FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Diego Antonio Roca Valls

Matheus Henrique Bueno do Nascimento

SISTEMA DE MARKETING ENTRE FREELANCER E CLIENTES

Americana, SP

2018

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Diego Antonio Roca Valls

Matheus Henrique Bueno do Nascimento

Sistema de marketing entre freelancer e clientes

Relatório técnico desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas sob a orientação do Prof. Esp. José William Pinto Gomes.

Americana, SP.

2018

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

V28s VALLS, Diego Antonio Roca;

Sistema de marketing entre freelancer e clientes. / Diego Antonio Roca Vals, Matheus Henrique Bueno do Nascimento. – Americana, 2018.

52f.

Monografia (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Esp. José William Pinto Gomes

1. Aplicativos I. NASCIMENTO, Matheus Henrique Bueno do II. GOMES, José William Pinto; II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.519

Diego Antonio Roca Valls
Matheus Henrique Bueno do Nascimento

SISTEMA DE MARKETING ENTRE FREELANCER E CLIENTES

Relatório técnico apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.

Americana, 03 de dezembro de 2018.

Banca Examinadora:

José Willian Pinto Gomes (Presidente)
Prof. Especialista
Faculdade de Tecnologia de Americana

Alberto Martins Junior (Membro)
Prof. Mestre
Faculdade de Tecnologia de Americana

Odilon Delmont Filho (Membro)
Prof. Doutor
Faculdade de Tecnologia de Americana

RESUMO

O documento apresentado acompanha o desenvolvimento de um WebApp, as ferramentas utilizadas e motivação para o projeto. A oportunidade de negócio encontrada para o aplicativo, seu impacto esperado no mercado autônomo brasileiro de arte digital e as atividades a serem realizadas são alguns dos pontos importantes discutidos. O WebApp por sua vez tem como propósito a criação de uma plataforma onde o artista freelancer possa criar um portfólio, comparar preços com outros artistas da área e mostrar seu trabalho para possíveis clientes. A criação de um ambiente seguro para a divulgação do trabalho de artistas freelancers no Brasil e a criação de uma área onde empresas possam divulgar suas vagas em aberto é um dos objetivos do projeto. O projeto, por sua vez, tem como objetivo geral o desenvolvimento do aplicativo, usando dos conhecimentos adquiridos em curso e por pesquisa própria, seja por livros na biblioteca da faculdade quanto por informações *online*.

Palavras Chave: *WebApp; freelancer; web designer.*

ABSTRACT

The opportunity of business found to the app, its impact on the autonomous Brazilian art market and the activities to be carried out are some of the important points discussed. This Web App which in turn has been the creation of one platform where the freelance digital artist can create a portfolio, compare prices witch other artists and show you work to possible clients.

The creation of a safe environment to show work of freelance artists in Brazil and the creation of one area where companies can spread their job vacancies it's one of most objectives to this project. The project, in turn, has as general objective the development of the application, using the knowledge acquired in the course and by own research, either by books in the college library or by information online.

Keywords: *WebApp; frelance; web designer.*

SUMÁRIO

LISTA DE FIGURAS

INTRODUÇÃO

Em um país onde a economia digital e o trabalho freelance mantêm um crescimento constante, a necessidade de um local onde o freelancer possa mostrar seu trabalho é de extrema importância. Levando em conta isso, a criação de um WebApp de marketing digital é imperativa para o desenvolvimento social e econômico do freelancer brasileiro.

O freelancer brasileiro e seu comportamento na área de marketing digital, o aprendizado de novas ferramentas durante o processo de produção da arte assim como custos envolvidos, estão sendo um problema atual para esta área específica.

Durante a análise dos sites wix e trampo.co foram identificadas as opções disponíveis e suas funcionalidades para artistas digitais.

Foram identificados que porem criar portfolio *online* e buscar serviços para candidatar-se para vagas de freelancer. Porém, não atingem o foco especifico para o público exclusivo de artistas digitais freelancers, tanto pelo orçamento quanto pela utilização das ferramentas.

As plataformas nacionais já existentes mencionadas entre outras como Workana e GetNinjas e Trampo.co, para a contratação dos serviços de freelancers, não são simplificadas como o projeto WebApp - Lancelart.

O WebApp desenvolvido tem como intuito unir a funcionalidade dessas duas modalidades, criando um espaço onde o trabalhador autônomo possa mostrar seu trabalho e possíveis contratantes podem exibir seus cargos e trabalhos disponíveis.

O desenvolvimento de um WebApp de Marketing digital para artistas *freelance* pode ser facilmente utilizado em computadores e *smartphones*, tem uma disponibilidade ao público geral, sejam eles clientes ou contratantes e é uma solução para este problema.

O WebApp apresentado foca no mercado freelance, ou seja, do trabalhador autônomo sem vínculos permanentes com seus clientes.

Os aplicativos citados anteriormente, em prática, são generalistas. Tem foco em serviços diversos, desde serviços de reparo e limpeza até escrita e design gráfico.

O aplicativo é desenvolvido com Node.js e seu banco de dados em PostgreSQL, duas linguagens que mostram grande capacidade para atender o público nacional.

O WebApp precisa ter uma plataforma simplificada e de fácil acesso ao cliente, uma vez que o aplicativo será disponibilizado para o público e gratuito para cadastro e utilização.

Um dos objetivos gerais destacados é incentivar o artista freelancer a procurar o uso do marketing digital no seu dia-a-dia, além da motivação ao artista amador a participar do mercado freelancer, eliminando inclusive o custo para o artista para criar seu portfólio e montar sua vitrine de artes online.

Este projeto é focado em solucionar um problema enfrentado pelo artista freelancer, que tem a maioria de seu público alcançado pelo Marketing de referência ou Marketing-boca-a-boca, além do desejo pessoal do alavancamento do cenário artístico freelancer nacional.

1 METODOLOGIA

1.1 Identificação de metodologias e análise do software

A metodologia Kanban aplicada para o desenvolvimento do projeto, resultou em um desenvolvimento ágil de acordo com as ferramentas utilizadas e respectivas linguagens de programação que serão listadas nos próximos capítulos.

Durante a análise do WebApp, foram definidos quais seriam os acessos ao sistema tanto por artistas digitais como por contratantes, modificando seu acesso apenas por variável booleana (Verdadeiro ou Falso) facilitando tanto a programação, quanto a leitura do banco de dados.

Também foram criados diagramas de sequência, caso de uso, classe e atividades, facilitando e documentando a ideia inicial do WebApp a ser desenvolvido.

Determinado também que seria um software Web, devido a facilidade existente nos dias de hoje a acesso à internet e pelo foco específico do mercado a ser atingido.

Na implementação, foram realizados o desenvolvimento do WebApp, execuções de metodologias aplicadas anteriormente, durante e depois do desenvolvimento, para manter o WebApp em crescente atividade evolutiva, buscando ao máximo atingir o público específico e suas objetividades.

Como todo software em desenvolvimento, é preciso garantir que ele tenha sua usabilidade e garantir também que atinja os objetivos específicos aos seus usuários com eficiência. Neste quesito foi validado através de testes e seguindo os diagramas desenvolvidos durante sua análise.

1.2 Identificação de linguagens e softwares utilizados

A proposta é de um aplicativo web, estará hospedado em um servidor e será acessado por qualquer computador ou *smarthphone* conectado à internet. Sendo apresentado e executado através de quaisquer navegadores como Internet Explorer, Mozilla Firefox, Google Chrome, Opera e Safari, de acordo com seu modelo responsivo determinado por seu *front-end* e somente após todo o *back-end* estar completo.

Deste modo, as ferramentas escolhidas para o desenvolvimento do sistema foram:

- **NodeJS:** é uma biblioteca de Java *script*, comumente usada para o desenvolvimento de *frameworks* escaláveis. Node pode suportar diversas conexões simultaneamente, é assíncrona e movida por eventos. Node foi lançada no dia 27 de maio de 2009, tendo um desenvolvimento corrente e funcionando em sistemas Linux, Microsoft *Windows*, OS X e outros. Node contém um código aberto, o que possibilita a criação e distribuição de módulos criados pela comunidade *online*. A biblioteca é escrita em Java *script*, uma linguagem já estabelecida no mercado, de fácil uso e de tipagem dinâmica e implícita. Java script foi criado por Brendan Eich em 1995 enquanto trabalhava para a Netscape. Tem a função de criar *scripts* que mudam o conteúdo da página do cliente sem que essas mudanças precisem ser passadas pelo servidor. Foi criada para ser usada em navegadores Netscape e se popularizou. Java *script* funciona em conjunto com HTML e por esse motivo, tanto Java *script* quanto Node.js, tem foco no cliente de uma comunicação servidor-cliente. As funções como o armazenamento de dados, imagens e acessos então são passadas para o gerenciador de banco de dados ligado ao Node.js, o PostgreSQL.

- **PostgreSQL:** foi lançada em 29 de janeiro de 1997 e funciona com um banco de dados objeto-relacional. Isso significa que PostgreSQL suporta objetos, classes e heranças. PostgreSQL, assim como Java *script* e Node.js, é *opensource*, isso é, tem código aberto e uma comunidade ativa. As ferramentas e o projeto foram pesquisados de antemão e escolhidas por motivos que serão descritos no próximo capítulo. O uso da linguagem Node.js junto ao gerenciador de banco de dados PostgreSQL, ambas com código fonte aberto, possibilita um sistema rápido e leve, economizando no custo de equipamento.

- **Kanban:** segundo David J. Anderson e Andy Carmichael, é um método para definir, gerenciar e melhorar serviços que fornecem trabalho de conhecimento, tais como serviços profissionais, criativos e *design* de produtos físicos e de *software*. Isto pode ser caracterizado como um método “comece pelo o que você faz agora” – um catalisador para uma mudança rápida e focalizada nas organizações – que reduz a resistência a mudanças benéficas pontualmente com os objetivos da organização. O método *Kanban* baseia-se em tornar visível o que é de outra forma conhecimento intangível, para garantir que a tarefa seja realizada com a quantidade certa de trabalho. Através de cartões (*post-it*, digitais, outros...) é sinalizado o andamento atual da tarefa.

- **Scrum:** O *Scrum* é um processo que ajuda as pessoas a solucionar problemas e concluir projetos com o máximo de produtividade possível e garantindo que os projetos em que trabalham tenham o maior valor possível. Com o *Scrum*, a equipe sempre vai ver o progresso de seus projetos em tempo real ao classificar as tarefas em estágios como “A fazer”, “Fazendo” e “Feito” facilitando desta maneira o alinhamento das informações.

- **Trello:** O sistema utilizado para implantar estas metodologias, Kanban e Scrum, no projeto foi o Trello. Lançado na TechCrunch Disrupt em setembro de 2011, o Trello vem conquistando milhões de usuários pelo mundo. Com este aplicativo fica fácil implantar as metodologias

citadas, podendo adicionar cores e prazos para entrega ou desenvolvimento do *software* WebApp.

