MEC-SETEC
INSTITUTO FEDERAL MINAS GERAIS - *Campus* Formiga
Curso de Ciência da Computação

PROTÓTIPO DE *WEB SERVICE* PARA A GERAÇÃO DE ESCALAS TRABALHISTAS DE ENFERMEIROS USANDO PROGRAMAÇÃO INTEIRA 0-1

Ana Paula Fernandes de Souza

Orientador: Prof. Me. Walace de Almeida Rodrigues

Formiga - MG
2019

ANA PAULA FERNANDES DE SOUZA

PROTÓTIPO DE *WEB SERVICE* PARA A GERAÇÃO DE ESCALAS TRABALHISTAS DE ENFERMEIROS USANDO PROGRAMAÇÃO INTEIRA 0-1

Monografia do trabalho de conclusão de curso apresentado ao Instituto Federal Minas Gerais - Campus Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Walace de Almeida Rodrigues

Formiga - MG

2019

Souza, Ana Paula Fernandes de
004 Protótipo de web service para a geração de escalas trabalhistas de enfermeiros usando programação inteira 0-1/ Ana Paula Fernandes de Souza. -- Formiga : IFMG, 2019.
57p. : il.

Orientador: Prof. Msc. Walace de Almeida Rodrigues
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1.Web Service. 2. Programação Inteira. 3 . Modelos Matemáticos. 4. Trabalhista. 5. REST. I. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária Msc. Naliana Dias Leandro CRB6-1347

ANA PAULA FERNANDES DE SOUZA

Protótipo de *web service* para a geração de escalas trabalhistas de enfermeiros usando programação inteira 0-1

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais-Campus Formiga, como Requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: 14 de junho de 2019.

BANCA EXAMINADORA

Prof°.Walace de Almeida Rodrigues

Prof.° Fernando Paim Lima

Prof.° Diego Mello da Silva

Sr. Henrique Meira Costa

Este trabalho é dedicado ao meu pai, minha mãe
meu namorado e amigos que me ajudaram nessa caminhada.

Agradecimentos

Primeiramente gostaria de agradecer a Deus, por ter me oferecido saúde, sabedoria e força para superar todas as dificuldades encontradas pelo caminho, e também agradeço ao Anjo São Miguel Arcanjo que me auxiliou em todas as batalhas.

Agradeço aos meus pais, Lucilene e Maurilo, que fizeram o possível para conseguir alcançar meus sonhos. Obrigada pelo amor incondicional, carinho, atenção e por todos os sacrifícios que fizeram para chegar aqui.

Agradeço também a ajuda do meu namorado, Fabiano, por todo companheirismo, amor, carinho e incentivo, principalmente por me auxiliar na revisão dos textos e conselhos durante a graduação.

Agradeço meus familiares que me ajudaram a seguir firme sem desistir nessa caminhada.

Agradeço a empresa *DoctorID*, principalmente o Henrique e o Rafael, por me ajudarem no desenvolvimento e avaliação deste trabalho.

Por fim gostaria de agradecer aos meus colegas que me ajudaram em todos esses anos, a evoluir como programadora e como pessoa. Agradecer também a instituição que colaborou com o conhecimento, principalmente o professor Walace que aceitou a ser meu orientador, Diego e Fernando por aceitar participarem da banca, e o professor Everthon pelas dúvidas esclarecidas em todo percurso na graduação.

"Porque sou eu que conheço os planos que tenho para vocês', diz o Senhor, 'planos de fazê-los prosperar e não de causar dano, planos de dar a vocês esperança e um futuro. Então vocês clamarão a mim, virão orar a mim, e eu os ouvirei. Vocês me procurarão e me acharão quando me procurarem de todo o coração. " (Jeremias 29:11-13)

Resumo

O problema de alocação de recursos é um problema clássico da computação, podendo ser encontrado em vários trabalhos como na alocação de horários em universidades, na distribuição de escalas de de trabalho dos enfermeiros, na designação de equipes para a manutenção de iluminação pública dentre outras formas. Este trabalho é sobre a geração de escalas para hospitais, que atualmente é pouco explorado devido as restrições de turnos, demanda e leis trabalhistas. Para o desenvolvimento desse trabalho foi criado dois modelos matemáticos, para as escalas 12x36 e 12x60, que para desenvolver podem ser programados utilizando programação inteira 0-1. No primeiro modelo, por ser o mais comum de acordo com um especialista na área, foi construído um *web service* com o padrão REST para disponibilizar o recurso.

Palavras-chave: Lei trabalhista, *web service*, programação inteira 0-1, REST.

Abstract

The problem of resource allocation is a classic problem of computing, and can be found in several works, such as the timetabling in universities, scheduling of nurses in hospital, and the designation of public lighting maintenance teams among other forms. This work is about the generation of scales for hospitals, which is currently little explored due to the restrictions of shifts, demand and labor laws. For the development of this work, two mathematical models were created for the 12x36 and 12x60 shedule, which can be programmed using integer programming. In the first model because it is the most common according to a specialist in the area the web service with the REST standard was built to make the feature available.

Keywords: *Employment law, web service, entire programming 0-1*, REST.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

12x36	12 horas trabalhadas e 36 horas de descanso
12x60	12 horas trabalhadas e 60 horas de descanso
PLB	Programação Linear Binária
PL	Programação Linear
REST	*Representational State Transfer* (Transferência de Estado Representativo)
SOAP	*Simple Object Access Protocol* (Protocolo de acesso a objetos simples)
URL	*Uniform Resource Locator* (Localizador padrão de recursos)
XML	*eXtensible Markup Language*(Linguagem de Marcação extensível)
JSON	*JavaScript Object Notation* (Notação de objeto JavaScript)
WSDL	*WebService Description Language*
VMs	Máquinas virtuais
HTML	*Hyper Text Markup Language* (Linguagem de marcação de hipertexto)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
SMTP	*Simple Mail Transfer Protocol* (Protocolo de transferência de correio simples)
JMS	*Java Message Service* (Serviço de Mensagens Java)
UTC	*Universal Coordinated Time* (Tempo Universal Coordenado)

Sumário

1 Introdução

A alocação de recursos é um problema clássico da computação, que tem como definição a atribuição e distribuição de recursos entre diversas tarefas ou atividades que devem ser realizadas. Geralmente os recursos disponíveis não são suficientes para permitir que as tarefas sejam executadas no nível mais alto, dessa forma é necessário encontrar a melhor forma para distribuir os recursos e alcançar o valor ótimo (ANDRADE, 2009). Esse problema pode surgir em diversas variações como por exemplo: na alocação de tabela de horários em universidades (FILHO, 2016), na distribuição de escalas de trabalho em hospitais (RANGER, 2006), na designação de equipes para manutenção de iluminação pública (SILVA, 2017), dentre outros. Geralmente a solução manual é complexa, pois decorre de uma análise combinatória e encontrar a melhor solução pode ser um processo custoso. Também pode acontecer o caso de nem mesmo existir uma solução que atenda todas às restrições.

Uma variante desse problema de alocação de recursos importante para esse trabalho é a geração de escalas de trabalho para funcionários de hospitais. Algumas características tornam essa variação particular: as leis trabalhistas criam restrições especiais, existindo demandas diferentes por setores e turnos, onde a quantidade de funcionários pode variar de um lugar para outro, tendo a possibilidade de favorecer os funcionários com preferência para trabalhar em determinado turno. Atender todas as restrições simultaneamente em um hospital de porte médio-grande e encontrar a melhor distribuição de escala é um problema custoso e que demanda tempo para os organizadores das escalas.

A proposta desse trabalho é desenvolver um *web service* que auxilie na geração da escala de horários de funcionários para cada setor de um hospital, atendendo apenas um setor por vez, obedecendo às leis trabalhistas, as restrições dos turnos e buscando atender as preferências dos funcionários, quando possível.

1.1 Justificativa

O problema de geração de escala para hospitais é pouco explorado devido ao fato das leis trabalhistas colocarem restrições que podem ser regionais: as restrições para um hospital de São Paulo podem ser diferentes das restrições para um hospital do Rio de Janeiro (SINDFIBERJ, 2016). Durante a pesquisa exploratória realizada, foi encontrada apenas uma tese de mestrado nessa área chamada: “Desenvolvimento de um sistema de apoio à decisão para a elaboração da escala periódica do pessoal de enfermagem”, desenvolvida em Ribeirão Preto em 2006, baseada na carga horária dos funcionários com máximo de 4, 4:48, 6, 7 e 8 horas diárias - um grupo de escalas não tão utilizadas

atualmente. (RANGER, 2006).

Junto ao fato do problema não ser muito explorado, existem poucas aplicações no mercado que realizam o gerenciamento de escalas, já as funções existentes apenas auxiliam ao usuário manipular uma escala já estabelecida. Ademais, todas as aplicações que foram encontradas são pagas, conforme detalhamento na fundamentação teórica.

Esse trabalho de conclusão de curso propõe o desenvolvimento de um protótipo de *web service* para atender hospitais na criação de escalas de trabalho para os funcionários dos diversos setores (pediatria, pronto-socorro, entre outros), obedecendo as leis trabalhistas vigentes, demandas dos setores, restrições de turnos e as preferências dos funcionários, quando possível. Uma vez implementado, esse trabalho será bastante útil para a comunidade, pois pode evitar processos trabalhistas e os modelos podem ser úteis para criação de novos projetos. Na construção deste trabalho será utilizado ferramentas gratuitas de modo que não será necessário adquirir a licença das empresas para seu uso.

1.2 Objetivos

Nessa seção serão relatados o objetivo geral e os específicos para o trabalho de conclusão de curso.

1.2.1 Objetivo Geral

Este trabalho tem como objetivo desenvolver um protótipo de *web service* para atender hospitais na criação de escalas de trabalho para os funcionários dos diversos setores, obedecendo às leis trabalhistas vigentes, demandas dos setores, restrições de turnos e quando possível, as preferências dos funcionários. Neste trabalho não será contemplado a criação da interface com o usuário final. Como trabalho futuro, uma interface poderá ser criada posteriormente, tendo referência a este trabalho, para atender os usuários do *web service*.

1.2.2 Objetivos Específicos

Para atingir o objetivo geral, é essencial alcançar as seguintes metas:

- Definir quais escalas e leis trabalhistas serão trabalhadas e vão influenciar o desenvolvimento;
- Pesquisar os aplicativos existentes nessa área;
- Desenvolver os modelos matemáticas para cada escala;

- Estudar quais ferramentas e linguagens serão utilizadas para desenvolver o *web service*;
- Desenvolver o protótipo do *web service* para a geração de escala.

1.2.3 Estrutura do trabalho

Este trabalho é composto por sete capítulos. No primeiro capítulo são apresentadas as ideias iniciais do projeto, com os objetivos e as justificativas. No segundo capítulo é apresentada uma revisão dos aspectos teóricos necessários para a compreensão desse trabalho, bem como algumas plataformas encontradas. No terceiro capítulo são apresentados os materiais e métodos usados para desenvolver o trabalho. No quarto capítulo são apresentados os cenários de trabalho junto com os respectivos modelos matemáticos. No quinto capítulo são apresentados os detalhes da implementação e utilização do *web service*, juntamente com informações sobre os testes e avaliações dos resultados. Por fim, no sétimo capítulo são feitas as considerações finais, seguidas pelas referências bibliográficas.

2 Fundamentação Teórica

Esse capítulo apresenta uma breve introdução sobre alguns conceitos necessários para a compreensão do trabalho, tais como: o problema da alocação de recursos, a questão da definição da escala empresarial para jornadas de trabalho, as principais escalas adotadas na área da saúde – 12x36 (trabalhar doze horas e folgar trinta e seis) e 12x60 (trabalhar doze horas e folgar sessenta) –, a programação inteira 0-1, além de detalhes sobre a construção de *web services* utilizando REST (*Representational State Transfer*) e JSON (*JavaScript Object Notation*). Também são apresentados alguns trabalhos relacionados que serviram como base para o referencial teórico, descrito neste capítulo.

2.1 O problema de alocação de recursos

O problema de alocação de recursos diz respeito ao trabalho de encontrar a melhor forma de distribuí-los para a realização de uma tarefa, pois é comum existir uma limitação da sua disponibilidade e por isso devem ser designados da melhor forma possível para obter um bom resultado. Esse problema normalmente trabalha com dois requisitos: (a) a existência de um objetivo,comumente representado por uma equação que utiliza variáveis de decisão, cujo resultado deve ser minimizado ou maximizado; (b) a existência de restrições, comumente representadas por inequações que utilizam variáveis de decisão para modelar as limitações dos recursos para o uso na atividade (ANDRADE, 2009).

