UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

Renê Luiz dos Santos Baldissera

Desenvolvimento de uma Ferramenta para Gestão de Carreira e Vendas voltada ao Mercado Audiovisual

Florianópolis
2019

Renê Luiz dos Santos Baldissera

Desenvolvimento de uma Ferramenta para Gestão de Carreira e Vendas voltada ao Mercado Audiovisual

Relatório submetido à Universidade Federal de Santa Catarina como requisito para a aprovação na disciplina **DAS 5511: Projeto de Fim de Curso** do curso de Graduação em Engenharia de Controle e Automação.
Orientador(a): Prof. Marcelo Stemmer

Florianópolis
2019

Renê Luiz dos Santos Baldissera

Desenvolvimento de uma Ferramenta para Gestão de Carreira e Vendas voltada ao Mercado Audiovisual

Esta monografia foi julgada no contexto da disciplina DAS5511: Projeto de Fim de Curso e aprovada na sua forma final pelo Curso de Engenharia de Controle e Automação.

Florianópolis, 08 de Fevereiro de 2019

Banca Examinadora:

Prof. Marcelo Stemmer
Orientador na Empresa
UFSC

Prof. Marcelo Stemmer
Orientador no Curso
Universidade Federal de Santa Catarina

Prof. Rodrigo Saad
Avaliador
Universidade Federal de Santa Catarina

Anelize Zomkowski Salvi
Debatedor
Universidade Federal de Santa Catarina

Gustavo dos Santos Gonçalves
Debatedor
Universidade Federal de Santa Catarina

Dedico esse trabalho a minha família e a meus amigos.

AGRADECIMENTOS

Agradeço aos amigos Pablo Andreus, Victor Skaetta, Rong Jiarui, Michael Camilo, Nilmar Júnior e principalmente ao Leonardo Werk, por fazerem desses anos todos na faculdade um processo bem mais legal.

RESUMO

O setor audiovisual brasileiro apresentou um grande crescimento nos últimos anos mesmo a uma forte crise econômica enfrentada no país (e também no mundo). Grande parte dos profissionais que trabalham no setor podem ser considerados Freelancers, ou seja, que não possuem de fato um contrato em regime CLT com as empresas do setor e são chamados para projetos específicos, de acordo com a necessidade dessas empresas. Isso faz com que esse tipo de profissional apresente algumas dificuldades tanto para levar um projeto adiante, quando são responsáveis por grande parte deles, como para conseguir projetos novos, i.e., o que fazer quando um projeto acaba, e muitos acabam confiando simplesmente em indicações do que na própria venda do seu trabalho. Uma pesquisa realizada no ano de 2017, por plataformas de ofertas de trabalhos para Freelancers, elencou as maiores dificuldades enfrentadas por esses profissionais. O objetivo desse trabalho é achar uma maneira de solucionar algumas dessas dificuldades, porém com o mercado audiovisual em foco. Para isso, como trabalho de conclusão de curso de Engenharia de Controle e Automação, se deu o desenvolvimento de um aplicativo SAAS (Software as a Service) com funcionalidades específicas para o profissional Freelancer do setor audiovisual.

Palavras-chave: Audiovisual. Software as a Service. Freelancers.

ABSTRACT

The Brazilian audio-visual market presented an extreme growth in the last years, even with a strong economic crises faced by the country (and over the world as well). The majority of professionals who work on this sector can be considered Freelancers, which means that those professionals don't have a regular contractual job with the companies of the market and they are called for specific projects, aligned with the necessities of these companies. With that in mind, those professionals present some difficulties to track the projects they are in, specially when they are responsible for a big percentage regarding one project, as well as getting new projects to work on, i.e., what can they do after finishing a project, and many freelancers just trust on clients appointing them to other clients, instead of actively sell their services. A research made on 2017, by online freelancer jobs platforms, pointed out the major difficulties that freelancers face and among them are keep track of their projects and defining a career direction. The objective of this assignment is to find a way to help with some of those difficulties, but with the audio-visual market in focus. For that, as a Control and Automation Engineering thesis, the development of a SAAS (Software as a Service) with specifically tools to help the audio-visual freelancer professional will be show here.

Key-words: Audiovisual. Software as a Service. Freelancers.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ANC – Agência Nacional do CInema

APRO – Associação Brasileira da Produção de Obras Audiovisuais

SEBRAE – Serviço Brasileiro de Apoio às Micro e Pequenas Empresas

DTL – Django Template Language

CBV – Class Based Views

CRUD – Creat, Retrieve, Update, Delete

SUMÁRIO

1 INTRODUÇÃO

1.1 O Audiovisual no Brasil

O setor Audiovisual brasileiro cresceu muito desde a década de 2000 até os dias de hoje. Segundo estudo da ANCINE (Agência Nacional do CInema) em 2014, atividades econômicas do setor audiovisual foram responsáveis por uma geração de valor de R$ 24,5 bilhões na economia, como pode ser observado na Figura 1. Em 2007, o montante era de apenas R$ 8,7 bilhões. Isso é praticamente o triplo em apenas 7 anos, um aumento expressivo frente a um cenário de crise econômica no país, entre essas datas.

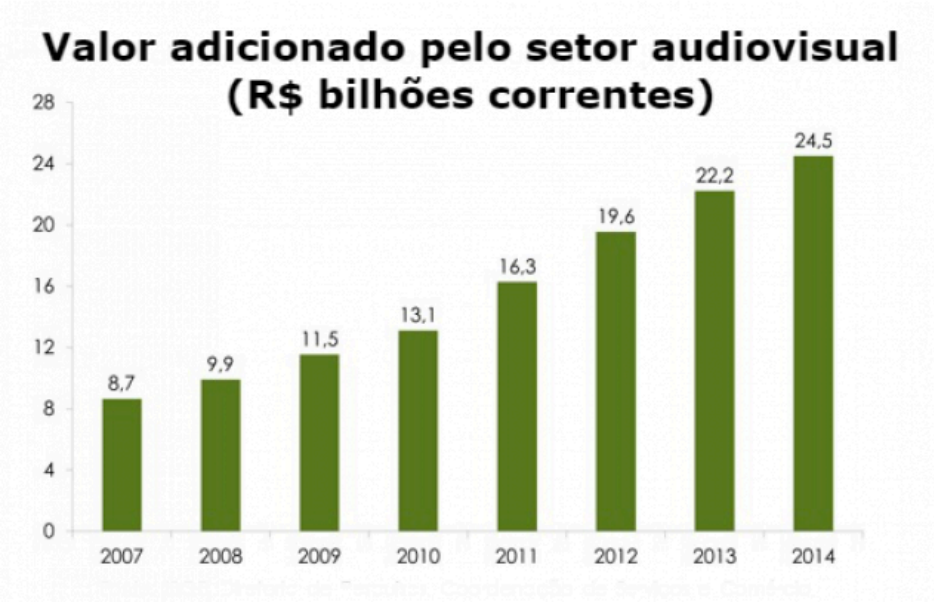

Figura 1 - Valor adicionado pelo setor audiovisual (ANCINE, 2014)

Segundos estudos da Apro (Associação Brasileira da Produção de Obras Audiovisuais) e do Sebrae (Serviço Brasileiro de Apoio às Micro e Pequenas Empresas), atualmente, estão inscritas 7.312 produtoras na Ancine. Os dados da agência também mostram que 56,7% das produtoras são classificadas como independentes, e 66% têm acima de cinco anos de operação.

O estudo também aponta que as produtoras independentes possuem poucos funcionários CLT e, em média, um número quatro vezes maior de freelancers.

1.2 Freelancers no Setor Audiovisual

Como observado anteriormente, o número de freelancers trabalhando para o setor representa uma quantidade bastante expressiva, visto que eles representam um número 4 vezes maior do que o de funcionários CLTs em mais da metade das produtoras registradas na ANCINE.

O contraste entre um estável emprego assalariado em regime CLT e um regime inconstante, como o do Freelancer, revela alguns desafios que essa classe empregatícia precisa enfrentar. Algumas das dificuldades que podem ser elencadas vão desde a organização financeira que um freelancer precisa ter (muito maior que um empregado CLT) até o contato frequente com clientes.

Em pesquisa realizada em 2017 pela Rock Content, We Do Logos e 99jobs, 9.561 pessoas responderam questionamentos sobre o mercado freelancer. Ao serem questionadas a respeito de suas maiores dificuldades, 59,5% dos participantes afirmaram passar pela dificuldade de conquistar novos clientes.

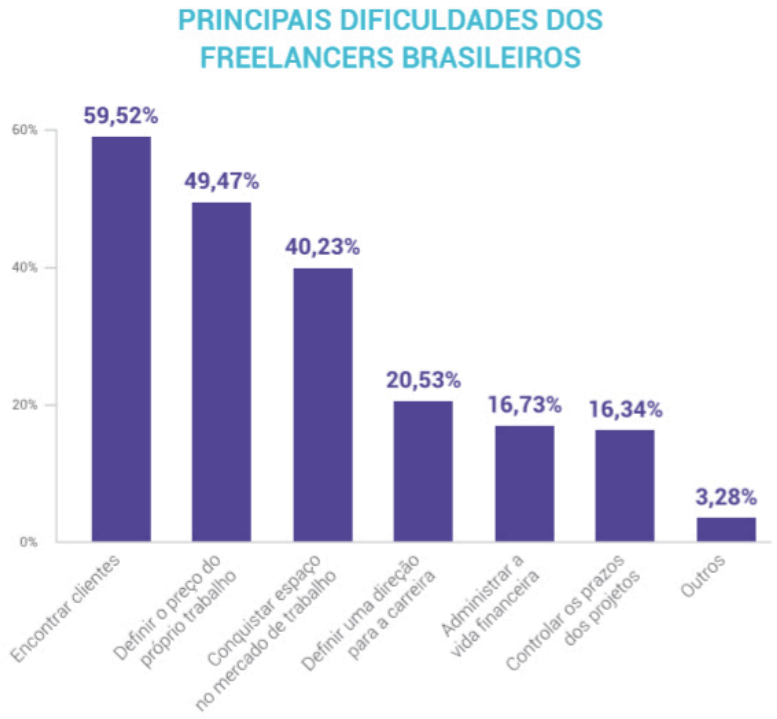

Figura 2 - Principais Dificuldades dos Freelancers Brasileiros

Como pode ser observado na Figura 2 outros problemas também foram elencados, como conquistar espaço no mercado de trabalho, administrar a vida financeira, controlar o prazo dos projetos e a dificuldade de definir o preço do próprio trabalho, um problema quase tão grande quanto o de conquistar novos clientes, com 49,47% dos pesquisados afirmando que passam por isso.

1.3 Solução desses problemas

O objetivo desse trabalho é atacar esses problemas tão comuns aos Freelancers, através de uma ferramenta que permita uma solução para cada um desses de forma independente, mas que no final todos trabalhem juntos para um melhor entendimento da vida profissional do usuário. O objetivo ainda é o desenvolvimento de um MVP (mínimo produto viável), ou seja, uma versão simples, mas funcional da ferramenta final que se denominará Freeland.

O foco, no entanto, será uma ferramenta para o profissional Freelancer do setor Audiovisual. Portanto as *features* presentes nessa ferramenta serão específicas para o uso desse profissional.
O objetivo é resolver 5 dos problemas apresentados na pesquisa demonstrada na seção 1.2:

1) Dificuldade de Encontrar Clientes;
2) Definir o preço do Próprio Trabalho;
3) Definir uma direção para a Carreira;
4) Conquistar espaço no Mercado de Trabalho;
5) Controlar os prazos dos Projetos.

Os problemas 1 e 5 não serão abordados nesse trabalho. Para a questão de encontrar clientes e o problema de administrar a vida financeira existem inúmeras ferramentas conhecidas com CRMs e ferramentas de controle financeiro, muitas gratuitas, já há muito tempo no mercado e que atendem perfeitamente um profissional do setor audiovisual. O foco do trabalho será em criar ferramentas

novas, que não são vistas com frequência, para esses outros problemas. No futuro uma implementação de uma *feature* de controle financeiro pode ser integrada a essa ferramenta.

As soluções propostas para a resolução desses problemas terão como base algoritmos utilizados e que serão implementados através de programação em linguagem Python, junto com framework Django, que serão explicados no próximo capítulo.

1.4 Metodologia

Passo 1 - Entendimento do Problema

A identificação do problema se deu a partir de conversas informais com os mais diversos freelancers de variados meios de atuação. Para corroborar as conversas informais se utilizou da leitura de pesquisas a respeito das maiores dificuldades dos freelancers no cenário brasileiro, como pode ser visto na seção 1.2 desse capítulo.

Para complementar, utilizou-se da experiência profissional do autor, que trabalha como freelancer no ramo audiovisual por cerca de 2 anos e compreende todas essas dificuldades apresentadas, o que também serviu com motivação para a criação da ferramenta.

Passo 2 – Especificação das Necessidades (Sugestões)

Após identificar os problemas é possível delimitar um escopo de necessidades ou requisitos que o projeto abordaria. Esse passo gera uma lista de necessidades a serem resolvidas e com isso pode se detalhar casos de uso da aplicação, assim como especificações do projeto.

As especificações do software serão planejadas usando alguns dos diagramas fornecidos pela Linguagem de Modelagem Unificada (UML), que permite descrever quais serão os elementos, características e comportamento da ferramenta através de desenhos e textos.

Passo 3 – Desenvolvimento

Com os casos de uso definidos se pode por em prática a implementação das funcionalidades com o objetivo de chegar a um MVP, ou seja, um Mínimo Produto Viável. Nessa parte do trabalho será botado em pratica todos os conceitos teóricos aprendidos e apresentados para a construção do projeto.