- **GitHub:** GitHub teve seu primeiro *commit* em San Francisco em outubro de 2007 e está presente na vida de muitos programadores, sendo por volta de 96 milhões de repositórios hospedados. É uma plataforma de repositórios potente com diversas funcionalidades para a área de desenvolvimento, como compartilhamento de código-fonte, voltado para softwares *opensource*. Vemos neste projeto que, através de análise realizada pelo repositório mencionado, temos uma quantidade em percentual do código utilizado, sendo esta uma das funções de análise do GitHub.

- **GitBash:** Gitbash é utilizado para acessar comandos dos quais atualizam pacotes do Node e executa o server.js. É possível também com estas execuções, visualizar o site no momento do desenvolvimento através do navegador de sua escolha. É um terminal Linux que pode ser utilizado diretamente no SO *Windows* (7, 8, 8.1, 10, entre outras versões).

- **Package.json:** O arquivo package.json é o *starter* de qualquer projeto Node.JS e é responsável por descrever seu projeto, informar a versão do Node e do Npm (*Node Package Manager* - gerenciador de pacotes do Node). Além disso, através dele, podemos adicionar as versões que estão sendo trabalhadas do projeto, o nome do projeto, url, descrição, palavras chaves, licenças de uso, autores do projeto entre outras informações.

- **SWOT:** A matriz SWOT é uma ferramenta amplamente utilizada em gestão empresarial, através dela podemos destacar forças e fraquezas da empresa, assim como suas oportunidades e ameaças. SWOT significa: S - *Strengths*, W - *Weaknesses*, O - *Opportunitie*s) e T - *Threats*.

2 IMPLEMENTAÇÃO

2.1 Informações sobre o tipo de usuário escolhido

Este sistema é específico para artistas freelancers, o que indica que não deverá manter o foco em qualquer freelancer nacional e sim em desenvolvedores de arte digital.

O sistema desenvolvido tem como principal cliente os artistas digitais sejam, por exemplo, eles desenvolvedores de artes 3D, 2D, a web art, as ilustrações digitais, entre outros.

Sendo assim, freelancers específicos como programadores de diversas linguagens, analistas de redes, entre outros da área de tecnologia da informação, não verão usabilidade para este software.

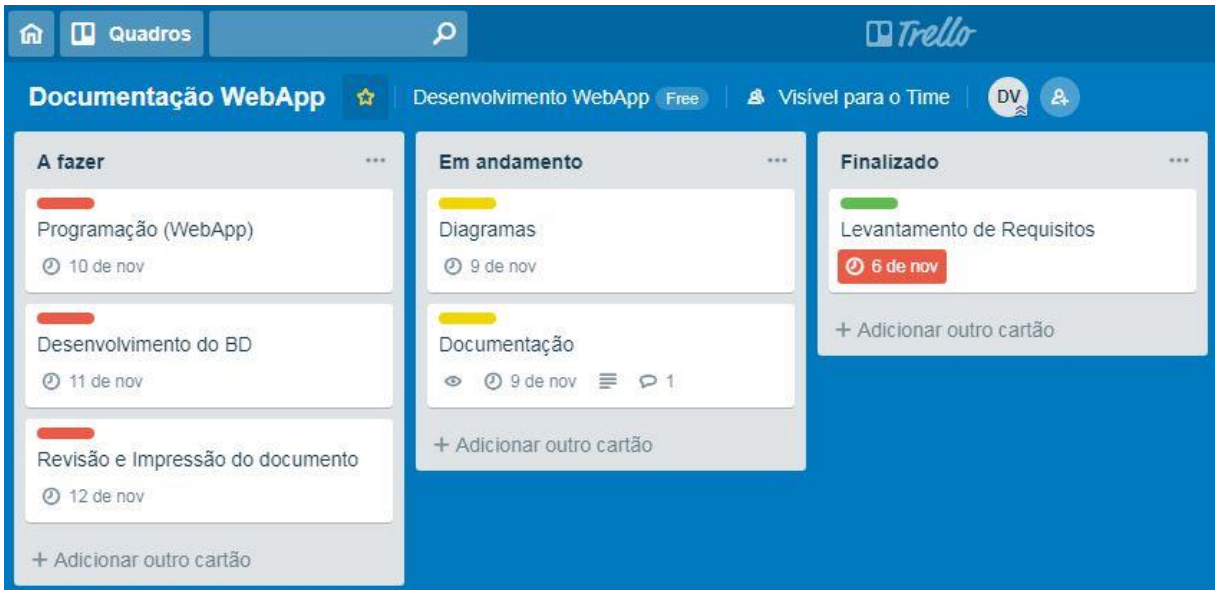
A seguir serão apresentadas as metodologias aplicadas, as definições de requisitos através de diagramas UML aplicados, o Banco de Dados desenvolvido do sistema e a explicação de código fonte utilizado no coração da aplicação (server.js) podendo assim entender também um pouco mais sobre Node.JS, que por sua vez é a linguagem principal utilizada.

2.2 Aplicação das metodologias

Para execução do projeto, tivemos um conjunto de ferramentas, metodologias e linguagens utilizadas com a finalidade de facilitar seu desenvolvimento e aqui explicaremos mais sobre duas principais em termo de metodologias ágeis para desenvolvimento, a união da metodologia Kanban com Scrum.

Acessando o sistema web da Trello, companhia anteriormente citada no capítulo 1, criamos nossos *charts* (cartões) dos quais foram transformados em uso da metodologia Scrum junto ao Kanban, agilizando e facilitando o processo em curto prazo do desenvolvimento do projeto como um todo e podendo adicionar comentários, conforme a figura 1 – Exemplo Trello:

Figura 1 - Exemplo Trello



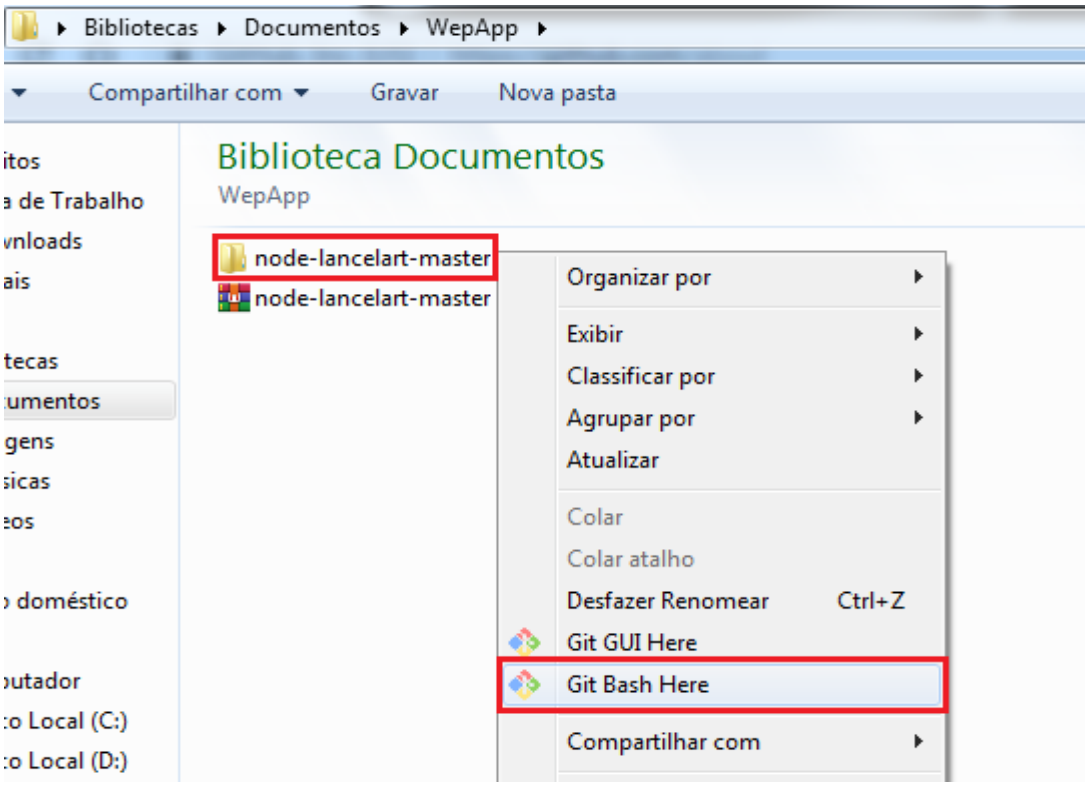
Fonte: Próprio autor.

Ao iniciar o desenvolvimento, foram determinados também em análise da dupla que teríamos um curto prazo para execução das tarefas mensais do *Scrum* como as *Sprints* mensais, reduzindo assim, para *Sprints* mais pontuais uma vez por semana.

2.3 Github e Bash, repositório do sistema e sua utilização

A utilização do Git não é novidade para desenvolvedores, porém quando se trata de sua utilização, o Bash do Git se torna um facilitador, podendo auxiliar na transferência de arquivos por comando. Sua utilização é baseada em disparar gatilho inicial através da pasta da qual deseja executar o Bash, como exemplo na figura 2 – “GitBash here”, a seguir:

Figura 2 - Gitbash Here

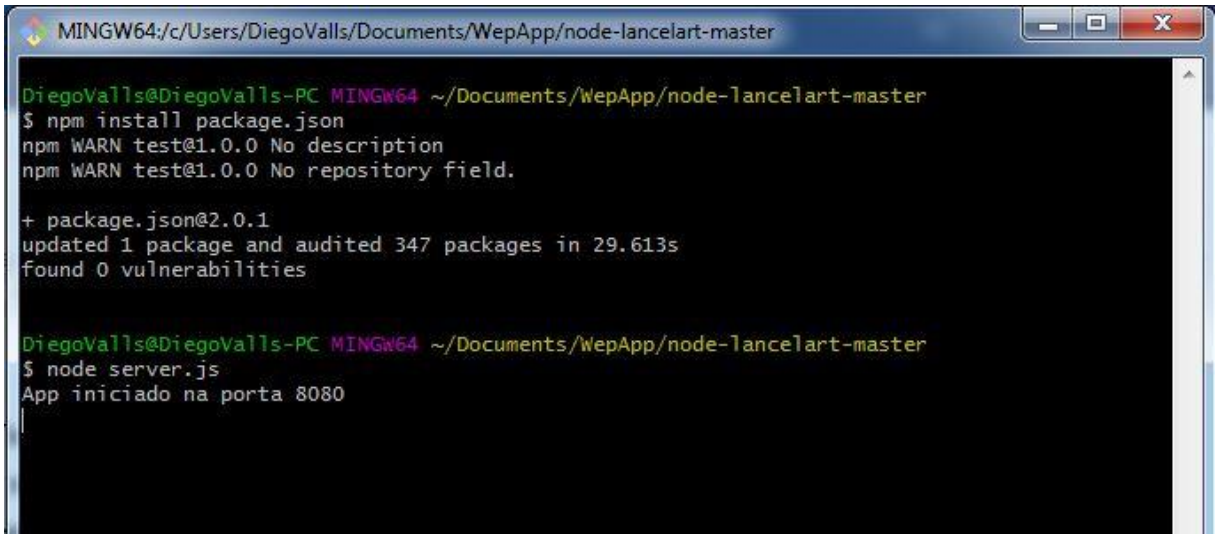


Fonte: Próprio autor.

