

Projeto de distribuição da vacina COVID 19

Lucas Nascimento¹, Matheus Rocha¹, Sillas Reis¹

¹Departamento de Ciência da computação – Instituto de Educação Superior de Brasília (IESB)
Brasília – DF – Brasil

Abstract. *Covid-19 due to its particularities resulted in a scenario where there is a need for organization to distribute vaccines to people who are at a higher risk of complications. Using a priority queue and a sorting algorithm an already sorted list will be made available with the names according to each region.*

Keywords—*Priority queue, Covid-19, Sorting Algorithm*

Resumo. *A Covid-19 devido a suas particularidades acarretou em um cenário onde existe uma necessidade de organização para distribuição de vacinas a pessoas que possuem um risco maior de complicações. A partir de uma fila de prioridade e utilizando um algoritmo de ordenação será disponibilizado uma lista já ordenada com os nomes de acordo com cada região.*

Palavras-Chave—*Fila de prioridade, Covid-19, Algoritmo de Ordenação*

1. Introdução

No final do ano de 2019 o planeta Terra se deparou com um dos maiores desafios na área da saúde, esse tal que causou uma drástica mudança em várias esferas da sociedade além da saúde como surgimento de novas soluções tecnológicas e novas maneiras de relacionamento interpessoal tanto no âmbito profissional quanto social por exemplo. Tendo em vista este cenário que afeta a todos sem diferenciação de classe social, cor ou nacionalidade é notório o aguardo de todos pela vacina, e visando o cenário onde uma vez que exista a disponibilidade da distribuição de tal é necessário o estudo, a modelagem e o mapeamento de como será feita uma vez que existem pessoas que possuem prioridade devido a comorbidades pré-existentes.

A OMS (Organização Mundial de Saúde) declarou que o COVID-19 oferece mais risco a determinadas pessoas com base em uma série de aspectos fisiológicos, dentre eles: Doenças crônicas como diabetes e hipertensão, Asma e Idade avançada (maior que 60 anos)

Diante do exposto os alunos pensaram na criação de uma aplicação que tem como objetivo automatizar e organizar a distribuição das vacinas para Covid-19 baseando-se em estudos que explicitam a maior fragilidade de uma parcela da população, isto é, em caso de contágio tal grupo de risco tem uma maior probabilidade de complicações que podem levar a óbito.

2. Objetivos gerais

Disponibilização de um sistema capaz de receber dados relativos a pessoas de uma região e conseguir processa-los de maneira a criar uma fila ordenada de acordo com prioridades pré-definidas baseadas em estudos médicos que relativizam a probabilidade de agravamento de casos de acordo com cada comorbidade existente.

3. Objetivos específicos

- Desenvolver banco de dados MYSQL que agrupe informações relevantes sobre o estado de saúde das pessoas;
- Desenvolver algoritmo para selecionar pessoas com maior risco;
- Desenvolver aplicação utilizando a linguagem Python para cadastrar pessoas e doenças, agendar vacinação das pessoas selecionadas e disponibilizar datas de vacinação e seus respectivos pacientes;
- Testar eficiência da aplicação.

4. Referencial teórico

4.1. Covid-19

Em 31 de dezembro de 2019, a Organização Mundial da Saúde (OMS) foi alertada sobre vários casos de pneumonia na cidade de Wuhan, província de Hubei, na República Popular da China. Tratava-se de uma nova cepa (tipo) de coronavírus que não havia sido identificada antes em seres humanos.[Gomes 2020]

Uma semana depois, em 7 de janeiro de 2020, as autoridades chinesas confirmaram que haviam identificado um novo tipo de coronavírus. Os coronavírus estão por toda parte. Eles são a segunda principal causa de resfriado comum (após rinovírus) e, até as últimas décadas, raramente causavam doenças mais graves em humanos do que o resfriado comum. [Gomes 2020]

Em 30 de janeiro de 2020, a OMS declarou que o surto do novo coronavírus constitui uma Emergência de Saúde Pública de Importância Internacional (ESPII) – o mais alto nível de alerta da Organização, conforme previsto no Regulamento Sanitário Internacional. Essa decisão buscou aprimorar a coordenação, a cooperação e a solidariedade global para interromper a propagação do vírus. Essa decisão aprimora a coordenação, a cooperação e a solidariedade global para interromper a propagação do vírus.[Gomes 2020]

A ESPII é considerada, nos termos do Regulamento Sanitário Internacional (RSI), “um evento extraordinário que pode constituir um risco de saúde pública para outros países devido a disseminação internacional de doenças; e potencialmente requer uma resposta internacional coordenada e imediata”. [Gomes 2020]

