UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DENIS AKIRA ISE WASHIO

Sistema de Controle de Equipamentos da FAAC webTV

BAURU
Novembro/2019

DENIS AKIRA ISE WASHIO

Sistema de Controle de Equipamentos da FAAC webTV

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Aparecido Nilceu Marana

BAURU
Novembro/2019

Denis Akira Ise Washio Sistema de Controle de Equipamentos da FAAC webTV/ Denis Akira Ise Washio. – Bauru, Novembro/2019- 46 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Dr. Aparecido Nilceu Marana
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Ciência da Computação, Novembro/2019.
1. Sistema de Informação 2. Aplicação web 3. Engenharia de Software 4. Banco de Dados

Denis Akira Ise Washio

Sistema de Controle de Equipamentos da FAAC webTV

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. Aparecido Nilceu Marana
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Professor Dr. Simone Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Professor Dr. Joao Pedro Albino
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Bauru, _____ de _____ de _____.

Agradecimentos

Começo agradecendo minha família, que me apoiou nos momentos de dificuldade e me ajudou a chegar até aqui. Também me deu liberdade para encontrar meu caminho.

Dedico este trabalho a todas as amizades que estiveram presentes sempre que eu precisava, que me ajudaram muito e estavam lá nos momentos bons e ruins. Dos mais antigos aos mais recentes, do curso ou de fora, todos aqueles que de alguma forma participaram.

Agradeço também à Faculdade de Ciência, a UNESP e o curso de Bacharelado de Ciência da Computação, que me proporcionou experiências incríveis num ambiente que fica para sempre comigo. Em especial, agradeço as extensões que tive a oportunidade de participar, a FAAC webTV e a Jr.COM. Foi um aprendizado gigante que só pôde ser completo por causa das pessoas empenhadas e dedicadas que faziam a engrenagem girar.

Também agradeço aos funcionários e professores, sobretudo meu orientador, pela pluralidade de opiniões que enriqueceram as conversas.

Resumo

Com a crescente importância da informação, há também um crescente benefício em ter um sistema de software para a gestão de organizações ou empresas. Sendo assim, este projeto tem o objetivo de desenvolver um sistema de gestão de equipamentos para uso dos membros da FAAC webTV. Para isso, são usados diversos conceitos que abrangem desde a natureza de processos de desenvolvimento até quesitos mais técnicos, envolvendo tópicos como bancos de dados ou *APIs*. A importância do projeto também é observada no descobrimento e utilização de novas tecnologias, *frameworks* de desenvolvimento e fluxos de trabalho de forma prática, também resolvendo um problema real enfrentado por uma organização da Faculdade de Arquitetura, Artes e Comunicação da UNESP Bauru. A conclusão é o sucesso na criação de um sistema de controle de equipamentos e seu lançamento em produção.
Palavras-chave: Engenharia de Software. Banco de dados. Sistema web.

Abstract

With the growing importance of information, there is also a growing benefit of having a software system to manage organizations or companies. This project has the purpose of developing a system to manage equipments used by FAAC webTV members. To accomplish that, numerous concepts are used, ranging from the nature of development processes to more technical elements, involving topics such as databases or APIs. The importance of this project is also observed in the discovery and use of new technologies, software frameworks and workflows in a practical way, also solving a real world problem faced by an organization based in FAAC UNESP Bauru. The conclusion is the success in the creation of a system of equipment management and its launch in production.

Keywords: Software engineering. Database. Web system.

Lista de códigos

Lista de figuras

Lista de abreviaturas e siglas

UNESP	Universidade Estadual Paulista "Júlio de Mesquita Filho"
FAAC	Faculdade de Arquitetura, Artes e Comunicação
QR	*Quick Response*
MER	Modelo Entidade Relacionamento
MVC	*Model-View-Controller*
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
CGI	*Common Gateway Interface*
PHP	*PHP: Hypertext Preprocessor*
CLI	*Command Line Interface*
API	*Appliaction Programming Interface*
URL	*Uniform Resource Locator*
HTTP	*Hypertext Transfer Protocol*
FTP	*File Transfer Protocol*

Sumário

1 Introdução

Com a evolução da tecnologia, a informação se torna o ativo mais importante no cenário mundial. Dessa forma, *softwares* são uma tecnologia muito importante, sobretudo devido à sua natureza dupla ao processar e também distribuir informações. (PRESSMAN, 2011)

Ainda segundo Pressman (PRESSMAN, 2011), *software* é um conjunto composto por:

- Instruções que, quando executadas, fornecem características, funções e desempenho desejados;
- Estruturas de dados que possibilitam aos programas manipular informações adequadamente;
- Informação descritiva, tanto na forma impressa quanto na virtual, descrevendo a operação e o uso do programa.

Para dar mais clareza à definição, seguem alguns exemplos de *software*: o *Microsoft Word* é um exemplo de software complexo cujo propósito é a criação de arquivos de textos, municiado de ferramentas que auxiliam no processo de edição do texto. A parte visual, o funcionamento das ferramentas, a criação do arquivo são todos componentes feitos através de um processo de desenvolvimento de software que é interpretado pelo computador.

O *Gmail* é outro exemplo de software cujo propósito é ler e manipular informações por meio de mensagens enviadas e recebidas pelo endereço de *e-mail* único de cada usuário. Nesse caso, se trata de um software que é interpretado pelo navegador e fornecido pela Internet.

Com os exemplos, fica claro que existem tipos abrangentes de *software*, que de forma geral, possuem funcionalidades computacionais distribuídas através de algum meio, como o computador pessoal ou Internet.

Nesse contexto, a comunidade da área tenta desenvolver métodos mais fáceis, ágeis e baratos para desenvolver e manter programas de computador. Dada a natureza complexa da construção e modificações que tendem a acontecer ao longo da vida útil do *software*, metodologias são necessárias para produzir um produto de alta qualidade.

Neste trabalho foi desenvolvido um sistema para a FAAC webTV, da Faculdade de Arquitetura, Artes e Comunicação, UNESP, campus de Bauru, com o uso de ferramentas *open-source* e com o intuito de aprimorar a forma com que a informação é manuseada por seus membros.

2 Problema

A FAAC webTV é vinculada à FAAC e tem como propósito realizar transmissões audiovisuais para televisão ou Internet, ao vivo. Visa oferecer experiência de trabalho em uma produção para os alunos dos cursos de Rádio e TV, Relações Públicas e Jornalismo. Para a produção acontecer são necessárias câmeras, microfones, cabos, além de outros equipamentos de transmissão, e uma locação onde acontece o evento. Portanto é importante existir um controle da entrada e saída desses equipamentos da sede para a locação do evento.

Sendo assim, entende-se a importância de um sistema para otimizar a organização dos equipamentos. Dado esse contexto, o Sistema FAAC webTV é um projeto elaborado para otimizar processos relacionados à gestão de equipamentos e eventos produzidos.

O problema então se resume a criar uma plataforma facilmente acessível, possibilitando que todos os membros da FAAC webTV possam usá-lo, que acessa um sistema de gestão. Esse sistema deve conter informações consistentes sobre todos os equipamentos da FAAC webTV, o evento onde ele está alocado e capacidade de alterar os equipamentos e eventos registrados. Também é necessário que as informações de equipamentos sejam facilmente acessíveis durante a montagem ou desmontagem das produções, pois é nesses momentos em que há mudança no estado dos dados. Paralelamente, eventos devem poder ser cadastrados em qualquer momento.

Como o sistema carrega informações restritas aos membros da FAAC webTV, e dentre os próprios membros existem graus diferentes de privilégio referente ao cargo, deve haver também uma forma de autenticação e autorização sobre os módulos, evitando acesso não autorizado e restringindo o que cada membro pode acessar.

Com relação ao desenvolvimento, pela complexidade do sistema, é necessária a elaboração de processos para finalizar o projeto dentro cronograma definido.

3 Objetivos

O objetivo deste trabalho é desenvolver um sistema para a FAAC webTV auxiliado por ferramentas que possibilitam a criação de um software com a complexidade desejada e metodologias baseadas na engenharia de software.

Depois, elaborar e utilizar processos adaptados para o contexto em questão para desenvolver o software, lançar o sistema em um ambiente de produção, ou seja, disponível para uso da FAAC webTV.

4 Justificativa

A justificativa do projeto se baseia na possibilidade de aplicar conceitos cruciais da Ciência da Computação, sobretudo relacionados a Engenharia de Software e a Banco de Dados, somada da oportunidade de contribuir de forma relevante com a comunidade. Sendo assim, o teor multidisciplinar permite a contribuição com a extensão FAAC webTV, que por sua vez, pode analisar as aplicação do sistema na prática.

Como aprimoramento de conhecimento, também se destaca a realização de um trabalho com ferramentas atuais e com grande relevância na comunidade de desenvolvimento. Dentre as usadas no projeto, destacam-se o *Laravel*, o *Git* e o *Docker*. As ferramentas servem para complementar os conceitos e tornar processos mais eficientes, garantindo a implementação das ideias em tempo e esforço constantes.

5 Fundamentação teórica

Esta seção é destinada a relatar sobre conceitos da Ciência da Computação e como eles se relacionam com ferramentas usadas no projeto.