Este tema pode ser encontrado em vários trabalhos com características diferentes, assim como a designação de equipes para manutenção de iluminação pública (SILVA, 2017) e alocação de professores para lecionar disciplinas (FILHO, 2016). Também pode ser encontrado em variações que combinam diferentes objetivos específicos para atingir um certo objetivo global, como pode ocorrer na busca do custo mínimo ou lucro máximo a partir de combinações de diversos ingredientes para produzir um ou vários produtos (BELFIORE; FAVERO, 2013).

A proposta do presente trabalho é resolver o problema das alocações de funcionários em escalas de um hospital considerando as leis trabalhistas, a preferência dos colaboradores e a demanda dos turnos.

2.1.1 Escalas de trabalhos

Segundo Ávila (2015), "o conceito de escala de trabalho refere-se à maneira como cada empresa organiza a jornada de trabalho dos seus funcionários". De acordo com informações coletadas de especialistas na área trabalhista, duas escalas muito utilizadas

são 12x36 e 12x60, razão pela qual elas foram escolhidas para serem tratadas neste trabalho (ÁVILA, 2015).

Para que a escala de trabalho contemple requisitos legais, os seguintes decretos de leis devem ser levados em consideração:

- De acordo com o Súmula nº444 do TST (Tribunal Superior do Trabalho) sobre escala de trabalho 12x36:

> "É valida, em caráter excepcional, a jornada de doze horas de trabalho por trinta e seis de descanso, prevista em lei ou ajustada exclusivamente mediante acordo coletivo de trabalho ou convenção coletiva de trabalho." (TST, 2012)

- De acordo com a convenção coletiva de trabalho 2016/2016 do Rio de Janeiro, Cláusula 28ª sobre a jornada de trabalho, sobre as escalas de trabalho 12x36, 12x48 ou 12x60:

> "A carga de horária dos enfermeiros será trinta e seis (trinta seis) horas semanais, respeitado o limite de seis horas diárias, exceto se o próprio enfermeiro optar pelo aumento da jornada diária para a compensação dos sábados não trabalhados, sendo permitida ainda a adoção da jornada de trabalho, 12x36, 12x48 ou 12x60, neles incluídos o período de refeição, assegurando aos empregados submetidos a tal regime, a marcação de cartões de ponto, somente à entrada e saídas dos plantões, sem que se caracterize jornada extraordinária, sendo necessária a concessão de uma hora de descanso durante esta jornada." (SINDFIBERJ, 2016)

- De acordo com o artigo 66 do Decreto Lei nº 5.452 de 01 de Maio de 1943: entre 2 (duas) jornadas de trabalho deve haver um período mínimo de 11 (onze) horas consecutivas para descanso (PLANALTO, 1943).

- O artigo 67 do Decreto Lei nº 5.452 de 01 de Maio de 1943, diz que deve ser assegurado um descanso de 24 horas consecutivos que deverá acontecer no domingo, no todo ou em parte (PLANALTO, 1943).

Nesse trabalho, o problema de alocação de horários em um hospital será resolvido com a programação inteira 0-1 utilizando modelos matemáticos construídos durante o desenvolvimento como parte deste trabalho. Além disso será implementado um *web service* para disponibilizar este recurso em rede, utilizando o padrão REST com troca de informações via JSON.

2.2 Programação NP-Completude

Uma instância de problema de Programação Linear (PL) é representada por uma função objetivo f e um conjunto de inequações que modelam todas as restrições

$g(i = 1, 2, ..., m)$ envolvidas. Uma instância de PL é apresentada como exemplo na Equação Eq.1, ilustrando como o objetivo e as restrições são modeladas, utilizando variáveis de decisão em funções e inequações. Neste trabalho serão consideradas apenas casos de funções e inequações lineares. Uma função é dita linear quando todas as variáveis envolvidas ocorrem em termos de primeira ordem (BELFIORE; FAVERO, 2013).

$$\begin{aligned} &\max \text{ ou } \min f(x_1, x_2, x_3, ..x_n) \\ &\text{sujeito a} \\ &g_1(x_1, x_2, ..., x_n) \quad \leq, =, \geq \quad b_1 \\ &g_2(x_1, x_2, ..., x_n) \quad \leq, =, \geq \quad b_2 \\ &\ldots \quad \ldots \quad \ldots \\ &\ldots \quad \ldots \quad \ldots \\ &\ldots \quad \ldots \quad \ldots \\ &g_m(x_1, x_2, ..., x_n) \quad \leq, =, \geq \quad b_m \\ &x_1, x_2, ..., x_n \quad \geq \quad 0 \end{aligned} \tag{Eq.1}$$

Uma estratégia possível para resolver um problema de programação linear seria resolver o sistema de equações lineares para encontrar as soluções possíveis e calcular o valor da função objetivo para cada uma delas até encontrar a que produz o melhor resultado. Obviamente este procedimento seria bastante trabalhoso, já que no pior caso deve resolver todos os sistemas antes de escolher a solução que retorna o melhor valor para a função objetivo (ANDRADE, 2009).

Um problema de Programação Inteira 0-1 ocorre quando todas as variáveis de decisão só podem assumir dois valores. Felizmente existem estratégias mais viáveis para buscar soluções exatas para os problemas de programação binária do que recorrer a uma simples busca exaustiva, e aqui serão citados dois métodos distintos:

1. Algoritmo de *branch-and-cut* — No método expande-e-corta, cada iteração introduz um novo corte ou uma nova restrição com o intuito de reduzir o espaço de soluções factíveis. O processo será repetido até encontrar a melhor solução (BELFIORE; FAVERO, 2013).

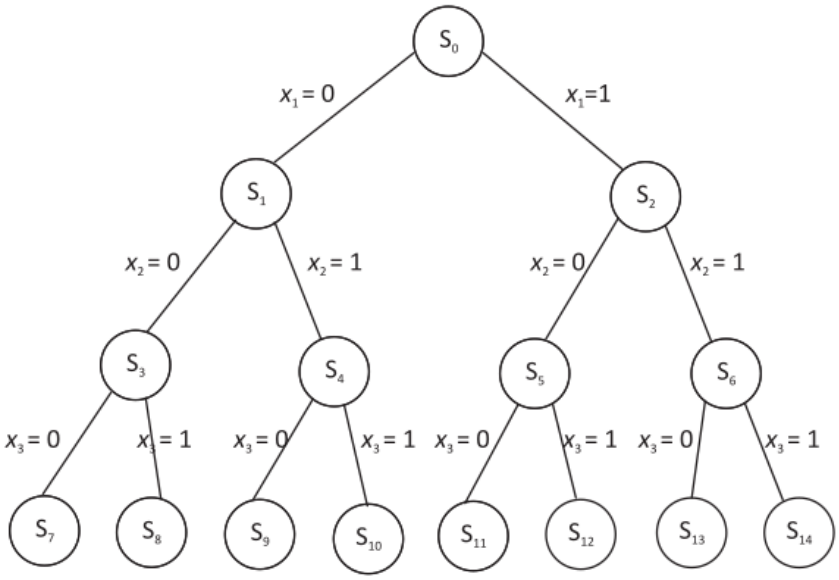
2. Algoritmo de *branch-and-bound* — No método expande-e-limita, uma estratégia de divisão e conquista é adotada na busca pela solução. O problema principal é divido em problemas menores e os subproblemas que não oferecem condições para atingir uma solução ótima são descartados (BELFIORE; FAVERO, 2013).

Uma instância do programação linear binária é apresentada como exemplo na Equação Eq.2:

$$\begin{aligned} min\ z(2x_1, 3x_2, 2x_3) \\ sujeito\ a \\ 2x_1, 2x_2 \geq 2 \\ 6x_1, 4x_2, 4x_3 \geq 6 \\ x_1, x_2, x_3 \in \{0, 1\} \end{aligned} \tag{Eq.2}$$

A Figura 1 ilustra a estrutura da busca da instância apresentada na Equação Eq.2:

Figura 1 – Árvore do modelo matemático Eq.2 com *brand-and-bound*



Fonte: (BELFIORE; FAVERO, 2013)

O problema de alocação de recursos é originalmente binário, ou ser conhecida como programção inteira 0-1, pois nele as variáveis podem assumir apenas dois valores: 0=livre e 1=alocado. O estado inicial começa em S_0 e, a cada nova restrição adicionada, mais uma aresta e um ramo são adicionados na árvore. Por exemplo: S_1 representa o novo estado criado a partir do estado inicial (S_0) alterado pela inclusão da restrição $x_1 = 0$. O estado S_6 representa o novo estado criado de (S_0) com a inclusão das restrições $x_1 = 1$ e $x_2 = 1$ (ou o estado (S_2) com a inclusão da restrição $x_2 = 1$). É óbvio que a árvore com a enumeração completa contém todas as soluções possíveis para o problema original (BELFIORE; FAVERO, 2013).

O problema da criação de escalas de trabalhos em hospitais é uma variação da Programação inteira 0-1, visto que todas as variáveis do modelo devem ser binárias, pois

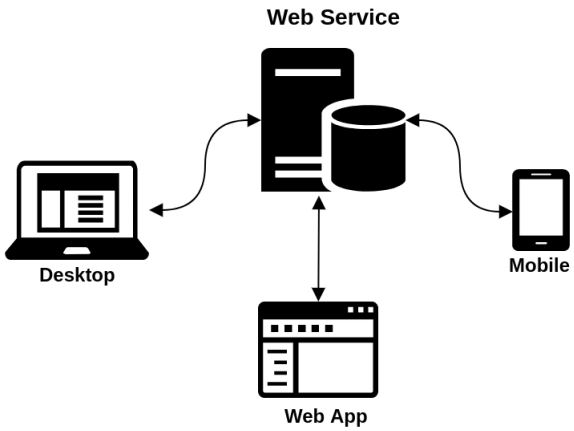
existem apenas duas possibilidades: estar de plantão (valor um) ou folgar (valor zero), não podendo haver resultados com valores contínuos ou outros valores discretos pois esse normalmente não tem significado na modelagem utilizada.

2.3 Web Service

Um *web service* pode ser definido como um conjunto de métodos que são invocados por outros *softwares* utilizando tecnologias *web*. Pode ser utilizado para transferir dados através de protocolos de comunicação para diferentes plataformas e independentemente das linguagens de programação. Permitem ser reutilizados em sistemas já existentes em uma organização e acrescentar-lhes novas funcionalidades sem que seja necessário recriar o sistema. Assim, é possível integrar novas funcionalidades ou atualizar uma já existente em apenas um lugar, e uma vez modificado o serviço todos os *softwares* que fazem uso terão acesso à alteração (OPENSOFT, 2016).

Por exemplo, na Figura 2, uma aplicação em *Desktop* desenvolvida em *Python*, um aplicativo móvel criado no *Android Studio*, e um sistema *web* construído em PHP, todas essas plataformas podem acessar o mesmo *web service* sem precisar saber em qual linguagem foi desenvolvido ou em qual plataforma está hospedado na rede.

Figura 2 – Exemplificando a utilização de *web service*

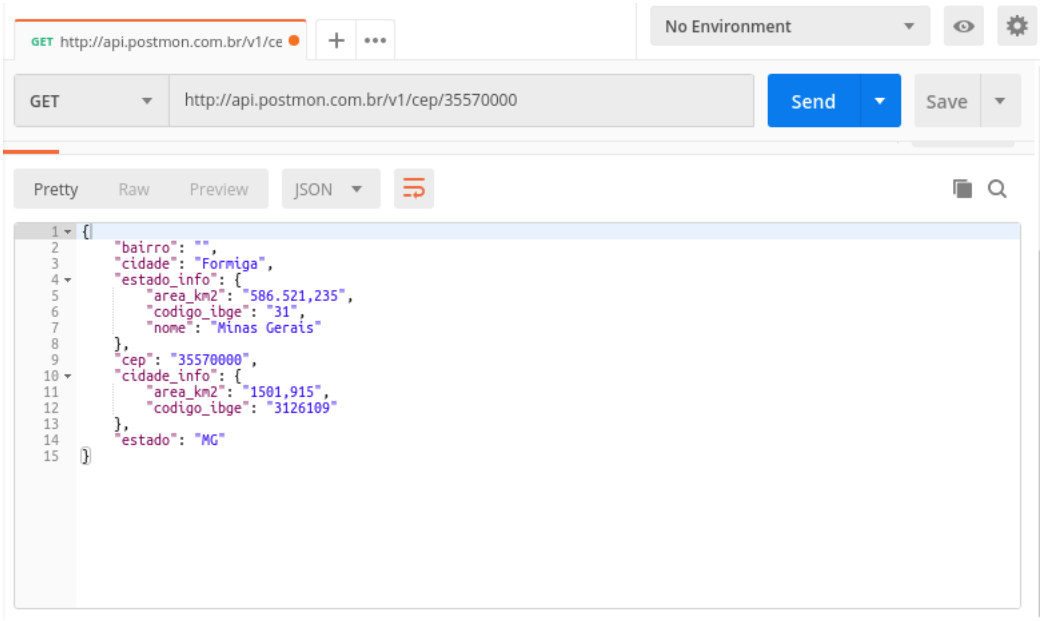


Fonte: Imagem ilustrativa da *internet*.