Além disso, é possível usar grande parte dos comandos em Shell, além de poder utilizar o *VIM* (que vem nativo ao Git Bash), que é um editor de texto altamente configurável para criar e alterar de forma eficiente qualquer tipo de texto.

Está incluído como "vi" na maioria dos sistemas UNIX e no Apple OS X. Além disso o Git Bash integra-se com muitas ferramentas realizar as conexões ssh, executar transferências de arquivos scp, e o mais importante, utilizar o Git por linha de comando no Windows necessário principalmente para a execução do server.js. Para isso veremos na Figura 3 – Exemplo Bash a seguir, a execução da aplicação em desenvolvimento utilizando o comando “npm install package.json” e em seguida “node server.js” será utilizado também, fazendo com que o servidor fique online e possamos utilizar o WebApp como teste através do endereço //localhost:8080.

Figura 3 - Exemplo Bash



Fonte: Próprio autor.

Durante o desenvolvimento, era possível com este repositório analisar a quantidade de código que era utilizada de acordo com a linguagem e exibir em um gráfico simples pelo GitHub, conforme Figura 4 – Código utilizado em (%), demonstrado em porcentagem temos 63% de Java Script, 31,8% de HTML e 4,7% de CSS.

Figura 4 - Código utilizado em (%)

Fonte: Próprio autor.

2.4 SCRUM

Nesta metodologia ágil para gestão e planejamento de projetos de software encontramos seu ponto principal as Daily *Scrum*, que são breve reuniões (normalmente no início do dia de toda sexta-feira) alinhando o trabalho a ser realizado e quais os conhecimentos disseminados no dia anterior.

Para esta metodologia não utilizamos os ciclos, tipicamente mensais, chamados apenas de *Sprints*, que representa um *Time Box* dentro do qual o

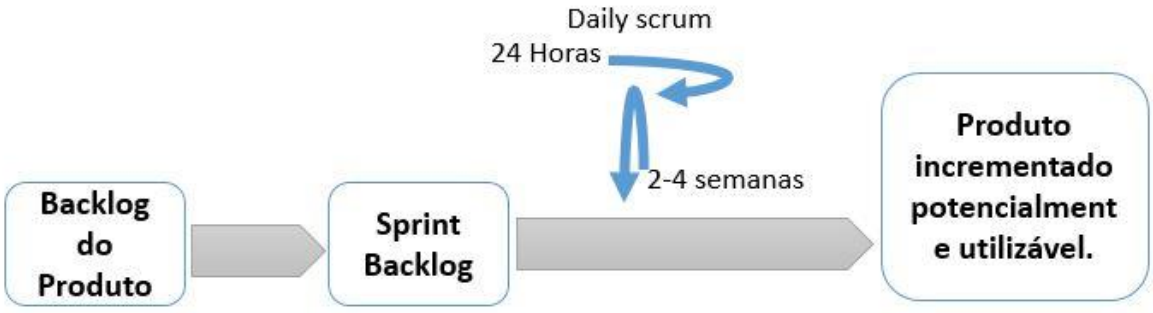
conjunto de atividades deve ser executado sendo assim as iterações do projeto, devido ao pouco tempo disponível para execução do mesmo.

Suas funcionalidades que serão implementadas, são mantidas dentro de *Product BackLogs*. No início de cada *Sprint*, faz-se um *Sprint Planning Meeting*, que é uma reunião de planejamento na qual o *Product Owner* prioriza os itens do Product Backlog e a equipe seleciona as atividades que ela será capaz de implementar durante o *Sprint* que se inicia.

Todas essas tarefas alocadas em um *Sprint* são transferidas do *Product Backlog* para o *Sprint Backlog*. Ao final de um *Sprint*, a equipe apresenta as funcionalidades implementadas em uma *Sprint Review Meeting*. Finalmente, faz-se uma *Sprint Retrospective* e a equipe parte para o planejamento do próximo *Sprint*.

Sendo desta maneira realizados todos através do sistema informado anteriormente, Trello. Assim reinicia-se o ciclo conforme Figura 5 – Scrum:

Figura 5 - Scrum



Fonte: Próprio autor.

2.5 Definição da implementação SWOT

A criação do Lancelart então tem como foco inicial um desejo que, após pesquisas relacionadas a área, tornou-se também uma estratégia de empreendedorismo. O WebApp então foi analisado através de uma tabela *SWOT*, também conhecida em nossa língua portuguesa como FOFA, simplificada que determina quais seus pontos fortes e fracos para serem analisados.

Ao ter noções das fraquezas e ameaças do projeto, principalmente sua fácil replicabilidade e instabilidade de mercado, as ferramentas utilizadas para o desenvolvimento do WebApp tem como intuito a criação de um sistema confiável e simples baseando-se também nas forças e oportunidades destacadas, conforme Figura 6 – SWOT.

Figura 6 - SWOT

Fonte: Próprio autor.

2.6 Detalhes do Banco de Dados

O recurso *auto increment* do PostgreSQL não é apenas uma opção na coluna e sim uma série de fatores que juntos determinam o efeito de somar um (ou n à sua chave primária), sendo eles:

- ***Sequence*** (sequência): Um recurso do PostgreSQL utilizado para gerar números sequenciais;

- ***Nextval*:** É uma função do PostgreSQL utilizada para obter o próximo valor de uma sequence;
- ***Default value*** (valor padrão): Um recurso disponibilizado em uma coluna para determinar um valor padrão que o campo assumirá caso nada seja informado no comando de insert;

Destes três temos em resumo à criação de um campo tipo *Serial*.

Exemplo:

```
ALTER TABLE exemplo ADD COLUMN id Serial;
```

Neste caso, foram utilizados em todas as chaves primarias do banco de dados (id) a tipagem *Serial*, facilitando a automatização do sistema no momento de criação de contas para usuários.

Entretanto, este foi apenas um exemplo de como foi implementado o banco de dados utilizado (PostgreSQL) e todo o seu código poderá ser analisado a seguir. De acordo com as figuras 7, 8, 9, 10, 11, 12 e 13 temos a visualização das tabelas do Banco de Dados:

Figura 7 - Usuário (BD)

TABELA DE USUÁRIOS				
Atributos	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Id	Determinante	Numérico		Chave primária
Login	Determinante	Texto		
Senha	Simples	Texto		
Nome	Determinante	Texto		
Email	Determinante	Texto		
Area	Multivalorado	Texto		Áreas de atuação do usuário
Contatp	Simples	Texto		Tipo de conta do usuário

Fonte: Próprio autor.

Figura 8 - Artistas (BD)

TABELA - ARTISTAS				
Atributos	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Biografia	Simples	Texto		biografia do artista
Redsoc	Simples	Texto		endereço de rede social do artista

Fonte: Próprio autor.

Figura 9 - Contratador (BD)

TABELA - Contratador				
Atributos	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Info	Simples	Texto		Informações do usuário contratante

Fonte: Próprio autor.

Figura 10 - Imagem (BD)

TABELA - "Imagem"				
Atributos	**Classe**	**Domínio**	**Tamanho**	**Descrição**
id	Determinante	Numérico		
Trabalho	Determinante	Numérico		Chave estrangeira do trabalho
Path	Simples	Texto		String do caminho até a imagem no filesystem

Fonte: Próprio autor.

Figura 11 - Organização (BD)

TABELA - Organização				
Atributos	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Email	Simples	Texto		
Telefone	Simples	Texto		
Local	Simples	Texto		
Nome	Simples	Texto		
Usuario	Determinante	Texto		Chave estrangeira do contratador
Id	Determinante	Texto		

Fonte: Próprio autor.

Figura 12 - Pedidos (BD)

TABELA - Pedidos				
Atributos	**Classe**	**Domínio**	**Tamanho**	**Descrição**
id	Determinante			
Texto	Simples			
organizacao	Determinante			Chave estrangeira da organização

Fonte: Próprio autor.

Figura 13 - Trabalhos (BD)

TABELA - Trabalhos				
Atributos	**Classe**	**Domínio**	**Tamanho**	**Descrição**
Id	Determinante	Numérico		
Tipo	Simples	Texto		Tipo de área na qual o trabalho se encaixa
Usuário	Determinante	Numérico		Chave estrangeira do artista

Fonte: Próprio autor.

Podemos analisar então que o banco de dados deste WebApp - Lancelart consiste de 7 tabelas, sendo que duas dessas possuem herança de uma principal através da própria programação utilizando o Node.JS.

A tabela usuário contém todos os dados que um usuário normal pode ter, como dados pessoais, dados de acesso e nesse banco de dados, o tipo de conta e as áreas artísticas na qual o usuário atua.

A tabela artista e contratador então ganham herança do banco de dados Usuário, tendo interações diferentes entre as duas tabelas.

A tabela artista interage com a tabela trabalho. A tabela trabalho cria um local onde imagens com o mesmo tema ou feitas em conjunto são postas.

Para cada Artista, é disponibilizado 5 (cinco) tabelas de trabalho. As tabelas de trabalho, por sua vez, interagem com a tabela Imagem, que mantém os dados de imagens individuais.

Essas imagens são salvas no sistema de arquivos *(filesystem)* do servidor e recuperadas pelo atributo *“path”* que armazena o caminho no sistema de arquivos para a imagem. As 5 (cinco) instancias da tabela imagem são aceitas para cada tabela trabalho.

A tabela contratador interage diretamente com a tabela organização, que tem apenas dados da organização.

Apenas uma tabela organização existe para cada tabela contratador. A tabela organização por sua vez interage com a tabela pedido. E a tabela pedido tem como função a criação de requisições aos artistas que a virem.

2.7 Diagramas UML (Unified Modeling Language)

O UML (Unified Modeling Language), conhecido como Linguagem de Modelagem Unificada, contém os diagramas necessários para entendimento simplificado e ao mesmo tempo pontual de como realizar o projeto.

O intuito do UML utilizado no WebApp é simplificar a comunicação entre Analista e Programador, agilizando o processo de programação final.