Passo 4 - Avaliação e Conclusão

Este passo teve por objetivo averiguar o que foi desenvolvido, o produto final frente a um conjunto de medidas de desempenho e as necessidades encontradas. Essas medidas e necessidades são diretamente relacionadas aos objetivos geral, específicos e proposição de valor do trabalho.

1.5 – Capítulos

O presente trabalho foi dividido em seis capítulos, abordando todos os conceitos teóricos e práticos necessários para o desenvolvimento e implementação do projeto. Os capítulos foram divididos conforme os passos da metodologia apresentada anteriormente.

O primeiro capítulo é referente à introdução do trabalho, explicando o problema a ser solucionado e os objetivos do projeto.

No capítulo 2 serão especificadas as necessidades que devem ser atendidas pela plataforma e seus casos de uso.

O capítulo 3 falará sobre a fundamentação teórica e explicará as ferramentas utilizadas para a implementação desse projeto o porquê da escolha das mesmas.

No capítulo 4 será apresentado o desenvolvimento em si do projeto. Serão mostrados os problemas enfrentados e as soluções propostas e como as *features* da ferramenta foram desenvolvidas.

No capítulo 5 serão mostrados os resultados obtidos com o MVP, onde profissionais conhecidos pelo autor do trabalho tiveram a chance de usar o aplicativo em suas máquinas e darem suas opiniões.

Por fim, no sexto e último capítulo, será apresentado a conclusão de todo o projeto e perspectiva para as próximas etapas da ferramenta e os novos *features* que possam ser incluídos no futuro.

2 ESPECIFICAÇÃO DAS NECESSIDADES

Nesse capítulo serão definidas quais são as ideias de solução para alguns dos problemas identificados no Capítulo 1 e posteriormente os casos de uso para cada uma delas.

Conforme apresentado os 4 problemas que serão atacados são os seguintes:

1) Definir o preço do Próprio Trabalho;
2) Definir uma direção para a Carreira;
3) Conquistar espaço no Mercado de Trabalho;
4) Controlar os prazos dos Projetos.

Assim o Freelancer possui 4 casos de uso como pode ser observado na Figura 3:

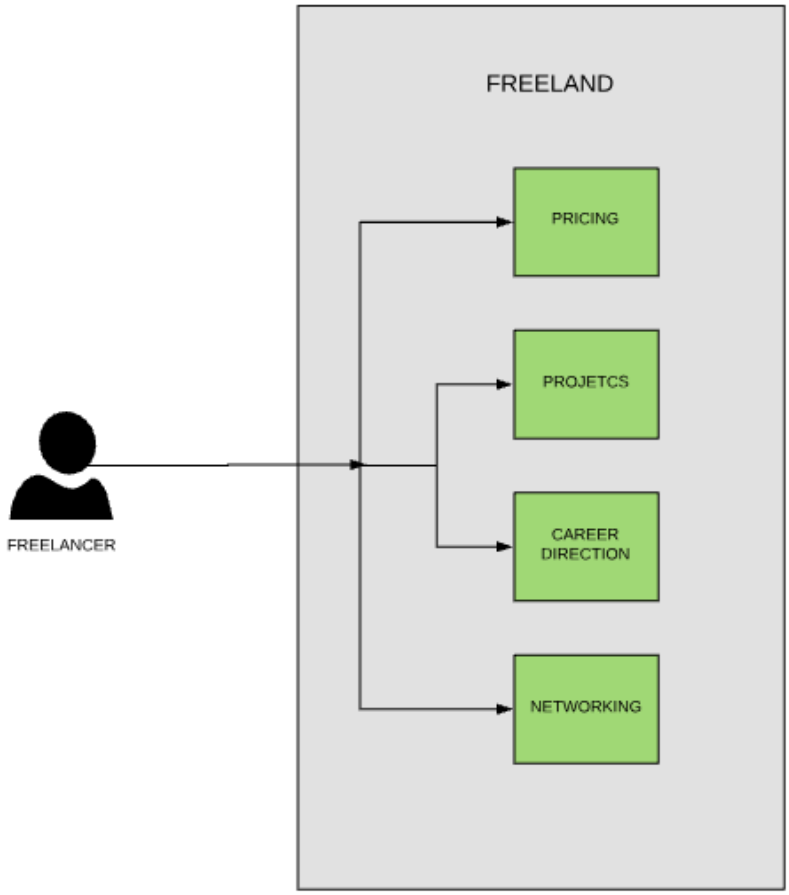

Figura 3 - Casos de Uso do App Freeland

2.1. Pricing - Definindo o preço do próprio trabalho

A solução para esse problema está no desenvolvimento de um algoritmo para se ter uma média e um valor sugerido do preço a ser cobrado pelo Freelancer. Os dados usados são o seguinte:

Preço Médio dos Freelancers Concorrentes
Experiência dos Freelancers Concorrentes (Iniciante, Intermediário, Profissional)
Experiência do Freelancer usuário do sistema (Iniciante, Intermediário, Profissional)

Os dados deverão ser buscados pelo usuário através de pesquisas no seu nicho de trabalho.

2.1.1 Caso de Uso UC01: Pricing

O Freelancer clica na aba Pricing e se depara com uma tela onde pode preencher alguns campos, como pode ser observado na Figura 4.

Fluxo Primário de Eventos:

1. O Freelancer preenche as informações necessários (preços, experiência dos outros Freelancers e seu próprio nível);
2. O sistema analisa os dados, processa o algoritmo e retorna na tela um preço sugerido de diária.

Fluxo de Exceção: O usuário escreve dados não aceitos pelo sistema (como símbolos ou letras)

1. O sistema informa ao Freelancer através da interface gráfica que os dados não são válidos;
2. O Freelancer volta ao passo 1 e preenche o formulário novamente.

Nas Figuras 5 e 6 estão representados, respectivamente, o diagrama de sequência e o diagrama de atividade desse caso de uso.

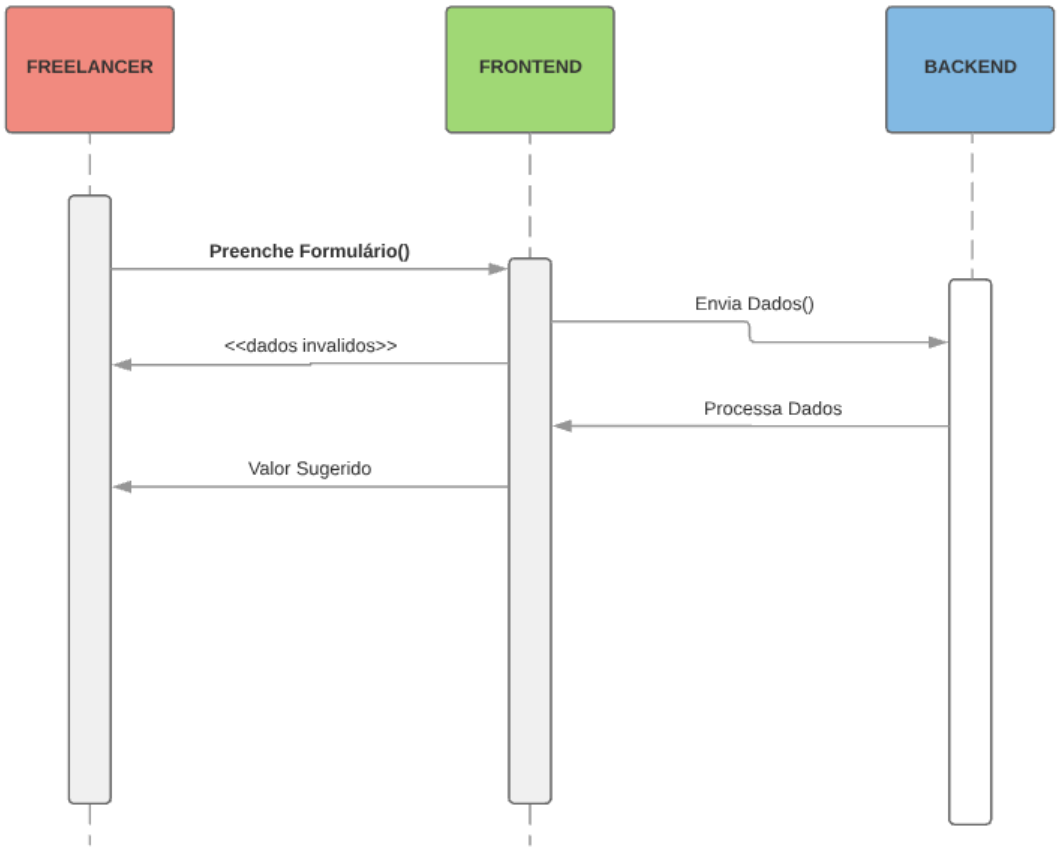

Figura 4 - Caso de Uso UC01

2.2. Projects - Controlando os prazos dos projetos

Ferramenta onde se possa criar projetos e elencar as fases dos mesmos. Permite a criação de deadlines e checklists. Ao criar um projeto o usuário irá definir uma classificação para esse projeto de acordo com o nicho e o nível de importância para sua carreira. Aqui o Freelancer também terá a oportunidade de elencar se um projeto foi finalizado ou não. Esses dados servirão de dados suporte para as outras duas ferramentas a seguir. Se já existirem projetos finalizados o Freelancer pode clicar em algum deles para saber como foi o andamento do mesmo e alterar o que quiser.

2.2.1 Caso de Uso UC02: Create New Project

Ao clicar na aba Projetos, se existir ao menos um projeto criado, uma lista contendo o(s) projeto(s) irá ser exibida e haverá a possibilidade de clicar em algum projeto para ver o andamento do mesmo, ou então criar um projeto novo.

Fluxo Primário de Eventos:

1. O Freelancer clica em criar novo projeto;
2. Um formulário com as informações do projeto é aberto e o freelancer completa esse formulário e o salva;
3. O sistema salva o projeto e mostra os dados do mesmo.

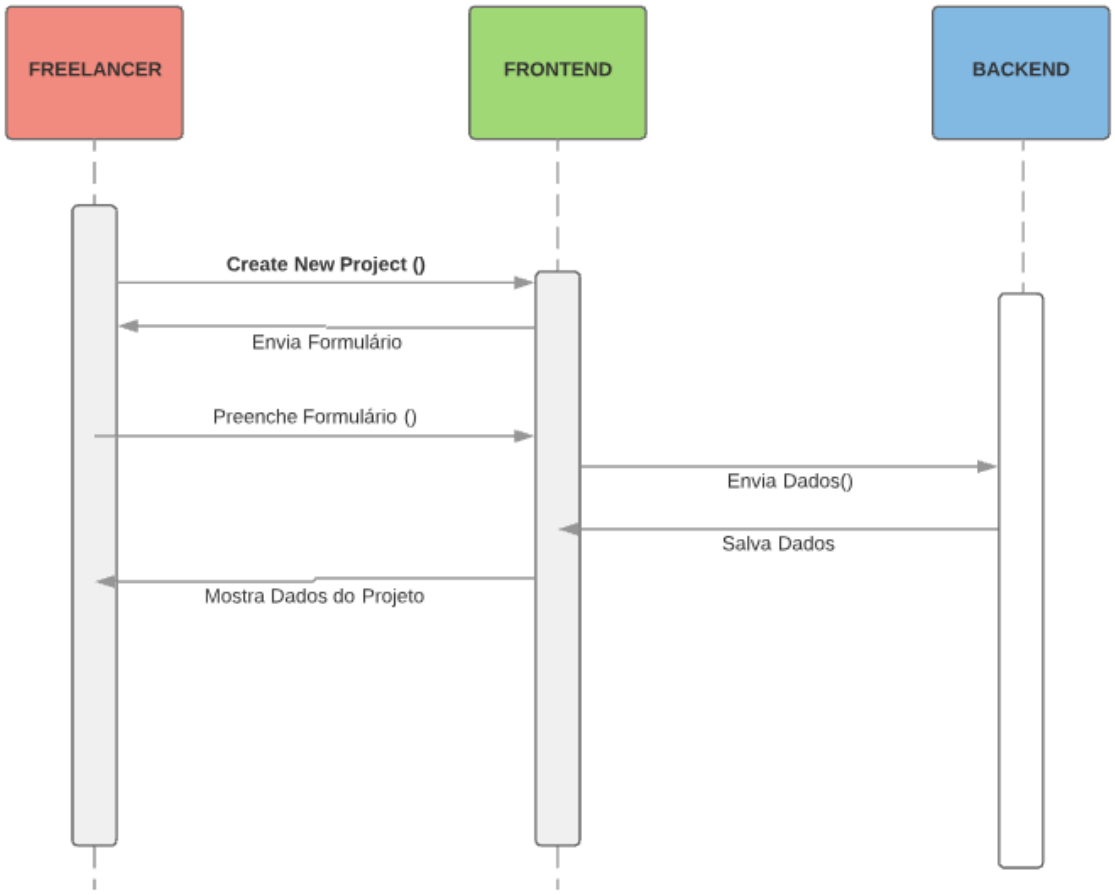

Figura 5 - Caso de Uso UC02

2.2.2 Caso de Uso UC03: Manage Project

Com uma lista de projetos já criados, abertos ou finalizados, o Freelancer pode clicar em algum deles para acompanhar o andamento do mesmo.

Fluxo Primário de Eventos:

1. O Freelancer clica em um Projeto;
2. Uma tela com todas as informações daquele projeto abre;
3. O Freelancer pode alterar o que quiser naquele projeto e incluir novas observações e tarefas e depois salvar. Além disso o Freelancer também pode Finalizar um Projeto ou Excluí-lo.
4. O sistema salva o projeto e volta para a tela de Projetos.