Uma funcionalidade que pode ser incorporada em vários sistemas seria, por exemplo, descobrir uma localização pelo CEP (Código de Endereçamento Postal): isso poderia ser realizado utilizando uma API((Interfaces de Programação de Aplicações) disponível para esse fim[1]. Na Figura 3 é mostrado como CEP 35570-000 pode ser utilizado para buscar as informações.

[1] Disponível em <http://api.postmon.com.br/v1/cep/cepPesquisa>. Acesso em 06/2019

Figura 3 – Consulta na API com o *PostMan* pelo CEP 35570-000 via *GET*



Fonte: Elaborado pela autora

Duas tecnologias normalmente utilizadas para a construção do *web service* são : REST (*Representational State Transfer*) ou SOAP (*Simple Object Access Protocol*). Na próxima seção será relatado a diferença entre as duas e o motivo para qual foi utilizado o REST.

2.3.1 REST vs SOAP

O SOAP foi projetado para garantir que programas diferentes possam trocar informações (ROZLOG, 2013). Ele é baseado em XML (*eXtensible Markup Language*) e pode ser utilizado de três maneiras distintas: a) com o envelope, que define o conteúdo da mensagem e informa como processá-la; b) com um conjunto de regras de codificação para os tipos de dados; c) com o *layout* para os procedimentos de chamadas e respostas. Uma das vantagens é o uso de um método de transporte variado, podendo ser utilizado qualquer meio de transporte existente para enviar sua requisição, desde SMTP (*Simple Mail Transfer Protocol* – um protocolo padrão da camada de transporte para envio de *e-mails* através da Internet) até JMS (*Java Message Service* – um serviço da camada de aplicação para permitir troca de mensagens entre diferentes programas).

REST é um estilo de arquitetura para criação de *web services*. Ele teve como origem a tese de doutorado de Fielding (2000), um co-autor dos protocolos HTTP (*Hypertext Transfer Protocol*). Assim é notável a influência do HTTP no REST (SAUDATE, 2015):

- Métodos HTTP — esse protocolo oferece vários métodos que foram incorporados ao REST, sendo os principais *GET* (recuperar os dados identificados pela URL (*Uniform Resource Locator*), *POST* (cria um novo recurso), *PUT* (atualiza um novo recurso) e *DELETE* (apaga o recurso).

- *Status* do HTTP — cada requisição enviada pelo servidor retorna um código, dividido em cinco famílias: 1XX (Informacionais), 2XX (Código de sucesso), 3XX (Código de redirecionamentos, 4XX (Erros causados pelo cliente) e 5XX (Erros originados pelo servidor). Essa estrutura também foi implementada no REST.

REST tem como fundamentos que tudo é definido como recursos, e estes são trafegados pelo protocolo. Todos os recursos são representados por URL's que é uma forma de identificar unicamente (SAUDATE, 2015).

A Tabela 1 apresenta as principais diferenças entre os duas tecnologias:

Tabela 1 – Principais diferenças entre SOAP e REST

SOAP	REST
O SOAP usa interfaces de serviço para expor sua funcionalidade a aplicativos clientes. No SOAP, o arquivo WSDL (*Web Service Description Language*) fornece ao cliente as informações necessárias que podem ser usadas para entender quais serviços o serviço da *web* pode oferecer.	REST usa localizadores *Uniform Service* para acessar os componentes no dispositivo de *hardware.*
SOAP requer mais largura de banda para seu uso. Como as mensagens contêm muitas informações dentro dela, a quantidade de transferência de dados é grande.	O REST não precisa de muita largura de banda quando as solicitações são enviadas ao servidor.
O SOAP apenas funciona no formato XML para troca de informações.	O REST permite diferentes formatos de dados, como texto simples, HTML(*Hyper Text Markup Language)*, XML, JSON, etc. Mas o formato mais comum para transferir dados é o JSON.

Fonte: GURU99 (2018)

O protocolo SOAP utiliza como troca de mensagens XML, enquanto o REST oferece maior diversidade, embora na maioria das vezes utilizado JSON. Este trabalho irá utilizar REST com JSON para trocar informações, evitando gastar muita largura de banda. Na próxima seção será apresentado mais detalhes sobre o JSON.

2.3.2 JSON VS XML

XML é uma sigla que significa *eXtensible Markup Language*, ou linguagem de marcação extensível. Por ter esta natureza extensível, conseguimos expressar grande parte de nossas informações utilizando este formato.

Da mesma forma que HTML (Hyper Text Markup Language), o XML utiliza etiquetas (*tags*) para representar os dados. Enquanto HTML especifica como os dados devem ser exibidos, XML se preocupa com o significado dos dados armazenados (MENENDEZ, 2002). Segue um exemplo de código 1 contendo alunos e médias de um curso, utilizando XML.

Código Fonte 1 Exemplo de um arquivo XML

```
1
2  <curso>
3   <aluno>
4       <nome>Danielle</nome>
5       <MGP>7.9</MGP>
6   </aluno>
7   <aluno>
8       <nome>Cinthia</nome>
9       <MGP>6.5</MGP>
10  </aluno>
11 </curso>
```

O JSON é uma sigla para *JavaScript Object Notation*. É uma linguagem de marcação criada por Douglas Crockford (JSON, 2019). Tem por principal motivação o tamanho reduzido em relação a XML. Acaba tendo uso mais propício em cenários onde largura de banda é um recurso crítico. Segue um exemplo de código 2 contendo alunos e médias de um curso, utilizando JSON, que pode ser comparado o código 1 em XML.

Código Fonte 2 Exemplo de um arquivo JSON

```
1   {
2      "curso": [
3               {
4                   "nome": "Danielle",
5                   "media": "7.9"
6               },
7               {
8                   "nome": "Cinthia",
9                   "preferencia": "6.5"
10              }
11              ]
12 }
```

Dessa forma, no desenvolvimento deste trabalho, foi necessário a utilização do JSON, devido a grande quantidade de informações necessárias para criação da escala e o seu retorno, trataremos as informações de entrada e saída no capítulo 5, que relata o desenvolvimento.

2.4 Trabalhos Correlatos

É fundamental o conhecimento das ferramentas próximas às áreas, para saber quais as limitações do mercado. Nas pesquisas foram encontradas uma tese de mestrado e duas ferramentas comerciais Pega Plantão e *DoctorID*.

2.4.1 Literatura sobre geração de escala automatizada

Na dissertação de mestrado "Desenvolvimento de um sistema de apoio à decisão para a elaboração da escala periódica do pessoal de enfermagem", desenvolvida em Ribeirão Preto em 2006, o autor construiu um sistema para auxiliar na elaboração da escala mensal dos funcionários do Hospital das Clínicas da Faculdade de Medicina de Ribeirão Preto da Universidade de São Paulo (HCFMRP-USP) no período de fevereiro de 2005 e a setembro de 2006 (RANGER, 2006).

A sua metodologia foi baseada nos conceitos de um sistema de apoio à decisão e seu desenvolvimento utilizou ferramentas livres (*Open Source*), para permitir que qualquer instituição/empresa possa adquirir e fazer uso do sistema sem que os custos de manutenção de *softwares* pagos.

O problema foi desenvolvido com programação inteira e sua função objetivo é minimizar a insatisfação dos funcionários com a estala de trabalho, tendo as seguintes restrições :

- O enfermeiro deve trabalhar um único turno por dia;
- O enfermeiro deve ter pelo menos uma folga por domingo por mês, assegurando o direito de um descanso de vinte e quatro horas consecutivas;
- O número de enfermeiros trabalhando por turno deve ser maior ou igual à quantidade demanda necessária de profissionais;
- Respeitar o número mínimo de folgas no período e as jornadas de trabalho, que pode ser visualizadas na Tabela 2.

Tabela 2 – Tabela de jornada de trabalho da dissertação de (RANGER, 2006)

Jornada semanal	Jornada diária máxima permitida
20 horas	04:00 horas
24 horas	04:48 horas
30 horas	6:00 horas
35 horas	07:00 horas
40 horas	8:00 horas

Fonte: Ranger (2006)

- Respeitar a carga horária de trabalho mensal;
- Folgar a cada seis dias no máximo;
- Cada enfermeiros deve trabalhar uma quantidade de noturnos;
- Noturnos não podem ser seguidos;
- Noturnos não podem ser seguidos de manhãs ou tardes pois devem descansar trinta e seis horas seguidas;
- Folgas seguidas depois de fazer noturno, auxiliando na regra anterior para cumprir o descanso;
- Enfermeiro trabalha em turno fixo, garantindo por exemplo quando ele trabalhar em turnos noturnos, preservar seu tempo de descanso de trinta e seis horas de modo a não trabalhar de manha ou tarde até completar o descanso;
- Agrupar turnos para que os enfermeiros organizem suas atividades fora do trabalho;
- Valor máximo de cada variável deve ser menor ou igual a um. De modo que o valor um representa que o funcionário irá trabalhar e o zero representa que ele está de folga.

Em sua conclusão do trabalho, mostrou que o resultado das escalas obtidas são de extrema rapidez, levando-se crer na aceitação por parte dos enfermeiros encarregados da geração. (RANGER, 2006). Todavia o código do Ranger não foi disponibilizado para realizar os testes, de modo que este trabalho serviu apenas como inspiração para o desenvolvimento do atual trabalho.

2.4.2 Pega Plantão

Pega Plantão[2] é uma plataforma que oferece soluções baseadas em plantões, escalas e vagas de emprego para médicos, enfermeiros e técnicos em enfermagem, auxiliando na gestão para hospitais e entidades de saúde.

Devido a utilização dessa ferramenta, o gestor realiza o cadastro dos funcionários de cada setor e depois constrói a escala dos seus empregados, podendo replicá-la nos meses seguintes e realizar modificações. Existe a possibilidade de deixar alguns plantões em aberto, contendo informações como valor do plantão, horário de início e fim, especificação do local (setor), para que profissionais cadastrados se candidate para a vaga.

Por um aplicativo disponível para celular, o funcionário poderá consultar todos os plantões que serão realizados; caso não possa cumprí-lo ou a plataforma aponte para choque de horários, ele pode anunciar para os colegas de trabalho, caso alguém se interessar em assumir, será apenas necessário se candidatar e o anunciante escolherá quem assumirá o plantão.

No setor financeiro, o responsável pelos pagamentos poderá consultar um relatório da produtividade e o valor a ser pago para cada funcionário, uma vez que este faz *check in* e *check out* pelo sistema a cada plantão realizado. O plantonista poderá acompanhar o valor que irá receber no final do mês atualizado.

Para realizar os testes nessa ferramenta é necessário entrar em contato com a empresa, pois não existe uma versão gratuita ou para teste. Mesmo entrando em contato pelo *site* através do formulário, não foi concedido o teste mas, em contra partida, foi marcado uma reunião com um dos donos para explicar o funcionamento da ferramenta. Depois da reunião foi disponibilizado manuais da ferramenta e arquivos da empresa, onde pode-se retirar as informações escritas nesse tópico.

2.4.3 *DoctorID*

O *DoctorID*[3] é uma empresa fundada em 2013, que busca desenvolver e comercializar *softwares* com uma plataforma *web*, criada para gerenciar equipes e escalas médicas.

Na plataforma uma escala fixa é estabelecida levando em consideração revezamentos quinzenais, quatro ou cinco semanas. Com essa funcionalidade é possível gerar as escalas reais de cada médico, o qual tem o controle em seu aplicativo celular de quando e onde deve realizar o plantão. Além disso, o sistema leva em consideração períodos de indisponibilidade para a geração de escala, tais como: férias, faltas, licenças, entre outras.

Qualquer médico pode transferir determinado plantão para outro membro da equipe, podendo trocar plantões entre sí ou pedir substituição. Como forma de segurança o gestor

[2] Disponível em <https://www.pegaplantao.com.br/> Acesso em 06/2019
[3] Disponível em <https://www.doctorid.com.br/> Acesso em 06/2019

detém a palavra final, ou seja autoriza ou rejeita cada uma dessas ações de alteração solicitadas pelo médico.