5.1 Engenharia de Software

A Engenharia de Software é a área da Ciência da Computação que descreve o processo de desenvolvimento de software, ou seja, a metodologia para as atividades, ações e tarefas necessárias para a produção de um software de alta qualidade. Além disso, também engloba conhecimento sobre tecnologias, métodos técnicos e ferramentas automatizadas. Como se trata de uma forma de conhecimento, é direcionada para pessoas que devem adaptar os processos ao contexto para a produção de um software maduro. (PRESSMAN, 2011)

5.1.1 Engenharia de requisitos

Engenharia de requisitos é o processo na engenharia de software que se inicia durante a comunicação e continua na modelagem. Ela deve ser adaptada às necessidades do processo, do projeto, do produto e das pessoas que estão realizando o trabalho. (PRESSMAN, 2011)

Sendo assim, a engenharia de requisitos fornece o mecanismo apropriado para entender o que o cliente deseja, analisando as necessidades, avaliando a viabilidade, negociando uma solução viável e especificando a solução sem ambiguidades.

5.1.2 Padrões de Projeto de Software

Na Engenharia de Software, *Design Patterns*, ou Padrões de Projeto, são abstrações de conceitos frequentemente usados e aceitos como boas soluções para resoluções de problemas comuns em projetos de software. Costumam apresentar uma mini arquitetura de como prosseguir quando interagindo com um problema, visto que é boa prática na computação utilizar conceitos já aceitos, sem apresentar diretamente linhas de código. (GAMMA et al., 1993)

Padrões de Projeto são, portanto, agnósticos a linguagem, ou seja, podem ser implementados independente da linguagem de programação escolhida, sendo mais relacionados à como objetos interagem, como devem ser instanciados e como devem ser estruturados de forma a garantir um padrão de código mais manutenível, reusável e mais fácil de ser compartilhado, visto que a comunicação também é beneficiada com seu uso. Isso acontece pois soluções podem ser melhor documentadas em cima de um conhecimento coletivo já estabelecido.

Os padrões de projeto possuem um papel chave na construção de *frameworks* por comunicar conceitos dentro da arquitetura do *framework* por meio de suas definições. Dessa forma, é mais fácil compreender o funcionamento do *framework* de forma semântica pelo padrões que ele utiliza, sem necessariamente conhecer detalhes de sua implementação.

5.1.3 *Framework*

Framework, em desenvolvimento de software, é uma estrutura de suporte que fornece blocos com funcionalidades genéricas prontas ou semi prontas para o desenvolvimento de *software*, definindo também arquitetura geral, ou seja, como esses blocos são compostos e como se comunicam entre si. (PREE, 1994)

A arquitetura dos *frameworks* consistem de partes sólidas e partes maleáveis do código. As seções maleáveis podem ser alteradas pelo usuário para se adequar à aplicação; as partes sólidas do código permanecem fixas, pois são responsáveis pelo funcionamento do *framework* em si.

Possuem características que os distinguem de bibliotecas:

- Inversão de controle: a definição do fluxo de controle é definido pelo *framework*, não pelo desenvolvedor.
- Extensibilidade: O desenvolvedor pode sobrescrever funcionalidades base do *framework* para se adequar à regra de negócio específica do projeto
- Código não modificável: No *framework* também há uma parcela de código que não é alterável pelo usuário, sendo a parte base para seu funcionamento.

5.1.4 Arquitetura de software

Arquitetura de software segue um conceito semelhante aos Padrões de Projetos de Software, em que uma solução bem definida e historicamente bem sucedida é aplicada para solucionar um conjunto definido de problemas na construção de um software, porém, a arquitetura se diferencia no escopo, pois se trata de uma visão mais macroscópica do projeto em comparação com os Padrões.

Nesse caso, há uma análise mais voltada para como componentes do software vão se comunicar, como a construção geral do projeto vai acontecer e até como os Padrões de Projeto se encaixam dentro da ideia do desenvolvimento.

5.1.5 Arquitetura MVC (*Model-View-Controller*)

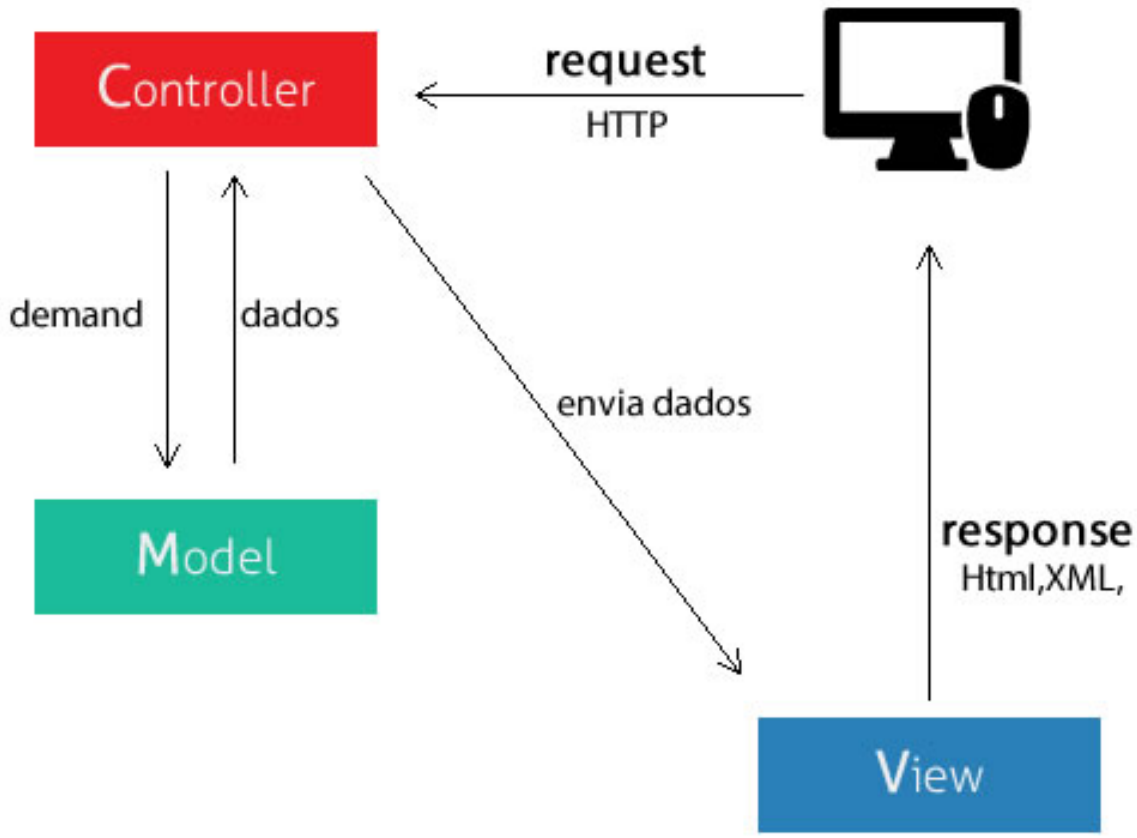
A arquitetura MVC oferece uma separação de conceitos de forma a separar a camada de interação com os dados e lógica de negócio, a camada de *Model*, da camada de interface

gráfica, a *View*. O *Controller* é responsável por lidar com eventos e preparar os dados de resposta. (POP; ALTAR, 2014)

A interação de usuário costuma ter um ciclo natural: o usuário realiza uma ação, que resulta numa alteração de dados e o retorno da informação por meio da interface gráfica. Isso cria um ciclo de requisições e respostas. (POP; ALTAR, 2014)

Na Figura 1, observa-se o fluxo de informações em sistemas arquitetura MVC. As requisições feitas são tratadas pelo *Controller*, o *Model* computa e retorna os dados e o *Controller* retorna esses dados para a *View* atualizar a resposta para o usuário final.

Figura 1 – Representação do modelo MVC.



Fonte: Página Tabeless. Link: https://tableless.com.br/mvc-afinal-e-o-que/

5.2 Banco de dados

Banco de dados é uma coleção organizada de informações estruturadas, normalmente armazenadas eletronicamente em um sistema de computador. Um banco de dados é normalmente acompanhado de um sistema de gerenciamento de banco de dados (SGBD). Juntos, os dados e o SGBD compõe o que é chamado de sistema de banco de dados. (ORACLE, 2019a)

Os dados são comumente modelados num formato de linhas e colunas em séries de tabelas para tornar o processamento e consulta de dados mais eficiente. Podem ser facilmente acessados, gerenciados, controlados e modificados, normalmente por meio da linguagem de consulta estruturada, o SQL. (ORACLE, 2019b)

Como área da Ciência da Computação, trata também abstrações mais amplas, com o objetivo de otimizar a estrutura das tabelas, evitando redundâncias ou dados ausentes.

Uma dessas formas de modelagem é conhecida como MER, que representa abstrações que visam descrever os elementos principais do sistema e a relação entre essas entidades.

6 Ferramentas

Esta seção descreve a funcionalidade e detalhes da implementação de ferramentas usadas no projeto.

6.1 PHP

O PHP (um acrônimo recursivo para *PHP: Hypertext Preprocessor*) é uma linguagem de *script open source* de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento *web* e que pode ser embutida dentro do HTML. O PHP é focado principalmente nos *scripts* do lado do servidor, portanto, é possível fazer qualquer coisa que outro programa *CGI* pode fazer, como coletar dados de formulários, gerar páginas com conteúdo dinâmico ou enviar e receber *cookies*. (PHP, 2019)

6.2 CLI

Command Line Interface é uma interface para execução de programas e gerenciamento de arquivos através de textos, usando o terminal de sistemas *Unix*, também conhecido como *Command Prompt* em sistemas *Windows*.