Como exemplo, veja a seguir os diagramas analisados e desenvolvidos:

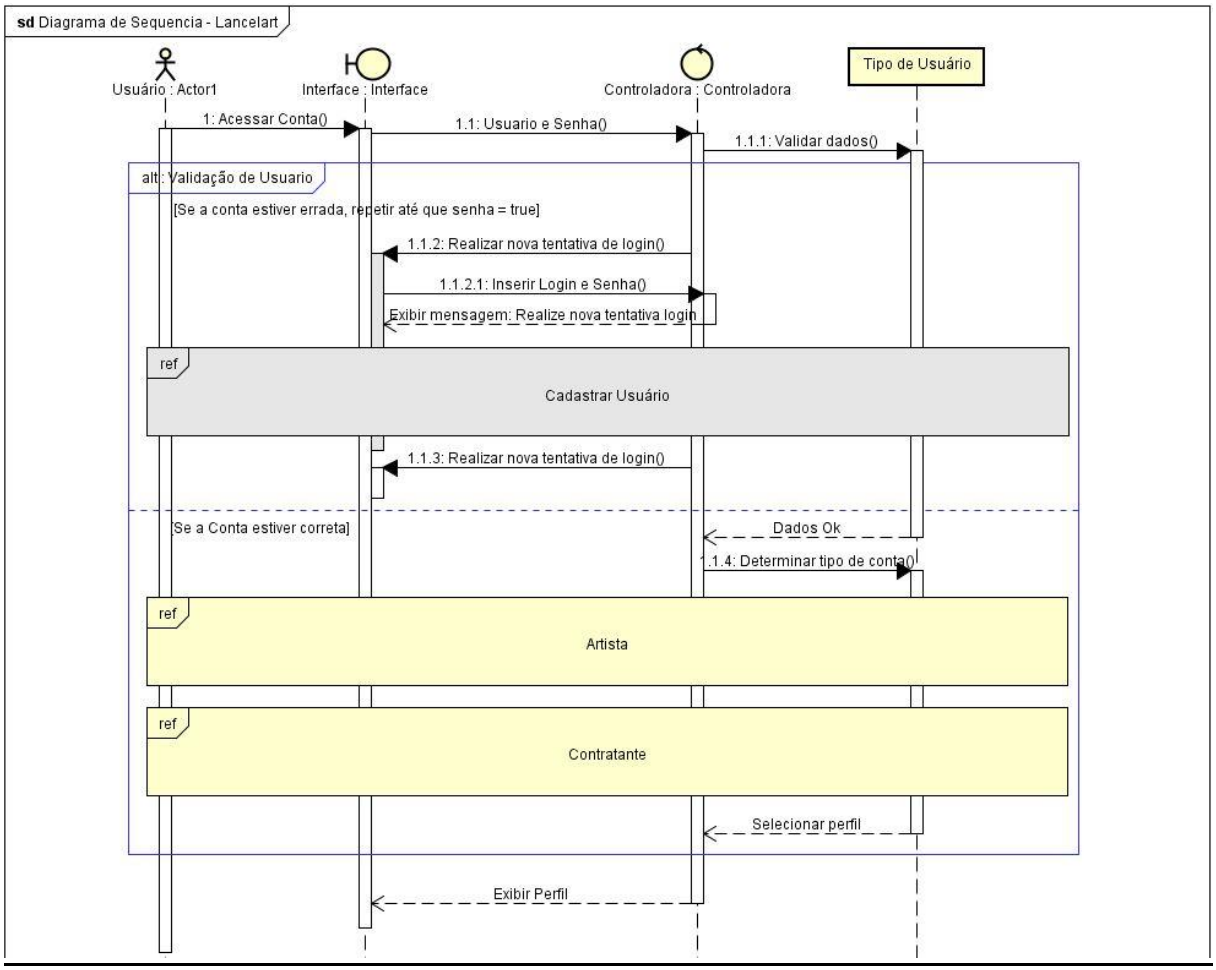
Diagrama de Sequência (Sequence)

Este, composto por mais 3 diagramas referenciados, dos quais são para artistas, contratantes e cadastro de usuários, facilitam o entendimento do acesso que será realizado por usuários no WebApp.

Neste diagrama o ator principal (Usuário) é generalizado anteriormente ao momento de seu acesso ao sistema, no qual será validado e encaminhado para área de acesso correta.

Antes de acessar o sistema em sua área específica, o usuário entrará em um laço de validação dos seus dados de acesso, podendo repetir até que os mesmos estejam corretos ou escolha a opção de referência “Cadastrar Usuário”.

Figura 14 - Diagrama de Sequência Principal

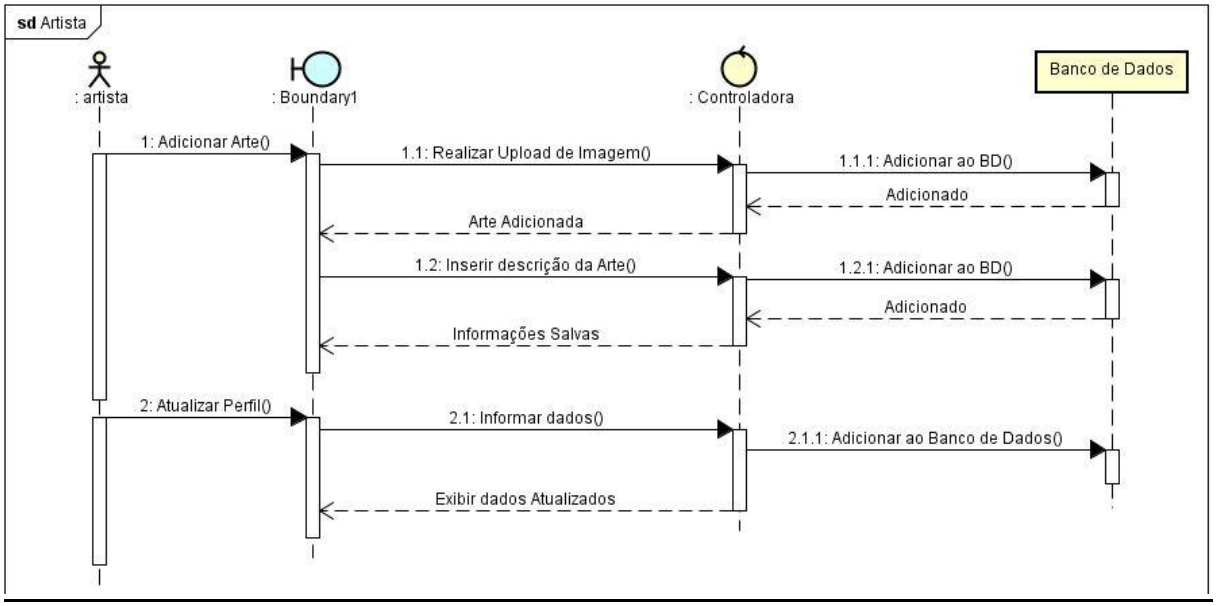


Fonte: Próprio autor.

Caso o usuário for um artista, seguirá para a referência do diagrama “Artista” e este módulo será informado em sequência.

Neste momento o Artista poderá executar algumas funções específicas de seu usuário, como realizar upload de suas artes e adicionar biografias, assim como poderá também manter seus dados cadastrais atualizados conforme mostra a figura 15 – Diagrama de Sequência Artista.

Figura 15 - Diagrama de Sequência Artista

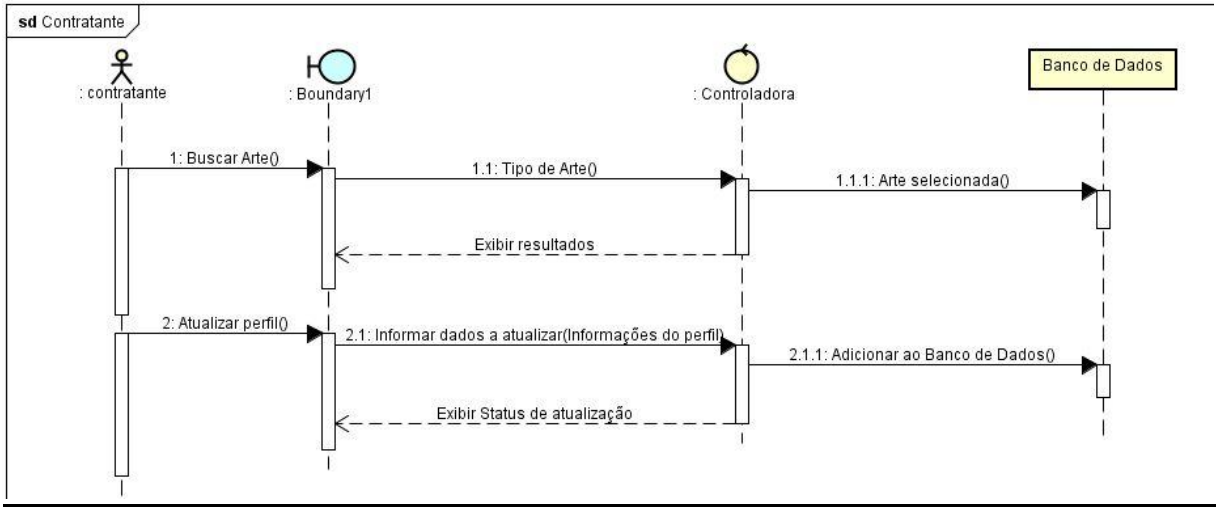


Fonte: Próprio autor.

Caso o usuário for um contratante, seguirá para a referência do diagrama “Contratante” e este módulo será informado em sequência.

Neste momento o usuário poderá executar algumas funções específicas de seu perfil, como realizar buscas do tipo de arte, visualizar contato do artista selecionado, assim como poderá também manter seus dados cadastrais atualizados conforme mostra a figura 16 – Diagrama de Sequência Contratante.