A responsabilidade de se determinar se um evento constitui uma Emergência de Saúde Pública de Importância Internacional cabe ao diretor-geral da OMS e requer a convocação de um comitê de especialistas – chamado de Comitê de Emergências do RSI.[Gomes 2020]

Esse comitê dá um parecer ao diretor-geral sobre as medidas recomendadas a serem promulgadas em caráter emergencial. Essas Recomendações Temporárias incluem medidas de saúde a serem implementadas pelo Estado Parte onde ocorre a ESPII – ou por outros Estados Partes conforme a situação – para prevenir ou reduzir a propagação mundial de doenças e evitar interferências desnecessárias no comércio e tráfego internacional. [Gomes 2020]

Em 11 de março de 2020, a COVID-19 foi caracterizada pela OMS como uma pandemia. O termo “pandemia” se refere à distribuição geográfica de uma doença e não à sua gravidade. A designação reconhece que, no momento, existem surtos de COVID-19 em vários países e regiões do mundo.[Gomes 2020]

Autoridades de todo o mundo decretaram estado de calamidade pública frente a doença que vinha se alastrando. No Brasil, esse feito foi realizado no dia 20 de março, quando o comércio, escolas, parques e vários outros segmentos da sociedade se viram com a necessidade de paralisar suas atividades rotineiras em prol da saúde pública. Na maioria do mundo, foi adotado o sistema de quarentena, em que as pessoas evitam ao máximo sair de casa para evitar possíveis aglomerações e por sua vez diminuir o contágio.[Gomes 2020]

Toda a situação detalhada anteriormente trouxe uma nova realidade a qual maioria da sociedade não estava preparada. No período de quarentena, setores como a economia e a educação foram amplamente afetados e isso impacta diretamente na vida de todo cidadão.[Gomes 2020]

4.2. Python

A tecnologia da informação, vinculada com a própria informação, tornaram-se requisito para grande parte das soluções que almejam atingir públicos de grande quantidade e variedade. Com tão grande evolução, diversas linguagens de programação surgiram, para que o hardware, cada vez mais poderoso, pudesse ser controlado.

Uma das linguagens mais populares atualmente é Python. De acordo com o Stack OverFlow Trends (Figura 1), em 2020 Python passou a marca de 14 por cento das perguntas do site por mês em todos os meses do ano, marca que só foi alcançada anteriormente pela linguagem C em alguns meses de 2009. Outra pesquisa do Stack OverFlow, a 2020 Developer Survey aponta Python como a 4ª tecnologia mais popular entre os desenvolvedores, perdendo nos votos apenas para JavaScript, HTML, CSS e SQL.

Python foi criada na década de 90 por Guido van Hossum. Seu objetivo era criar uma tecnologia fácil e intuitiva que mesmo programadores não tão experientes fossem capazes de compreender, o que não era o caso em linguagens como C. No decorrer dos anos, Python recebeu diversas atualizações que acrescentaram cada vez mais componentes em sua estrutura. Hoje, Python é mantido pela empresa PSF (Python Software Foundation), empresa criada em 2001 como uma organização sem fins lucrativos.[Silva 2019]

Python é uma linguagem de uso geral e que trabalha de forma isolada, contudo pode ser integrada com outras linguagens e tecnologias. Seus usos comerciais são os mais variados, e alguns exemplos são: buscadores do Google processando pesquisas, transmissões de vídeo do YouTube, e também nos algoritmos bem elaborados da Netflix.[Rocha 2010]

Outras características do Python são: linguagem de alto nível, interpretada, linguagem de script, tipagem dinâmica, boa legibilidade e obrigatoriedade na indentação. Um exemplo de código em Python pode ser observado na Figura 2 e gera o seguinte resultado: Hello World! Hello World! Hello World! Hello World! Hello World!

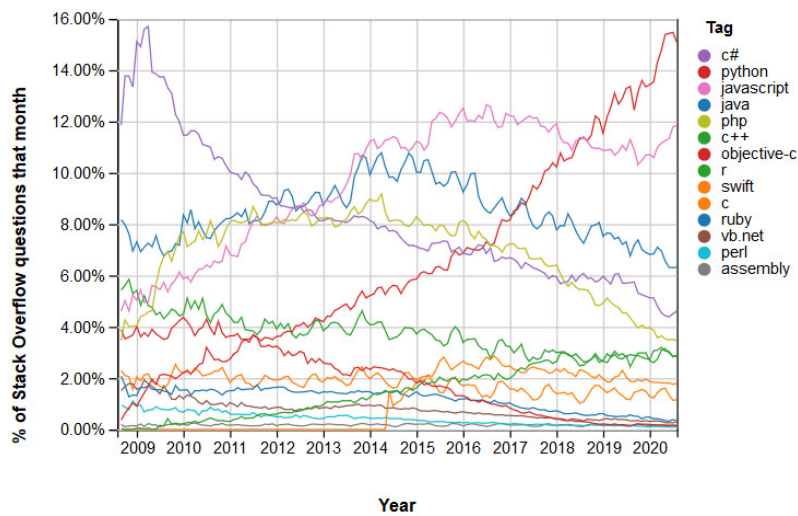


Figura 1. Gráfico popularidade no stack overflow

```

1  x = 'Hello'
2  y = 'World!'
3
4  for i in range(5):
5      print(x, y)
6