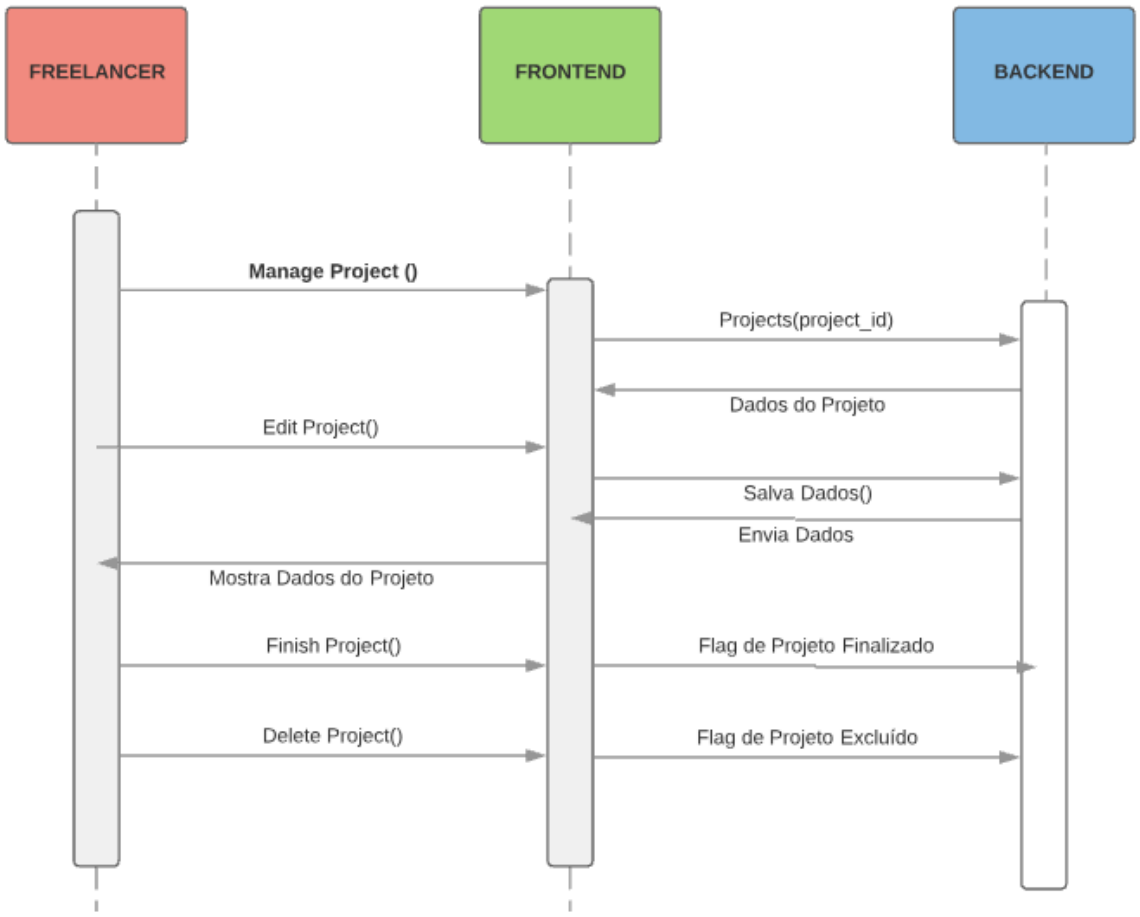

Figura 6 - Caso de Uso UC03

2.3. Career Direction - Definindo uma direção para a carreira

Cada projeto na ferramenta de controle de prazos, Project, será contabilizado nessa ferramenta. Ela irá definir caminhos através de classificação dos projetos de acordo com o nicho dos mesmos e o seu nível de importância. Conforme mais trabalhos de um caminho vão sendo feitos maior a pontuação desse caminho. Esse mapa permitirá saber para onde está andando sua carreira com o tipo de trabalho que está fazendo.

2.3.1 Caso de Uso UC04: Follow Career Direction

O Freelancer clica na aba Career Direction e caso já exista uma pontuação de acordo com os projetos criados o sistema mostrará um gráfico com a pontuação de cada caminho de acordo com a classificação dos Projetos. Caso não exista nenhum projeto, um aviso aparecerá na tela com essa observação.

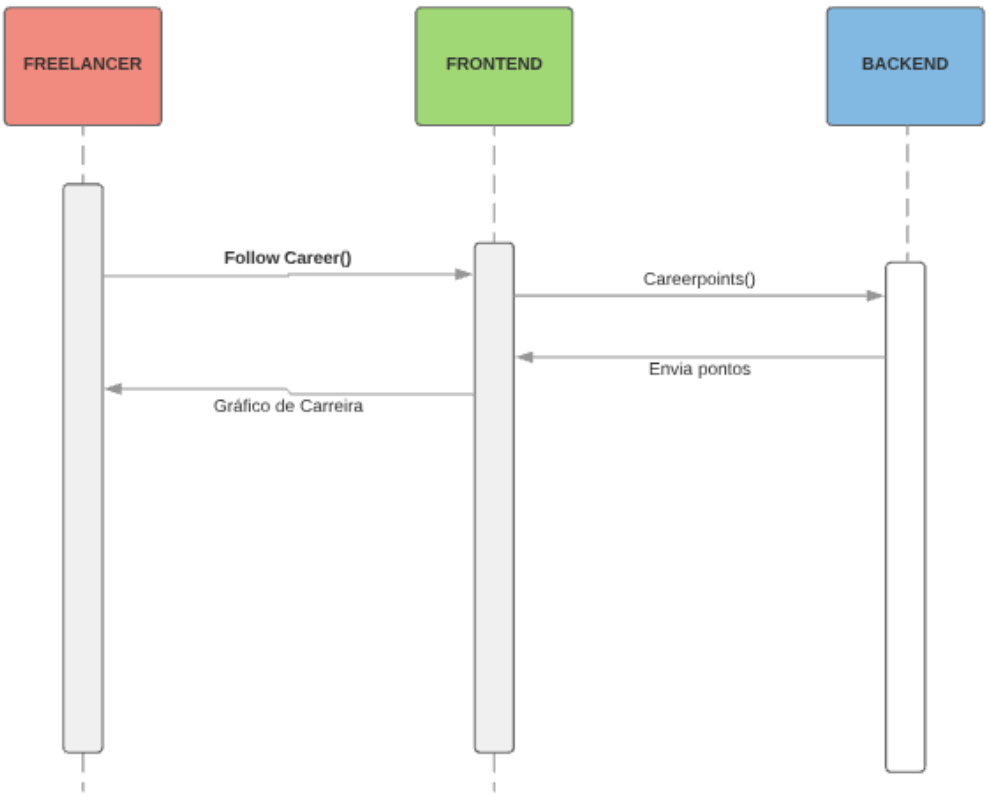

Figura 7 - Caso de Uso UC04

2.4. Network - Conquistando espaço no mercado de trabalho

Após a criação de um projeto a ferramenta Network permite ao Freelancer que ele elenque os profissionais que fizeram parte daquele trabalho para ter aqui uma ferramenta de networking de fácil recordação. Ex.: O fotógrafo que trabalhou na campanha, maquiadores, produtores, atores, donos de marcas, agências, etc.

2.4.1 Caso de Uso UC05: Create Network

O Freelancer clica na aba Rede e a nova janela listará todos os projetos abertos e finalizados. Nessa parte ele também poderá adicionar todos os profissionais que fizeram parte do projeto e assim criar uma lista para fácil networking.

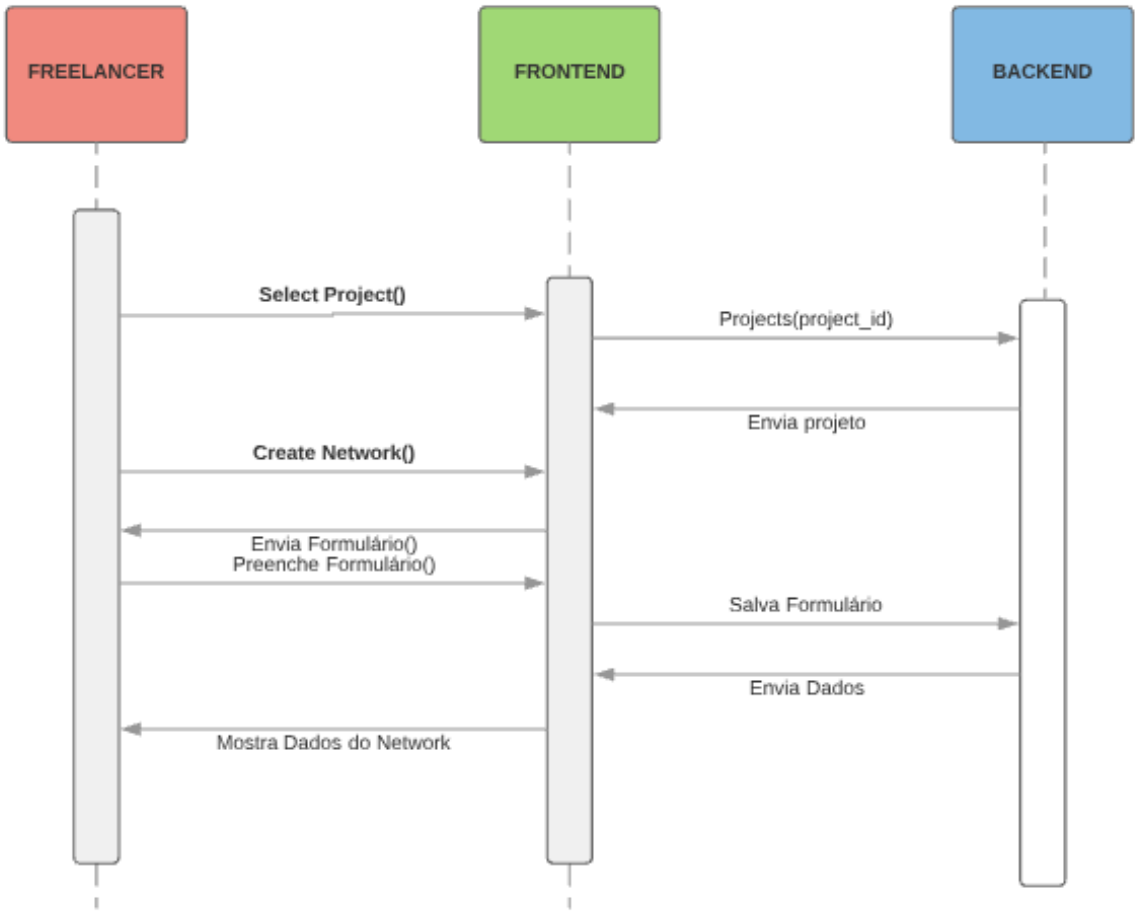

Figura 8 - Caso de Uso UC05

Fluxo Primário de Eventos:

1. O Freelancer clica em Networking;
2. O sistema mostrará a lista de projetos;
3. O Freelancer clica em um projeto e o sistema o leva para a página de Rede desse projeto;
4. Aqui, através de um formulário, ele adiciona os profissionais que fizeram parte do projeto e seus contatos e salva a lista.
5. O sistema salva o formulário e apresenta para o freelancer a página de Rede do projeto com todas as informações salvas.

3 FUNDAMENTAÇÃO TEÓRICA

Para uma melhor compreensão de como foi realizado o projeto, é necessário a apresentação de alguns conceitos teóricos usados durante o desenvolvimento do trabalho que explicarão o porquê do uso de cada ferramenta.

3.1 Python

O Python é uma linguagem de programação criada por Guido von Rossum em 1991 com o objetivo de ter uma linguagem mais produtiva e legível. Em outro linguajar, um código bom e fácil de manter de maneira rápida. Algumas das características da linguagem:

- Baixo uso de caracteres especiais, o que torna a linguagem muito parecida com *pseudo-código executável*;
- Uso de identação para marcar blocos;
- Quase nenhum uso de palavras-chave voltadas para a compilação.
- Bastante intuitiva e fácil de aprender;
- Multiplataforma.

Como o autor desse projeto não tinha conhecimento aprofundado em nenhuma linguagem de programação a escolha pela linguagem Python se deu por causa de alguns fatores.

- Popularidade da Linguagem: Hoje o Python é a terceira linguagem mais utilizada;
- Portátil e Extensível: O Python é suportado pela maioria das plataformas, desde Windows até o Playstation, e permite integrar componentes Java e .NET por exemplo;
- Simplicidade e Facilidade de Aprendizado: O Python é uma linguagem de alto nível e interpretada, além de possuir uma das maiores comunidades na internet;

- Desenvolvimento Web: O Python possui uma matriz variada de frameworks para desenvolvimento Web o que torna o código muito mais rápido e estável.

Um desses frameworks é o Django: um framework de alto nível para desenvolvimento web, que tem como característica a agilidade para o desenvolvimento de aplicações.

Figura 9 - Símbolo da Linguagem Python

3.2 Framework Django

O Django é um *framework* de alto nível, escrito em Python para o desenvolvimento limpo de aplicações Web. Basicamente esse *framework* cuida das atividades pesadas como como tratamento de requisições, mapeamento objeto-relacional, preparação de respostas HTTP, entre outros.

É um *framework* bastante escalável, foi um *framework* desenvolvido para tirar vantagem da maior quantidade de hardware possível. Ele utiliza uma arquitetura onde se pode adicionar mais recursos em qualquer nível: servidores de banco de dados, servidores de aplicação, etc.

Além disso ao desenvolver o Django houve uma preocupação com a segurança ao evitar os mais comuns dos ataques: *cross site scripting (XSS), Cross site request forgery (CSRF), SQL Injection* entre outros.

Diferente da maioria dos frameworks web o Django não é um *framework* MVC (*Model View Controller*) e sim um MTV (*Model Template View*). Aqui a *View* representa qual informação você vê e não como você vê. No caso do Django, uma *View* é uma forma de processar os dados de uma URL específica, por que esse

método descreve qual informação é apresentada. Resumidamente uma *View* descreve **qual** informação é apresentada, e essa *View* delega para um *Template*, que descreve **como** a informação é apresentada.