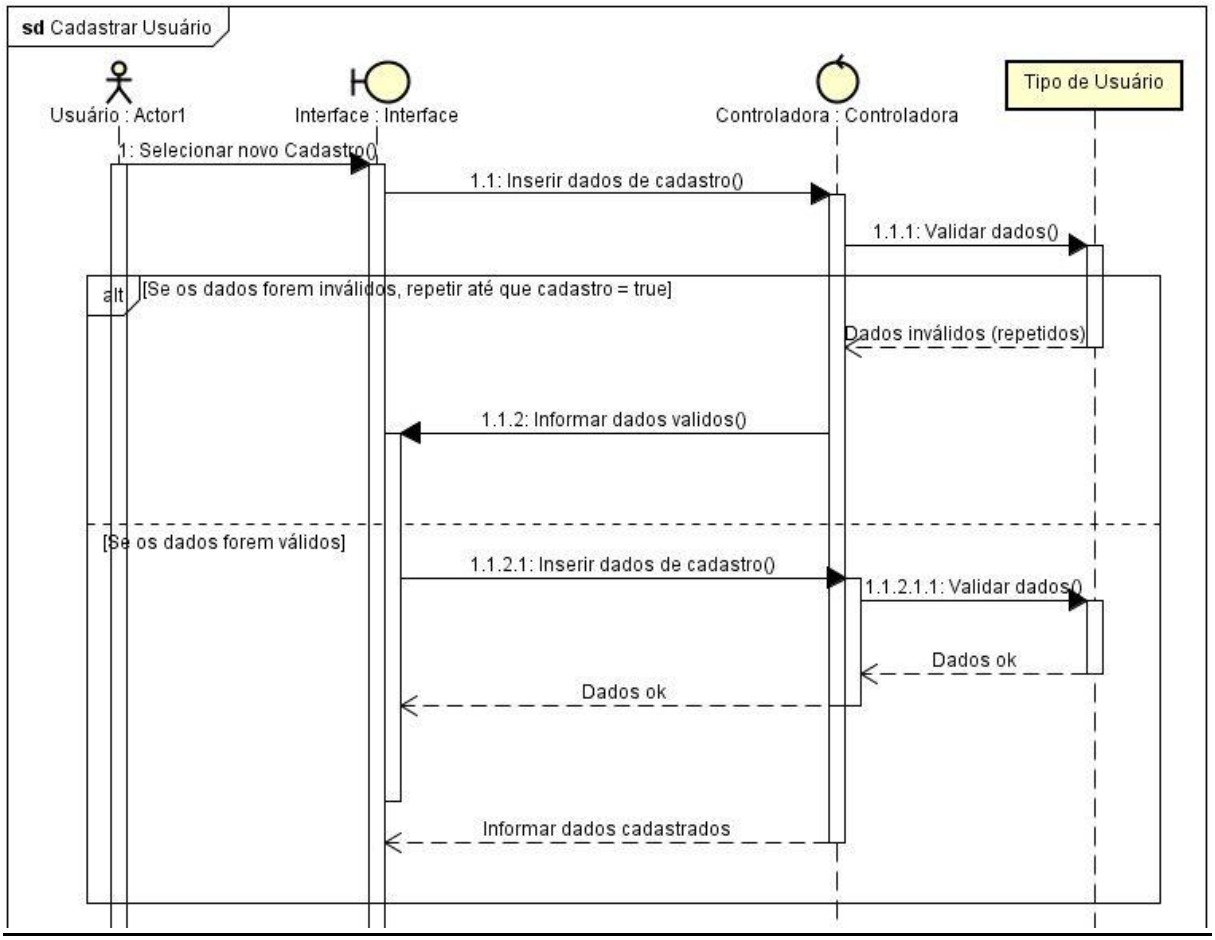
Figura 16 – Diagrama de Sequência Contratante



Fonte: Próprio autor.

Caso o usuário não possuir perfil cadastrado e selecionar esta opção, seguirá para a referência do diagrama “Cadastro” e este módulo será informado em sequência. Neste momento o usuário deverá inserir seus dados cadastrais a serem enviados para o banco de dados.

Figura 17 - Diagrama de Sequência Cadastro

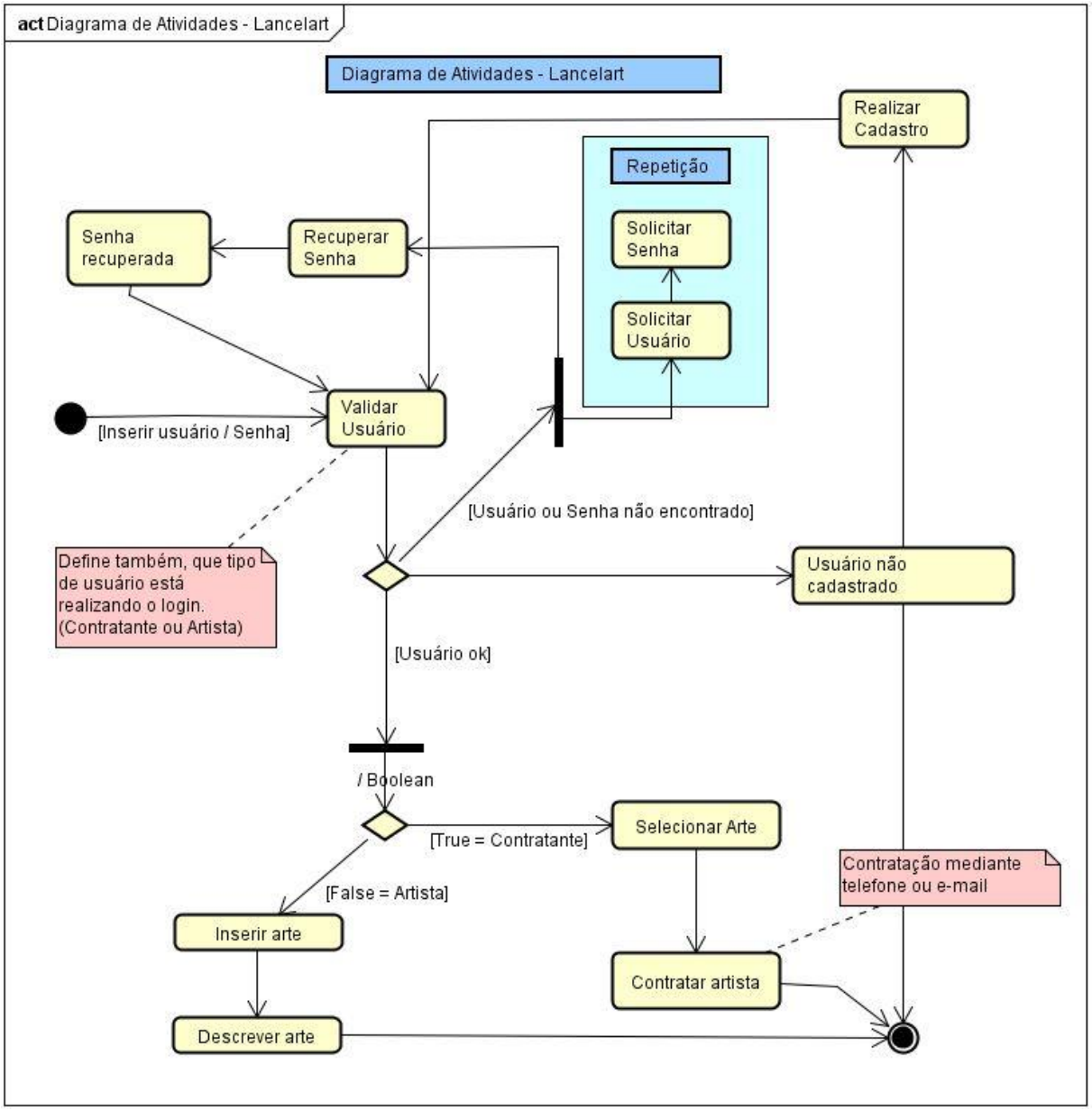


Fonte: Próprio autor.

Diagrama de Atividades (Activity)

É o gráfico de fluxo, para controle de uma atividade para outra, sendo possível também analisar acessos e funções de determinadas rotinas dinâmicas do sistema.

Figura 18 - Diagrama de Atividades.



Fonte: Próprio autor.

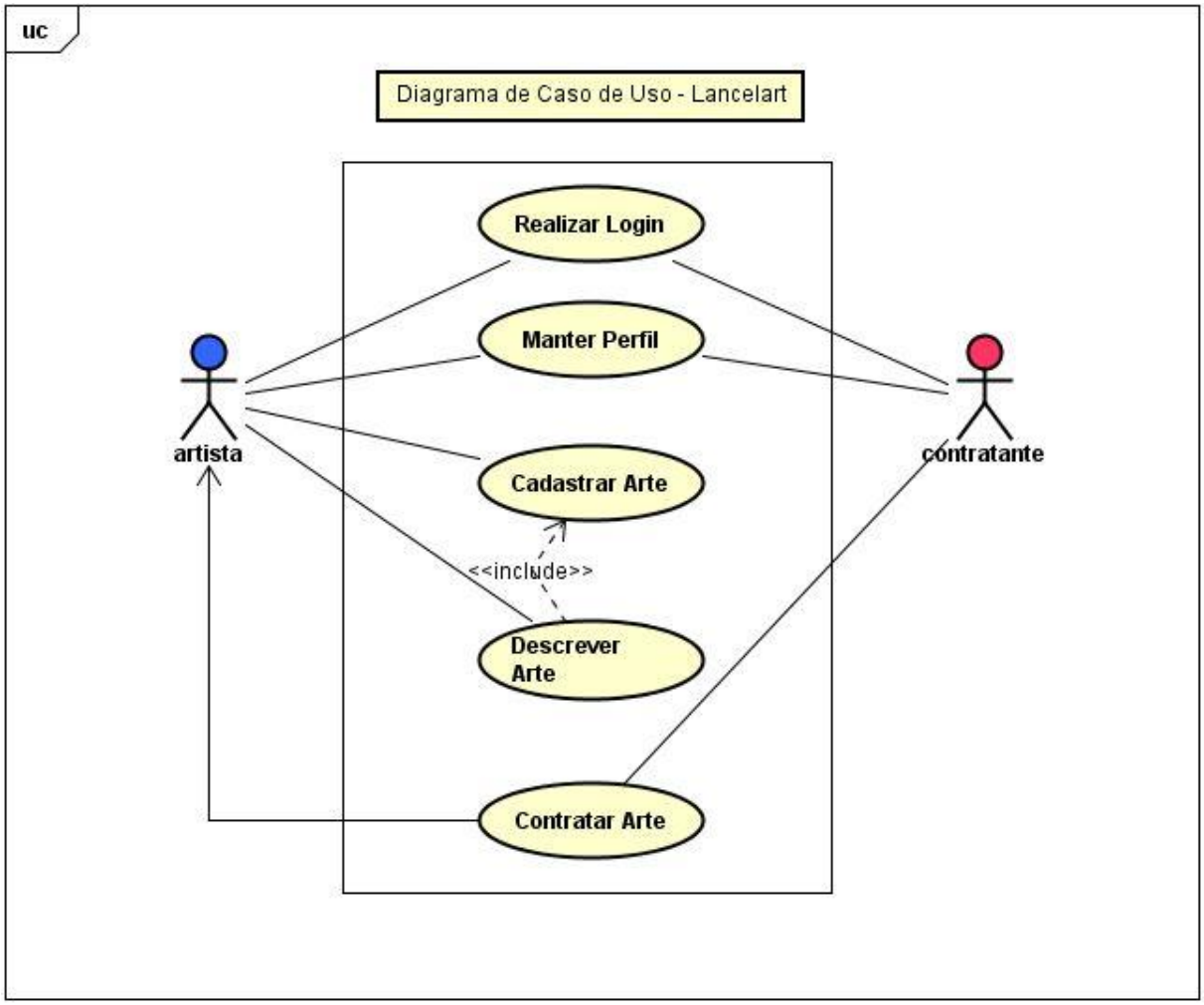
Diagrama de Caso de Uso (Use Case).