```

Figura 2. Exemplo Código

4.3. Fila de Prioridade

A fila de prioridade nada mais é que uma fila comum que permite que elementos sejam adicionados associados com uma prioridade. Cada elemento na fila deve possuir um dado adicional que representa sua prioridade de atendimento.[Abdala 2017]

Imagine um conjunto S de números. Os elementos de S são às vezes chamados chaves ou prioridades (especialmente se há outros dados associados a cada elemento de S). Uma fila-com-prioridades (ou fila priorizada, ou priority queue) é um tipo abstrato de dados que permite executar as seguintes operações sobre S : [Feofiloff 2015]

- encontrar um elemento máximo de S ,
- extrair um elemento máximo de S ,
- inserir um novo número em S ,
- aumentar o valor de um elemento de S ,
- diminuir o valor de um elemento de S .

Tanto a pilha como a fila são estruturas de dados cujos elementos estão ordenados com base na sequência na qual foram inseridos. A operação `pop` recupera o último elemento inserido, e a operação `remove` recupera o primeiro elemento inserido. A fila de prioridade é uma estrutura de dados na qual a classificação intrínseca dos elementos determina os

resultados de suas operações básicas. Existem dois tipos de filas de prioridade: Fila de prioridade ascendente e a fila de prioridade descendente.[Farias 2009]

Uma fila de prioridade ascendente é um conjunto de itens no qual podem ser inseridos itens arbitrariamente e a partir do qual apenas o menor item pode ser removido. Assim que a função designada para retirar o menor elemento for aplicada a fila de prioridade ascendente, ela poderá ser aplicada novamente para recuperar o menor elemento seguinte, e assim por diante. Dessa forma, a função recuperará sucessivamente elementos de uma fila de prioridade em ordem ascendente.[Tanenbaum 1995]

Uma fila de prioridade descendente é semelhante, mas só permite a eliminação do maior item. De maneira análoga a fila descrita anteriormente, é utilizado uma função na qual irá retornar ao maior elemento e realizando a retirada de tal, de forma a caracterizar a retirada em ordem descendente.[Tanenbaum 1995]

Os elementos de uma fila de prioridade não precisam ser números ou caracteres que possam ser comparados diretamente. Eles podem ser estruturas complexas, classificadas por um ou vários campos.[Tanenbaum 1995]

4.4. Heap

Um heap é uma estrutura de dados usada para representar uma coleção de elementos organizados de uma maneira específica. Heaps são árvores binárias, porém é importante deixar claro que não são árvores binárias de pesquisa. Mais especificamente, duas propriedades definem o Heap. A primeira propriedade difere um Heap de uma árvore binária de pesquisa (BST). Na BST, os valores à esquerda de um nó são menores do que ele e os valores à direita são maiores. No Heap, ambos são menores ou iguais. Heaps que seguem essa propriedade são Heaps Máximos porque o maior valor sempre está na raiz. Há também Heaps Mínimos, onde o nó tem valor sempre menor ou igual ao seus filhos. Neste caso, o menor valor sempre está na raiz. Neste material, vamos utilizar o termo Heap como sinônimo de Heap Máximo. A segunda propriedade garante que o Heap é uma árvore binária completa ou quase-completa da esquerda para a direita, quer dizer que, se ela não for completa, todos os níveis estão preenchidos, exceto o último, que deve estar preenchido da esquerda para a direita até um certo ponto.[Brunet 2019]

Nem toda fila segue a política de acesso First In First Out (FIFO). Na verdade, em vários cenários do dia a dia, as filas que entramos possuem uma política diferente: são filas de prioridade. No contexto de estrutura de dados, precisamos pensar em como manter a estrutura ordenada tendo como critério essa prioridade. Então, vamos primeiro analisar alternativas para implementar filas de prioridade usando estruturas de dados lineares, como LinkedList ou ArrayList. Em primeiro lugar, os objetos passam a ter uma prioridade, que é representada por um atributo inteiro. No nosso exemplo, quanto maior esse número, maior a prioridade. Nesse caso, para implementarmos uma fila de prioridade, temos que tomar uma decisão: manter a fila ordenada ou não?[Brunet 2019]