Para o controle de horário, existem duas opções: o uso de Código *QR - Quick Response*, onde o médico utiliza o aplicativo instalado em seu celular para registrar o horário de chegada e saída do hospital. A segunda opção é utilizar a localização fornecida pelo *GPS-Global Positioning System*.

Alguns problemas que podem ser evitados utilizando a plataforma, segundo os autores:

- Conflitos de horários: quando o médico é alocado duas ou mais vezes no mesmo horário, sendo em equipes e locais diferentes.
- Conflito de deslocamento: quando o médico termina um plantão em um endereço e inicia outro em localidades diferentes.
- Maiores e menores alocações: quando um médico possui uma alocação muito alta dentro de uma escala, há um risco de comprometimento de seus plantões quando por exemplo, ele ficar doente e não puder realiza-lo. Nessa situação vários plantões ficariam sem o médico. No entanto, existindo um planejamento para balancear a quantidade de horas de cada médico da escala, minimizando o impacto.

Essa ferramenta não possui uma versão para teste e foi necessário entrar em contato com a empresa para poder para realizar testes. A empresa entrou em contato e foi marcado uma reunião para conversar sobre a plataforma. Um dos sócios achou interessante o trabalho e auxiliou no desenvolvimento do mesmo. Depois da reunião foi disponibilizado manuais e arquivos da empresa, onde pode-se retirar as informações desse tópico.

3 Materiais e métodos

Este capítulo tem como objetivo fornecer uma visão geral dos materiais *hardware* e/ou *software*) utilizados no desenvolvimento do *web service*, juntamente com a metodologia empregada na construção dos modelos matemáticos utilizados para o tratamento das escalas de trabalhistas. Serão detalhados os equipamentos físicos, as tecnologias *web*, as linguagens de programação e dentro outros detalhes pertinentes.

3.1 *Hardware*

Todo trabalho foi desenvolvido em uma máquina com a seguinte configuração:

- Processador Intel i5-3210M CPU 2.500 GHz;
- 8 GB de memória RAM;
- Sistema operacional Linux *Mint* 18.3 "Sylvia- *Cinnamon* (64-bit).

3.2 *Software*

Essa seção retrata de todos os *softwares* que foram analisados e depois explica o motivo da utilização no *web service*.

3.2.1 *Docker*

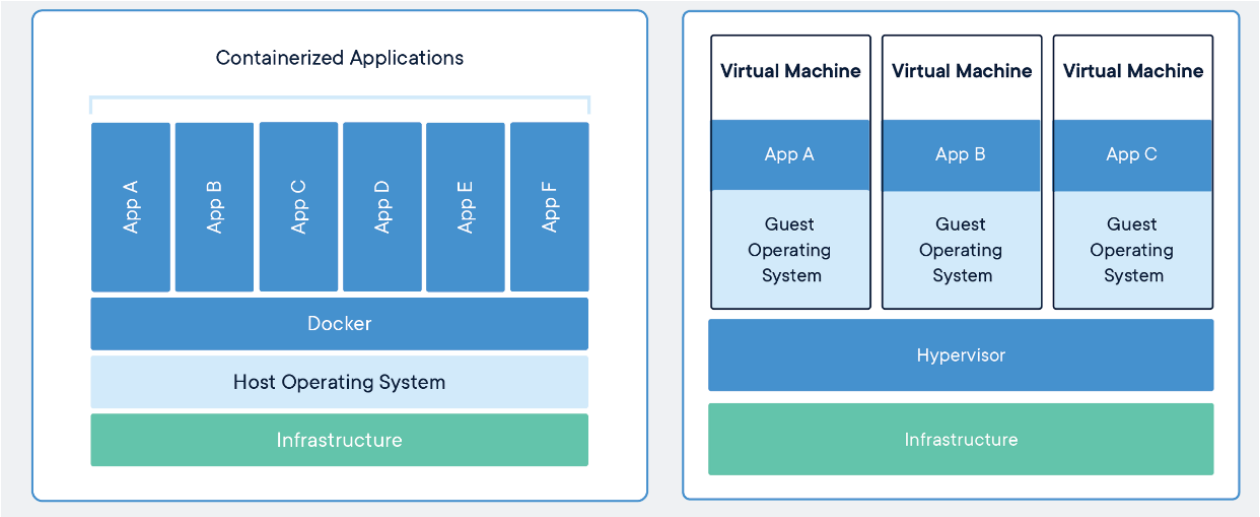
Docker[1] é uma tecnologia que foi lançada em 2013, para criar e manter contêineres, ou seja, é responsável por armazenar vários serviços de forma isolada do sistema operacional. Um contêiner é uma unidade padrão que empacota o código e todas as suas dependências para que o aplicativo seja executado de maneira rápida e confiável de um ambiente de computação para outro, por exemplo, pode-se desenvolver uma aplicação no *Windows* 10 e colocar em um servidor Linux, que não sofrerá nenhum conflito com as dependências (DOCKER, 2013).

Máquinas virtuais (VMs) são uma abstração de *hardware* físico, transformando um servidor em vários servidores. O *hypervisor* permite que várias VMs sejam executadas em uma única máquina. Cada VM inclui uma cópia completa de um sistema operacional, com seus binários necessários e bibliotecas do sistema. Já com a utilização do *docker* vários contêineres podem ser executados na mesma máquina e compartilhar apenas o *kernel*

[1] Disponível em <https://www.docker.com/> Acesso em 06/2019

do sistema operacional com outros contêineres, cada um sendo executado em processos isolados no espaço do usuário (DOCKER, 2013). A diferença ilustrativa pode ser observada na Figura 4.

Figura 4 – Diferença entre *Docker* e as máquinas virtuais



Fonte: Docker (2013)

Para o desenvolvimento desse trabalho foi utilizado o *Docker*, pois a aplicação poderá ser levada para qualquer plataforma sem sofrer interferência das dependências e ficará isoladas do sistema operacional.

3.2.2 Linguagem *Python*

Python é uma linguagem lançada em 1991, interpretada de alto nível e que suporta múltiplos paradigmas de programação: imperativo, orientado a objetos e funcional. Possui algumas estruturas de dados embutidas como tuplas, listas e dicionários (CRUZ, 2015). Segue algumas características para a escolha da utilização nesse trabalho:

- Por ser uma linguagem de código aberto: a comunidade utiliza e compartilha soluções para diversos problemas;
- A sintaxe básica é bem simples e pode ser compreendida ao analisar o código;
- Possui uma grande variedade de bibliotecas disponíveis para serem utilizadas.

Devido ao *python* possuir um gerenciador de pacotes chamado PIP, é possível remover, atualizar, instalar os pacotes nos projetos de maneira simples, sem precisar ficar instalando as dependências de forma manual (PINHEIRO, 2018).

3.2.3 Biblioteca PuLP

PuLP [2] é uma biblioteca que fornece os subsídios para desenvolver modelos de programação linear incluindo a inteira e binária (0-1). Essa biblioteca foi escolhida entre as demais, devido às seguintes características: por ser gratuita, a documentação é simples, baseada em explicações com exemplos práticos e utilização de estruturas padrões do *python* (dicionários).

3.2.4 *Django Rest Framework VS Flask*

O *Flask* [3] é um *web framework*, em *python*, de código aberto lançado em 2010. Este *framework* é minimalista, de modo que seu funcionamento é estendido com a utilização de bibliotecas, e dessa forma os programadores podem construir o que desejarem de acordo com as bibliotecas (IMASTER, 2018).

A *Django Rest Framework* [4] é uma biblioteca *Django* que viabiliza de forma simples a criação de API'S REST em projetos *Django* usando de meios já conhecidos dos desenvolvedores para proporcionar produtividade na criação das API'S (IMASTER, 2018).

A escolha deste trabalho foi a utilização de *Django Rest Framework* devido ao seu código já possuir o padrão REST, otimizando assim no tempo de desenvolvimento, pois será necessário programar apenas os recursos de entrada para o *web service* funcionar.

3.2.5 *PostMan*

Devido a esse trabalho ser desenvolvido pelo modelo REST e a troca de mensagens ser realizadas pelo HTTP, e este não possui interface gráfica, atuando apenas no recebimento e movimentação dos dados através de requisições. Dessa forma para testar esse tipo de aplicação, envolve simular requisições a partir de um *software* (cliente), para que se tenha controle sobre os dados trafegados.

Para suprir essa necessidade, foi utilizado o *PostMan* [5], uma ferramenta gratuita, que permite realizar requisições HTTP a partir de uma interface simples e intuitiva, facilitando no teste e depuração dos serviços criados nesse trabalho.

3.2.6 *PyCharm Community*

O *Pycharm Community* [6] é uma ferramenta gratuita e multiplataforma com versões para *Windows*, *MacOS* e Linux é desenvolvido pela empresa *JetBrains*[7], fornece análise de

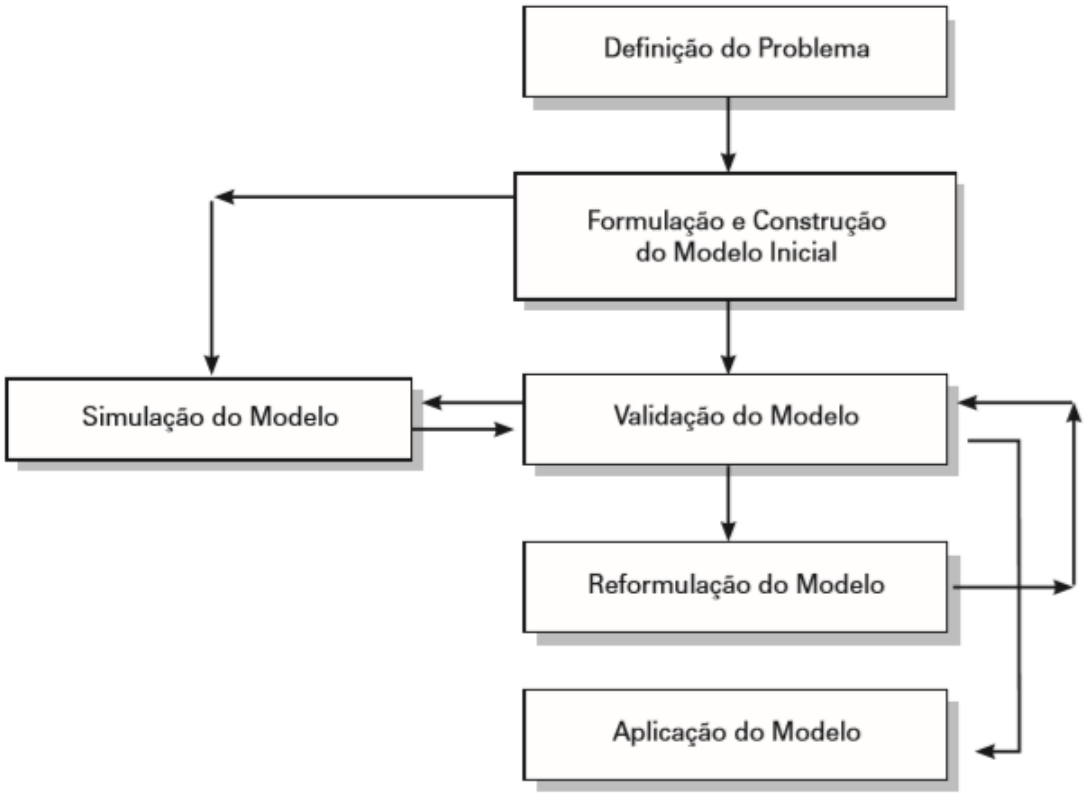
[2] Disponível em <https://pythonhosted.org/PuLP/> Acesso em 06/2019
[3] Disponível em <http://flask.pocoo.org/> Acesso em 06/2019
[4] Disponível em <https://www.django-rest-framework.org/> Acesso em 06/2019
[5] Disponível em <https://www.getpostman.com/> Acesso em 06/2019
[6] Disponível em <https://www.jetbrains.com/pycharm/> Acesso em 06/2019
[7] Disponível em <https://www.jetbrains.com/> Acesso em 06/2019

código, depurador gráfico, autocompletamento e capacidades de navegação entre os códigos, de modo a facilitar a escrita e o desenvolvimento. Possui suporte para desenvolvimento de aplicações web como: *Django*, *Flask* e também suporta HTML, CSS, *JavaScript* e XML.

3.3 Metodologia

O desenvolvimento dos modelos nesse trabalho seguiram uma metodologia no roteiro de construção de modelos proposto por Goldbarg (2005), cujas etapas são mostradas na Figura 5.