A utilização é comum no desenvolvimento de software pela simplicidade em contraste com sistemas que apresentam uma interface gráfica completa. (CODECADEMY, 2019)

6.3 Gerenciador de Pacotes

Gerenciador de pacotes é um conjunto de programas que tem como objetivo automatizar o processo de instalar, atualizar, configurar e remover programas instalados no Sistema Operacional. São responsáveis por também instalarem ou sugerirem pacotes que são requerimentos de outros pacotes, chamadas dependências, ou pacotes que usam dependências com versões conflitantes. (DEBIAN, 2017)

Também existem Gerenciadores de Pacotes de Aplicações, mais relevantes no trabalho em questão, por se tratarem de gerenciadores de pacotes a nível de aplicação, gerenciando bibliotecas que são instaladas para uso mais restrito. Nesse caso, os pacotes tendem a ser instalados a nível de diretório da aplicação, evitando conflitos com bibliotecas globais do sistema operacional.

6.4 Composer

Composer é um gerenciador de dependências voltado para o PHP, normalmente usado a nível de aplicação, embora possa ser usado a nível global de sistema também. O Laravel e outros *frameworks* PHP utilizam o *Composer* para gerenciar suas dependências e pacotes externos. (COMPOSER, 2019)

6.5 Laravel

Laravel é um *framework* de desenvolvimento para aplicações web com uma arquitetura MVC escrito na linguagem PHP. Teve sua primeira versão lançada em junho de 2011 e está atualmente na versão 6.0. O *Laravel* possui funcionalidades que aceleram o desenvolvimento, com uso de padrões de projeto, que além de torná-lo mais robusto devido a solidez dos padrões, também facilitam o entendimento de suas funcionalidades, que são explicadas pelos nomes dos padrões. (LARAVEL, 2019) Alguns exemplos são:

- Injeção de Dependência: aplica funcionalidades em uma classe através de parâmetros de função. Esse padrão permite a separação de funções dentro da aplicação, pois funcionalidades podem ser injetadas quando necessário partindo da vasta biblioteca de APIs do *framework*;
- *Facade*: cria uma estrutura intermediária entre o desenvolvimento e as aplicações internas do *framework*, evitando que código sólido não entre em contato com partes maleáveis;
- *Active Record*: um objeto que representa uma entidade do banco de dados, carregando seus dados e comportamentos. Assim, é possível usá-lo como porta de acesso à determinados dados do banco, com as possíveis operações como *update* ou *delete*, tornando mais abstrata e direta a interação com os dados. No *Laravel*, é implementada através do *Eloquent*, a classe base para criação dos modelos da aplicação. Sua composição apresenta diversos métodos para acesso e edição de dados no banco.

Além desses, o *Laravel* também usa outros padrões, que contribuem com sua natureza reutilizável e escalável.

Dentro da estrutura de projeto do *Laravel*, destacam-se suas funcionalidades base:

- *Artisan*: Executado por meio do arquivo *artisan*, se trata de um CLI para executar comandos referentes à aplicação em *Laravel*;
- *Controllers*: Usando arquivos presentes em *app/Http/Controllers*, o *controller* representa a camada dos controladores da arquitetura e estendem da classe base fornecida pelo *Laravel*;

- *View*: Usando arquivos presentes em *resources/views*, a *view* representa a camada de visão da arquitetura, e através do *Blade Engine* integrado nativamente no Laravel, arquivos HTML podem ser divididos em múltiplos arquivos de forma a desacoplar a aplicação e reutilizar funcionalidades em múltiplas páginas;

- *Model*: Usando arquivos presentes em *app/Models*, o *model* representa a camada de modelo da arquitetura, e a convenção é que cada classe represente uma tabela do banco de dados, funcionando como *Active Record* e tendo nos métodos base da classe as operações de leitura, inserção, edição e deleção. Nessa camada também se convenciona estruturar as regras de negócio do projeto. É implementada estendendo a classe *Model* do *Eloquent*, como já mencionado previamente;

- Rotas: Usando o arquivo *routes/web.php*, é possível cadastrar as rotas de forma que cada link resulte na ativação de um método através de um controller, que por sua vez retorna uma instância de uma view;

- *Migration*: Usando arquivos presentes em *database/migrations*, é possível usar métodos semânticos para criação e edição de tabelas no banco de dados. As instruções inseridas nessas classes são convertidas em *queries* de SQL e efetuam operações no banco. São ativadas através de comandos usando o *Artisan*;

- *Trait*: São funcionalidades que podem ser inseridas em classes quando necessário.

6.6 *Git*

Git é um sistema de versionamento de código *Open Source* feito para lidar com base de códigos grandes e pequenas. O sistema possui uma variedade de comandos que auxiliam na união de bases de códigos de usuários diferentes. (GIT, 2019)

A composição básica de um projeto *Git* consiste em uma pasta com uma representação do histórico de mudanças do projeto. Os objetos são adicionados na história local do sistema por meio de *commits*. É possível criar ramificações da pasta de trabalho que são chamadas *branches* para trabalhar em mudanças e não afetar outros ramos, permitindo a manutenção de ramos funcionais e trabalho paralelo em diferentes *branches*.

As mudanças registradas nos *commits* podem ser adicionadas em um sistema online, tal como o *Github* ou *Gitlab*.

Essa separação permite o desenvolvimento paralelo de diferentes funcionalidades de software que podem ser juntadas ao código base.

6.7 Gitflow

Gitflow é um fluxo de trabalho definido com o uso do *Git* na incrementação de funcionalidades em um software. Através da funcionalidade das *branches*, cada ramo é separado em seu objetivo da seguinte forma:

- Uma *branch* única chamada *Develop*, que contém o software em estado estável e recebe novas iterações a cada *sprint*
- Uma *branch* única chamada *Master*, que contém o software em estado de produção, só é alterado em novos lançamentos e atualizações de versão
- Para funcionalidades novas, cria-se uma *branch feature/id-da-funcionalidade*
- Para conserto de *bugs*, cria-se uma *branch bugfix/id-do-bug*
- Para conserto de *bugs* direto na Develop ou Master, cria-se uma *branch hotfix/id-do-bug*

As *branches* são juntadas ao código base com *Pull requests* feitas em plataformas de desenvolvimento, tais como o *Github*, através das *Merge Requests*. O *Merge Request* compara as diferenças de código de forma visual e possibilita a alocação de outro membro do projeto para revisar as alterações. O processo reduz erros e mantém o versionamento de forma organizada.

6.8 API

API é um conjunto de rotinas e padrões de programação para acesso a um software ou plataforma na web. API significa *Application Programming Interface*, que em tradução livre significa Interface de programação de aplicativos.

APIs costumam ser usadas quando desenvolvedores de software querem disponibilizar suas aplicações para uso conveniente por outros desenvolvedores.

Através de APIs, aplicativos podem se comunicar sem saber detalhes de implementação ou interação do usuário. Isso permite baixo acoplamento da aplicação, visto que as funcionalidades não são dependentes entre si. (CANALTECH, 2019)

6.9 Mysql

MySQL é um software voltado para uso por vários usuários que apresenta uma estrutura robusta e funciona como servidor de banco de dados em SQL. É preparado para utilização em projetos com alta carga de dados e lançamento em servidores em produção. (MYSQL, 2019)

Um sistema comum de acesso e gerenciamento dos dados do banco é o *phpMyAdmin*, sistema que permite a visualização de dados e execução de comandos para analisar e manipular a informação contida nas tabelas. (PHPMYADMIN, 2019)

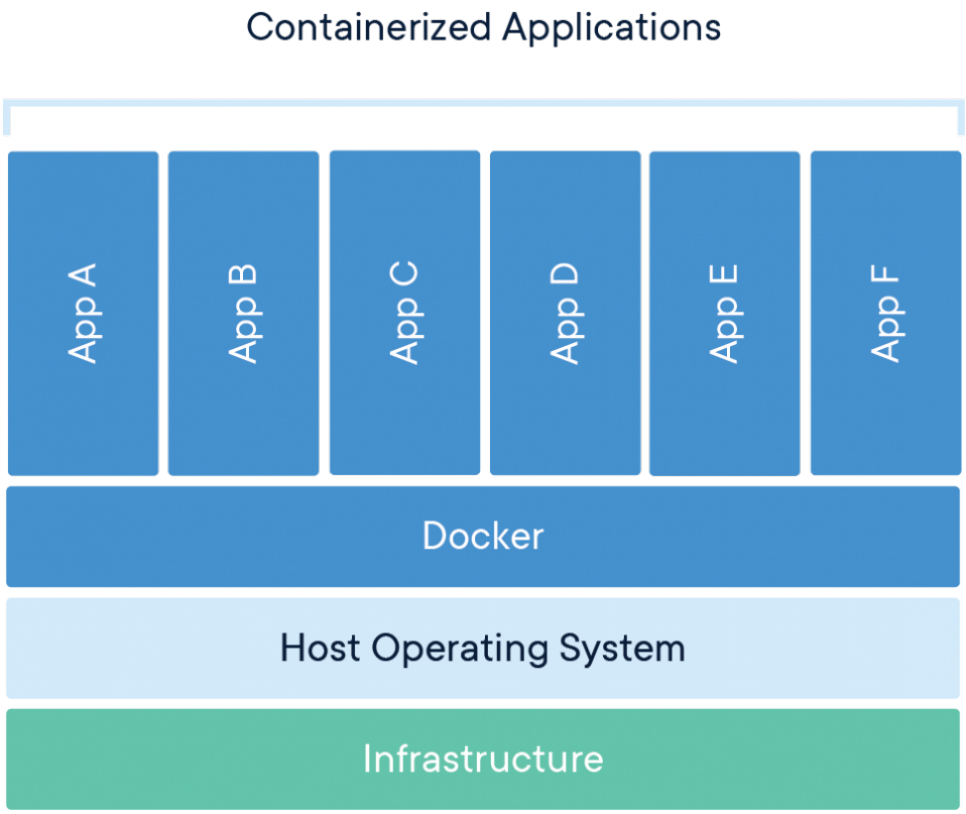
6.10 Docker

Docker é um software *Open Source* que auxilia na criação de containers para gerenciar aplicações de forma isolada.

