.fatectq.edu.br/index.php/interfacetecnologica/article/view/567. Acesso em: 31 de out. 2020.

SOMMERVILLE, I. **Engenharia de Software.** Editora Pearson Education, 2011. Edição 9. Disponível em: file:///D:/Users/Eduardo%20Souza/Desktop/FATEC/Jogos%20-%20TCC2/Referencia/engenhariaSoftwareSommerville.pdf. Acesso em: 21 de set. 2020.

TANEMBAUM, A. S. **Redes de Computadores**. Editora Campus, Rio de Janeiro, 2003. Edição 7. Acesso em: 15 de set. 2020.

VIEIRA, R. **UML – Diagrama de Casos de Uso.** Disponível em: https://medium.com/operacionalti/uml-diagrama-de-casos-de-uso-29f4358ce4d5. Acesso em: 14 de out. 2020.