Figura 5 – Passos para a construção de modelos matemáticos



Fonte: Goldbarg (2005)

Segundo Goldbarg, a fase de definição do problema é a mais importante pois o mesmo deve ser traduzido em elementos palpáveis englobando objetivos, variáveis de decisão ou controle e níveis de detalhamentos. Em seguida encontrada-se a fase de formulação, é onde as fórmulas ou equações do modelo não existem prontas e elas têm de ser identificadas ou criadas, desta forma são definidos os tipos de variáveis, bem como o nível apropriado de agregação dessas variáveis. O modelo deverá ser adequado à natureza dos dados de entrada e de saída, bem como ser capaz de expressar as funções de desempenho que possivelmente

serão exigidas no processo de otimização. Já a construção de modelos determina a inclusão de parâmetros e constantes que serão responsáveis pela definição e dimensionamento das relações entre as variáveis do modelo.

Subsequente na fase de validação do modelo, compara-se seu comportamento com a realidade e, se necessário, atuar sobre esses elementos de forma a aproximar ao máximo o comportamento do sistema modelo ao do sistema real prospectado da fase de simulação tendo como regra geral. A validação exige um procedimento iterativo que vai refinando o ajuste entre o modelo e a realidade utilizando a fase de reformulação para realizar modificações no modelo. A última etapa é a aplicação do modelo.

Na primeira fase, foi considerado o que é essencial para o desenvolvimento do modelo matemático, que pode ser citado :

- Escala de trabalho 12 x 36;
- Será estabelecido a existência de apenas dois turnos (diurno/noturno);
- O funcionários deverá ter pelo menos uma folga por semana, sendo que uma dessas ocorrer no domingo;
- Quando o funcionário trabalhar no turno noturno e ele for mudar de turno, ele deverá receber duas folgas subsequentes, como forma de garantir as trinta e seis horas mínimas de descanso após o noturno;
- Funcionários podem escolher o dia de preferência para trabalhar de formar a garantir a satisfação da classe trabalhista dessa escala, mas será atendido assim que possível.

Depois de estabelecidos os fundamentos para o desenvolvimento da escala, foi criado um modelo inicial, e o resultado deste foi mostrado para um especialista e acabou sendo necessário algumas modificações do modelo para torna-lo mais otimizado e próximo da realidade.

No próximo capitulo será relatado com detalhes os modelos matemáticos criados nesse trabalho, juntamente com a função objetivo e suas restrições.

4 Modelos

Os modelos matemáticos baseiam-se na pressuposição que todas as informações e variáveis relevantes do problema de tomada de decisão podem ser quantificadas, ou seja contadas. Isso nos leva a utilizar símbolos matemáticos para apresenta-las e usar funções matemáticas para descrever as ligações entre elas (ANDRADE, 2009).

Os modelos matemáticos de otimização são compostos por uma função denominada função objetivo, que consiste na maximização ou minimização de um problema de decisão, essa função deve respeitar um sistema linear de igualdades ou desigualdades que recebem o nome de restrições do modelo.

Neste capítulo são apresentados dois cenários trabalhistas diferentes presentes em alguns hospitais. Primeiro serão definidas as características, depois será apresentado o modelo matemático para cada caso com sua função objetivo e suas restrições.

4.1 Características do 1° Cenário

No modelo de escala 12x36, o funcionário tem direito a folga pelo menos em um domingo no mês. O funcionamento da escala é um dia de trabalho - o que corresponde à doze horas de trabalho e o restante do dia em descanso mais um dia de folga-, completando as trinta e seis horas de descanso.

Todos os funcionários compõem um único conjunto onde cada um pode ser alocado em apenas dois turnos, diurnos e noturnos. De acordo com um consultor especialista na área, o profissional pode preferir um turno para trabalhar, e este deverá será escalado a maior quantidade possível.

4.1.1 Formulação do modelo

O problema foi representado conforme a Função Objetivo Eq.1a, de modo a maximizar a quantidade de funcionários nos turnos, e em todos os casos cumprir com a escala de trabalho (12x36) e respeitar todas as restrições. As variáveis foram nomeadas $W * Xijt$, onde os conjuntos são:

- $N = \{1, 2, ...,$ Total de Enfermeiros$\}$
- $M = \{1, 2, ..., 31\}$
- $S \subseteq M\{s|s$ é um domingo do mês $\}$

- $T = \{1, 2\}$

  Conjunto N representa todos os enfermeiros disponíveis para construir a escala, de modo que os que estão de ferias, licença ou outros fatores não entram no conjunto. Conjunto M representa os dias do mês que será gerado a escala,(nesse trabalho é contemplado no máximo 31 dias). Conjunto S são valores de M que representam domingos no determinado mês da escala. E as variáveis são construídas a partir de :

- i: representa o funcionário de código i;

- j: representa o dia j do mês;

- t: representa o turno, podendo assumir valores diurno(1) ou noturno(2);

  Os parâmetros de entrada para o modelo são:

- W: representa a preferência do funcionário em trabalhar em determinado turno. Por exemplo, se um funcionário optar por trabalhar no turno diurno, em todas as variáveis que faz referência a esse turno será atribuído o valor dois e as restantes o valor um.

- D: representa a quantidade de demanda de funcionários em determinado turno.

$$max \sum_{i \in N} \sum_{j \in M} \sum_{t \in T} W_i^t * X_{ij}^t \tag{Eq.1a}$$

$$subject \quad to$$

$$X_{ij}^1 + X_{ij}^2 + X_{i(j+1)}^1 + X_{i(j+1)}^2 \leq 1, \quad \forall i \in N, \forall j \in M/\{31\} \tag{Eq.1b}$$

$$X_{ij}^2 + Xij^1 + X_{i(j+1)}^2 + X_{i(j+1)}^1 \leq 1, \quad \forall i \in N, \forall j \in M/\{30, 31\} \tag{Eq.1c}$$

$$X_{i30}^2 + X_{i31}^1 + X_{i31}^2 \leq 1, \quad \forall i \in N \tag{Eq.1d}$$

$$\sum_{i \in N} X_{ij}^t \geq D_j^t, \quad \forall i \in N, \forall t \in T \tag{Eq.1e}$$

$$\sum_{s \in S} \sum_{t \in T} X_{is}^t \leq |S| - 1, \quad \forall i \in N \tag{Eq.1f}$$

$$X_{ij}^1 + X_{ij}^2 \leq 1, \quad \forall i \in N, \forall j \in M \tag{Eq.1g}$$

$$X_{ij}^t \in \{0, 1\} \quad \forall i \in N, \forall j \in M, \forall t \in T \tag{Eq.1h}$$

Restrições :

O problema está sujeito às seguintes restrições:

- **Garante que os funcionários trabalhem apenas 12x36:**

  As restrições nas Eq.1 e Eq.1c garantem que cada funcionário trabalhe em um turno e folgue três turnos em sequência – essas restrições são construídas com quatro

variáveis. Existe uma exceção para o dia trinta noturno, representada nas restrições na Eq.1d.

- **Garante a quantidade necessária de funcionários por turno:**

  As restrições na Eq.1e garantem que em cada um dos turnos de determinado dia trabalhe uma certa quantidade de profissionais – uma restrição é construída para cada um dos turnos (diurno/noturno) e dias separadamente, de forma que a soma de todos os funcionários em determinado dia e turno seja maior ou igual à demanda.

- **Garante aos funcionários pelo menos uma folga por domingo no mês:**

  As restrições na Eq.1f garantem que durante o mês pelo menos uma das folgas do mês deve ser concedida em um domingo. Exemplo: no mês de agosto de 2018, j assume os valores $5, 12, 19, 26$ e a desigualidade à direita será atribuído valor três.

- **Garante que os funcionários trabalhem apenas um turno por dia:**

  As restrições na Eq.1g garantem que todos irão trabalhar apenas um único turno por dia, desde que não exista turnos de vinte e quatro horas. Essa restrição também reforça as Eq.1 e Eq.1c para garantir que cada funcionário tenha direito a folga de vinte e quatro horas consecutivas.

- **Garante que o valor de cada variável seja zero ou um:**

  As restrições na Eq.1h garantem que todas as variáveis do problema recebam valor zero ou um, sendo que o valor um indica que o profissional foi escalado para trabalhar e zero representa folga. Dessa forma essas restrições garantem que nenhum funcionário trabalhará duas vezes ou mais no mesmo turno.

De acordo com o *DoctorID* - uma empresa especialista na área trabalhista - esse cenário é o mais utilizado na geração de escalas, dessa forma apenas esse modelo foi implementado no *web service*, por causa da sua importância.

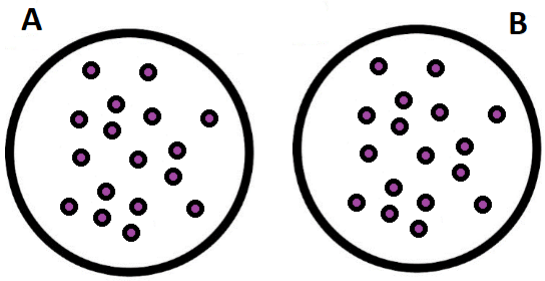
4.2 Características do 2º Cenário

Na escala 12x60, o funcionário tem direito a folga pelo menos em um domingo no mês. O funcionamento da escala é um dia de trabalho - o que corresponde à doze horas de trabalho e o restante do dia em descanso - mais dois dias de folga, completando as sessenta horas de descanso. Os conjuntos de enfermeiros são separados por turnos diurno ou noturno e o número de profissionais trabalhando por cada turno deve ser maior ou igual à quantidade necessária de profissionais.

4.2.1 Formulação do modelo

Uma forma de ilustrar o modelo é mostrado na Figura 6. O conjunto A, representa todos os funcionários que trabalham no turno diurno e o B trabalham noturno, dessa forma cada restrição deve ser construída para os dois conjuntos.

Figura 6 – Conjunto do cenário 12x60



Fonte: Elaborado pela autora

O cenário foi modelado conforme a Função Objetivo Eq.2a, de modo a minimizar a insatisfação dos funcionários com a escala de trabalho (12x60) respeitando todas as restrições, onde os conjuntos são:

- $N = \{1, 2, ...,$ Total de Enfermeiros$\}$
- $M = \{1, 2, ..., 31\}$
- $T = \{1, 2\}$

  Conjunto N representa todos os enfermeiros disponíveis para construir a escala, de modo que os que estão de ferias,licença não entram no conjunto. Conjunto M representa os dias do mês que será gerado a escala,(nesse trabalho é contemplado no máximo 31 dias). E as variáveis são construídas a partir de :

- i: representa o funcionário de código i;
- j: representa o dia j do mês;
- t: representa o turno, podendo assumir valores diurno(1) ou noturno(2);

  Os parâmetros de entrada para o modelo são:

- D: representado a quantidade de demanda de funcionários em determinado turno.

$$min \sum_{i \in N} \sum_{j \in M} \sum_{t \in T} X_{ij}^{t} \tag{Eq.2a}$$

$$subject \quad to$$

$$X_{ij}^{t} + X_{i(j+1)}^{t} + X_{i(j+2)}^{t} \leq 1, \quad \forall i \in N, \forall t \in T, \forall j \in M/\{30, 31\} \tag{Eq.2b}$$

$$X_{i(30)}^{1} + X_{i(31)}^{1} \leq 1, \quad \forall i \in N \tag{Eq.2c}$$

$$X_{i(30)}^{2} + X_{i(31)}^{2} \leq 1, \quad \forall i \in N \tag{Eq.2d}$$

$$\sum_{i \in N} X_{ij}^{t} \geq D_{j}^{t}, \quad \forall i \in N, \forall t \in T \tag{Eq.2e}$$

$$X_{ij}^{t} \in \{0, 1\} \quad \forall i \in N, \forall j \in M, \forall t \in T \tag{Eq.2f}$$

- **Garantir que os funcionários trabalhem apenas 12x60:**

  Na restrição Eq.2b garante que irá trabalhar doze horas e descansar 60 horas, ou seja irá trabalhar doze horas, e o restante do dia de folga e depois folgará quarenta e oito horas. Desse modo essa restrição deve ser aplicada o conjunto A e B.

- **Quantidade necessária de funcionários por turno:**

  Na restrição Eq.2e garante que a cada um dos turnos de determinado dia, deve-se trabalhar certa quantidade de enfermeiros (demanda).

- **Valor máximo de cada variável tem que ser um:**

  Na restrição Eq.2f garante que o valor de cada variável seja zero ou um, se for encontrado zero está de folga caso o contrário estará trabalhando. Dessa forma essa restrição garante que o mesmo funcionário não irá trabalhar mais de uma vez no mesmo turno.