Um container é uma unidade de software que oferece um ambiente para desenvolvimento e lançamento em produção de uma aplicação. Ele engloba todas as dependências, códigos, bibliotecas e funcionalidades necessárias para o funcionamento da aplicação. (DOCKER, 2019)

Como é possível observar na Figura 2, os containers se colocam numa camada acima do Sistema Operacional, se aproveitando do poder computacional do computador e criando uma camada de separação entre o ambiente local e o do container. Com isso é possível evitar conflitos de variáveis de ambiente ou versionamento de aplicação, pois se tornam padronizadas pelo *Docker*.

Figura 2 – Containers



Fonte: Docker.com

O *Docker* possui uma ferramenta para utilização de diversos containers de forma concorrente, inclusive podendo ter comunicação entre si, no caso de aplicações co-dependentes.

7 Desenvolvimento

A criação de um software costuma se iniciar a partir da definição de como ele resolverá um problema existente. No caso deste trabalho, se refere à um problema relacionado a um cliente, a FAAC webTV. Portanto, o desenvolvimento começa a partir da engenharia de requisitos.

7.1 Pré-projeto

7.1.1 Engenharia de requisitos e Modelagem

O processo inicial para entender o escopo do projeto se deu através de reuniões e comunicação direta com o cliente para entender como o *software* poderia se adaptar aos processos da FAAC webTV. A partir das conversas, foi possível iniciar uma modelagem com os conceitos chaves do sistema e elaborar um fluxo de trabalho que permitisse a incrementação gradual de funcionalidades.

Sendo assim, como já definido ao longo da monografia, o projeto consiste de um sistema com acesso restrito aos membros e autorização baseada em funções, com acesso por meio da Internet. O sistema deve ser acompanhado de um banco de dados para garantir a consistência do estado de cada entidade que compõe o sistema, e deve também ter um método de rápido acesso à página de edição de equipamentos específicos.

Com relação ao último tópico, a solução proposta foi o uso de um *QR Code*, que deve ser gerado a partir de cada página única de um equipamento, e colado junto ao equipamento. Dessa forma, se um equipamento é utilizado em uma transmissão, seu *QR Code* é escaneado, e a leitura resulta em uma URL que direciona para a página do equipamento, possibilitando a alteração de seu estado, seja num evento ou na sede da FAAC webTV.

Com relação à restrição de *login* para os usuários da FAAC webTV, foi possível aproveitar o contexto para elaborar uma solução. Pelo fato dos membro possuírem um *e-mail* com domínio *@faacwebtv.com.br* oferecido pelo *Google*, o sistema pode se aproveitar desse fato e restringir o acesso somente a *e-mails* deste domínio. Dessa forma, usa-se o *e-mail* já cadastrado como forma de login, evitando acesso indesejado e autenticando o membro.

Assim, os requisitos ficam definidos da seguinte forma:

- Modelagem do banco de dados;
- Painel administrativo;
- Autenticação restrita aos membros da FAAC webTV;

- Autorização baseada em função do membro dentro da estrutura da FAAC webTV;
- Listagem de equipamentos registrados;
- Capacidade de editar, adicionar e remover equipamentos;
- Listagem de eventos registrados;
- Capacidade de editar, adicionar e remover eventos;
- Integração com API do *Google* para o login;
- Integração com API de *QR Code*;
- Listagem de funções de usuário;
- Capacidade de editar, adicionar e remover funções.

Definidos os requisitos, o próximo passo foi definir o processo de desenvolvimento, de forma que houvesse um progresso constante na integração de novas características do sistema.

Para o projeto, a metodologia escolhida tem inspirações no desenvolvimento ágil, porém com modificações para se encaixar no contexto. No caso do projeto, o essencial foi a definição de *sprints*, onde há uma incrementação gradual que melhora o fluxo de desenvolvimento dividindo o trabalho em diversas partes constantes e permitindo a avaliação do progresso por parte do cliente. Esses aprimoramentos são integradas ao projeto no *Github*, usando o fluxo do *Gitflow* com o uso de *Pull Requests* feitas no *Github*. Isso possibilita análise do código que será inserido, garantindo o padrão e minimizando a quantidade de erros.

7.1.2 Modelagem de dados

Com os modelos de processo definidos, iniciou-se a próxima etapa da produção do software, a modelagem de dados. Com ela, foi possível criar uma estrutura sucinta no banco, contendo os campos essenciais para as funcionalidades.

Considerando a existência de relações entre tabelas e a natureza imutável das colunas, a escolha do banco de dados do projeto foi por um banco de dados relacional, o *MySQL*. Sendo assim, as entidades foram definidas no formato tabela: colunas da seguinte forma:

- Usuário: id, email, senha, funcaoId;
- Equipamento: id, nome, eventoId;
- Evento: id, nome, descricao;
- Função: id, nome;

- Permissão: id, chave;
- Permissao_função: permissaoId, funcaoId.

Em que as colunas eventoId, funcaoId e usuarioId representam chaves estrangeiras que referenciam respectivamente o id das tabelas Evento, Função e Usuário. A coluna chave na tabela Permissão é a chave que será usada na implementação do software para verificar a autorização da função com relação ao conteúdo acessado.

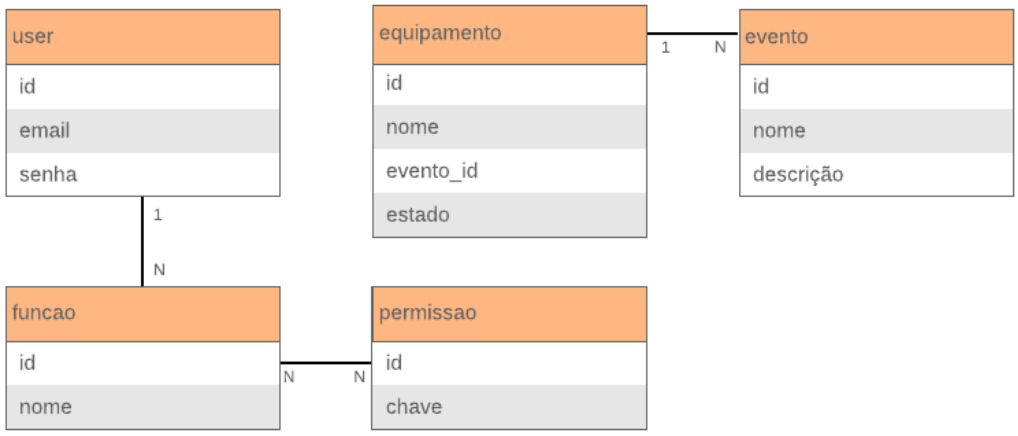
O relacionamento entre Equipamentos e Eventos é de cardinalidade 1 para N, dado que um equipamento pode estar associado a somente um evento e um evento pode ter vários equipamentos.

Usuário tem um relacionamento de cardinalidade N para 1 com Função, visto que usuários podem ter a mesma função.

Por fim, Função tem relacionamento de cardinalidade N para N com Permissão, implementada na tabela Permissão_função, pois cada função possui diversas permissões e cada permissão é referenciada por diversas funções.

O MER do projeto descrevendo os relacionamentos e entidades descritas pode ser visto na Figura 3.

Figura 3 – MER do projeto.



Fonte: Elaborado pelo autor.

Após a conclusão desta etapa, seguiu-se para a definição de tecnologias.

7.1.3 Arquitetura, tecnologias e ferramentas

Com o escopo do problema e as metodologias definidas, foi necessário pensar na arquitetura do software e em tecnologias que permitissem a implementação das ideias de forma simples.

Sendo assim, a arquitetura idealizada para o projeto foi a MVC, possibilitando uma separação de conceitos e reutilização de código, visto que algumas características são compartilhadas entre as entidades principais do sistema: tanto equipamentos como eventos necessitam de uma listagem e das operações de criação, edição e exclusão. Dessa forma, tanto a parte de Visão como a de Modelo tiveram elementos semelhantes e isso permitiu a utilização do princípio de não repetição da Engenharia de Software.

Das ferramentas *Open Source* que oferecem esse tipo de característica, o *Laravel* foi escolhido pelos seguintes motivos:

- Arquitetura condizente com a escolhida;
- Documentação detalhada;
- Comunidade ativa;
- Padrões de projeto bem definidos;
- Fácil integração com APIs;
- Disponibilidade do PHP no servidor de produção;
- Métodos semânticos de acesso ao banco de dados.

Dentre esses elementos, destacam-se os métodos semânticos para acesso ao banco de dados, visto que é nesse aspecto que tendem a se encontrar as regras de negócio de um projeto que envolve um banco, pois são operações no banco que alteram o estado da aplicação. Com o *Laravel*, por meio do *Active Record*, as regras ficam armazenadas na classe referente à entidade, ao invés de *queries* em SQL, que tornariam o sistema mais difícil de manter.