O *controller*, no caso do Django, é o próprio *framework* que faz o trabalho de processor e rotear uma requisição para a *View* apropriada de acordo com a configuração URL que foi feita.

A seguir pode ser observado na Figura 10 o funcionamento do Django e o fluxo de uma requisição.

O Django é dividido em três camadas: camada de modelos, camada de visualização e camada de *templates*. A seguir uma explicação sobre elas para que um melhor entendimento sobre esse trabalho possa ser obtido.

3.2.1 Camada Modelo (arquivo models.py)

A descrição básica da Camada Modelo é: um modelo é a descrição do dado que será gerado pela aplicação. É nele onde os campos e os comportamentos desses dados estão. Assim, no final, cada modelo equivalerá a uma tabela no banco de dados. O Django gera automaticamente uma API, que faz o trabalho de gerenciar os dados (adicionar, excluir e atualizar).

Ainda relacionado a camada Modelo, está o banco de dados. Por padrão o Django utiliza o SQLite. Depois de criado o modelo ainda é necessário executar a criação das tabelas do banco de dados através dos comandos ***makemigrations*** e ***migrate.***

Quando executamos o ***makemigrations***, o Django cria o banco de dados e as *migrations*, mas não as executa, não aplica as alterações no banco de dados. O comando ***makemigrations*** também analisa se foram feitas mudanças nos modelos e se caso tenham sido feitas mudanças mesmo, ele cria novas migrações para alterar a estrutura do seu banco de dados. Então toda vez que se faz algum tipo de alteração no modelo precise-se aplicar essa alteração a estrutura presente no banco de dados. De acordo com a documentação do próprio Django:

> Migração é a forma do Django de propagar as alterações feitas em seu modelo (adição de um novo campo, deleção de um modelo, etc…) ao seu esquema do banco de

dados. Elas foram desenvolvidas para serem (a maioria das vezes) automáticas, mas cabe a você saber a hora de fazê-las, de executá-las e de resolver os problemas comuns que você possa vir a ser submetidos. (DJANGO, 2018)

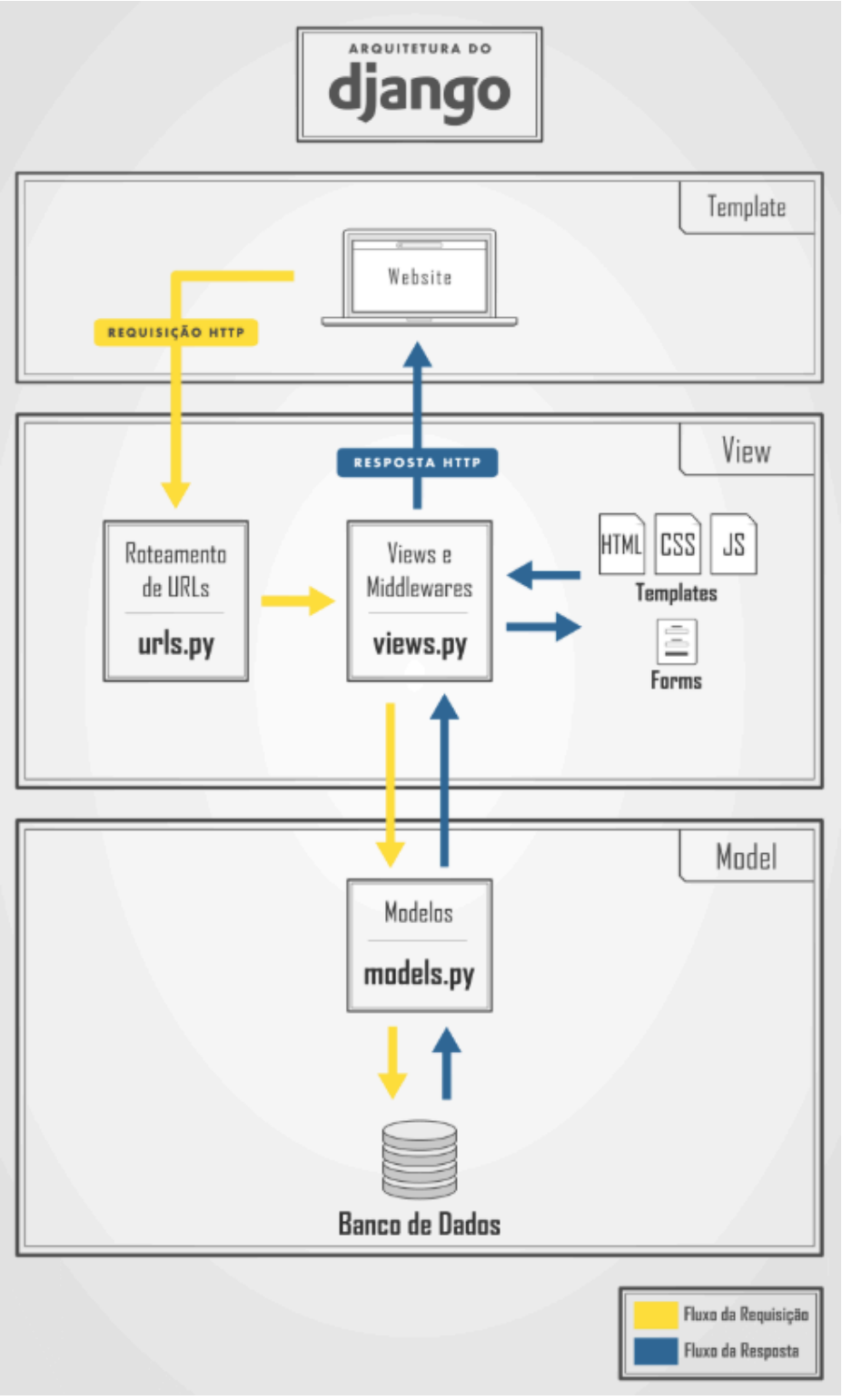

Figura 10 – Arquitetura do Django

Para aplicar as alterações no banco de dados, precisa-se usar o comando ***migrate.*** Com a execução do ***migrate***, o Django irá criar diversas tabelas no banco, caso você tenha diferentes modelos no arquivo ***models.py***.

3.2.2 Camada Views (views.py)

A camada *View* processa as informações vindas do usuário, forma uma resposta e a envia novamente ao usuário. É aqui que ficam as lógicas de negócio e a camada *View* trabalha junto com o roteamento de URLs.

A partir da URL que o usuário acessar o Django irá fazer o roteamento para quem irá tratar essa requisição. Para isso precisa-se informar para onde irá essa requisição. Configuramos esses caminhos no arquivo ***urls.py***. Cada *app* (*feature*) do programa que está se criando possui um arquivo de rotas, um arquivo ***urls.py***. Assim as requisições são roteadas para as *Views*, que irão processar as mesmas.

Pode-se tratar as requisições de duas formas: com funções ou através de CBVs (Class Based Views).

Com as CBVs se tem classes que herdam diversas funcionalidades e facilitam a vida de quem está desenvolvendo um software. As CBVs ou *Views,* descrevem o comportamento para as funcionalidades mais comuns (listagem, exclusão, busca, criação, atualização) que sempre acabam sendo implementadas. O Django possui várias *Views*. As utilizadas nesse trabalho serão descritas abaixo:

3.2.2.1 CreateView

Essa é a *View* de Criação. Nela se informa o modelo que aquela *View* irá se basear, qual template será usado, a classe do formulário e a URL para a qual o programa irá retornar caso a criação de um novo objeto seja feita com sucesso. Essa *View* é o *Create* do CRUD (*Create, Retrieve, Update and Delete*)

3.2.2.2 DetailView

A *DetailView* nos retorna as informações do objeto desejado que foi criado pela CreateView. Ela é o Retrieve do CRUD.

3.2.2.3 UpdateView

Para atualizações do objeto. Nela se seta o modelo, os campos que devem ser atualizados e o nome do template. O Django precisa também saber o id do objeto a ser buscado para atualização. Isso pode ser feito nos arquivos de rotas (urls.py).

3.2.2.4 DeleteView

View para deletar dados de um objeto. Sua configuração se dá similar ao do UpdateView, com a diferença que se precisa pegar o nome do objeto que estará disponível no template (para confirmação com o usuário, e.g., “Você tem certeza que deseja excluir o projeto de nome tal? “) e uma URL de retorno, caso se tenha sucesso na deleção das informações.

3.2.2.5 TemplateView

A TemplateView basicamente renderiza um *template* em *html* e mostra na tela do usuário.

3.2.2.6 ListView

Na ListView se configura o modelo que deve ser buscado e ela automaticamente faz a busca por todos os registros presentes no banco de dados da entidade informada e entrega isso ao usuário na forma de uma lista.

3.2.3 Camada Template

A camada template é a camada que dá a “cara” à aplicação web. É nela onde temos o código Python, responsável por renderizar as páginas web e os arquivos HTML, CSS e JavaScript.

Essa camada faz a interface do código Django/Python com o usuário. Resumidamente um template é um arquivo de texto que pode ser convertido em outro arquivo (um HTML, um CSS, etc..). Um *template* contém variáveis que podem ser substituídas por valores, *tags* que controlam a lógica do *template* e filtros que adicionam funcionalidades ao *template*.

O Django possui uma linguagem padrão de Templates, a DTL (Django Template Language) e ele utiliza um template-base em código HTML que servirá de esqueleto e se reptirá nas páginas que se deseja.

Alguns Templates são mais comuns e estão presentes no código desse trabalho. São eles: Detail, List, Update e Form.

Os templates com nomes Detail são para mostrar na tela as informações dos objetos. Os templates com nomes List, para mostrar uma lista. Os templates com nome Update servem para editar informações dos objetos e os templates com nome Form servem para criar um formulário onde se criará um novo objeto dentro de uma classe. Também nesse projeto existem alguns Templates para deletar objetos, porém esses são bastante simples, como pode ser observado na Figura 11. Conforme visto anteriormente esses templates estão ligados aos arquivos Views e possuem código HTML e CSS.

Figura 11 – Template Delete

3.2.3.1 HTML e CSS

HTML e CSS estão presentes em todas as aplicações Web. São responsáveis pelo FrontEnd, ou seja, o que aparece para os usuários. Nesse trabalho foram usados HTML e CSS de forma básica. Para a renderização das páginas se usou o Bootstrap CDN, um framework baseado em HTML, CSS e Javascript para aplicações Web. Com o Bootstrap CDN se pode carregar CSS e Javascript remotamente através de seus servidores. A figura 12 mostra a linha de

código para carregar o Bootstrap CDN. Elas estão presentes em todos os templates do do aplicativo.

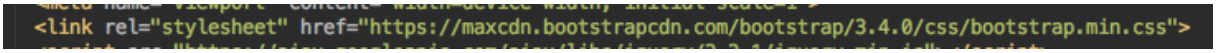

Figura 12 - Bootstrap CDN

4 DESENVOLVIMENTO

Para a explicação do desenvolvimento da ferramenta Freeland irá se separar o produto final, o MVP, nos 4 apps do Freeland. Assim se mantém uma coerência no documento e permite uma melhor interpretação das linhas de código e explicações para cada app.

4.1 Os Apps do Freeland

As diferentes funcionalidades de um aplicativo feito com o framework Django são chamadas de Apps. No caso do Freeland os diferentes apps são:

- Pricing;
- Projects;
- Network;
- Career Direction.

Cada um com seus models, views e templates. Na figura 13 se pode observar a estrutura do Freeland:

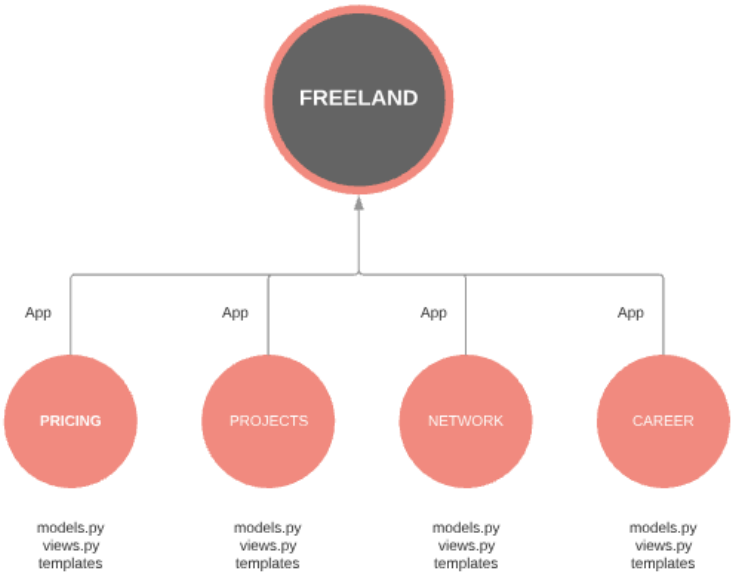

Figura 13 - Arquitetura de Apps do Freeland

4.2 APP PRICING

O App Pricing como falado no capítulo 2, auxilia o Freelancer a ter uma base sobre qual preço cobrar pela sua diária. Aqui ele irá botar os dados da sua pesquisa de mercado e ter um valor sugerido sobre o próprio valor do seu trabalho, como pode ser observado na Figura 14.