4.2.2 Breve análise do modelo

Não existe nenhuma interseção entre os conjuntos, de forma que deixa o problema determinístico. Dessa forma é proposto algumas modificações no modelo original para retirar o determinísticos, criando uma interseção entre os conjuntos. Seguem as seguintes modificações:

- **Possibilidade de folgas durante o mês:**

  As folgas dos funcionários durante o mês podem ocorrer por meio de um banco de horas. O banco de horas trata-se de um sistema de compensação de horas extras mais flexível . Assim, pode acontecer que um funcionário tenha direito a uma ou mais folga(s) durante o mês, dessa forma é necessário representar também essa restrição.

Tal possibilidade deve ser aplicada apenas para os funcionários que possuírem esse recurso:

$$\sum_{j \in M} \sum_{t \in T} X_{is}^{t} \leq Y, \qquad \forall i \in N \tag{Eq.3}$$

Y representa a quantidade de dias que deveria ser trabalhados durante o mês menos a quantidade de folgas.

- **Existência do folguista:**

  O funcionário conhecido como folguista é aquele que pode trabalhar em qualquer turno, mas deve cumprir o horário de descanso. Ele pode ser necessário com demanda de dias e turnos diferentes, ou cumprir alguma folga de um funcionário, para realizar essa modificação foi preciso criar uma regra nova para cada folguista:

$$X_{kj}^{1} + X_{kj}^{2} + X_{k(j+1)}^{1} + X_{k(j+1)}^{2} + X_{k(j+2)}^{1}, \forall k \in K, \forall j \in M/\{30, 31\} \tag{Eq.4}$$

$$X_{kj}^{2} + X_{kj}^{1} + X_{k(j+1)}^{2} + X_{k(j+1)}^{1} + X_{k(j+2)}^{2}, \forall k \in K, \forall j \in M/\{29, 30, 31\} \tag{Eq.5}$$

  O conjunto K representa todos os funcionários que são folguistas.

  Essas regras devem substituir a restrição Eq.2c, e o folguista deve seguir as outras restrições normalmente

No capitulo subsequente será relatado o desenvolvimento do primeiro modelo em *web service* para a utilização em *softwares* já existentes.

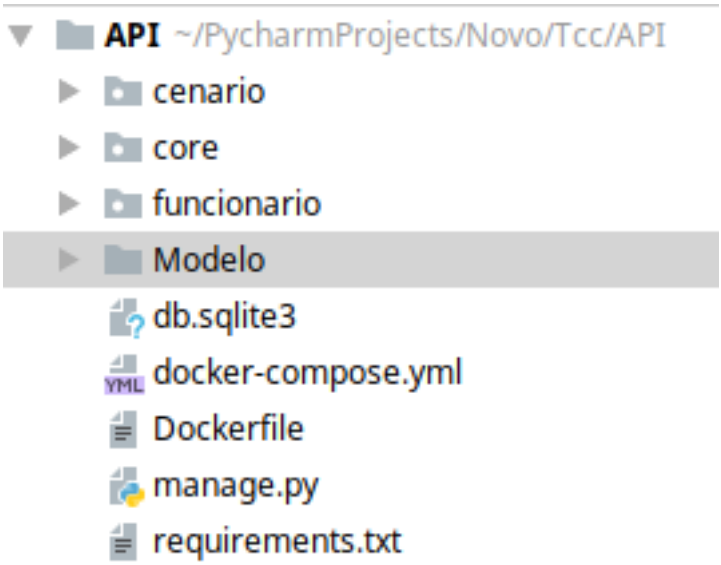
5 Desenvolvimento

Este capítulo tem como objetivo fornecer informações sobre o desenvolvimento do protótipo de *web service* para a geração de escalas usando programação inteira 0-1.

5.1 Estrutura dos arquivos

Nessa seção será apresentado a estrutura dos arquivos do trabalho. Segue a descrição do conteúdo das pastas apresentadas na Figura 7.

Figura 7 – Organização dos arquivos



Fonte: Elaborado pela autora

- cenario — arquivos de configuração do *web service*, que especificam quais campos devem ser enviados na quais na requisição;
- core — arquivos de configuração da API, é onde fica a configuração do *Django Rest Framework*, registrando as rotas existentes no trabalho. Nesse caso existe apenas uma rota /cenario;
- funcionario — arquivos referentes aos parâmetros de cada funcionário: código, preferencia de turno;
- Modelo — arquivos fonte com a implementação do modelo.

Na seção seguinte serão detalhados os arquivos: Dockerfile, requirements.txt e o docker-compose.yml, que são essenciais para a construção do contêiner utilizando o *docker*.

5.2 Estrutura *Docker*

Para criar um contêiner é necessário criar um arquivo imagem *Dockerfile*. No Código Fonte 3 na segunda linha *FROM Python3* informa qual imagem deve-se basear, nesse caso foi utilizado o *python3*. Mais informações/referências de outros contêineres [1].

Na linha quadro tem uma variável de ambiente: *ENV PYTHONUNBUFFERED 1*, que instrui o não armazenamento em *buffer*. A saída será enviada diretamente para o terminal.

Na linha dez será criado o diretório, onde irá ficar o código dentro do contêiner. Na linha dez é copiado o arquivo com os requerimentos para instalar as dependências.

Na próxima linha, dezesseis, o comando *run* executa qualquer comando como se estivesse executando-o em um terminal, nesse caso está sendo chamado o PIP para instalar as dependências do projeto. Por fim é enviado o código local para dentro dentro do contêiner.

Código Fonte 3 *Arquivo Dockerfile*

```
1  # Container base: python 3 Alpine Linux
2  FROM python:3
3
4  ENV PYTHONUNBUFFERED 1
5
6  # Cria diretorio onde vao ficar o codigo fonte
7  RUN mkdir /code
8
9  # Define o diretorio de trabalho
10 WORKDIR /code
11
12 # Copia arquivo requirements.txt para o diretorio code
13 ADD requirements.txt /code/
14
15 # Executa o pip e estalar as dependencias para o projeto
16 RUN pip install -r requirements.txt
17
18 # Copia os arquivos locais para o diretorio code no conteiner
19 ADD . /code/
```

Para gerenciar as dependências do projeto foi utilizado o Código Fonte 4 (requirements.txt). Esse arquivo será utilizado no *Dockerfile* na construção da imagem do

[1] Disponível em <https://hub.docker.com/>Acesso em 06/2019

contêiner As dependências necessárias nesse caso são o *Django*, o *DjangoRest Framework* e a biblioteca para criar a programação inteira 0-1, PuLP.

Código Fonte 4 Arquivo *requirements.txt*

```
1 Django
2 djangorestframework
3 pulp
```

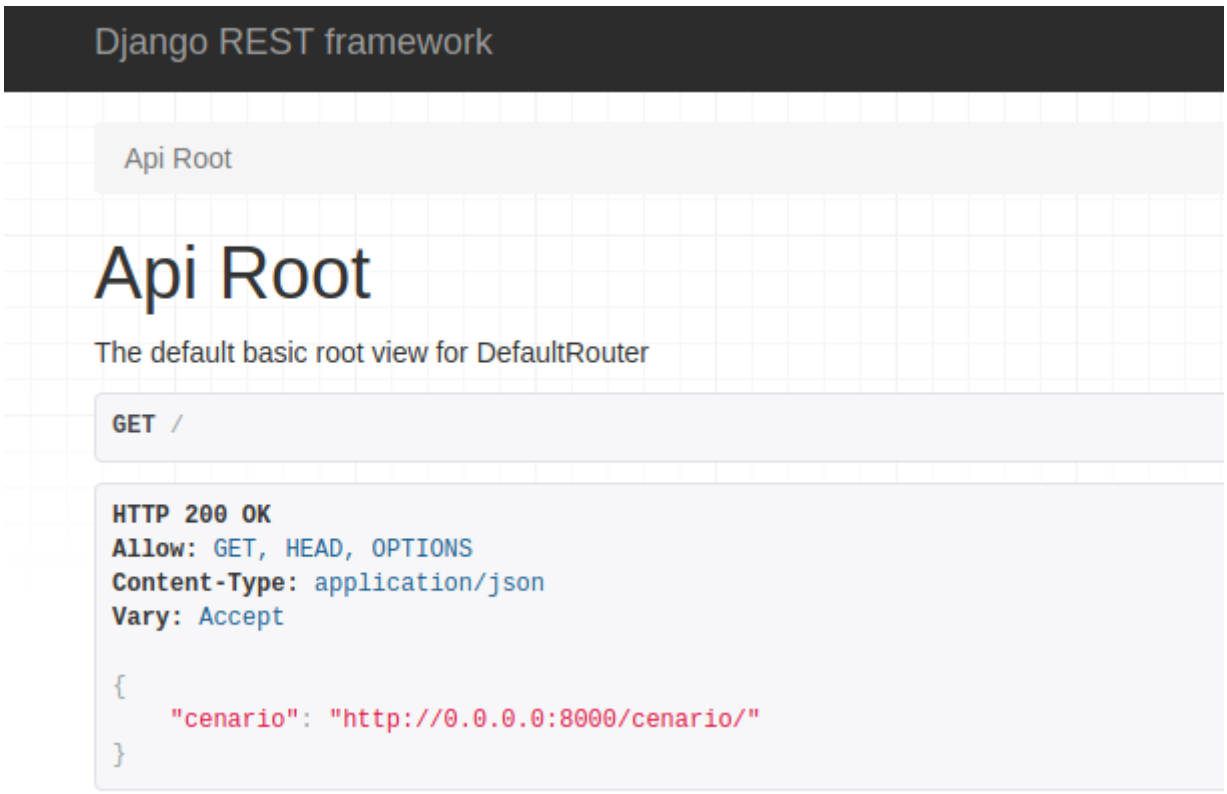
O Código Fonte 5.2 descreve os serviços que vão compor o *stack* do projeto. A primeira linha especifica qual é a versão de sintaxe do *Docker*. Em seguida, define-se um *service* chamado *web*. Na *build* é informado para o *docker* o diretório raiz contendo a imagem. O *command* informa o comando padrão para executar o *Django*.

O campo *volumes* constrói o diretório raiz do projeto no contêiner, ou seja, ele garante que qualquer mudança realizada fora do contêiner será atualizado imediatamente no contêiner. Por fim, o campo *ports* indica a porta para expor para comunicar com o contêiner. Para construir e executar o contêiner, pode-se utilizar o comando: *docker-compose up* e podendo ser acessado pela porta 0.0.0.0:8000 ou *localhost*:8000. Como pode ser visualizado na Figura 8.

Código Fonte 5 Arquivo *r*equirements.txt

```
1  version: '2'
2  services:
3      web:
4          build: .
5          image: django-sandbox
6          command: python manage.py runserver 0.0.0.0:8000
7          volumes:
8              - .:/code
9          ports:
10             - "8000:8000"
```

Figura 8 – Web service Funcionando no *localhost*



Fonte: Elaborado pela autora

Na próxima seção serão tratadas informações adicionais como: o formato da data, e o motivo por não utilizar o banco de dados.

5.3 Informações Adicionais

As datas nesse trabalho serão tratadas como *epoch*, que é definido como o número de segundos decorridos desde o tempo universal coordenado proléptico UTC (Tempo Universal Coordenado) de 1 de janeiro de 1970, sem contar os segundos bissextos, e pode ser calculada até os dias atuais. Deve-se ressaltar que esse ponto no tempo não muda, não importando onde esteja no planeta, isto é muito útil para sistemas de computador, onde o horário é importante (CONVETER, 2018).

Nesse trabalho não foi necessário empregar nenhum banco de dados para salvar as requisições ou os registros. O *web service* implementado não guarda os estados anteriores, pois para seu consumo é preciso que toda requisição envie as informações necessárias para desenvolvimento da escala. Dessa forma o trabalho não irá gastar recursos computacionais

para salvar nenhum registro da requisição, em vez disso será utilizado para retornar o valor da escala.

Nesse modo, esse trabalho pode ser utilizado para qualquer fuso horário, independente do horário do local e não terá a necessidade de salvar nada em banco de dados.

Na próxima seção será relatado, a estrutura para o desenvolvimento do modelo, juntamente com a classe principal e suas funções.

5.4 Estrutura para criar o modelo

Para o desenvolvimento do modelo matemático utilizando programação inteira 0-1 foi necessário construir um arquivo contendo Código Fonte 6 com métodos da classe Modelo.