É mais específico para validar simplificadamente o levantamento dos requisitos funcionais do sistema em desenvolvimento, como apresentado na figura 19 – Diagrama de Caso de Uso.

O usuário do tipo artista, poderá realizar login, manter seu perfil, cadastrar arte adicionando sua descrição, ou apenas como extensão adicionar uma nova descrição já cadastrada.

Já o contratante poderá realizar seu login, manter seu perfil atualizado, e solicitar a contratação da arte recebendo as informações do artista selecionado.

Figura 19 - Diagrama de Caso de Uso



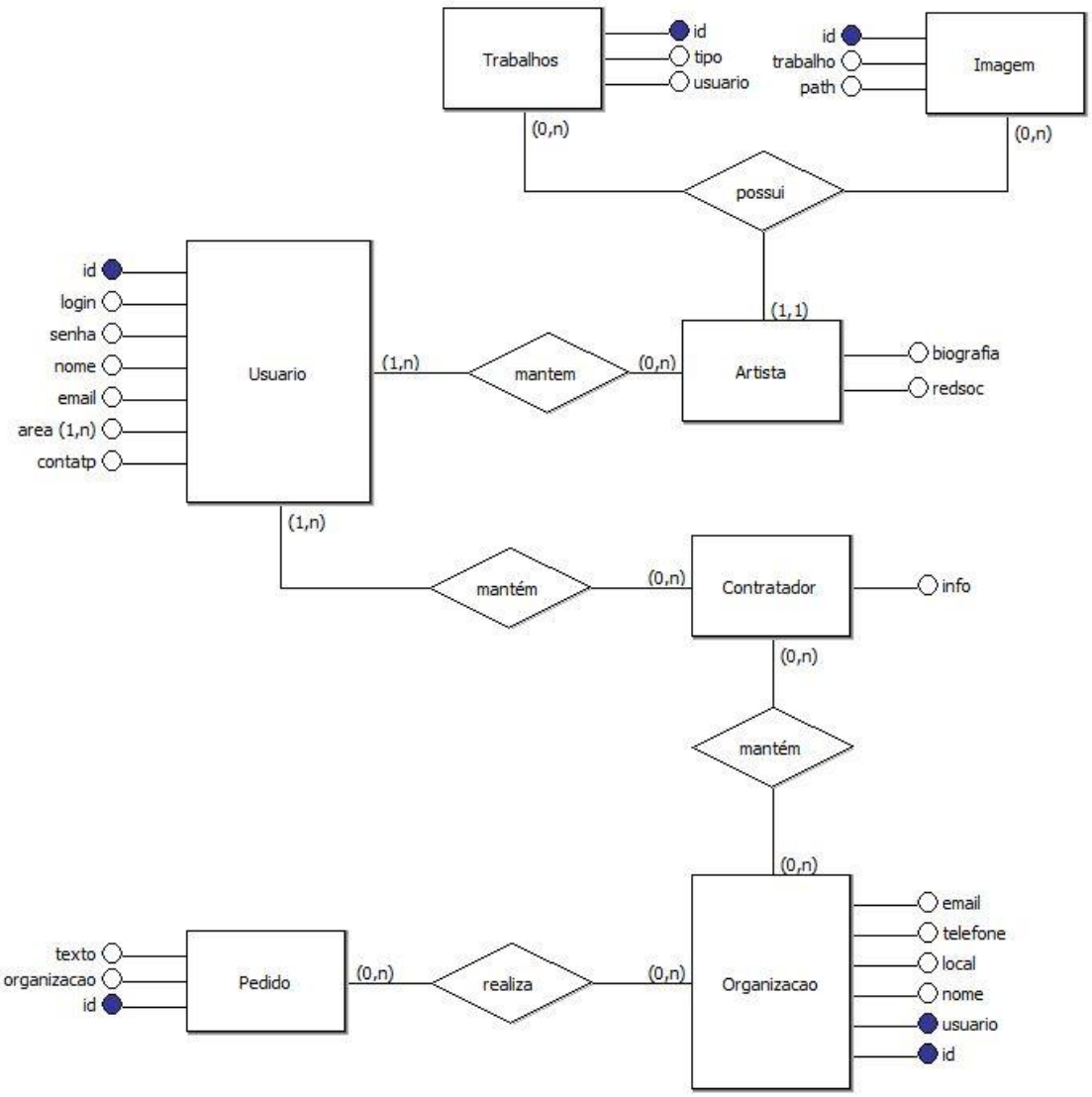
Fonte: Próprio autor.

Banco de Dados (Conceitual, Lógico e Físico)

Para entendimento do banco de dados através de diagramas, foram desenvolvidos os modelos conceituais, lógicos e físicos.

No modelo conceitual poderemos analisar a aproximação do mundo real ao banco de dados, analisando seus pontos conforme a figura 20 – Modelo Conceitual, antes de transforma-lo em lógico.

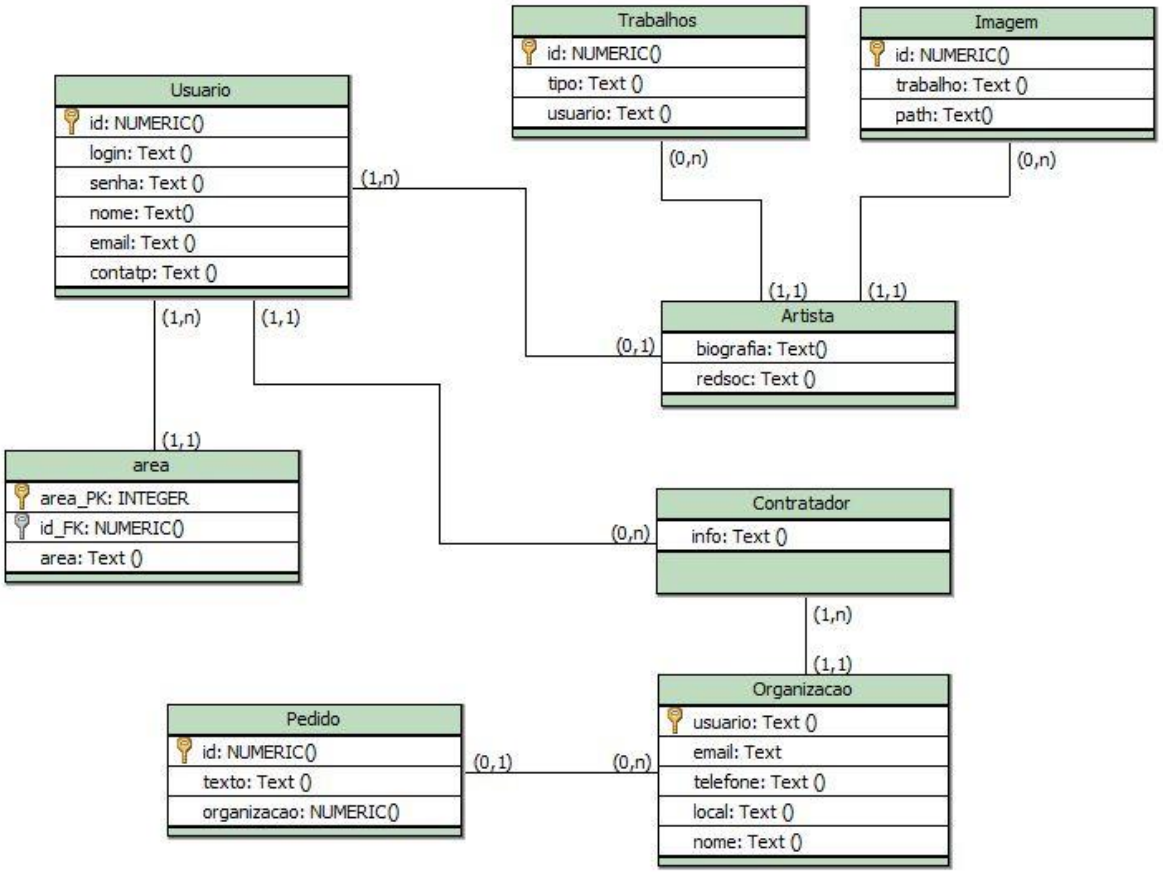
Figura 20 - Modelo Conceitual



Fonte: Próprio autor.

No modelo lógico teremos a estruturação, com base conceitual, do banco de dados em tabelas conforme figura 21 – Conceito Lógico, sendo possível assim transferir esta lógica para modelo físico.

Figura 21 – Conceito Lógico



Fonte: Próprio autor.

No modelo físico temos a programação do banco de dados junto ao código a ser executado, conforme descrito a seguir para criar a tabela Usuário:

```
CREATE TABLE Usuario (
id NUMERIC() PRIMARY
KEY,
login Text (),
senha Text (),
nome Text(),
email Text (),
contatp Text ()
)
REFERENCES Usuario (id)
)
```

Para criar a tabela Trabalhos:

```
CREATE TABLE Trabalhos (
id NUMERIC() PRIMARY
KEY,
tipo Text (),
usuario Text ()
)
```

Para criar a tabela Artista:

```
CREATE TABLE Artista (
biografia Text(),
redsoc Text ()
)
```

Para criar a tabela Contratador:

```
CREATE TABLE Contratador (
info Text ()
)
```

Para criar a tabela Organização:

```
CREATE TABLE Organizacao
(
usuario Text () PRIMARY
KEY,
email Text,
telefone Text (),
local Text (),
nome Text ()
)
```

Para criar a tabela Pedido:

```
CREATE TABLE Pedido (
id NUMERIC() PRIMARY
KEY,
texto Text (),
organizacao NUMERIC()
)
```

Neste caso, conforme informado durante o capítulo de implementação, o PostgreSQL por se tratar de um banco de dados com suporte a orientação ao objeto, podemos apenas criar as chaves primaria e determinar durante a programação realizada com Node quais serão suas heranças e dependências através do atributo inherit.

2.8 Implementação da linguagem

A primeira parte a ser criada em um programa Node é o package.json, onde todos os dados do programa ficam.