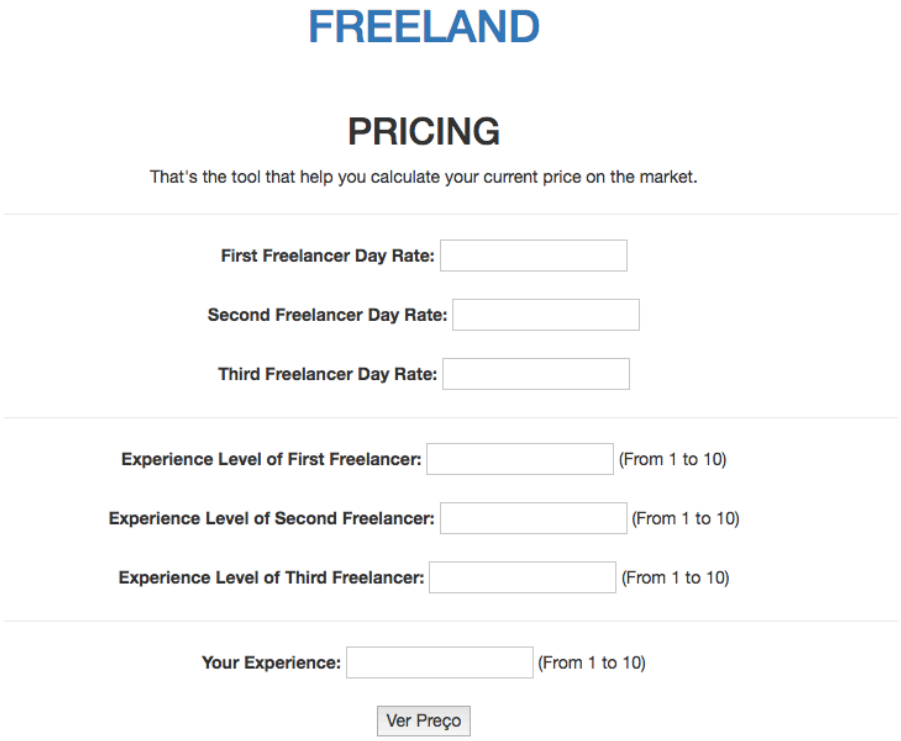

Figura 14 - Página Inicial do App Pricing

4.2.1 Pricing Model

A ferramenta de *pricing* possui um *model* bem simples, conforme visto na Figura 15, isso se dá pois ela só recebe informações numéricas e de inteiros. Os campos de ***menornumero*** e ***maiornumero*** são utilizados para cálculos no View e não são digitados pelo usuário, assim como ***lowerdayrate***, ***biggestdayrate*** e ***price***. Porém esses últimos são mostrados na tela como informação no fim da requisição. Os campos de ***first_freelancer*** até o ***third_freelancer***, correspondem ao preço que eles cobram.

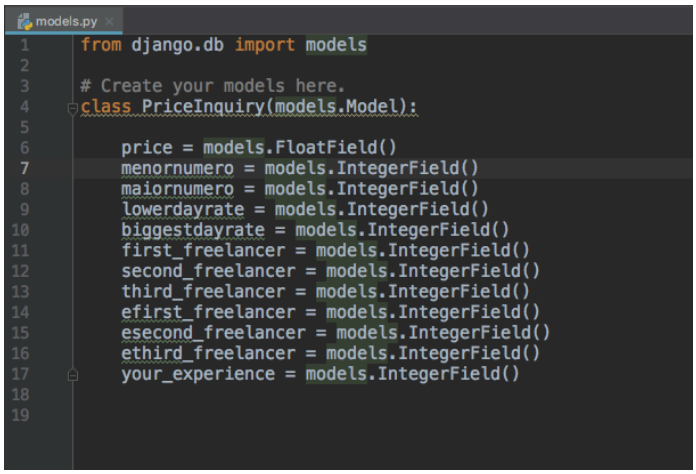

Figura 15 - Arquivo model.py do App Pricing

Os campos de ***efirst_freelancer*** até ***ethird_freelancer***, correspondem a experiência dos freelancers concorrentes.

4.2.2 Pricing View

O arquivo view do projeto Pricing se destaca pelas funções Get e Post, que basicamente mostra o formulário na tela (função Get) e salva a informação e a processa retornando um dado diferente (função Post). Dentro da função post se pega os dados obtidos do formulário e se calcula através de uma média aritmética ponderada qual o valor sugerido que o Freelancer deve cobrar. Isso pode ser observado nas Figuras 16 e 17. Além disso se cria um objeto price_object que guardará todas essas informações e será o resultado dessa função para ser usado pela View do Pricing.

views.py

```python
from django.shortcuts import render
from django.views.generic import FormView

# Create your views here.
from pricing.forms import PriceForm
from pricing.models import PriceInquiry


class PriceView(FormView):
    form_class = PriceForm
    template_name = "pricing/price.html"

    def get(self, request, *args, **kwargs):
        form = self.form_class(initial=self.initial)
        return render(request, self.template_name, {'form':form})

    def post(self,request):
        form = self.get_form()
        form.is_valid() #erro de validação
        data = form.cleaned_data #dado validado

        sum = data['first_freelancer']*data['efirst_freelancer']+data['second_freelancer']*data['esecond_freelancer']\
              +data['third_freelancer']*data['ethird_freelancer']
```

Figura 16 - Arquivo view.py do App Pricing

```python
        sum = data['first_freelancer']*data['efirst_freelancer']+data['second_freelancer']*data['esecond_freelancer']\
              +data['third_freelancer']*data['ethird_freelancer']

        price = sum/(data['efirst_freelancer']+data['esecond_freelancer']+data['ethird_freelancer'])

        dayrates = [data['first_freelancer'], data['second_freelancer'], data['third_freelancer']]
        lowerdayrate = min(dayrates)
        biggestdayrate = max(dayrates)
        experience = [data['efirst_freelancer'], data['esecond_freelancer'], data['ethird_freelancer']]
        menornumero = min(experience)
        maiornumero = max(experience)

        price_object = PriceInquiry.objects.create(price=price, menornumero=menornumero, maiornumero=maiornumero,
                                                   lowerdayrate=lowerdayrate, biggestdayrate=biggestdayrate,
                                                   first_freelancer=data['first_freelancer'],
                                                   second_freelancer=data['second_freelancer'],
                                                   third_freelancer=data['third_freelancer'],
                                                   efirst_freelancer=data['efirst_freelancer'],
                                                   esecond_freelancer=data['esecond_freelancer'],
                                                   ethird_freelancer=data['ethird_freelancer'],
                                                   your_experience=data['your_experience'])

        return render(request, self.template_name, {'form': form, 'price_inquiry': price_object})
```

Figura 17 - Arquivo view.py do App Pricing - parte 2

4.2.3 Pricing Template

Para o template de pricing foi preciso separar cada campo feito pela FormView do Django para poder customizá-los e não ficar com o layout *default* do Django e isso pode ser visto na figura 18 com os campos dentro das "* *" na classe CSS "fieldWrapper".

Na figura 19 se observa o *template* para o resultado da pesquisa feita pelo usuário e onde se pode observar o resultado do teste de experiência. Se a experiência do Freelancer for menor que a de um concorrente ele não deverá cobrar mais que o valor daquele concorrente. Caso a experiência seja maior do que todos, ele não deverá cobrar menos do que o maior preço. Caso ele esteja no meio o

programa irá mostrar quanto ele deveria cobrar com base no cálculo feita pela View, como pode ser visto na Figura 20.

4.3 APP PROJECT

O App Project auxilia no gerenciamento dos Projetos que o usuário está fazendo parte. A página inicial contém uma lista com esses projetos, como pode ser observado na Figura 21.

price.html

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
  </head>


<BODY style="...">

<h1 style="..."><a href='/'>FREELAND</a></h1>
<br>

<h2>PRICING</h2>
<p>That's the tool that help you calculate your current price on the market.</p>
<hr>


<form action="{% url 'pricing:create' %}" method="post">
{% csrf_token %}

  <div class="fieldWrapper">
    {{ form.first_freelancer.errors }}
  <label for="{{ form.first_freelancer.id_for_label }}">First Freelancer Day Rate:</label>
    {{ form.first_freelancer }}
  </div>

  <br>

  <div class="fieldWrapper">
    {{ form.second_freelancer.errors }}
  <label for="{{ form.second_freelancer.id_for_label }}">Second Freelancer Day Rate:</label>
    {{ form.second_freelancer }}
  </div>

  <br>

  <div class="fieldWrapper">
    {{ form.third_freelancer.errors }}
  <label for="{{ form.third_freelancer.id_for_label }}">Third Freelancer Day Rate:</label>
    {{ form.third_freelancer }}
  </div>

```

Figura 18 - Template do App Pricing

```
    <br>
<input type="submit" value="Ver Preço">
</form>

{% if request.method == "POST" %}
  <hr>
  <h2>Competing Freelancer's Day Rates</h2>
  <p>First Freelancer: R$ {{price_inquiry.first_freelancer}},00</p>
  <p>Second Freelancer: R$ {{price_inquiry.second_freelancer}},00</p>
  <p>Third Freelancer: R$ {{price_inquiry.third_freelancer}},00 </p>
  <hr>
  <h2>Competing Freelancer's Experience Level</h2>
  <p>Experience Level of First Freelancer - {{price_inquiry.efirst_freelancer}} Experience Points</p>
  <p>Experience Level of Second Freelancer -  {{price_inquiry.esecond_freelancer}} Experience Points</p>
  <p>Experience Level of Third Freelancer -  {{price_inquiry.ethird_freelancer}} Experience Points </p>
  <hr>
  <h2>Your Level of Experience</h2>
  <p>Nowadays Your Level of Experience is: {{price_inquiry.your_experience}} Experience Points</p>
  <p>The Less Experienced Freelancer has: {{price_inquiry.menornumero}} Experience Points</p>
  <p>The More Experienced Freelancer has: {{price_inquiry.maiornumero}} Experience Points</p>
  <hr>

  {% if price_inquiry.your_experience <= price_inquiry.menornumero %}
      <br><br><p style="...">You should not charge more than: R$ {{price_inquiry.lowerdayrate}}</p><br><br><br>
  {% endif %}

  {% if price_inquiry.your_experience >= price_inquiry.maiornumero %}
      <br><br><p style="...">You should not charge less than: R$ {{price_inquiry.biggestdayrate}}</p><br><br><br>
  {% endif %}

   {% if price_inquiry.your_experience > price_inquiry.menornumero and price_inquiry.your_experience < price_inquiry.maiornumero %}
     <br><br><p style="...">You should charge around: R$ {{price_inquiry.price}}</p><br><br><br>
  {% endif %}

{% endif %}
```

Figura 19 - Template do App Pricing part 2

Competing Freelancer's Day Rates

First Freelancer: R$ 2000,00

Second Freelancer: R$ 2000,00

Third Freelancer: R$ 4000,00

Competing Freelancer's Experience Level

Experience Level of First Freelancer - 2 Experience Points

Experience Level of Second Freelancer - 3 Experience Points

Experience Level of Third Freelancer - 5 Experience Points

Your Level of Experience

Nowadays Your Level of Experience is: 4 Experience Points

The Less Experienced Freelancer has: 2 Experience Points

The More Experienced Freelancer has: 5 Experience Points

You should charge around: R$ 3000.0

Figura 20 - Tela do App Pricing Renderizada

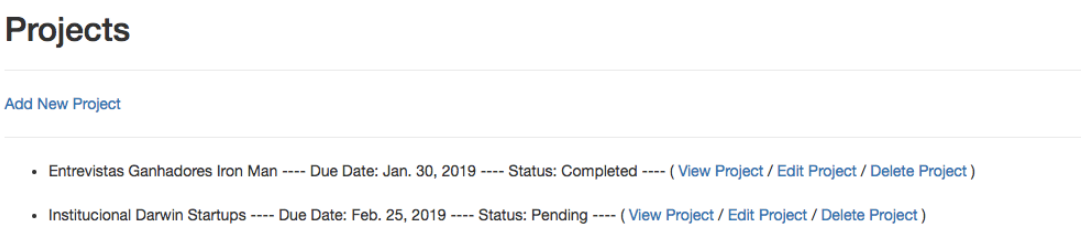

Figura 21 - Tela Inicial do App Projetc

4.3.1 Projects Model

O *model* do App Projects na verdade começa com um *model* de cliente, como pode ser visto na Figura 22. Isso porque foi analisado, durante o desenvolvimento, que faz muito mais sentido ter uma tabela de clientes do que simplesmente ter um cliente adicionado no Projeto sem mais informações. Ao mesmo tempo não se fez necessário a criação de um novo app apenas para essa função. Os campos de *company_name* e *contact_name* são obrigatórios, já os de *email, celular* e *website* não o são e isso se dá pelo comando *"blank=True"*.

No model de Projeto, como ser observado na Figura 23, podemos ver um model um pouco maior dos que já apresentados. Nele constam campos de caracteres, inteiros, booleanos e datas, além de uma chave estrangeira para a tabela de clientes. Além disso temos uma função que calcula o tempo que falta para a entrega do projeto *(def time_left*).