Se decidirmos manter a fila sempre ordenada tendo como critério a prioridade, precisamos utilizar o algoritmo de inserção ordenada, cujo custo é $O(n)$. Contudo, a extração do maior elemento é $O(1)$, pois ele sempre está no início da fila.[Brunet 2019]

Se optarmos por não manter a fila ordenada por prioridade, temos o cenário oposto. A adição passa a ser $O(1)$, mas a remoção do maior passa a ser $O(n)$, pois te-

remos que pesquisar em toda a fila a maior prioridade.[Brunet 2019]

Em resumo, temos de um lado adição $O(n)$ e remoção $O(1)$ e do outro lado temos adição $O(1)$ e remoção $O(n)$. [Brunet 2019]

A estrutura que veremos neste material, Heap, resolve essa questão permitindo que a adição e extração do máximo sejam ambas realizadas em $O(\log n)$, o que é muito desejável do ponto de vista de eficiência. Além disso, o máximo fica sempre na raiz dessa estrutura, o que permite sua inspeção em $O(1)$. [Brunet 2019]

4.4.1. Max-Heap

Pode-se definir um heap descendente (conhecido também como max heap ou árvore descendente parcialmente ordenada) de tamanho n como uma árvore binária quase completa de n nós tal que o conteúdo de cada nó seja menor ou igual ao conteúdo de seu pai. Se a representação sequencial de uma árvore binária quase completa for usada, essa condição se reduzirá à inequação: [Tanenbaum 1995]

$$\text{info}[j] \geq \text{info}[(j-1)/2] \text{ para } 0 < (j-1)/2 < j \leq n-1$$

Por esta definição de heap descendente, é evidente que a raiz da árvore (ou o primeiro elemento do vetor) contém o maior elemento do heap. Observe também que qualquer percurso da raiz até uma folha (ou, na realidade, qualquer percurso na árvore que inclua não mais do que um nó em qualquer nível) é uma lista ordenada em ordem descendente. Resumindo a organização dos elementos em um heap, temos: [Schouery 2018]

- Nó pai (para $i > 0$): $\lfloor (i-1)/2 \rfloor$,
- Nó filho esquerdo: $2 * i + 1$,
- Nó filho direito: $2 * i + 2$,

Um heap permite uma implementação muito eficiente de uma fila de prioridade. Vimos que uma lista ordenada contendo n elementos permite que a inserção na fila de prioridade (pqinsert) seja implementada usando uma média de aproximadamente $n/2$ acessos de nós, e a eliminação do mínimo ou máximo (pqmindelete ou pqmaxdelete) usando apenas um acesso a nó. Sendo assim, uma sequência de n inserções e n eliminações de uma lista ordenada conforme exigidas por uma classificação por seleção poderia exigir $O(n^2)$ operações. Embora a inserção na fila de prioridade usando uma árvore de busca binária exigisse somente um mínimo de $\log_2 n$ acessos a nós, ela poderia exigir até n acessos de nós se a árvore estivesse desbalanceada. Sendo assim, uma classificação por seleção usando uma árvore de busca binária poderia também demandar $O(n^2)$ operações, embora na média só fossem necessárias $O(n \log n)$. [Tanenbaum 1995]

Conforme veremos, um heap permite que a inserção e a eliminação sejam implementadas em $O(\log n)$ operações. Por conseguinte, uma classificação por seleção consistindo em n inserções e n eliminações pode ser implementada usando um heap em $O(n \log n)$ operações, mesmo no pior caso. Uma vantagem adicional é que o próprio heap pode ser implementado dentro do vetor de entrada x , usando uma implementação sequencial de árvore binária quase completa. O único espaço extra necessário é para as variáveis do programa. [Tanenbaum 1995]

5. Metodologia

5.1. Desenvolvimento do Banco de Dados

Para armazenar dados de forma que fiquem disponíveis à aplicação, foi usado o banco de dados relacional MySQL.

Em relação as pessoas, é necessário armazenar dados tais como documento identificador (CPF), nome, meios para contato (e-mail e telefones), data de nascimento e endereço. Especificamente sobre o último item, endereço, sua importância se torna grande para os casos nos quais os hospitais aplicarão a vacina na casa do paciente.

Pessoas cadastradas podem estar agendadas/vacinadas ou esperando na fila pelo agendamento. Uma data de vacinação pode ter várias pessoas agendadas para ela, contudo só deve ser possível cadastrar uma pessoa para uma única data de vacinação por vez.