Desde o nome do programa, sua versão, descrição, documento principal, autor, tipo de licença e suas dependências. Estas dependências do sistema em questão são mostradas a seguir:

```
"dependencies": {
   "cookie-parser": "^1.4.3",
   "ejs": "^2.6.1",
   "express": "^4.16.3",
   "express-busboy": "^7.0.0",
   "express-ejs-layouts": "^2.4.0",
   "express-promise-router": "^3.0.2",
   "express-session": "^1.15.6",
   "package.json": "^2.0.1",
   "pg": "^7.4.3" }
```

Cada uma dessas dependências tem ao lado sua versão, como o "pg" que ao lado tem sua versão, "7.4.3". Essas dependências então são passadas para o documento principal, também listado no package.json como:

```
"main": "server.js",
```

O server.js então é o documento que será lido para a execução do programa. As dependências por sua vez são passadas pela atribuição das bibliotecas a variáveis dentro do código. Logo no topo do documento, antes de tudo, são inseridas as seguintes dependências:

```
var pg = require('./db'),
        expressLayouts = require('express-ejs-layouts'),
        ejs = require('ejs'),
        express = require('express'),
        bb = require('express-busboy'),
        cookieParser = require('cookie-parser'),
        session = require('express-session')
```

Dentro do *require()* é inserido o módulo a ser usado pela variável. É importante citar uma dependência em específico:

```
pg = require('./db')
```

Diferente dos outros, que só inserem o nome dos módulos, que passarão a ser procurados na pasta *node_modules*, no *filesystem* do programa no momento de execução, a variável “pg” é direcionada para o diretório “db”, que se encontra no mesmo lugar que o documento server.js.

Dentro desse diretório, por sua vez, existe apenas um arquivo, index.js, que contém módulos utilizados pelo sistema. Dentro do index.js, é possível encontrar a dependência “pg”:

```
const Pool = require('pg');
```

E a exportação do módulo a ser usado:

```
module.exports = {
  query: (text, params, callback) => pool.query(text, params, callback)
}
```

De volta ao server.js, após as variáveis serem declaradas, é necessário estabelecer o que cada módulo irá fazer. Primeiro, é inserido o framework a ser utilizado pelo sistema, fazendo possível a comunicação com o servidor. Para o sistema desenvolvido, é usado Express, como visto no código:

```
express = require('express')
```

Essa variável que contém os módulos usados pelo express agora inicializa seu framework em outra variável com o código:

```
var app = express();
```

Agora, todas as comunicações feitas com o sistema web serão passadas pelo express. Após assimilar o framework, é preciso decidir qual será a *‘view engine’*, ou seja, como os dados html serão passados para a web. Para isso, é utilizado o EJS, com o código:

```
app.set('view engine', 'ejs');
```

E em sequência, é inserido um meio de comunicação dos layouts do EJS com o express, com o módulo ‘express-ejs-layouts’:

```
app.use(expressLayouts);
```

Agora temos o framework para o sistema web e um modo de enviar as páginas e suas formatações para o browser. Em seguida, é definido um modo de receber arquivos do cliente para o servidor nos elementos *form* do *html*.

O *Busboy*, podendo fazer o upload tanto de texto quanto imagens, é vital para o funcionamento do sistema *web* e é utilizado com o código:

```
bb.extend(app, {
    upload: true,
    allowedPath: /./
});
```

Dentro de uma função do próprio *Busboy*, é feita a conexão com o *sistema WebApp* e suas configurações inseridas em efeito.

As configurações em questão são a capacidade de upload e os caminhos aceitos para esse upload.

Temos como conversar com o servidor, visualizar as páginas *html* e fazer *upload* de imagens, agora para a permanência de dados durante o acesso, é utilizada duas dependências, a primeira sendo o *cookie-parser*, que habilita a edição de cookies dentro do browser:

```
app.use(cookieParser('secret'));
```

Dentro do *cookieParser*, por medida de segurança, é inserido um “*secret*”, como um *password* para dificultar a abertura do cookie.

Com o cookie criado, é necessário escolher quais dados serão guardados e por quanto tempo. O módulo *Session* então precisa de algumas configurações:

```
app.use(session({
    cookie: { maxAge: 60000 },
    store: sessionStore,
    saveUninitialized: true,
    resave: true,
    secret: 'secret',
    key: '', //id
    nome: '', //nome
    email: '', //email
    ctp: '', //tipo de conta
    flag:''
}));
```

2.9 Os atributos server.js

Cookie: que por sua vez tem seu tempo útil estabelecido em minutos.

Store: Escolhe onde os dados serão guardados, em um ambiente de desenvolvimento, *sessionStore* é o suficiente, mas não é recomendado para o uso efetivo do sistema.

Para a criação do sessionStore, é posto no código é criada a variável que por sua vez faz parte do módulo session:

```
var sessionStore = new
```

SaveUnitialized: Com cada acesso ao sistema web, uma sessão é criada vazia. Cada sessão contém um id, salvando as sessões que continuam vazias, o servidor consegue então economizar espaço e manter o sistema rápido.

Desligado, as sessões vazias continuaram com o mesmo usuário, mesmo vazia, o que possibilita visibilizar vários acessos vindo do mesmo usuário.

Resave: com esse atributo ligado, a sessão é salva mesmo quando o usuário se ausenta do contato com o servidor, mantendo a permanência por diferentes acessos do usuário.

Secret: novamente, é possível estabelecer uma palavra chave para o *cookie*.

key: esse atributo e os próximos a serem apresentados são salvos durante o funcionamento da sessão de usuário.

O atributo *'key'* copia o id do usuário que se encontra no banco de dados.

nome: O atributo 'nome' copia o nome do usuário que se encontra no banco de dados.

email: O atributo *'email'* copia o *e-mail* do usuário que se encontra no banco de dados.

ctp: O atributo 'ctp' copia o contatp do usuário que se encontra no banco de dados.

flag: O atributo *'flag'* é usado durante o funcionamento da sessão para a passagem de mensagens ao usuário.

Agora com todos os módulos em funcionamento, é apenas necessário o roteamento das páginas, que é estabelecido com uma atribuição de uma variável a um arquivo do *filesystem* do sistema.

```
var router = require('./app/routes');
```

Colocar o sistema para usar esse arquivo quando roteamento for necessário:

```
app.use('/', router);
```

E, por último, definir onde os arquivos estáticos que estão sendo utilizados no código, com o uso de uma função do módulo express:

```
app.use(express.static(__dirname + '/public'));
```

Diferente de outras instancias do código, o endereço usado precisa ser estabelecido desde a raiz do servidor, usando a variável ‘__dirname’ que localiza onde o arquivo em executado se encontra, e então tem seu atributo como o caminho até o arquivo.

Agora com isso, só é necessário apenas a inicialização do sistema, com um pedido de app.listen, que recebe uma porta de acesso e uma função.

```
app.listen(port, function(){
console.log ('App iniciado na porta 8080');
});
```

A porta que está sendo utilizada tem seu valor na variável *port,* que é inicializada com:

```
var port = process.env.PORT || 8080;
```

Deste modo aceita uma variável de ambiente ou a porta 8080. E por fim, a função executa ao iniciar o *listen* é a mensagem “App iniciado na porta 8080”.

3 RESULTADOS

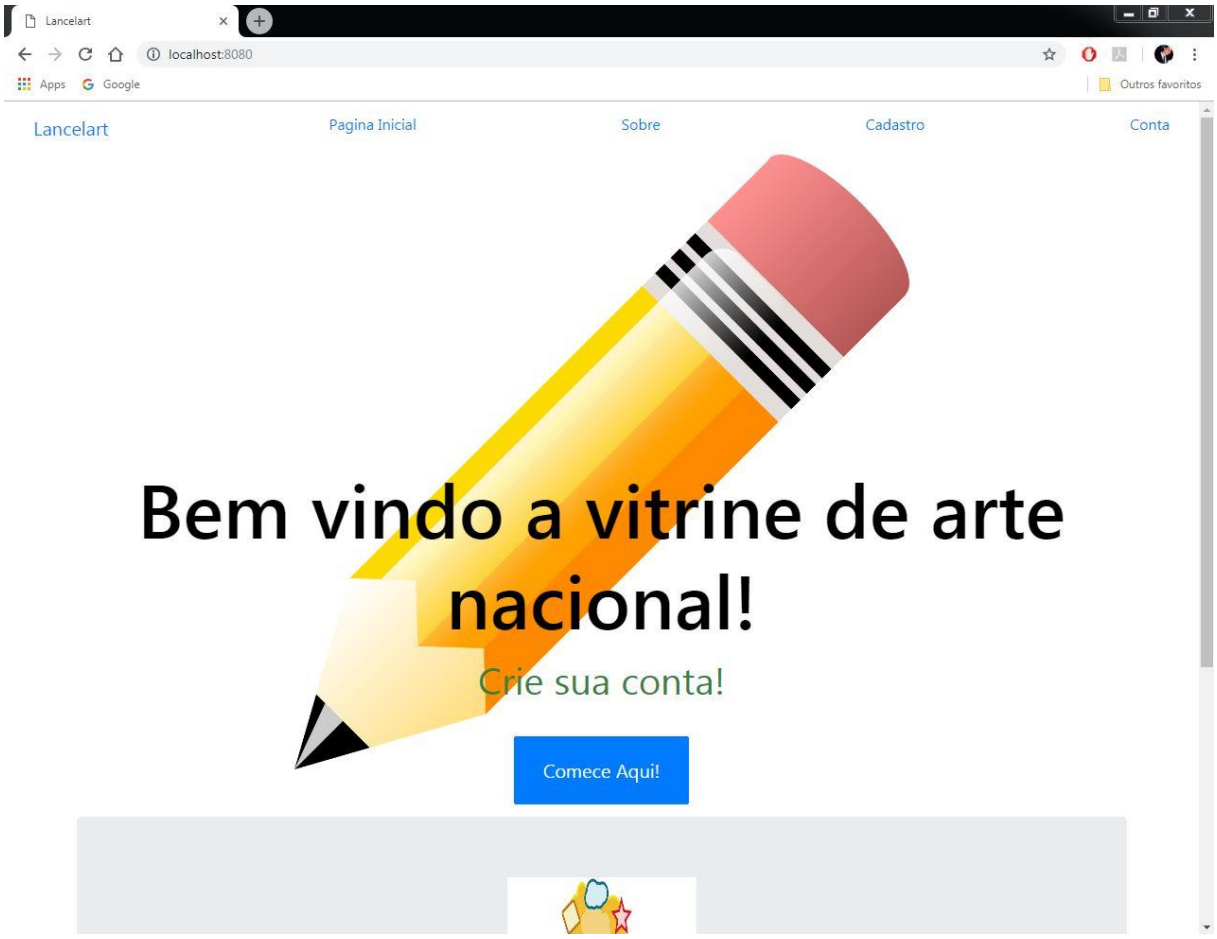
Neste tópico, será abordado a respeito da interface do sistema e do desenvolvimento até o momento da execução deste relatório, contudo, novas versões do sistema estão sendo lançadas com o tempo.

3.1 Apresentação das telas do sistema

A página inicial do sistema, temos uma mensagem de boas-vindas junto a um botão “Comece Aqui! ”, que leva o usuário diretamente a área de cadastro do WebApp.

No momento do desenvolvimento deste projeto, possuí uma tela simples e objetiva, destinada a usuários do mercado a ser atingido.