```python
from django.db import models

# Create your models here.
from django.urls import reverse
from django.utils import timezone


class Client(models.Model):

    company_name = models.CharField(max_length=255)
    contact_name = models.CharField(max_length=255)
    email = models.CharField(blank=True, default='', max_length=255)
    cellphone = models.CharField(blank=True, default='', max_length=255)
    website = models.CharField(blank=True, default='', max_length=255)


    def get_absolute_url(self):
        return reverse('client-detail', kwargs={'pk': self.pk})

    def __str__(self):
        return self.company_name

```

Figura 22 – Arquivo model.py do App Project parte do Cliente

4.3.2 Projects View

As Views dos projetos e dos clientes seguem as CBVs baseadas no CRUD discutidas no capítulo 3. A única diferença aqui se dá na *ProjectListView* onde temos um *def_queryset* que organiza os Projetos por data de entrega. Os que estão mais pertos de serem entregues vem primeiro. Isso pode ser observado na figura 24.

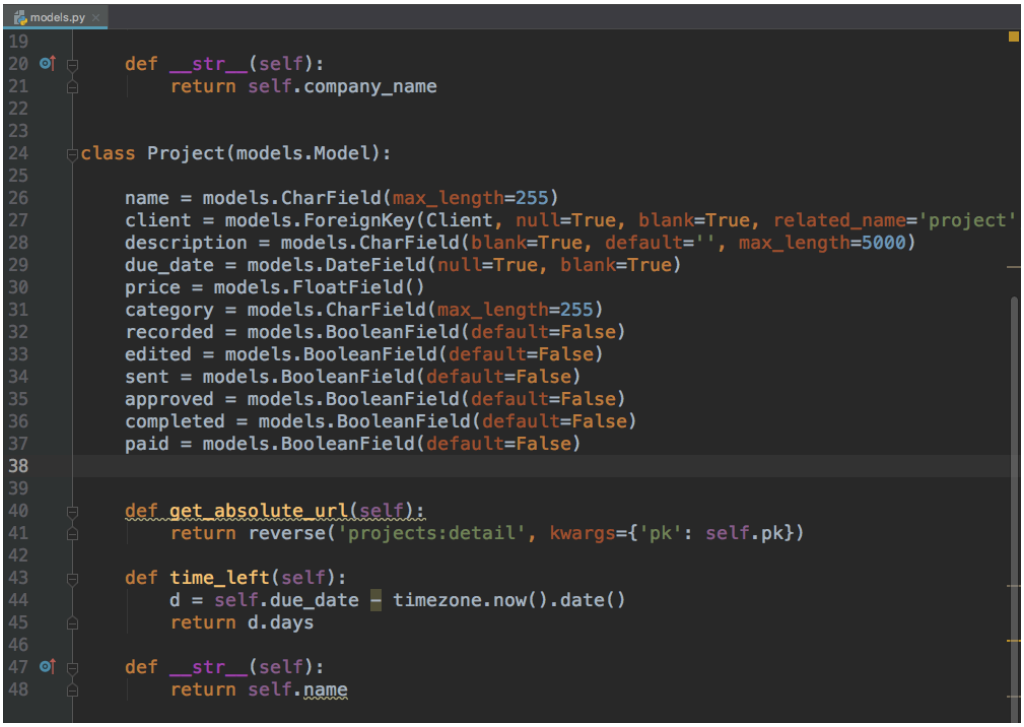

Figura 23 - Arquivo model.py do App Project parte de Projetos

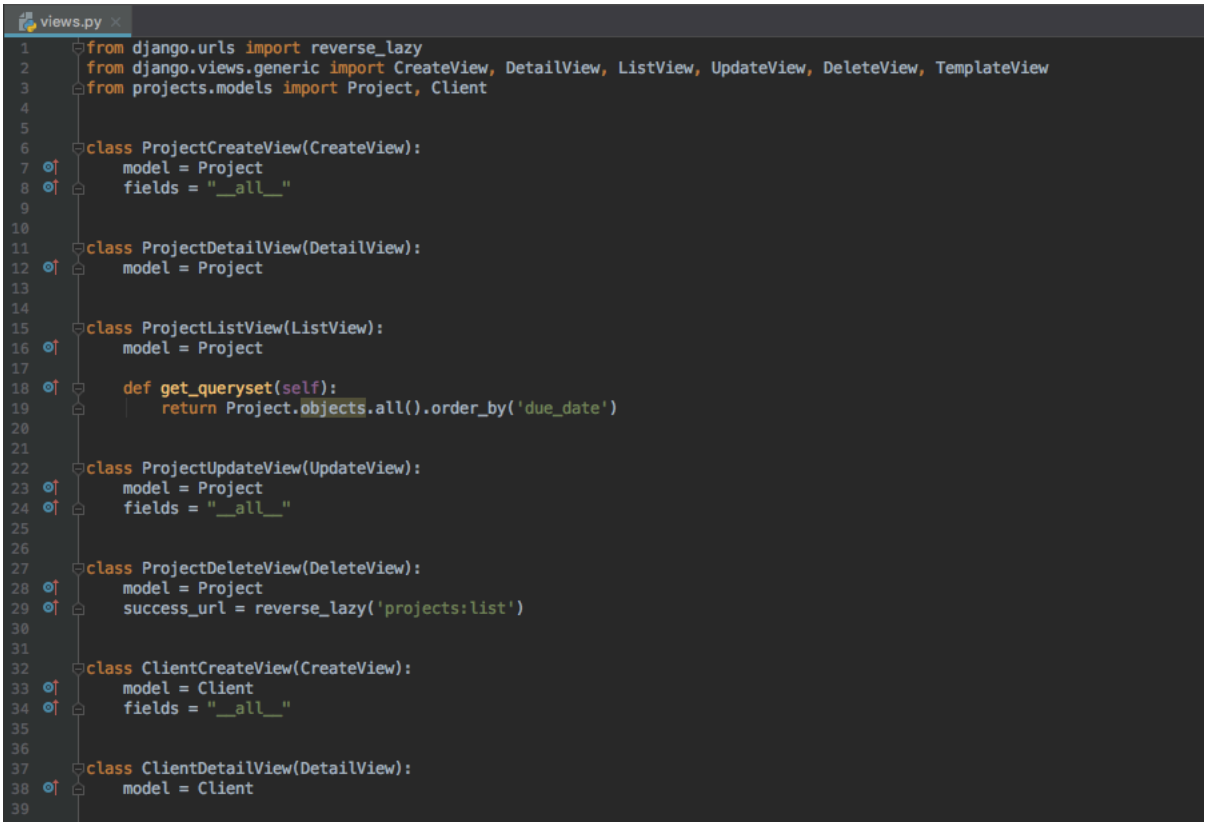

Figura 24 - Arquivo views.py do App Project

```
    model = Client


class ClientListView(ListView):
    model = Client


class ClientUpdateView(UpdateView):
    model = Client
    fields = "__all__"


class ClientDeleteView(DeleteView):
    model = Client
    success_url = reverse_lazy('clientlist')


class IndexView(TemplateView):
    template_name = 'index.html'
```

Figura 25 - Arquivo views.py do App Projects parte 2

4.3.3 Projects Template

Para o app Projects foram feitos 4 *templates* para o app Project e 4 *templates* para o Clients, conforme pode ser visto na Figura 26. Eles são de certa forma bem parecidos um com os outros e por isso aqui se mostrarão apenas os *templates* do app Projects.

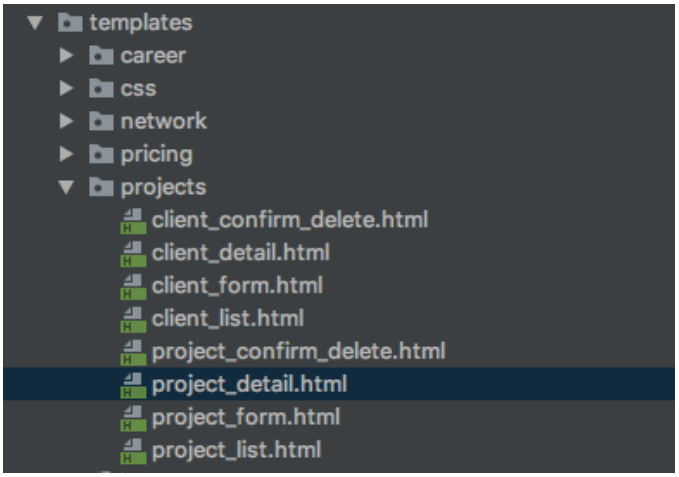

Figura 26 - Templates Projects e Clients

Como podemos observar na Figura 27, 28 e 29 os *templates* do Project são praticamente escritas todas em HTML. Na Figura 27, correspondente ao *template* do formulário para preenchimento temos o *{{ form.as_p }},* um comando em Pyhton que fornece o formulário com todos os campos no formato *paragraph* em HTML, o que

simplifica muito a criação desses *templates*. Já no *template project_detail, da figura 29,* pode-se observar que é feito a customização campo por campo, chamando os devidos dados dos objetos também com a dupla chave ({{}}). Isso se repete no *template project_list*.

project_form.html | project_list.html | project_detail.html

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
  </head>


<BODY style="...">

<h1 style="..."><a href='/'>FREELAND</a></h1>
<br>

<form method="post">{% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Save">
</form>

<br>


<a href="{% url 'clientcreate' %}"> Add New Client </a>
<br>
<a href="{% url 'projects:list' %}"> Back to Projects </a>


</BODY>

</html>
```

Figura 27 - Arquivo project_form.html do App Projects

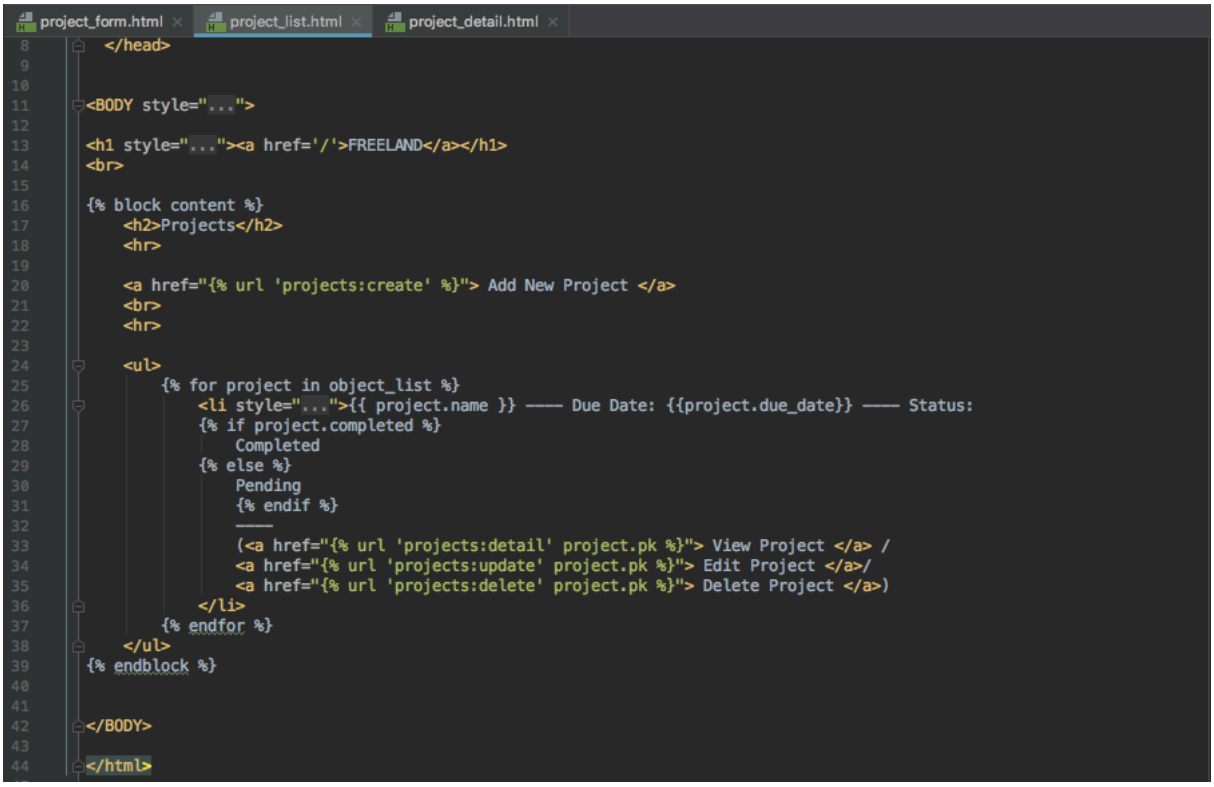

Figura 28 - Arquivo project_list.html do App Project

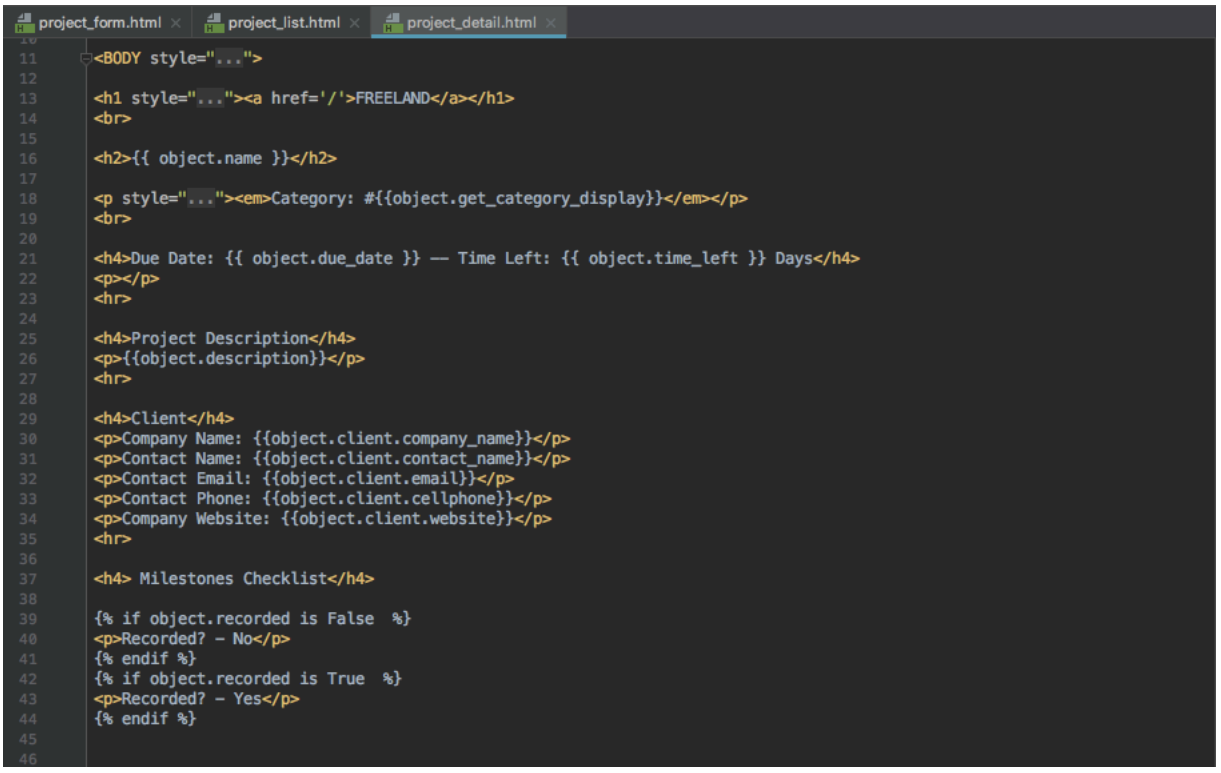

Figura 29 - Arquivo project_detail do App Projects

As figuras 30, 31 e 32, mostram o resultado dos *templates* renderizados para o usuário. O *template project_list* pode ser visto na Figura 21, do começo dessa seção.