Além das informações referentes as pessoas, é necessário guardar informações sobre doenças que aumentam o risco em relação ao COVID-19, para que assim seja calculada a prioridade de cada paciente. Cada doença deve conter nome e peso, além de um código identificador.

Cada cadastro de pessoa pode estar relacionado com várias doenças e cada doença pode estar relacionada com várias pessoas.

5.1.1. Projeto Conceitual

O modelo conceitual conta com uma entidade geral pessoa com todos os atributos requisitados (Figura 3). Essa entidade se especializa em: em_fila e atendida.

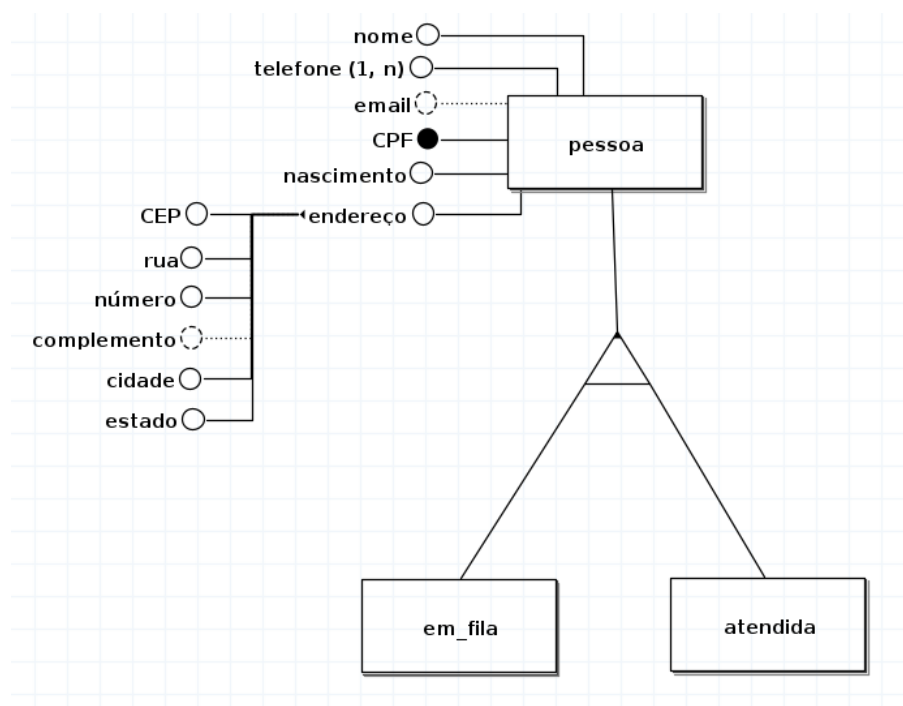


Figura 3. Entidade geral pessoa e suas especializações

Cada pessoa atendida tem uma relação n:1 com uma data de vacinação (Figura 4). Para remarcar uma pessoa atendida, é necessário alterar o registro já existente da relação.

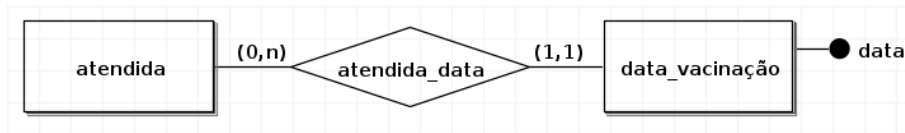


Figura 4. Relação de pessoa atendida com data de vacinação

Cada pessoa pode se relacionar com uma ou mais doenças numa relação n:n (Figura 5).

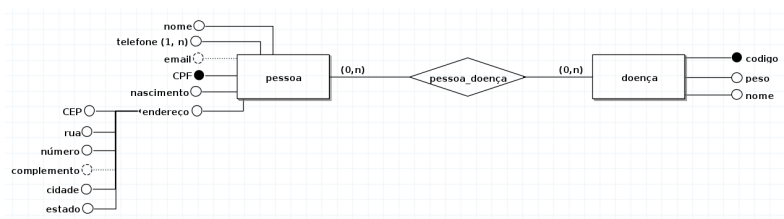


Figura 5. Relacionamento de pessoa com doença

O projeto conceitual completo está representado na Figura 6.