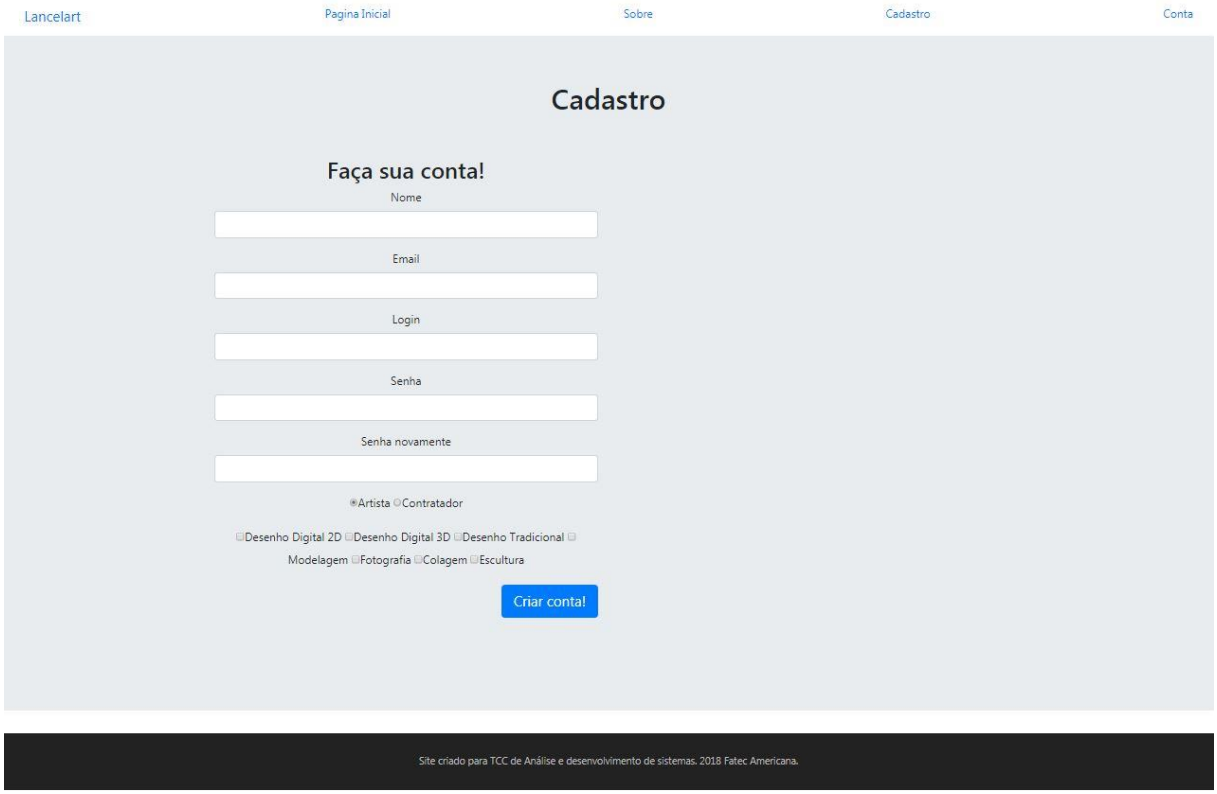
Figura 22 - Página Inicial



Fonte: Próprio autor.

Logo temos também a página de cadastro, dos quais devem ser preenchidos os campos Nome, E-mail, Login, Senha, Senha novamente, a escolha entre o perfil Artista e Contratador, assim como os tipos de arte que você busca ou desenvolve (Desenho Digital 2D, Desenho Digital 3D, Desenho Tradicional, Modelagem, Fotografia, Colagem ou Escultura).

Figura 23 - Página de Cadastro

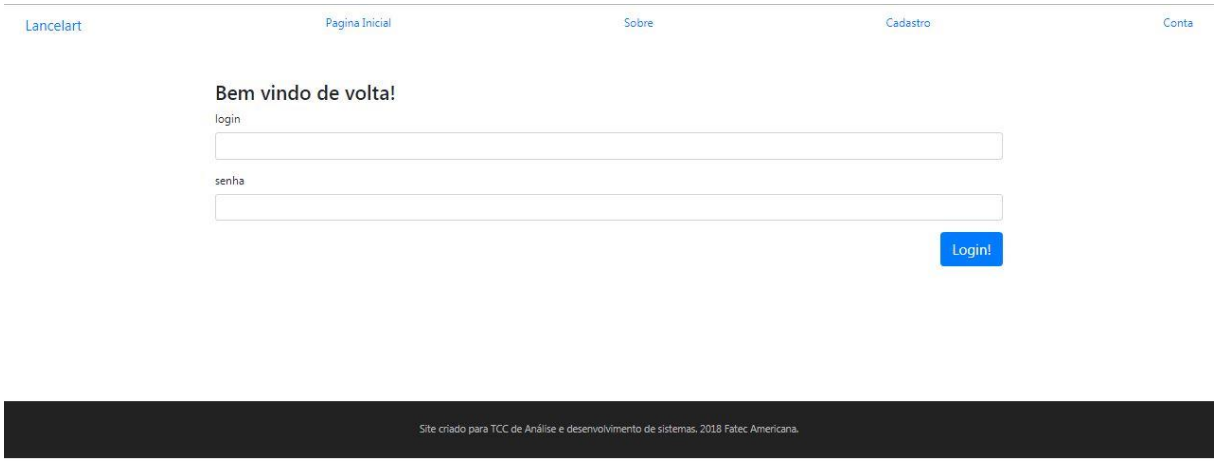


Fonte: Próprio autor.

Após inserir seus dados e validá-los, poderá seguir para tela de Login.

Na tela de login, será necessário inserir e validar os dados de acesso ao sistema.

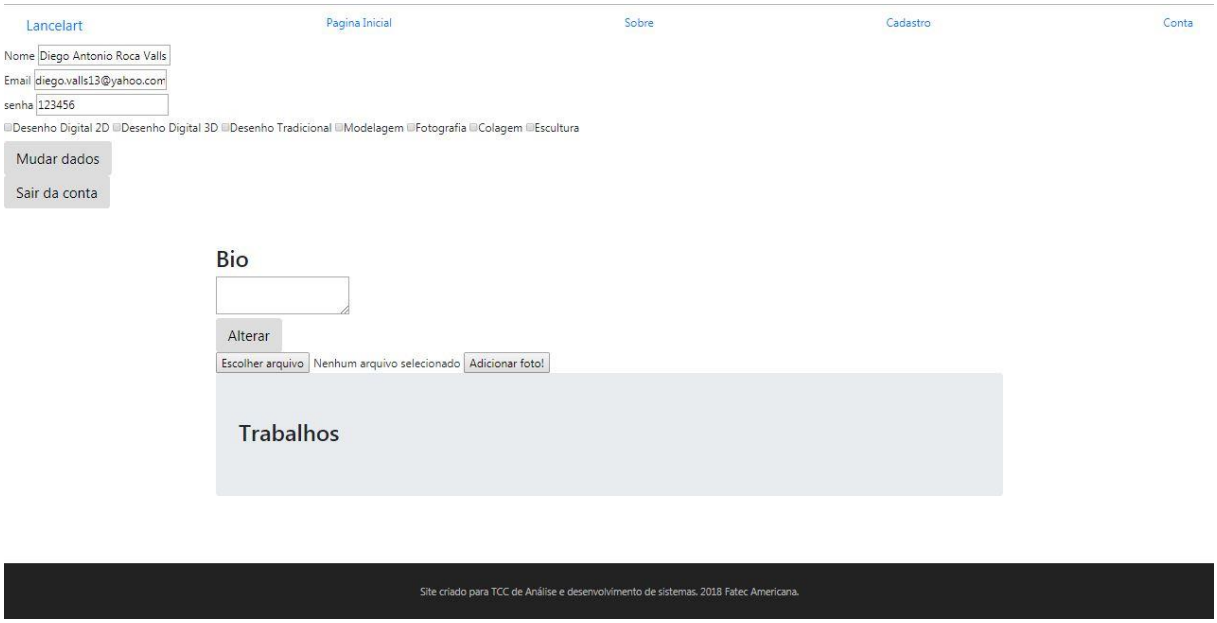
Figura 24 - Página de Login



Fonte: Próprio autor.

Após realizar o login, a página do usuário (artista neste exemplo) será exibida, e suas funções disponibilizadas.

Figura 25 - Página de usuário artista



Fonte: Próprio autor.

No usuário artista, podemos notar que já é possível adicionar sua Biografia, assim como imagens para exibição na vitrine da página inicial.

[INSERIR RELATÓRIO DE USO DO SISTEMA]

4 CONSIDERAÇÕES FINAIS

O projeto aqui desenvolvido buscou apresentar uma solução para uma das grandes demandas do mercado de marketing digital nacional. O objetivo foi o de colaborar para atender artistas e contratantes de serviços de arte digital, de maneira simples e com baixo orçamento para ambos, se tratando da utilização do sistema propriamente dito, eliminando assim o uso de diversas ferramentas separadas já existentes e agregando uma unificação a estes profissionais.

Neste desenvolvimento, novas linguagens de programação foram estudadas e utilizadas, e além de modernas estão em alta no mercado neste momento.

Acredita-se que neste projeto, não há nada que seria alterado para sua execução, mesmo linguagens de programação utilizadas, quanto ferramentas e metodologias aplicadas.

Para implementações futuras, está destinado a este projeto uma estratégia de marketing e buscar soluções para faturamento e cobrança de serviços junto a implementação de um novo designer mais moderno e atrativo em sua estrutura front-end.

O aplicativo também incluirá uma área para a divulgação de cargos em aberto de contratantes.

REFERÊNCIAS

ANDERSON, David J. **Essential Kanban Condensed.** First digital version. Pg 1. – “What is Kanban?” Versão 28 de Julho de 2016.

“Como saber se o Scrum é o melhor método para sua equipe? ”, Disponível em: <https://blog.trello.com/br/tutorial-scrum> Acesso em: 20 de novembro de 2018, às 18:17.

Github, Disponível em: <https://github.com/> Acesso em: 10 de outubro de 2018; às 19:25;

NODEJS, Disponível em: <https://nodejs.org/en/about/> Acesso em: 10 de Junho de 2018, as 22:32;

NODE, PostgreSQL, Disponível em: <https://node-postgres.com/> Acesso em:10 de Junho de 2018, as 22:44;

POSTEGRESQL, Disponível em: <https://www.postgresql.org/about/> Acesso em: 10 de Junho de 2018, as 21:57;

Package.json, Disponível em: <https://docs.npmjs.com/files/package.json> Acesso em: 02 de novembro de 2018; às 11:56;

Scrum, Disponível em: <https://www.desenvolvimentoagil.com.br/scrum/> Acesso em: 20 de novembro de 2018, às 18:39.

VIM, Disponível em: <https://www.vim.org/> Acesso em: 12 de novembro de 2018; às 11:44;