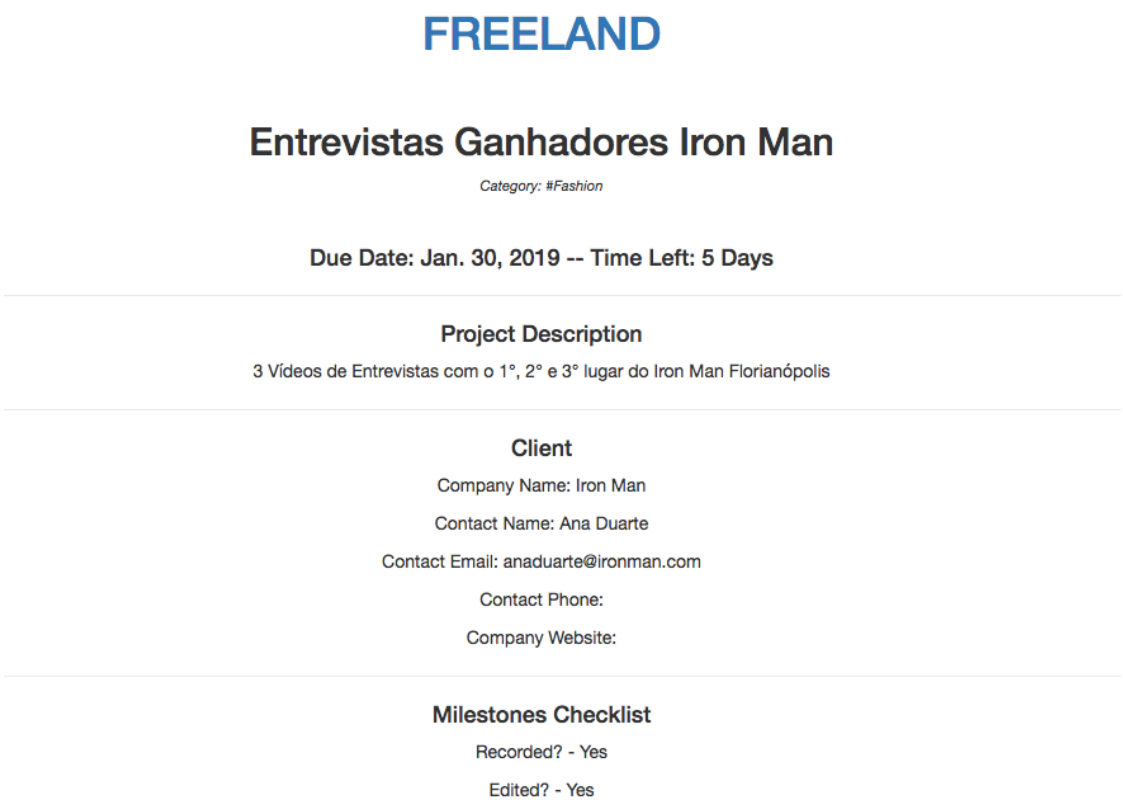

Figura 30 - Página do Projeto "Entrevistas com Ganhadores Iron Man"

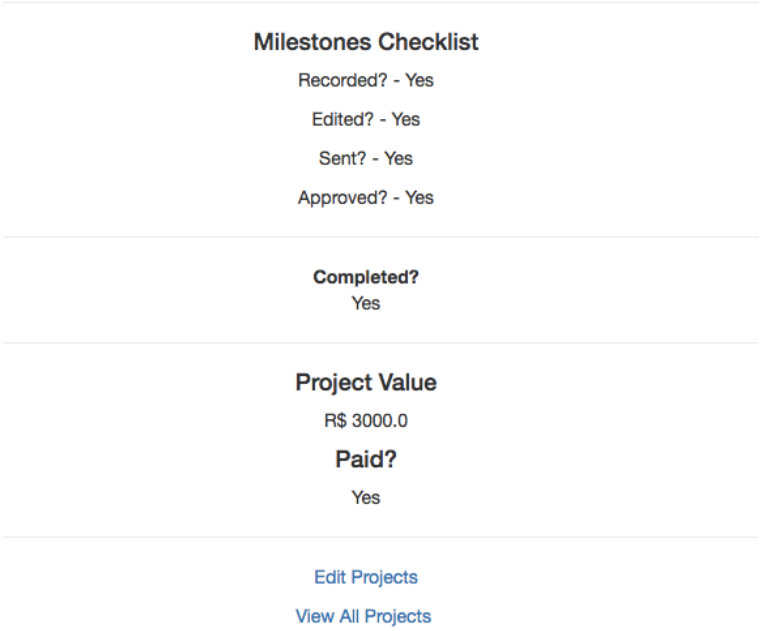

Figura 31 - Página do Projeto "Entrevistas com Ganhadores Iron Man" (cont.)

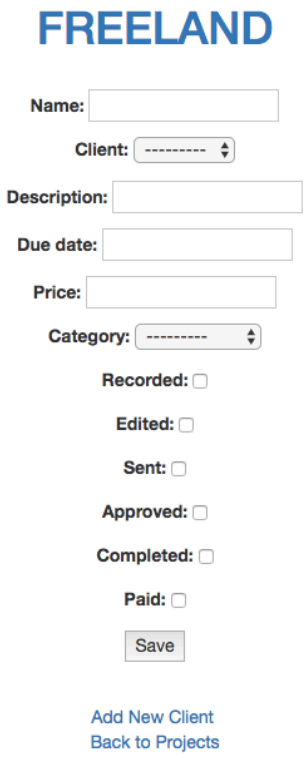

Figura 32 - Página do Formuáriorio para Adicionar Novo Projeto

4.4 NETWORKS

O App Network permite a criação de networks por projeto, onde o usuário pode descrever quem trabalhou com ele em cada projeto e botar mais informações sobre cada membro da equipe. A página Inicial pode ser vista na Figura 33.

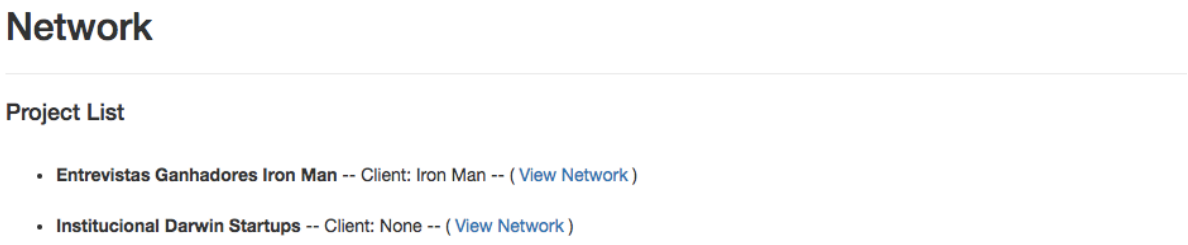

Figura 33 - Página Inicial do App Network

4.4.1 Network Model

O Network Model não possuiria nenhum segredo se não fosse o campo projects com o ManyToManyField(). A arquitetura ManyToMany, conforme mostra a Figura 34, relaciona a tabela de projetos com a tabela de membros de forma automática. Assim, poderemos adicionar membros nos projetos certos sem dificuldade de correlacionar essas duas tabelas manualmente. O Django faz isso para nós de forma automática.

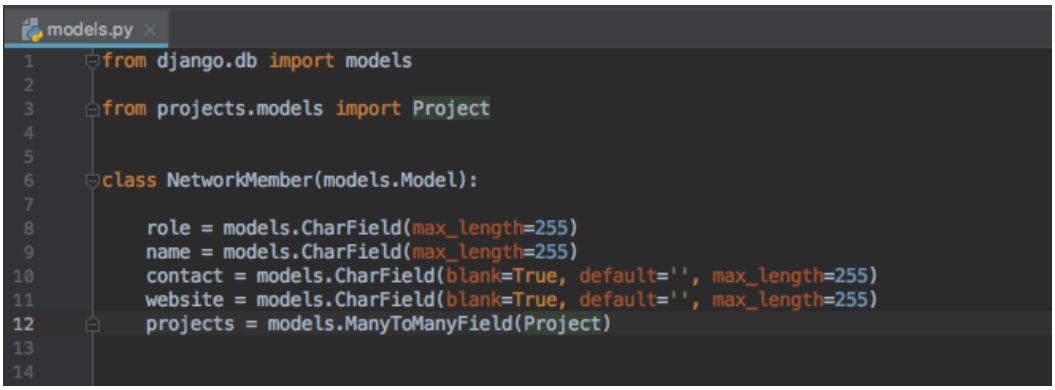

Figura 34 - Arquivo model.py do App Network

4.4.2 Network Views

Aqui não temos nenhuma View diferente e todas seguem o modelo CRUD exemplificado no capítulo anterior, conforme pode ser visto na Figura 35.

views.py

```python
from django.http import Http404, HttpResponseRedirect
from django.urls import reverse, reverse_lazy
from django.views.generic import ListView, DetailView, CreateView, UpdateView, FormView, DeleteView

from network.forms import NetworkMemberForm
from network.models import NetworkMember
from projects.models import Project


class NetworkListView(ListView):
    model = Project
    template_name = 'network/network_list.html'


class NetworkDetailView(DetailView):
    model = Project
    template_name = 'network/network_detail.html'


class NetworkCreateView(CreateView):
    model = Project
    fields = "__all__"


class NetworkDeleteView(DeleteView):
    model = NetworkMember
    template_name = 'network/networkmember_confirm_delete.html'
    success_url = reverse_lazy('network:list')


class NetworkUpdateView(FormView):
    form_class = NetworkMemberForm

    template_name = 'network/network_update.html'
```

Figura 35 - Arquivo views.py do App Network

4.4.3 Network Template

Por ser um App parecido com o de Projetos, na questão de construção, os *templates* aqui são bem similares, conforme pode ser visto nas Figuras 36, 37 e 38 que correspondem aos *templates* de Detalhes, Lista e Update, respectivamente. As figuras 39 e 40 mostram os *templates* renderizados. O Figura 33 mostra o template *network_list* renrederizado, que é a página inicial do app Network.

network_detail.html

```
<BODY style="...">

<h1 style="..."><a href='/'>FREELAND</a></h1>
<br>

{% block content %}
    <h2>{{ object.name }}</h2>
    <hr>
    <h4>Project Members</h4>

    <ul>
        {% for member in object.networkmember_set.all %}
            <li style="..."> <strong>Role: {{ member.role }} </strong>
                <br> Name: {{ member.name }}
                <br>  Contact: {{ member.contact }}
                <br>  Portfolio/Website: {{ member.website }}
                <br>
            </li>
            <a href="{% url 'network:update' member.pk %}"> Edit Member </a> —
            <a href="{% url 'network:delete' member.pk %}"> Delete Member </a>
        {% endfor %}
    </ul>

    <br>
    <a href="{% url 'network:update' project.pk %}"> Add Member </a>

{% endblock %}

</BODY>


<hr>
<a href="{% url 'network:list' %}"> View All Networks </a>
```

Figura 36 - Template Detail do App Network

network_detail.html | network_list.html

```
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
  </head>


<BODY style="...">

<h1 style="..."><a href='/'>FREELAND</a></h1>
<br>



{% block content %}
    <h2>Network</h2>
    <hr>
    <h4>Project List</h4>
    <br>
    <ul>
        {% for project in object_list %}
            <li style="..."><strong>{{ project.name }}</strong> — Client: {{ project.client}} —

                (<a href="{% url 'network:detail' project.pk %}"> View Network </a>)

            </li>
        {% endfor %}
    </ul>
{% endblock %}

</BODY>

</html>
```

Figura 37 - Template List do App Network

network_detail.html | network_list.html | network_update.html

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
  </head>


<BODY style="...">

<h1 style="..."><a href='/'>FREELAND</a></h1>
<br>
<h4>Add a New Member to your Network of — <strong>{{ project.name }}'s</strong> — Project:</h4><br>

<form method="post">{% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Save">
</form>

(<a href="{% url 'network:detail' project.pk %}"> Back to Network </a>)