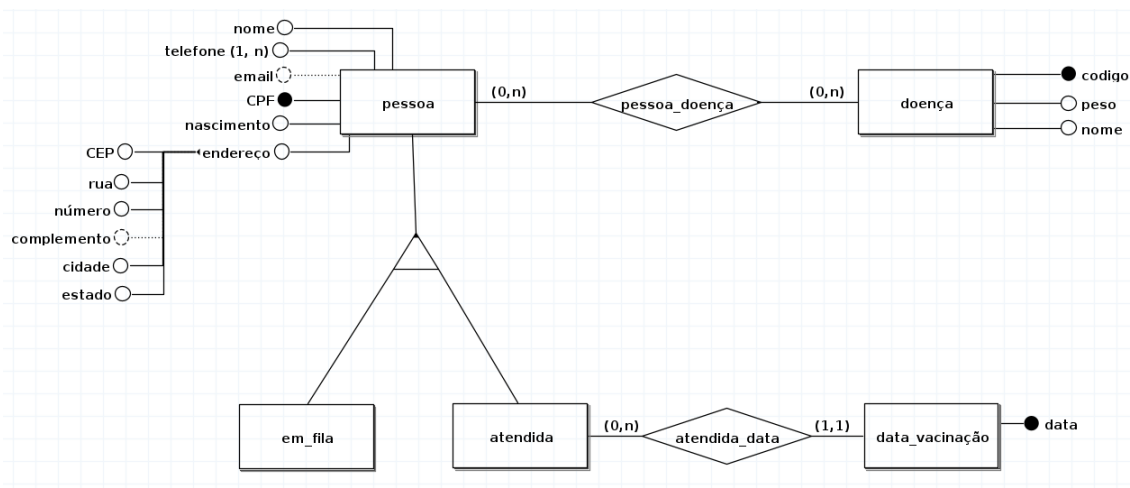


Figura 6. Projeto conceitual completo

5.1.2. Projeto Lógico

Para retirar, o atributo multi-valorado e o atributo composto da entidade pessoa, foram utilizados respectivamente os métodos de criação de tabela e decomposição de atributo.

O método de conversão da generalização/especialização escolhido foi o de fusão de tabelas. A entidade pessoa recebeu os atributos que seria de sua especialização, data, e um atributo para indicar seu tipo (Figura 7).

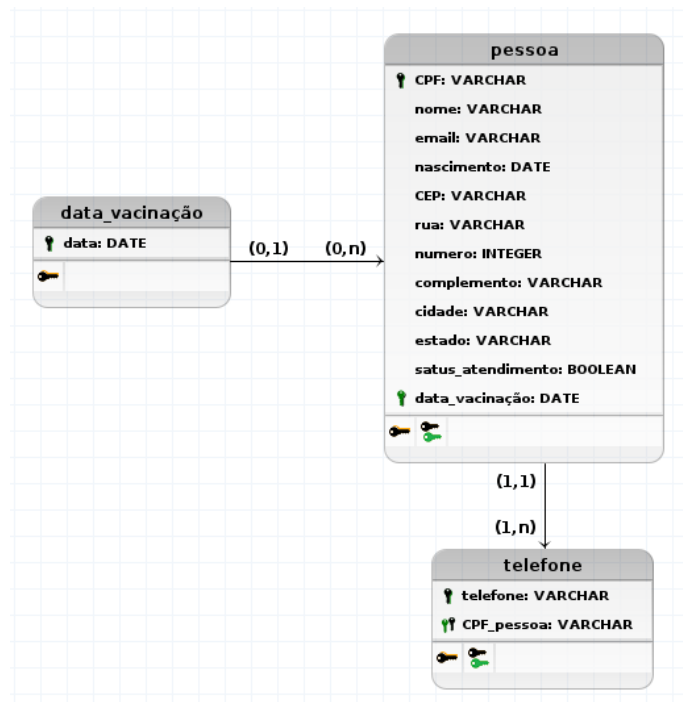


Figura 7. Entidade pessoa e suas especializações mapeadas no modelo lógico

A relação n:n de pessoa com doença foi mapeada para o modelo lógico como uma nova tabela que tem como chave primária a combinação das duas chaves estrangeiras vindas de pessoa e doença (Figura 8).

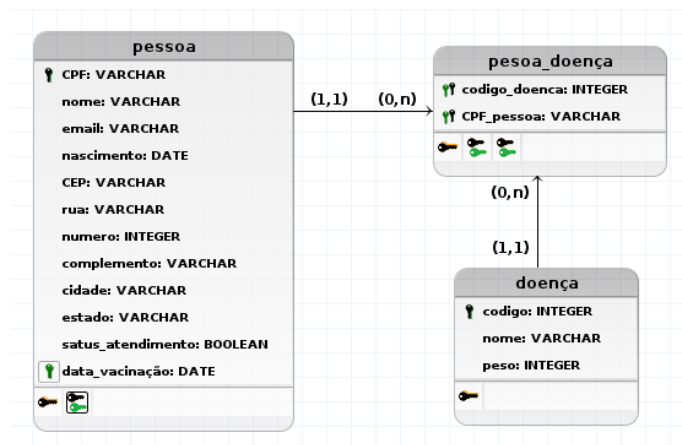


Figura 8. Mapeamento do relacionamento de pessoa e doença

O projeto lógico completo é descrito na Figura 9.