Para acelerar o processo de desenvolvimento, também foi utilizado o *Docker* para criar imagens e evitar a necessidade de instalar dependências, sobretudo porque são necessárias extensões do PHP que não são nativamente instaladas para executar o *Laravel*, assim como a instalação do *MySQL* e de um gerenciador de banco relacional.

Para usar o *Docker*, um arquivo chamado *docker-compose.yml* é necessário detalhando informações de preparo e instalação, permitindo a construção dos containers através de um comando no terminal.

Pelo exemplo do Código 1, é possível fazer uma análise do funcionamento do *Docker*.

O atributo *image* define a imagem base da aplicação, e com ela o cliente do *Docker* faz a busca no *Docker Hub*, cria os containers e baixa a aplicação, além de suas dependências. Os outros atributos definem detalhes da implementação no ambiente onde ele é executado, ou seja, o campo *ports* define qual porta será usada no servidor local, o campo *environment* define variáveis de ambiente para uso do *Docker*, *network* define internamente que o container descrito terá uma conexão com outro container. No caso do exemplo, relata a conexão entre os containers *database* e *dbadmin*, ou seja, o *MySQL* e o *phpMyAdmin*.

Código 1 – Parte do *dockerfile* do projeto.

```
database:
  image: mysql:5.7
  command: --default-authentication-plugin=mysql_native_password
  volumes:
    - ./docker/dbdata:/var/lib/mysql
    - ./docker/mysql.cnf:/etc/mysql/conf.d/mysql.cnf
  environment:
    - "MYSQL_DATABASE=webtv"
    - "MYSQL_USER=homestead"
    - "MYSQL_PASSWORD=secret"
    - "MYSQL_ROOT_PASSWORD=secret"
  ports:
    - "33062:3306"
  networks:
    - network

dbadmin:
  image: adminer
  volumes:
    - ./docker/adminer/adminer.css:/var/www/html/adminer.css
  links:
    - database
  ports:
    - 8085:8080
  environment:
    - "PMA_HOST=database"
    - "PMA_PORT=3306"
    - "MYSQL_USER=homestead"
    - "MYSQL_PASSWORD=secret"
```

```
    - "MYSQL_ROOT_PASSWORD=secret"
  networks:
    - network
```

Ao todo, o arquivo do projeto inclui as aplicações:

- app: container contendo dependências do *Laravel* e o *Composer*;
- web: container contendo *nginx* que serve a aplicação contida dentro da pasta de trabalho;
- database: container contendo *MySQL*;
- dbadmin: container contendo o *phpMyAdmin*.

Com os containers em execução, é possível executar comandos por meio do container, ou seja, é possível usar um comando no *Laravel* no container *app*, um comando para o banco de dados no container *database*, e o mesmo para os outros serviços.

7.1.4 Pacotes

Na próxima etapa de pré-projeto, foram decididos quais pacotes essenciais para o projeto eram os mais indicados, se aproveitando do ecossistema do *Laravel* e da facilidade de incluí-los no projeto através do *Composer*. Sendo assim, uma análise do projeto indicou a necessidade de um pacote de painel de administrador, um pacote de integração com a API do *Google* e um pacote de criação de *QR Codes*.

7.1.4.1 *Voyager*

Para a pacote de painel de administrador, foi escolhido o *Voyager* e os quesitos levados em consideração para sua escolha foram:

- Interface de administrador completa e fácil customização;
- Capacidade de criar funções e permissões para os usuários;
- Capacidade de listar tabelas do banco de dados;
- Capacidade de criar páginas de leitura, criação e edição de informação baseado em tabelas do banco de dados;
- Capacidade de deletar linhas de tabelas do banco de dados;
- Capacidade de escalar sobretudo devido ao gestor de mídia, que possibilita a edição de imagens dinamicamente pelo usuário final.

7.1.4.2 Socialite

Para o pacote de integração com a API do *Google*, destaca-se o fato da única necessidade ser o login através de uma conta *Google*. Sendo assim, e dado o fato de que a operação de *login* em plataformas como o *Google* ou *Facebook* é uma operação trivial, o *Laravel* oferece um pacote oficial chamado *Socialite* que foi usado no projeto.

7.1.4.3 *Simple QR-Code*

Para o pacote de criação de *QR Codes* foi escolhido o pacote *Simple QR-Code* devido à simplicidade de integração, considerando que essa operação envolve pouca complexidade de implementação e se resume em receber um objeto que contém o *QR Code*.

7.2 Projeto

Esta etapa se iniciou com o desenvolvimento do código do projeto e durou até a finalização de todas as funcionalidades e homologação com o cliente.

Usando o *Composer* foi possível gerar uma aplicação em *Laravel* contendo a arquitetura com funcionalidades básicas. Com essa estrutura, a primeira etapa foi a criação de *migrations* para criar o banco de dados no formato planejado. Contudo, é relevante observar que o *Voyager* oferece funcionalidades nativas para organização e gerenciamento tanto das tabelas como de funcionalidades relacionadas à função de usuário. Sendo assim, por meio de *migrations*, são criadas as estruturas de Equipamentos e Eventos, enquanto a tabela Usuários, Funções e Permissões_funções foram adicionadas com a instalação do *Voyager*, que por sua vez foi inserida através do *Composer*.

O Código 2 mostra a classe *CreateEquipamentoTable* que estende a classe *Migration*, herdando assim os métodos que transformam o código em SQL para realizar operações no banco, e representa uma tabela e as colunas que serão inseridas. Cada coluna necessita da chamada de um método que define qual será sua estrutura. Então, por exemplo, a coluna nome, tem uma estrutura de *string*, que é convertida para *varchar* quando o código em PHP é interpretado pelo *Laravel* e traduzido para SQL. O *Laravel* também permite detalhar outras informações de coluna, como definir que id é um campo cujo valor se incrementa, ou como é possível definir se um campo pode ser nulo, como é o caso de eventoId. Isso torna o projeto mais manutenível, pois o código do *Laravel* é semântico e tem uma interpretação mais simples que *queries*.

Código 2 – *Migration* da tabela Equipamento.

```
class CreateEquipamentosTable extends Migration
{
```

```
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('equipamentos', function (Blueprint $table) {
            $table->increments('id');
            $table->string('nome');
            $table->string('estado');
            $table->integer('evento_id')->nullable();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('equipamentos');
    }
}
```

Também foi necessário criar um *Model* para representar a tabela, então cria-se *Models* Equipamento, Evento e Usuário. No *Model* pode-se definir quais colunas podem ser editadas ou quais devem ser protegidas, além de ser possível definir relações entre tabelas, com o *Laravel* lidando com a implementação a mais baixo nível, analisando a coluna que referencia outra tabela e trazendo a instância relacionada. O *Model* otimiza a implementação de regras complexas, também contribuindo com a manutenibilidade do projeto.

O Código 3 mostra a estruturação base do *Model* Equipamento do projeto. Nela, pode-se observar a implementação da relação com a tabela Evento.

Código 3 – *Model* da entidade Equipamento.

```
<?php
```

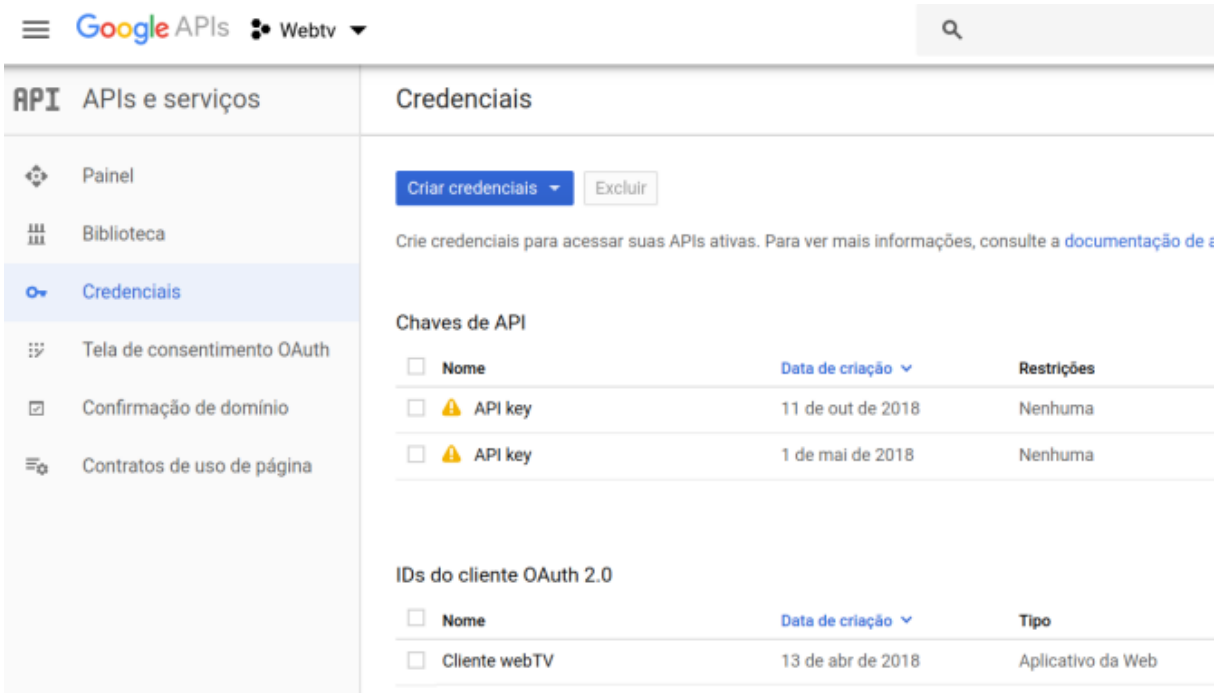
```
namespace webTV\Models;

use Illuminate\Database\Eloquent\Model;

use webTV\Models\Evento;

class Equipamento extends Model
{
    public function evento() {
        return $this->belongsTo(Evento::class);
    }
}
```

O *Laravel* também apresenta funcionalidades de autenticação prontas para uso, que são implementadas via *Traits* e *Controllers*.