Código Fonte 6 Classe Modelo

```
1  def __init__(self, conjuntoFuncionarios, conjuntoPrefeD ,
       diaInicio, dias, demandaDiurnas, demandaNoturnas, domingos)
2
3  def modificaConjuntoDias(self,diaInicio,dias)
4
5  def modificaConjuntoDiasDomingo(self,domingos)
6
7  def geraModelo(self)
8
9  def geraListaVar(self)
10
11 def geraListaObj(self)
12
13 def geraDemanda(self,conjunto)
14
15 def trabalharUnicoTurnoDia(self,conjunto)
16
17 def garantirTurnosDiurno(self,conjunto)
18
19 def garantirTurnosNoturno(self,conjunto)
20
21 def umaFolgaDomingoMes(self,conjunto)
```

Na primeira linha tem o protótipo da função construtor que recebe como parâmetros o conjunto de funcionários, dia inicial para a escala, dias demanda do turno diurno e noturno, e os domingos do mês.

Na terceira e quinta linhas tem uma função que adiciona a letra "m"entre o código do funcionário e o dia do mês, de modo a diferenciar os dois parâmetros e retornando a lista criada.

Na linha sete tem a função principal que chama todas as outras para geração da escala, será explicando com mais detalhes.

Na linha onze tem a função responsável por gerar uma lista que será parâmetro gerar a função objetivo do problema de acordo com Eq.1a.

Na linha treze tem a função responsável por gerar uma lista com a demanda necessárias em cada um dos turnos, seguindo a restrição do modelo. Eq.1e.

Na linha quinze tem a função para gerar uma lista com as restrições de trabalhar apenas um único turno por dia, ou seja diurno ou noturno Eq.1g.

Na linha dezessete e dezenove têm as duas próximas funções são criadas listas para retornar as restrições de forma a garantir turnos diurnos e noturnos respeitando as restrições 12x36 Eq.1, Eq.1c.

Na última função são cridas as restrições para que o funcionário tenha pelo menos uma folga acontecendo em um domingo do mês Eq.1f.

O próximo código relata a função principal para construir modelo, dividido em três partes, a primeira parte (7) é responsável por declarar o problema de maximização e gerar a função objetivo com todas as variáveis do problema.

Código Fonte 7 Função para Geração do modelo Parte 1

```
1  def geraModelo(self):
2
3      #declarar o problema
4      prob = LpProblem("Escala", LpMaximize)
5
6      #lista de variaveis do problema
7      lista=self.geraListaVar()
8
9      #declara as variaveis no modelo
10     conjunto = LpVariable.dicts("x", lista, 0, 1, LpInteger)
11
12     #geracao da funcao objetivo.
13     valorobjetivo=self.geraListaObj()
14     prob += lpSum([valorobjetivo[i] * conjunto[i] for i in lista
           ]), "Maximize"
```

Na segunda parte (5.4), nas próximas linhas são chamadas as funções que irão construir as referências das variáveis para as restrições, depois será construída cada restrição dentro do comando *for*.

Código Fonte 8 Função para Geração do modelo Parte 2

```
1       #lista de referencia para demanda Diurna e Noturna
2       listaDemandaDir,listaDemandaNor=self.geraDemanda(conjunto)
3
4       cont=0
5       #adicionar restricao de demanda diurna e noturna
6       for demanda in(listaDemandaDir):
7           prob += lpSum(demanda) >=self.demandaDiurnas, "Demanda"+
                str(cont)
8           cont+=1
9
10      for demanda in(listaDemandaNor):
11          prob += lpSum(demanda) >=self.demandaNoturnas, "Demanda"+
                str(cont)
12          cont+=1
13
14      #lista de referencia para unico turno por dia
15      listaUnicoTurno=self.trabalharUnicoTurnoDia(conjunto)
16
17      cont=0
18      #adicionar a restricao de unico turno no modelo.
19      for unicoTurno in(listaUnicoTurno):
20          prob += lpSum(unicoTurno) <=1, "UnicoTurno"+str(cont)
21          cont+=1
22
23     #lista de referencia para turnos diurno
24      garantirEscala=self.garantirTurnosDiurno(conjunto)
25
26      cont=0
27      #adicionar a restricao de turnos diurnos
28      for turno in(garantirEscala):
29          prob += lpSum(turno) <=1, "garantirEscalaD"+str(cont)
30          cont+=1
31
32      #lista de referencia para turnos noturno
33      garantirEscala = self.garantirTurnosNoturno(conjunto)
34
35      cont = 0
36      #adicionar a restricao de turnos diurnos
37          for turno in (garantirEscala):
38              prob += lpSum(turno) <= 1, "garantirEscalaN" + str(
                    cont)
39              cont += 1
40
41      #lista de referencia para uma folga no domingo por mes
42      garantirEscala = self.umaFolgaDomingoMes(conjunto)
43
44      cont = 0
45      #adicionar a restricao de turnos diurnos
46      for turno in (garantirEscala):
47          prob += lpSum(turno) <= len(self.domingos)-1, "
                folgaDomingo" + str(cont)
48          cont += 1
```

Na segunda parte (5.4), próximas linhas é resolvido o modelo criado, e retornado para o soliciatante o resultado.

Código Fonte 9 Função para Geração do modelo Parte 3

```
1          #resolver o modelo
2          prob.solve()
3
4          Dictresultado={}
5
6          #colocando os valores das variaveis em um dicionario.
7          for v in prob.variables():
8              Dictresultado[v.name]=v.varValue
9
10         #adicionar no dicionarios os valores.
11         Dictresultado["resultado"]=value(prob.objective)
12         #adicionar o status.
13         Dictresultado["status"]=LpStatus[prob.status]
14
15         return Dictresultado
```

Em seguida é informado a função *create* do *Django Rest Framework* que utiliza o método POST, que irá receber as requisições do *web service* os parâmetros necessário e depois criará um objeto Modelo passando os mesmo e retornando a resposta para o solicitante.

Código Fonte 10 Função create presente no *Django REST framework*

```
1      #POST
2      def create(self, request,*args,**kwargs):
3
4          #adquiri os valores da requisicao
5
6          #converter datas em numeros de 1 ao 31
7
8          #passar Parametro para a classe modelo
9          modelo = Modelo1(conjuntoCodigo, conjuntoPrefeD,
               diaInicio, diasEscala, posicioesDiurnas,
               posicoesNoturnas,
10                                listaDomingo)
11             respostaModelo = modelo.geraModelo()
12
13             #chamar funcao para desenvolver o modelo
14             resposta=self.instanceResult(respostaModelo,
                   conjuntoCodigo, diaInicio,diasEscala,mes,ano)
15             resposta["status"]=respostaModelo.get("status")
16
17             #retornar o resultado da geracao de escala
18             return Response(resposta)
```

Na próxima seção serão explicados os detalhes do funcionamento do *web service* e seus componentes, juntadamente com a estrutura necessária para a sua utilização.

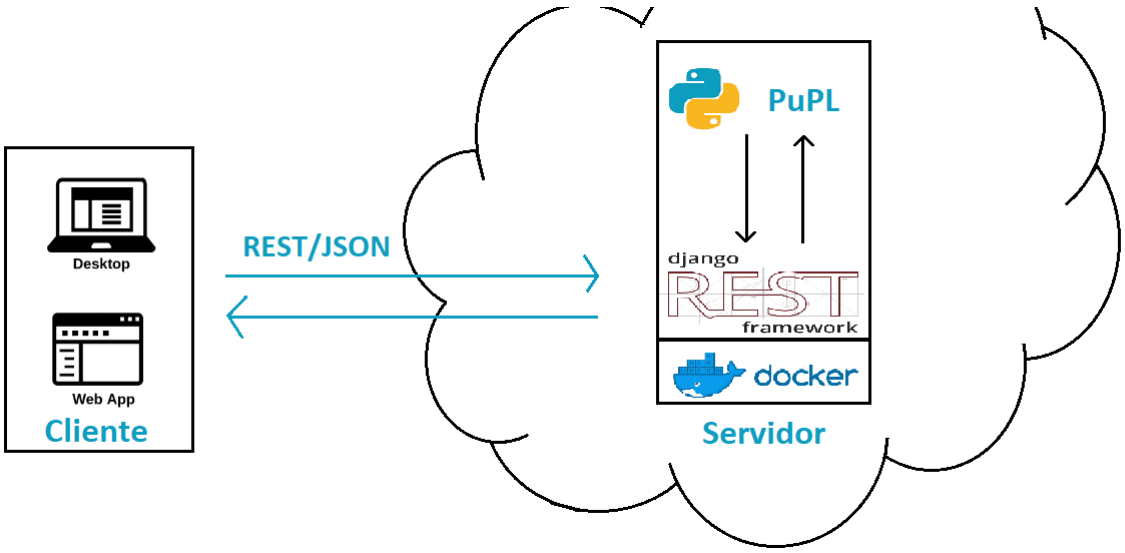
5.5 Funcionamento do *web service*

Um solicitante do serviço é um programa que faz requisição a um serviço específico e este pode ser comparável ao cliente dentro de um modelo cliente-servidor padrão. O servidor tem recursos, permitindo que muitos computadores possam ter acessos a essas informações. Geralmente o servidor é um computador com capacidade mais potente do que as máquinas que rodam as atividades, pois estas não precisam ter uma configuração elaborada para trocar informações, apenas fazem acesso de conexão de rede. Nesse trabalho o *web service* será responsável pela criação da escala trabalhista 12x36.

Para o funcionamento do *web service* precisa de dois componentes, mostrado na Figura 9.

- *Cliente* : Pode ser um sistema de qualquer tipo, por exemplo *web* ou *desktop*, de modo que essa parte será responsável por organizar as informações em formato JSON e enviar para o servidor que contém o *web sevice* e este irá retornar o resultado para o solicitante.
- *Servidor* : Receberá uma requisição via POST e realizará a construção da escala e depois retornará um JSON com o resultado para o solicitante.

Figura 9 – Arquitetura do trabalho



Fonte: Elaborada pela autora

Para a criação da escala é necessário fornecer as seguintes informações, via POST para o *web service*:

- **funcionarios:** Representa um conjunto de colaboradores que possui dois campos: o código para ser diferenciado entre os demais e a prefêrencia de turno podendo receber os valores de D para diurno e N para noturno.
- **dataInicio:** Representa a data para inicializar a geração de escala, devendo ser em *epoch*, para converter pode ser utilizado o *site*[2].
- **dataTermino:** Representa o último dia para a realização da escala, considerando que este trabalho realiza apenas a escala de um mês.
- **posicoesDiurna:** Representa a quantidade necessária de funcionários no turno diurno.
- **posicoesNoturnas:** Representa a quantidade de funcionários necessário para o turno noturno.

Para o entendimento dos parâmetros explicados acima, , tem o exemplo de Código Fonte 11.

[2] Disponível em <https://www.epochconverter.com/> Acesso em 06/2019.

Código Fonte 11 Exemplo de arquivo de entrada para o *web service*

```
{         "funcionarios": [
            {   "codigo": "1",
                "preferencia": "D"
            },
            {  "codigo": "2",
                "preferencia": "D"
            },
            {   "codigo": "3",
                "preferencia": "N"
            },
            {   "codigo": "4",
                "preferencia": "N"
            },
            {   "codigo": "5",
                "preferencia": "N"
            },
            {   "codigo": "6",
                "preferencia": "D"
            },
            {   "codigo": "7",
                "preferencia": "D"  } ],
        "dataInicio": "1543629600",
        "dataTermino": "1546221600",
        "posicioesDiurnas": 2,
        "posicoesNoturnas": 1
}
```

A resposta do *web service* será representado por várias datas *epoch*, onde dentro dela estará contido o dia, mês e seu turno, em seguida será informado os códigos dos funcionários que realizaram aquele turno. Por fim será mostrado um *status* sobre o resultado: *Optimal* conseguiu gerar a escala com os parâmetros passados, *Not Solved* não foi possivel resolver, *Infeasible* é inviável, *Unbounded* sem limite e *Undefined* indefinido. Exemplo do retorno da requisição do Código Fonte 11, no Código Fonte 12.

Código Fonte 12 Exemplo de retorno da requisição

```
{"1543647600.0":["2","7"],
"1543690800.0":["4","5"],
"1543734000.0":["1","6"],
"1543777200.0":["3"],
"1543820400.0":["2","7"],
"1543863600.0":["4","5"],
"1543906800.0":["1","6"],
"1543950000.0":["3"],
"1543993200.0":["2","7"],
"1544036400.0":["4","5"],
"1544079600.0":["1","6"],
...
"1546153200.0":["1","6"],
"1546196400.0":["3","5"],
"status":"Optimal"}
```

Em seguida serão relatados os testes que foram necessário para a validação do *web service*, construindo no primeiro modelo 4.1.1 presente no capitulo anterior.