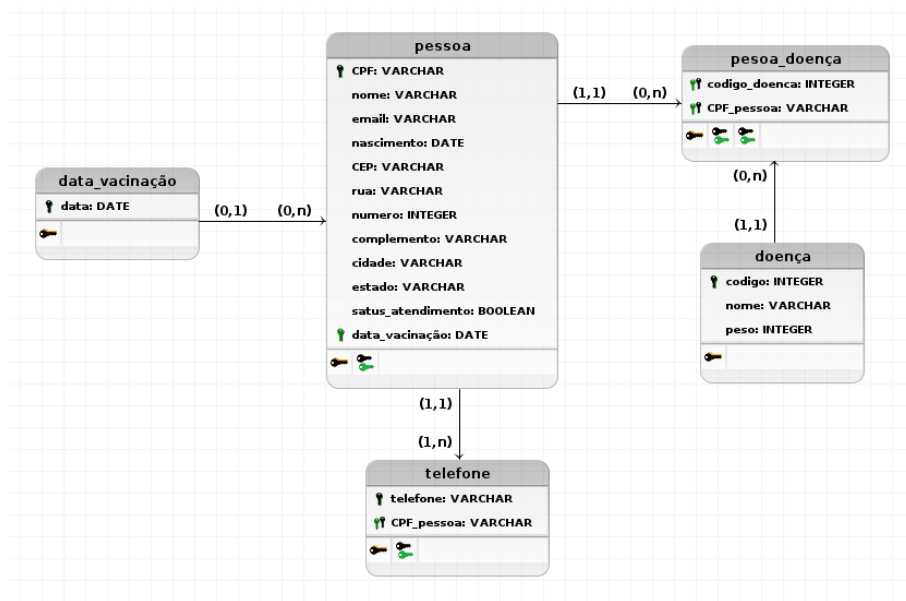


Figura 9. Projeto lógico completo

5.1.3. Conexão MySQL-Python

Para acessar o banco de dados, fazer consultas, inserções e atualizações, a aplicação usa a biblioteca `mysql.connector` para o Python. Após importar a biblioteca é necessário conectar com o banco de dados local. A conexão é feita da seguinte forma:

```

1 db = mysql.connector.connect (
2     host="localhost",
3     user="root",
4     passwd="Ss3131",
5     database="dados_saude"
6 )
7
8 mycursor = db.cursor()
  
```

Após conectar a aplicação ao banco, é gerado um cursor intitulado de *mycursor* que executará as operações. Para cada execução, é utilizado o método *mycursor.execute(operation)* e a query a ser executada é colocada como uma string no lugar de *operation*.

Uma das principais operações utilizadas é a de pesquisa por dados específicos no banco usando *SELECT*. Quando uma query que retorna dados é usada, o *mycursor* se torna iterável e cada iteração retorna uma linha da tabela gerada. Outra forma de acessar os dados retornados é usando o método *mycursor.fetchall()* que fornece uma lista com todas as linhas da tabela resultante.

```

1 # Retorna as datas de vacinação cadastradas - formato: [data1, data2
2   ...]
3 def get_datas():
4     mycursor.execute('SELECT * FROM data_vacinacao ORDER BY data_vac')
5     return [i[0] for i in mycursor]
  
```

```

6 # Retorna todos os cpfs agendados para o dia de vacinacao informado -
  formato: [cpf1, cpf2 ...]
7 def get_cpf_em_data(data):
8     mycursor.execute(f'SELECT CPF, nome FROM pessoa WHERE data_vac="{
  data}"')
9     return mycursor.fetchall()

```

Outra operação muito usada é a de inserção, pelo comando *INSERT INTO*. Para todas as operações que fazem alterações no banco, é necessário o uso do método *db.commit()* para efetivar a alteração. Caso haja algum erro, a aplicação usa o método *db.rollback()* para desfazer alterações já realizadas na operação.

```

1 # Adiciona doença a um cpf
2 def add_pessoa_doenca(cpf, doenca):
3     try:
4         mycursor.execute(f'INSERT INTO pessoa_doenca (cod_doenca,
  CPF_pessoa) VALUES ({doenca}, "{cpf}")')
5         db.commit()
6     except Exception as e:
7         db.rollback()
8         print(e)