O passo seguinte foi integrar o *Socialite* e ajustar a página de *login* de forma a direcionar para o *login* pelo *Google* e definindo o domínio *@faacwebtv.com.br* como o único permitido para autenticação. A adição do *Socialite* no projeto aconteceu via *Composer* e a instalação usando o *Artisan* pelo terminal. Dentro da estrutura de projeto, foi necessário adicionar rotas de *login* da plataforma escolhida e preencher os dados de permissão no *Console* de desenvolvedor do *Google*, plataforma que gerencia permissões para uso das APIs de acordo com o projeto registrado.

A Figura 4 mostra o *Console* de desenvolvedor do *Google* e as chaves criadas para uso no projeto. Autenticando na plataforma, as requisições para o *Google* se tornaram funcionais no projeto.

Figura 4 – Console de desenvolvedor do Google.



Fonte: Elaborado pelo autor.

Com as permissões funcionando, foi necessário ajustar os métodos do *controller* para lidar com as informações fornecidas pelo *Socialite*, que serve para ligar informações providas pela API de *login* do *Google* com o projeto por meio de um objeto.

No Código 4, é possível notar o primeiro método, que define a plataforma de *login*, o domínio escolhido e as informações desejadas para retorno. No segundo método, que é chamado após o *login* pelo *Google* ter sido bem sucedido, obtém-se as informações fornecidas pelo usuário através do *Socialite*, que é usado para criar o registro no banco e depois autenticar o usuário. Com isso feito, é possível testar a implementação dessa funcionalidade.

Código 4 – Código de implementação do *Socialite*.

```
public function redirectToProvider()
{
    return Socialite::driver('google')
        ->with(['hd' => 'faacwebtv.com.br'])
        ->scopes(['openid', 'profile', 'email'])
        ->redirect();
}

public function handleProviderCallback()
{
```

```
        $user = Socialite::driver('google')->user();

        $authUser = $this->findOrCreateUser($user);

        Auth::login($authUser, true);

        return redirect($this->redirectTo);
    }
```

Com o *Socialite*, é possível acessar o *login* do *Google* para ter autorização de acesso ao sistema. É possível observar o formulário de *login* do *Google* na Figura 5 com o domínio fixo *@faacwebtv.com.br*.

Figura 5 – *Login* do Google.

Fonte: Elaborado pelo autor.

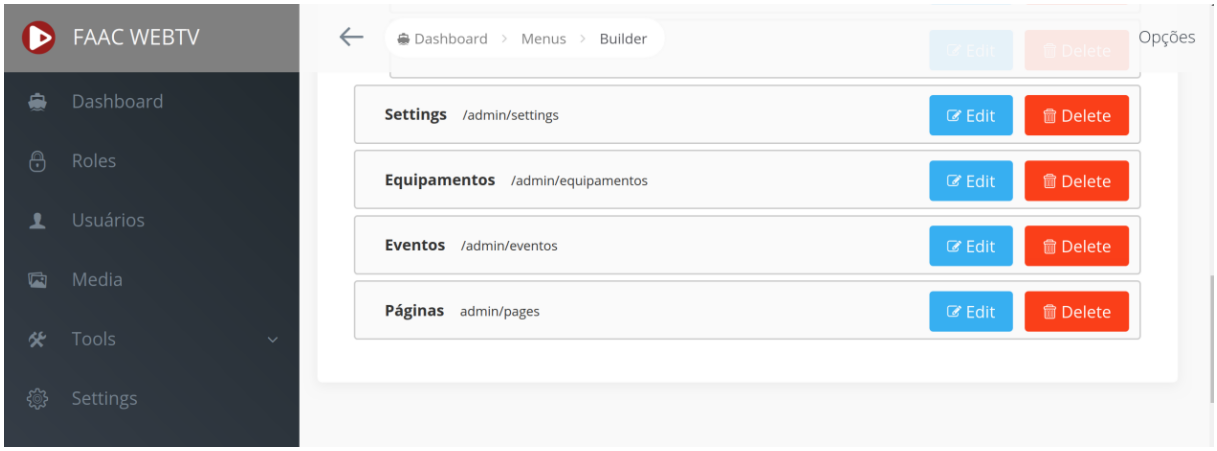
Com o sucesso dessa implementação, a próxima etapa foi continuar a construção

do painel de administrador, sobrescrevendo partes do *Voyager* de forma a se encaixar nas necessidades do projeto. Também encaixando os *Models* essenciais do projeto, Eventos e Equipamentos dentro das funcionalidades já providas.

Sendo assim, a interface do painel pode ser customizada para mostrar as entidades principais e o menu do painel foi criado dinamicamente, de acordo com o construtor de menus do *Voyager*. Através da ferramenta de páginas dinâmicas do *Voyager*, também foi possível criar páginas com a listagem de tabelas do banco de dados.

O Construtor de menu pode ser observado na Figura 6. Nele, é possível adicionar os campos Eventos e Equipamentos no menu lateral, onde é possível acessar listagens de tabelas do banco de dados.

Figura 6 – Construtor de menu do *Voyager*.

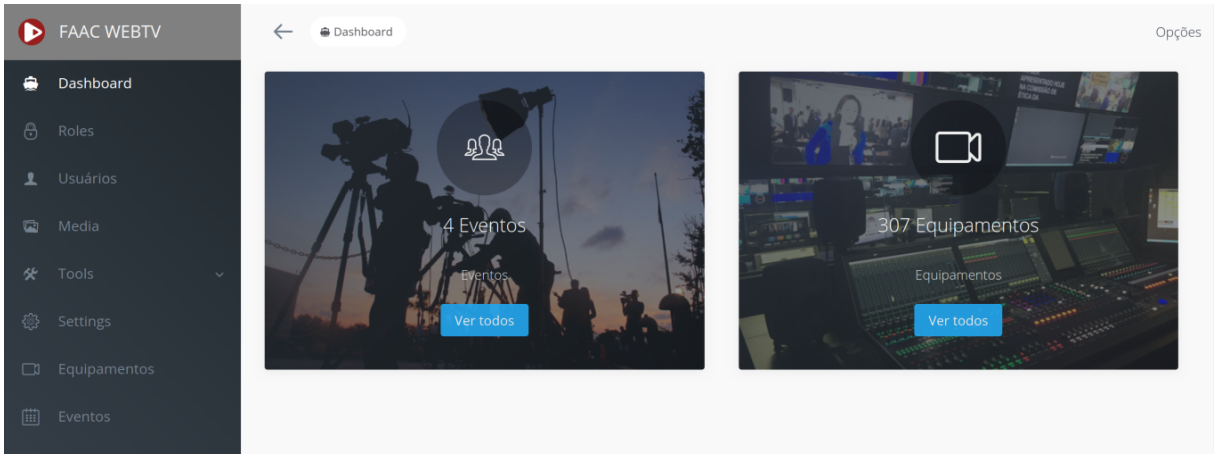


Fonte: Elaborado pelo autor.

Também foi possível adicionar ícones na página inicial do painel para construir a interface da página e direcionar a interação do usuário para os elementos mais importantes do sistema, que pode ser feito sobrescrevendo o conteúdo padrão do *Voyager*. Os ícones levam para as páginas de listagem da respectiva entidade.

O painel pode ser observado na Figura 7 contendo botões que levam para as páginas de Equipamentos e Eventos.

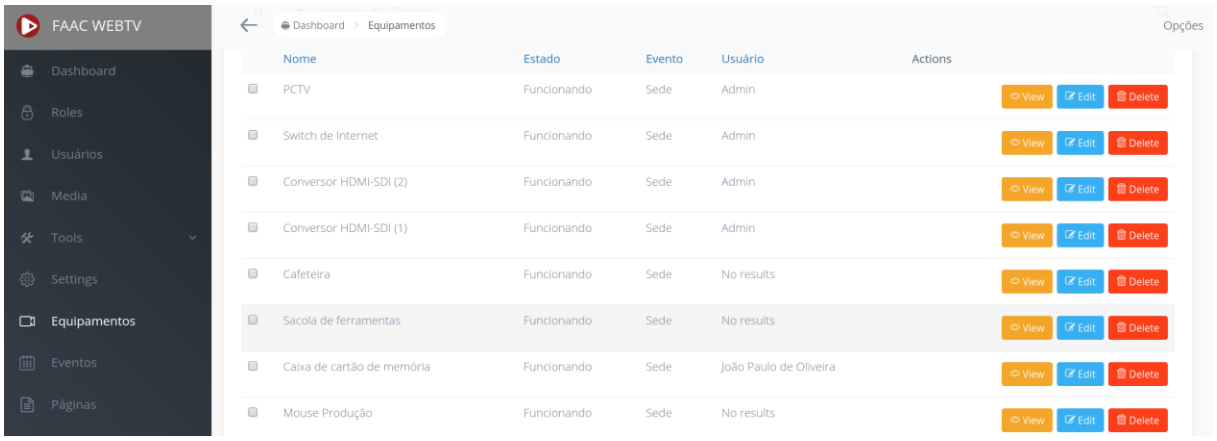
Figura 7 – Interface do painel.