</BODY>

</html>
```

Figura 38 - Template Update do App Network

Institucional Darwin Startups

Project Members

- **Role: Diretor de Fotografia**
  Name: Jeann Frederico Mette
  Contact: 47 999723543
  Portfolio/Website: jeannfm.com
  Edit Member -- Delete Member

- **Role: Câmera**
  Name: Rômulo Cruz
  Contact: 47 99927-7724
  Portfolio/Website: rcruzworld.com
  Edit Member -- Delete Member

Add Member

View All Networks

Figura 39 - Página de membros do Projeto Institucional Darwin Startups

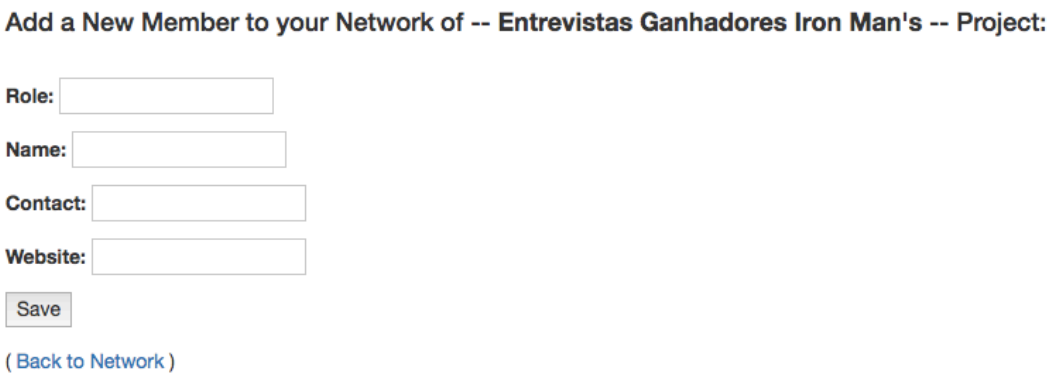

Figura 40 - Formulário de Novo Membro de Projeto

4.5 CAREER DIRECTION

O App Career Direction pega as categorias dos projetos e soma a quantidade de projetos dentro dessas categorias que o Freelancer está fazendo. Esse App só possui uma página, conforme visto na Figura 41, que contabiliza os pontos de acordo com a classe de projetos.

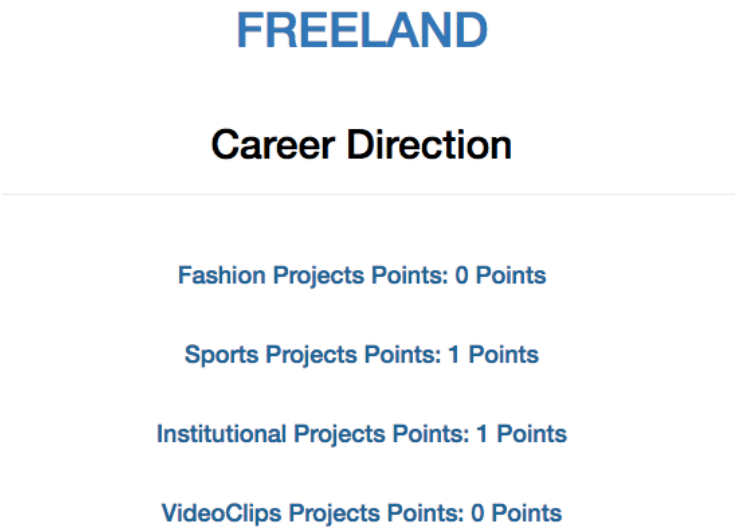

Figura 41 - Página Inicial de Career Direction

Esse App, como não escreve e nem posta nada do banco de dados não possui um arquivo *model.py*, porém possui o arquivo *views.py* e *template.*

4.5.1 Career Direction View

A view do Career Direction não pertence aos CBVs comuns. Precisou-se usar uma função *get_context_data* para contar as categorias presentes no campo de categorias do banco de dados da tabela Projetos. Assim a View de Career Direction consegue contar quantas categorias existem e quantos são os projetos presentes nelas, retornando isso para o usuário. A figura 42 mostra isso.

views.py

```python
from django.views.generic import TemplateView

from projects.models import Project


class CareerView(TemplateView):
    template_name = 'career/career.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data()

        counts = {}

        for category in Project.CATEGORY_CHOICES:
            counts[category[0]] = Project.objects.filter(category=category[0]).count()

        context["counts"] = counts

        return context
```

Figura 42 - Arquivo view.py do App Career Direction

4.5.2 Career Direction Template

O template de Career Direction conta com uma lista em HTML que retorna a contagem de projetos em cada categoria da View anterior, conforme pode se ver na Figura 43.

career.html

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
  </head>

  <BODY style="...">

  <h1 style="text-align:center;"><a href='/'>FREELAND</a></h1>

  <br>


<h2 style="text-align:center; color: black" >Career Direction</h2>

<HR>
<br>

<h4 style="text-align:center; color: #006699">Fashion Projects Points: {{ counts.FASHION }} Points</h4>
  <br>
<h4 style="text-align:center; color: #006699;"> Sports Projects Points: {{ counts.SPORTS }} Points</h4>
  <br>
<h4 style="text-align:center; color: #006699">Institutional Projects Points: {{ counts.INSTITUCIONAL }} Points</h4>
    <br>
<h4 style="text-align:center; color: #006699">VideoClips Projects Points: {{ counts.VIDEOCLIP }} Points</h4>
    <br>

<br>
  <br>
  </BODY>
```

Figura 43 - Arquivo Template para Career Direction

5 RESULTADOS

Como pôde ser observado no decorrer desse documento o objetivo de criar um MVP foi atingido. O Freeland, como um produto mínimo que atende algumas das dificuldades dos Freelancers, foi criado.

Nessa seção discutiremos os resultados através das pesquisas de opinião que foram realizadas com profissionais conhecidos pelo autor do trabalho que de fato usaram a ferramenta. O Fotógrafo Jeann Frederico Mette, o Cinegrafista Fábio Koerich Souza e o cinegrafista e fotógrafo Rômulo Cruz fizeram parte dos testes e apresentaram algumas conclusões:

5.1 Pontos fortes apresentados pelos Freelancers

- Lista de network separada por projeto de extrema utilidade;
- Career Direction como ferramenta realmente útil para fazer o tracking de projetos feitos e para onde se está indo.

5.2 Pontos a serem melhorados apresentados pelos Freelancers

- Parte visual precisa ser reformulada;
- Checklist de projetos precisa ser editável a gosto do Freelancer;
- Tabela de Pricing com possibilidade de adicionar mais concorrentes;
- Mais opções para adicionar informações nos formulários.

5.3 Features adicionais interessantes apresentados pelos Freelancers

- Adicionar preços dos profissionais na lista de Network;
- Uma ferramenta de tracking para saber como anda o contato com os Clientes.

Algumas das considerações já estavam podiam ser esperadas pelo autor, principalmente sobre a parte gráfica da ferramenta, que até a finalização desse documento apresenta-se bastante básica, conforme o conhecimento do autor com a

parte FrontEnd do projeto, HTML, CSS e Javascript. Isso já era esperado e a o planejamento para o futuro é ter alguém especializado nessa área para a mudança completa da parte gráfica do sistema.

O interessante de fazer esse tipo de pesquisa é que algumas coisas inesperadas, problemas ou ideias, são enumeradas. Um exemplo é o fato de a lista de Network ter sido a ferramenta que todos os profissionais falaram muito bem. Isso devido a grande participação dos mesmos em projetos variados, na maioria com pessoas diferentes entre si. O fato de se ter uma lista com as pessoas que trabalharam em cada projeto facilita na hora de lembra e acionar um dos profissionais quando necessário. Adicionar um campo onde é possível preencher o custo da diária dos mesmos foi uma solicitação feita que não foi pensada pelo autor.

Uma outra reclamação foi a pouca possibilidade de customização dos campos, onde fosse possível renomear alguns deles, como por exemplo no Checklist dos Projetos. Deixar essa função estática não serviu como método organizacional para os profissionais da pesquisa, visto que cada um lida com seu workflow de uma maneira diferente.

Foi interessante ver na pesquisa que os dois pontos fortes presentes foram na ferramenta Network e Career Direction, o que não foi o que o autor imaginava. Isso motiva a torná-las melhores e com mais *features* que serão discutidos na seção de Perspectivas Futuras no próximo capítulo.

6 CONCLUSÃO E PERSPECTIVAS FUTURAS

Como se pode observar ao longo do documento o objetivo principal foi alcançado: desenvolver um mínimo produto viável que atendesse algumas das maiores dificuldades enfrentadas por Freelancers, voltado ao mercado audiovisual utilizando metodologias de desenvolvimento de software e de negócios.

É interessante ressaltar que durante todo o desenvolvimento pôde ser observado que o MVP do Freeland também pode ser usado por Freelancers de outros mercados. Apesar de na hora do desenvolvimento as justificativas para a criação de cada ferramenta terem sido pensadas no ambiente de um profissional do mercado audiovisual, se pôde observar que muitos outros profissionais podem utilizar o Freeland. E isso é ponto positivo, pois permite a escalabilidade do software para outros mercados e outros públicos.

Atualmente a indústria de software é movida majoritariamente pelo desenvolvimento de aplicações para a internet. Este trabalho e as atividades nele presente serviram como um excelente ponto de partida e de aquisição de experiências nessa área, visto que o aluno não possuía nenhum conhecimento nas linguagens de programação aqui presentes. As ferramentas de desenvolvimento utilizadas no desenvolvimento do sistema, são recursos utilizados largamente e cujo conhecimento será de grande valia. Mesmo não se tornando um profissional da área o fato de conseguir entender as diferentes linguagens será de fato muito importante, principalmente para contratar desenvolvedores para levar o Freeland adiante.

A construção deste documento foi importante para que os conhecimentos adquiridos nele se consolidassem e que para que se tomasse consciência de como o projeto se conecta com o curso de Engenharia de Controle e Automação. De fato, os conhecimentos e experiências trazidos pelo curso foram de grande ajuda no dia a dia de desenvolvimento. De maneira mais evidente, foram úteis os conteúdos obtidos em toda a parte do curso que se relaciona à Tecnologia da Informação de alguma forma. As matérias Introdução à Informática para Automação, Fundamentos da Estrutura da Informação e Metodologia para desenvolvimento de Sistemas foram as que mais diretamente contribuíram como base para o desenvolvimento do software, assim como na escrita da monografia.

O curso como um todo contribuiu para o desenvolvimento do pensamento analítico e abstrato necessário para o aprendizado e a compreensão detalhada das diferentes linguagens (que nunca foram de fato aprendidas durante o curso).

6.1 Perspectivas Futuras

Como a versão do Freeland desenvolvida nesse trabalho foi apenas um MVP, existem muitas possibilidades de aprimoramento. A partir da pesquisa feita com os profissionais do mercado, apresentada no capítulo 5, vários pontos de melhoria foram apresentados, assim como as ferramentas mais importantes.

As primeiras melhorias devem de fato ser feitas na parte gráfica da ferramenta. Uma melhoria no design da plataforma é essencial para a veiculação do aplicativo. Paralelo a isso, formulários que possam ser customizados por cada Freelancer devem ser implementados, visto que cada um trabalha à sua maneira e para usarem a plataforma devem se sentir confortáveis dentro dela.

Em questão de novas ferramentas podemos começar com as 2 que foram deixadas de lado logo no início do projeto por já haverem inúmeras do mesmo tipo no mercado atualmente, um CRM e uma plataforma de Controle Financeiro. Com estudos e ideias criativas se pode desenvolver tanto um CRM como um app de Controle Financeiro dentro da Freeland com alguns diferenciais. Mas não se pode descartar também a possibilidade de trabalhar com empresas já bem segmentadas no mercado e fazer algum tipo de parceria, com os apps dessas empresas funcionando em sincronia com a plataforma da Freeland ou até uma API de integração com essas plataformas.

Um outro ponto dentro da ferramenta a ser levado em consideração, que não estava no escopo desse trabalho, é a geração de alertas para o usuário na ocorrência de algum evento importante, como por exemplo, deadlines de um projeto chegando ao fim. Esses alertas serão importantes quando um Freelancer estiver com muitos projetos em andamento ou com projetos complicados e com bastantes deadlines.

Importante ressaltar que as ferramentas de Network e Career Direction foram tidas como as melhores e pensar em novos *features* para elas em paralelo com as melhorias anteriores é essencial, como por exemplo o sugerido de ter a opção de adicionar o custo da diária dos profissionais de cada Projeto. Uma outra ideia para a Career Direction talvez seja um processo de *gamefication* entre usuários do sistema e com isso a possível troca de informações entre usuários de um mesmo nicho.

REFERÊNCIAS

ESTUDO AVALIA IMPACTO ECONOMICO DO SETOR AUDIOVISUAL NO BRASIL. Disponível em: http://www.meioemensagem.com.br/home/midia/2016/11/30/estudo-avalia-impacto-economico-do-setor-audiovisual-no-brasil.html. Acesso em: 06 set. 2018.

ESTUDOS DA ANCINE APONTAM QUE O MERCADO AUDIOVISUAL BRASILEIRO SEGUE CRESCENDO. Disponível em: https://www.ancine.gov.br/pt-br/sala-imprensa/noticias/estudos-da-ancine-apontam-que-o-mercado-audiovisual-brasileiro-segue. Acesso em 10 set. 2018

UML. Disponível em: https://www.devmedia.com.br/uml/. Acesso em 10 nov. 2018.

DESENVOLVIMENTO WEB COM PYTHON E DJANGO. Disponível em: https://pythonacademy.com.br/blog/desenvolvimento-web-com-python-e-django-introducao. Acesso em: 10 nov. 2018

DOCUMENTAÇÃO DJANGO. Disponível em: https://docs.djangoproject.com/pt-br/2.1/ . Acesso em 15 nov. 2018

RELACIONAMENTO MANY TO MANY. Disponível em: https://ohmycode.com.br/post/relacionamento-many-to-many-no-django/. Acesso em 12 dez. 2018.

TIOBE – INDEX. Disponível em: https://www.tiobe.com/tiobe-index/. Acesso em 08 out. 2018

BOOTSTRAP. Disponível em: https://www.codecademy.com/articles/bootstrap. Acesso em 10 jan. 2018