6 Avaliações e Testes

Este capítulo tem como objetivo fornecer informações sobre avaliações e testes realizados nesse trabalho. Para auxiliar nos testes desse trabalho foi utilizado o conhecimento de um especialista na área (um dos sócios da empresa *DoctorID*), que comercializa soluções para a geração de escalas na área da saúde, para a fase de validação do modelo.

6.1 Avaliações

No processo de avaliação da escala foi criado um documento relatando as principais restrições da escala trabalhista (12x36), de forma a obedecer as leis trabalhistas e adicionar a possibilidade de atender assim que possível a preferência dos funcionários. Na própria cidade da autora, foram pesquisadas quais escalas são utilizadas nos dois hospitais da cidade: a Casa de Saúde Santa Marta de Formiga/MG e a Santa Casa de Caridade. Ambos organizaram suas escalas trabalhistas no método de 12x36.

Além de modelar o problema de acordo com o Diagrama 5, foi pensando a melhor forma para desenvolvimento do trabalho, buscando reusabilidade para que não seja necessário construir ferramentas novas quando existem bibliotecas ou *frameworks* que atendem as necessidades.

Para que a aplicação não sofra com dependências de bibliotecas e do próprio sistema operacional foi construindo um contêiner com *docker* para armazenar o serviço de forma isolada do sistema operacional. Essa estratégia permite a transferência facilitada da aplicação para outros sistemas operacionais, junto com as bibliotecas necessárias para o trabalho funcionar, de modo a cumprir com o requisito das dependências.

No elaboração do núcleo foi utilizado programação inteira 0-1, que explora combinação dos possíveis resultados que no pior caso, pode demorar horas/dias/meses para executar de acordo com a quantidade de entrada. Para tentar amenizar o problema foi idealizada uma solução como *web service*, disponibilizada em um servidor mais robusto onde é possível investir financeiramente em processamento e memoria RAM para disponibilizar a geração de escala para outros *softwares* que necessitam da nova funcionalidade.

Para não consumir muita banda na comunicação, foi estabelecida a implementação do método REST – apenas com a requisição via *POST* – com o padrão de troca de mensagens JSON para criação da escala. Não será necessário em nenhum momento salvar registros no banco de dados, pois a requisição carrega os dados necessários para fazer o cálculo, e o serviço pode retornar prontamente para o solicitante o resultado da escala.

Desse modo o projeto cumpre dois objetivos: (1) aplicação independente de biblio-

tecas e sistema operacional; (2) processamento remoto para diminuir o gargalo de uso da CPU local, pois o uso de programação inteira 0-1 pode demorar no piores casos.

6.2 Testes

Para facilitar a análise dos resultados foram implementadas funções para coletar as respostas do serviço e apresentá-las em formato mais amigável como uma tabela em HTML. A Figura 10 ilustra o formato da saída dos testes, onde na tabela são mostrados os intervalos de tempo (data e turno) no cabeçalho e para cada funcionário uma linha indicando se ele estará ou não de plantão.

Figura 10 – Exemplo para o refinamento dos modelo 1

	01/12 05:00	01/12 17:00	02/12 05:00	02/12 17:00	03/12 05:00	03/12 17:00	04/12 05:00	04/12 17:00	05/12 05:00	05/12 17:00	06/12 05:00	06/12 17:00	07/12 05:00	07/12 17:00	08/12 05:00	08/12 17:00	09/12 05:00	09/12 17:00	10/12 05:00	10/12 17:00	11/12 05:00	11/12 17:00	12/12 05:00	12/12 17:00	13/12 05:00	13/12 17:00	14/12 05:00	14/12 17:00
4	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
7	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
6	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
5	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
2	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Fonte: Elaborado pela autora

Durante a fase de testes foram construídos alguns modelos, entre eles estão de vinte e cinco funcionários, com demanda de cinco em todos os turnos, cinquenta funcionários demanda de dez, e cem funcionários com demanda de vinte, mas todos foram aplicados em apenas um mê. Por fim os testes foram mostrados para os sócios do *DoctorID*, para avaliar a qualidade dos resultados e auxiliar nos refinamentos do modelos. Depois de refinar os dois modelos, foi fornecido mais atenção ao *web service* que foi implementado apenas o primeiro modelo, otimizando os parâmetros de entrada, conforme consultado o especialista na área.

No próximo capitulo será sobre considerações finais e as referencias bibliográficas do trabalho.

7 Considerações Finais

Inicialmente o trabalho foi desenvolvido criando um modelo matemático que representa o mais próximo possível a realidade, considerando a criação de escalas trabalhistas, leis trabalhistas e preferências do trabalho.

Uma solução foi implementada com base no modelo criado e disponibilizada como *web service*. Objetivando a portabilidade, no desenvolvimento da solução foi utilizado o *docker*, que facilita possíveis mudanças de sistema operacional e atendimento dos requisitos de dependências do projeto. Além disso, devido ao fato da solução adotada nesse trabalho utilizar programação inteira 0-1, que pode recair em busca combinatória no pior caso, fez-se a opção por hospedar o serviço em um servidor mais robusto que disponível para outras aplicações.

No desenvolvimento desse trabalho foram utilizadas as seguintes ferramentas: *Python*, por ser uma linguagem gratuita de código aberto, juntamente com o *Django Rest Framwork*, que utiliza o padrão REST e permite o uso do formato JSON para a troca de mensagens. O desenvolvimento de um cliente para consumir o serviço não era um foco do trabalho, entretanto, para realização dos teste foi utilizado um cliente de requisição chamado *PostMan*.

Este trabalho poderá contribuir com a comunidade deixando o modelo matemático disponível para ser utilizado como inspiração para o desenvolvimento de novos trabalhos. É um incentivo para que essa área seja mais explorada, pois o setor trabalhista pode ser melhorado no seu planejamento, por exemplo, com a automatização da geração das escalas. Atualmente, muitos gestores ainda demoram várias horas para gerar uma escala, e com a automatização eles poderiam utilizar melhor esse tempo para fazer algo de maior prioridade.

Mesmo um trabalho decorrente poderá utilizar programação inteira para resolver problemas combinatoriais, pois como em um hospital os funcionários são divididos em setores (números pequenos de colaboradores), é possível dividir o problema e construir uma escala para cada setor por vez, obtendo depois uma solução para um hospital inteiro através da união das soluções parciais, utilizando a ideia de divisão e conquista.

7.1 Trabalhos Futuros

Por ser um tema que não foi abordado em várias pesquisas, e as ferramentas de trabalho atuais ainda não oferecem muitos recursos de automatização. Muito ainda pode ser feito nessa área, dessa forma seguem algumas sugestões para trabalhos futuros:

- Implementação do segundo cenário do *web service* Eq.2a e construir as restrições da análise do modelo, citadas no capitulo 4;
- Criação de novos cenários que não foram abrangido neste trabalho, por exemplo 12x48, 5x1, 5x2 dentro outras.
- Melhorar o primeiro cenário, passando a requisição para GET e considerando o risco de manter a conexão aberta em todo o tempo, podendo ser tratado na hora que tiver resultado, por exemplo enviar por *e-mail*.
- Adicionar restrições que considere a semana anterior do mês, para fazer a escala a partir dela.
- Construção dos modelos em outra biblioteca ou outras linguagens.
- Utilização de heurísticas para atacar o problema, seguido de uma comparação da qualidade dos resultados.
- Desenvolvimento de um sistema completo para atender os gestores, um que permita automatizar todo o controle, por exemplo: gerenciamento de funcionários, folhas de pagamento, *check in* e *check out*, etc. Esse trabalho futuro é muito maior e mais ambicioso, e poderia consumir o *web service* desenvolvido no presente trabalho para fazer geração de plantões na escala trabalhista.

Referências

ANDRADE, E. L. *Introdução à pesquisa operacional métodos e modelos para análise de decisões*. 4a edição. ed. [S.l.]: LTC, 2009. Citado 4 vezes nas páginas 15, 18, 20 e 35.

BELFIORE, P.; FAVERO, L. P. *Pesquisa Operacional Para cursos de Engenharia*. 1a edição. ed. [S.l.]: Elsevier, 2013. Citado 3 vezes nas páginas 18, 20 e 21.

CONVETER, E. *Epoch Unix Timestamp Conversion Tools*. 2018. Disponível em <https://www.epochconverter.com/>. Acesso em 19 fev. 2019. Citado na página 44.

CRUZ, F. *Python Escreva seus primeiros programas*. 2a edição. ed. [S.l.]: Casa do Codigo, 2015. Citado na página 31.

DOCKER. *What is a Container?* 2013. Disponível em <https://www.docker.com/resources/what-container>. Acesso em 19 fev. 2019. Citado 2 vezes nas páginas 30 e 31.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architecture*. Tese (Doutorado) — University of California, Irvine, 2000. Citado na página 23.

FILHO, F. S. d. F. *UMA ABORDAGEM PARA O PROBLEMA DE ALOCAÇÃO DE PROFESSORES EM DISCIPLINAS UTILIZANDO PROGRAMAÇÃO LINEAR INTEIRA*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DO CEARÁ, 2016. Citado 2 vezes nas páginas 15 e 18.

GOLDBARG, M. C. *Otimização combinatória e programação linear: modelos e algoritmos*. 2a edição. ed. [S.l.]: Elsevier, 2005. Citado na página 33.

GURU99. *SOAP Vs. REST: Difference between Web API Services*. 2018. Disponível em <https://www.guru99.com/comparison-between-web-services.html>. Acesso em 19 fev. 2019. Citado na página 24.

IMASTER. *Flask x Django: como escolher o framework correto para seu aplicativo web*. 2018. Disponível em <https://imasters.com.br/back-end/flask-x-django-como-escolher-o-framework-correto-para-seu-aplicativo-web>. Acesso em 19 fev. 2019. Citado na página 32.

JSON. *Introdução ao JSON*. 2019. Disponível em <https://www.json.org/json-pt.html>. Acesso em 26 mai. 2019. Citado na página 25.

MENENDEZ, A. I. M. *Uma ferramenta de apoio ao desenvolvimento de Web Services*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DE CAMPINA GRANDE, 2002. Citado na página 25.

OPENSOFT. *Web service: o que é, como funciona, para que serve?* 2016. Disponível em <https://www.opensoft.pt/web-service/>. Acesso em 19 fev. 2019. Citado na página 22.

PINHEIRO, F. *Gerenciando Pacotes Em Projetos Python Com O PIP*. 2018. Disponível em <https://www.epochconverter.com/>. Acesso em 19 fev. 2019. Citado na página 31.

PLANALTO. *DECRETO-LEI N.º 5.452, DE 1º DE MAIO DE 1943*. 1943. Disponível em <http://www.planalto.gov.br/ccivil_03/decreto-lei/Del5452.htm>. Acesso em 19 fev. 2019. Citado na página 19.

RANGER, A. L. *Desenvolvimento de um sistema de apoio à decisão para a elaboração da escala periódica de pessoal de Enfermagem*. Tese (Doutorado) — Universidade de São Paulo, 2006. Citado 5 vezes nas páginas 11, 15, 16, 26 e 27.

ROZLOG, M. *REST e SOAP: Usar um dos dois ou ambos?* 2013. Disponível em <https://www.infoq.com/br/articles/rest-soap-when-to-use-each>. Acesso em 19 fev. 2019. Citado na página 23.

SAUDATE, A. *REST construa API'S inteligentes de maneira simples*. 2a edição. ed. [S.l.]: Casa do Codigo, 2015. Citado 2 vezes nas páginas 23 e 24.

SILVA, L. G. *Modelagem para resolução do problema de designação de equipes para manutenção da iluminação pública no médio Piracicaba*. Tese (Doutorado) — Universidade Federal de Ouro Preto, 2017. Citado 2 vezes nas páginas 15 e 18.

SINDFIBERJ. *CONVENÇÃO COLETIVA DE TRABALHO*. 2016. Disponível em <http://sindfiberj.org.br/convencoes/Convencao_Enfermeiros_2016.pdf>. Acesso em 19 fev. 2019. Citado 2 vezes nas páginas 15 e 19.

TST. *Súmulas do Tribunal Superior do Trabalho*. 2012. Disponível em <http://www3.tst.jus.br/jurisprudencia/Sumulas_com_indice/Sumulas_Ind_401_450.html#SUM-444>. Acesso em 19 fev. 2019. Citado na página 19.

ÁVILA, R. *As principais escalas de trabalho e como escolher qual utilizar*. 2015. Disponível em <https://blog.luz.vc/o-que-e/principais-escalas-de-trabalho-e-como-escolher-qual-utilizar/>. Acesso em 19 fev. 2019. Citado 2 vezes nas páginas 18 e 19.