Fonte: Elaborado pelo autor.

Um exemplo de página de listagem de entidade do trabalho pode ser observada na Figura 8. A listagem inclui informações salvas no banco de dados de cada coluna da entidade e botões para as páginas de visualização, edição e deleção.

Figura 8 – Listagem de equipamentos.



Fonte: Elaborado pelo autor.

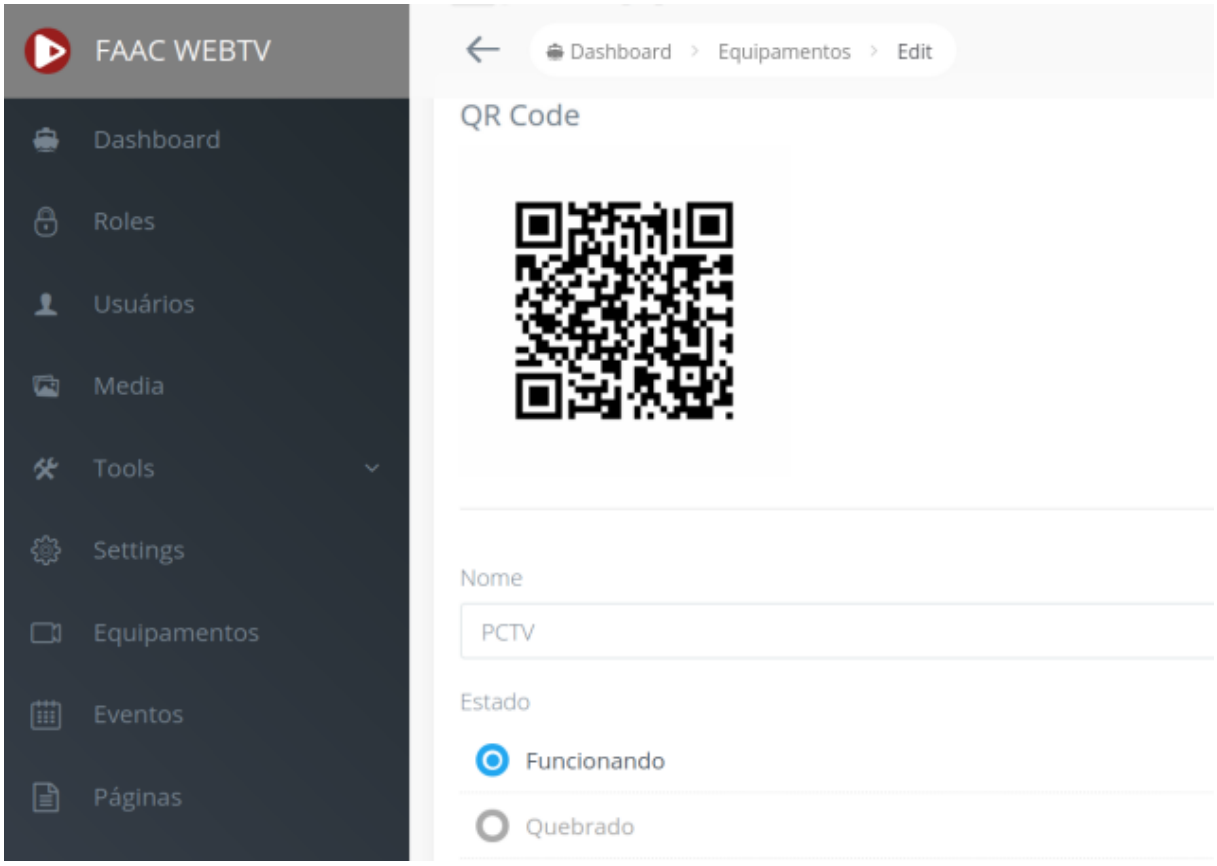
Com a implementação dessas funcionalidades, seguiu-se para a próxima etapa do projeto, a implementação dos *QR Codes* com o *Simple QR Code*. Sua instalação no projeto também foi feita por meio do *Composer* e pode ser usado acessando a classe inserida no projeto. Com uso de sua instância, é possível criar um *QR Code* baseado em uma informação de texto.

Usando a classe e inserindo o *QR Code* na página de edição de cada equipamento, a URL fornecida para a criação do *QR Code* é a própria URL acessada. O *QR Code* fica disponível no formato de imagem, portanto pode ser aberta em uma outra página do navegador para impressão.

A Figura 9 demonstra um exemplo de página de edição de equipamento com o *QR Code*, além dos campos de Estado, que define se o Equipamento está funcionando ou quebrado

e o nome.

Figura 9 – Página de edição de equipamentos com o *QR Code*.



Fonte: Elaborado pelo autor.

Tendo em vista que a URL fornecida pelo *QR Code* leva a própria página, ele só pode ser apresentado em páginas de edição de equipamento. Sendo assim, o fluxo de implementação se inicia com o registro do equipamento, e posteriormente, editando o equipamento a imagem de *QR Code* aparece.

O projeto usa a mesma *view* na edição e criação de registros, se aproveitando do fato do *Blade* poder reutilizar partes repetidas de código. Elas são complementados pelo contexto fornecido pelos *Controllers* em toda requisição de rota. Portanto a identificação do método, e se o *QR Code* deve ou não aparecer, deve ser feita através da rota de URL.

A arquitetura do projeto segue um padrão estruturado com relação a rotas, também usando os protocolos HTTP em consideração. Para as operações triviais, a rota descreve a entidade desejada e a ação descrita, sendo assim o padrão fica da seguinte forma, usando o exemplo de equipamentos:

- *GET* /equipamentos : Listagem de todos os equipamentos
- *GET* /equipamentos/id : Detalhes de um equipamento específico

- *POST* /equipamentos : Criação de um novo equipamento
- *PATCH* /equipamentos/id : Edição de um equipamento específico
- *DELETE* /equipamentos/id : Deleção de um equipamento específico

Sendo assim, por exemplo, ao acessar a página na rota *faacwebtv.com.br/equipamentos*, o sistema faz uma requisição para a rota /equipamentos e seguindo o padrão de nomeação, o sistema retorna o a listagem de equipamentos. A rota ativa um método do *controller*, que preenche informações necessárias para a *view* e faz o redirecionamento que deve acontecer para o usuário. Uma rota de criação seria uma requisição *POST* para a URL equipamentos, já uma de edição seria uma requisição *PATCH* para URL *equipamentos/id* do equipamento, pois a edição requer a identificação do equipamento que será modificado.

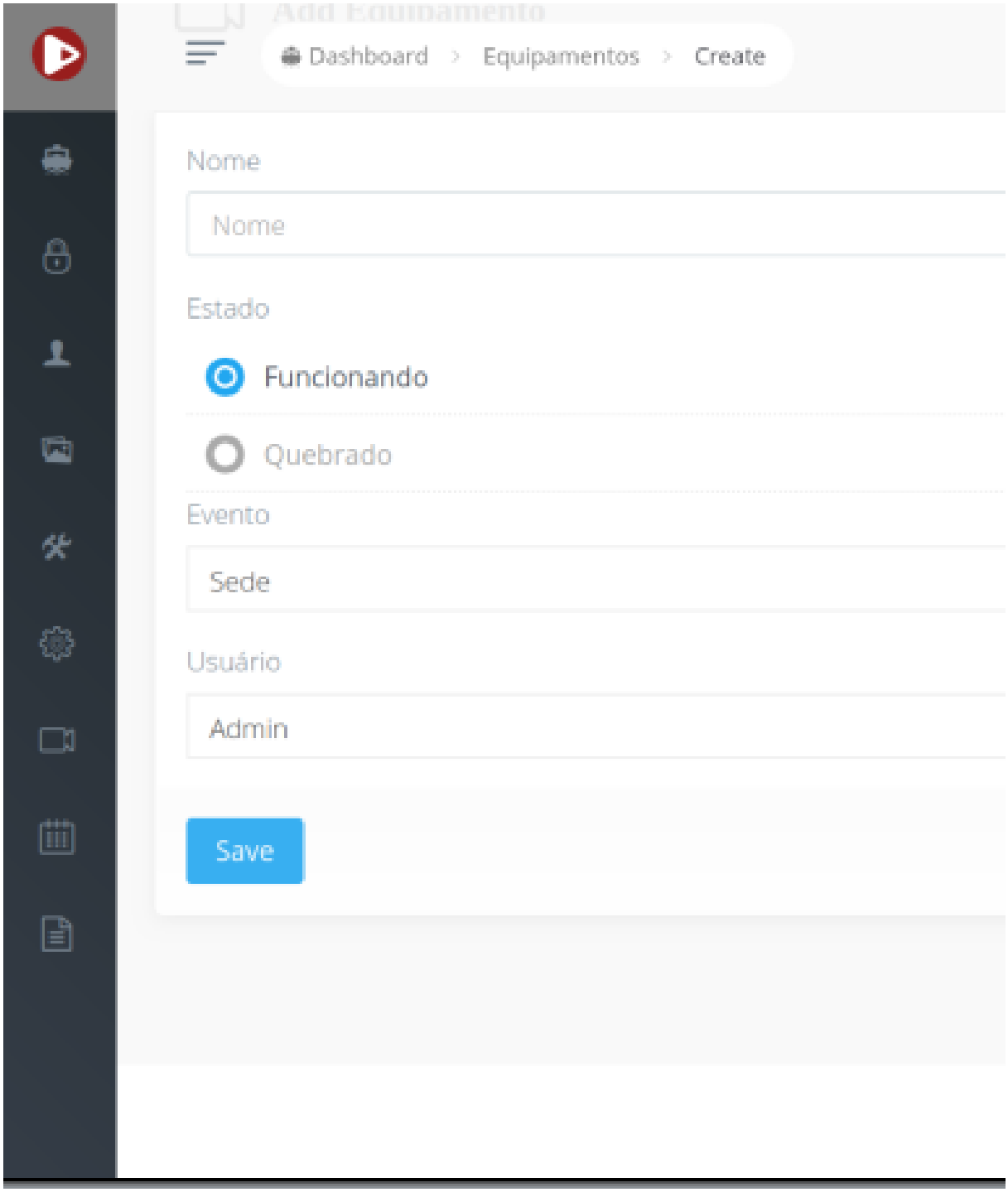
A mesma ideia é usada para a visualização de um equipamento específico, que requer uma requisição *GET* para equipamentos/id do equipamento.

É importante lembrar que é possível executar código PHP dentro da própria HTML, portanto é também possível acessar métodos providos pelo *Laravel* ou criados pelo desenvolvedor, de forma que acessar os elementos de rota da página se torna possível.

Sabendo disso, faz-se uma verificação na *view* de criação e edição de equipamento que adiciona o campo *QR Code* caso a rota da página possua um elemento de id, que significaria que a página já é uma página de edição, e não mais de criação.

Na Figura 10 observa-se a página de criação de Equipamento sem a presença do *QR Code*. Também é possível observar os campos de Evento associado e o usuário que registrou o equipamento.

Figura 10 – Página de criação de equipamentos.



Fonte: Elaborado pelo autor.

Dentro das funcionalidades que o *Voyager* oferece também se encontra a capacidade de gerenciar funções de usuários e suas permissões dentro do sistema. Acontece com o uso da tabela auxiliar permissao_funcao. Assim, o menu pode apresentar menos opções dependendo da função do membro.

Assim, todas as funcionalidades estão implementadas e se encerra a etapa de projeto após homologação e aprovação do cliente.

Também é feita uma reunião com o usuário final visando explicar o funcionamento do sistema e como ele se encaixa dentro dos processos da própria organização.

7.3 Pós-projeto

A etapa de pós-projeto consistiu em lançar o produto em ambiente de produção para uso do usuário final, neste caso, os membros da FAAC webTV. Sendo assim, uma pasta que serve a página e um banco em *MySQL* foram as ferramentas a disposição. A página é acessada através de uma conexão FTP, apropriada para a transferência de arquivos, e o *MySQL* através do *phpMyAdmin*.

As *migrations* são executadas localmente para criar a estrutura de tabelas, que são exportadas para serem importadas no *phpMyAdmin* do ambiente de produção. Já o código do *Laravel* é copiado para a pasta usando o FTP. A página fica então disponível para acesso.

Assim, os membros podem passar a utilizar o sistema para transmissões. O registro de equipamentos e eventos é feita na página de registro do sistema. Com os equipamentos registrados, a imagem do *QR Code* fica disponível nas páginas de edição e então é possível imprimir as imagens e colá-las nos equipamentos.

Os eventos produzidos pela FAAC webTV são registrados no sistema. Assim, na montagem, os equipamentos são registrados nos eventos, e na desmontagem o equipamento volta para a sede.

No acompanhamento com o cliente, existe o *feedback* e foi ressaltada a melhora na organização proporcionada pelo sistema. O uso do *QR Code* garante a integridade do estado dos equipamentos na medida que o processo é incorporado na organização dos eventos.

8 Conclusão

A crescente importância da informação no mundo moderno destaca a importância de *softwares* para o crescimento sustentável de organizações. Dessa forma, empresas e organizações que precisam organizar seus ativos tendem a usar sistemas de *softwares*, tanto para alterar dados como também para distribuí-los, auxiliados por ferramentas e processos que visam oferecer estabilidade e integridade para o *software* e a informação que ele oferece.

No caso do sistema elaborado neste trabalho, o envolvimento multidisciplinar para a construção de uma ferramenta que atende um problema real enfrentado pela FAAC webTV apresenta uma oportunidade que explora o potencial abrangente de uma universidade. Num aspecto técnico da Ciência da Computação, valoriza-se a criação de um sistema com regras e ideias complexas que exigem conceitos de diversas disciplinas, bem como num elemento de veracidade no que diz respeito à interação interpessoal, pressão para entrega e elaboração de soluções técnicas para questões abstratas.

Também existe uma importância tecnológica, sobretudo com relação a como a Ciência da Computação se relaciona com o entendimento de ferramentas que representam o estado da arte. No caso do projeto, pode-se citar o *Laravel*, ferramenta importante na área de construção de aplicações para web, o *Docker*, ferramenta de criação de ambientes de desenvolvimento, e o *Git*, ferramenta de versionamento de código que estimula a organização a aprimora a realização de trabalho em equipes.

É imprescindível que haja essa mescla de conceitos e tecnologias na busca pela excelência.

Considerando esses aspectos, o projeto aborda o problema de forma sistêmica, através de processos que formam uma estrutura que dá o suporte necessário para a criação do sistema. Os resultados, no caso, o sucesso na elaboração e lançamento do produto, corroboram com conceitos muito difundidos na Ciência da Computação, e de forma geral, na ciência, relacionados à elaboração de processos de acordo com o contexto acompanhada da pesquisa, análise e aplicação.

9 Trabalhos futuros

A etapa de pós-projeto ofereceu conhecimento da utilização na prática da ferramenta e ideias que também surgiram ao longo do processo de desenvolvimento que serão exploradas nessa seção, a fim de arquitetar possíveis próximas etapas para o avanço do projeto.

Uma sugestão foi da adaptação do painel de administrador para possibilitar o gerenciamento do conteúdo do site institucional da FAAC webTV e com relação a isso também se destaca a versatilidade do *Voyager*, que possui um editor de texto com ferramentas de edição e gerenciador de mídia. Com essas ferramentas, é possível definir os parâmetros através dos *Controllers* e que seriam exibidos na *View*, com informação fornecida pelo usuário final.

Também foi pensado ao longo do processo elaborar uma aplicação mobile, ao invés do site, que requer o navegador dos celulares. Seria uma funcionalidade que otimizaria o processo, sem ter a necessidade de usar uma aplicação externa para ler os *QR Codes*. Essa melhoria poderia utilizar a estrutura do banco de dados e a capacidade do *Laravel* de funcionar como uma *API Restful*, retornando informações sem necessitar de uma sessão.

Referências

CANALTECH. *O que é API*. 2019. Disponível em: <https://canaltech.com.br/software/o-que-e-api/>. Acesso em: 20 de outubro de 2019.

CODECADEMY. *CLI*. 2019. Disponível em: <https://www.codecademy.com/articles/command-line-interface>. Acesso em: 5 de setembro de 2019.

COMPOSER. *Documentação Composer*. 2019. Disponível em: <https://getcomposer.org/doc/00-intro.md>. Acesso em: 5 de setembro de 2019.

DEBIAN. *What is Package Manager?* 2017. Disponível em: <https://web.archive.org/web/20171017151526/http://aptitude.alioth.debian.org/doc/en/pr01s02.html>. Acesso em: 20 de outubro de 2019.

DOCKER. *Documentação Docker*. 2019. Disponível em: <https://docs.docker.com/>. Acesso em: 20 de outubro de 2019.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design Patterns Abstraction and Reuse of Object Oriented Design*. 1993. Disponível em: <http://people.cs.pitt.edu/~mock/cs1530/lectures2/ecoop93-patterns.pdf>. Acesso em 10 de outubro de 2019.

GIT. *Documentação Git*. 2019. Disponível em: <https://www.git-scm.com/>. Acesso em: 20 de outubro de 2019.

LARAVEL. *Documentação Laravel*. 2019. Disponível em: <https://laravel.com/docs/6.x>. Acesso em: 20 de outubro de 2019.

MYSQL. *Documentação MySQL*. 2019. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/>. Acesso em: 20 de outubro de 2019.

ORACLE. *Banco de Dados*. 2019. Disponível em: <https://www.oracle.com/br/database/what-is-database.html>. Acesso em: 12 de outubro de 2019.

ORACLE. *Documentação ORACLE*. 2019. Disponível em: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Data-Types.html#GUID-A3C0D836-BADB-44E5-A5D4-265BA5968483>. Acesso em: 15 de outubro de 2019.

PHP. *Documentação PHP*. 2019. Disponível em: <https://www.php.net/manual/pt_BR/intro-whatis.php>. Acesso em: 5 de setembro de 2019.

PHPMYADMIN. *Documentação phpMyAdmin*. 2019. Disponível em: <https://docs.phpmyadmin.net/en/latest/intro.html>. Acesso em: 20 de outubro de 2019.

POP, D. P.; ALTAR, A. *Designing an MVC Model for Rapid Web Application Development*. 2014. Disponível em: <https://www.sciencedirect.com/science/article/pii/S187770581400352X>. Acesso em: 10 de outubro de 2019.

PREE, W. *Meta Patterns — A Means For Capturing the Essentials of Reusable Object-Oriented Design*. 1994. Disponível em: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.7935&rep=rep1&type=pdf>. Acesso em 1 de setembro de 2019.

PRESSMAN, R. *Engenharia de Software Uma Abordagem Profissional*. 2. ed. [S.l.]: AMGH Editora LTDA, 2011.