```

5.2. Ordenação de Pessoas

5.2.1. Definição de Prioridade

A prioridade de cada pessoa pode ser entendida pela soma dos pesos de cada doença que ela possui. A seguinte query, no banco de dados mostrado anteriormente, retorna uma lista com todas as pessoas não vacinadas/agendadas juntas de suas respectivas prioridades:

```

1 SELECT pessoa.CPF, x.peso
2 FROM (
3     SELECT pd.CPF_pessoa AS CPF, SUM(d.peso) AS peso
4     FROM pessoa_doenca AS pd
5     INNER JOIN doenca AS d
6     ON pd.cod_doenca=d.cod_doenca
7     GROUP BY pd.CPF_pessoa
8 ) AS x
9 RIGHT JOIN pessoa
10 ON x.CPF=pessoa.CPF
11 WHERE pessoa.data_vac IS NULL;

```

Para os fins da aplicação, não é necessário carregar todos os dados das pessoas, apenas o CPF e a prioridade.

A query exibida acima é executada em um programa Python que interage com o banco de dados, e para tornar a manipulação e interpretação dos dados mais simples, para cada linha da tabela gerada, é criado um objeto Pessoa dentro da aplicação.

A definição da classe Pessoa implementa métodos de comparação com outros objetos do mesmo tipo. Os métodos de comparação levam em consideração o peso de cada objeto Pessoa, ou seja, se uma pessoa tem mais prioridade que outra, pode ser considerada "maior". Sendo assim, duas ou mais pessoas podem ser comparadas da mesma forma com que se comparam números inteiros.

```

1 class Pessoa:
2     def __init__(self, cpf, peso_doencas=None, nome=None, nascimento=
      None):
3         self.cpf = cpf
4         self.nascimento = nascimento
5         self.nome = nome
6
7         if peso_doencas is None:
8             self.prioridade = 0
9         else:
10            self.prioridade = peso_doencas
11
12    def __gt__(self, other):
13        return self.prioridade > other.prioridade
14
15    def __lt__(self, other):
16        return self.prioridade < other.prioridade
17
18    def __eq__(self, other):
19        return self.prioridade == other.prioridade
20
21    def __ne__(self, other):
22        return not self.__eq__(other)
23
24    def __le__(self, other):
25        return self.prioridade <= other.prioridade
26
27    def __ge__(self, other):
28        return self.prioridade >= other.prioridade

```

5.2.2. Algoritmo de Fila

Para organização das pessoas por ordem de prioridade a aplicação usa um algoritmo de fila de prioridade implementada com base em Max-Heap. A implementação da fila seguiu os padrões do Max-Heap explicados no referencial teórico.

```

1 # Lista de prioridade com Max-Heap
2
3 class NoFatherException(Exception):
4     pass
5
6 class NoSonException(Exception):
7     pass
8
9 class EmptyQueueException(Exception):
10    pass
11
12 class Fila:
13     def __init__(self, elementos:list=None):
14         # Elementos da fila
15         self._elementos = []
16
17         # Popular fila a partir de uma lista de elementos já existentes
18         if elementos is not None:

```

```

19         self._criar(elementos)
20
21     def __repr__(self):
22         return str(self._elementos)
23
24     @staticmethod
25     def _get_pai(indice_elemento):
26         indice_pai = (indice_elemento-1)//2
27         if indice_pai < 0:
28             # Levanta a exceção NoFatherException caso elemento seja a
raíz da árvore
29             raise NoFatherException()
30         return indice_pai
31
32     @staticmethod
33     def _get_filho_esquerda(indice_elemento):
34         return 2*indice_elemento+1
35
36     @staticmethod
37     def _get_filho_direita(indice_elemento):
38         return 2*indice_elemento+2
39
40     # Escolhe filho de maior prioridade
41     def _escolher_filho(self, indice_pai):
42         indice_esquerda = Fila._get_filho_esquerda(indice_pai)
43         indice_direita = Fila._get_filho_direita(indice_pai)
44
45         indice_limite = len(self._elementos)-1
46
47         if indice_esquerda > indice_limite and indice_direita >
indice_limite:
48             raise NoSonException()
49         elif indice_direita > indice_limite:
50             return indice_esquerda
51         elif self._elementos[indice_esquerda] > self._elementos[
indice_direita]:
52             return indice_esquerda
53         else:
54             return indice_direita
55
56     # Trocar elementos de posição
57     def _troca(self, indice1, indice2):
58         self._elementos[indice1], self._elementos[indice2] = self.
_elementos[indice2], self._elementos[indice1]
59
60     # Descer elemento para posição correta
61     def _descida(self, indice_elemento):
62         try:
63             indice = indice_elemento
64             elemento = self._elementos[indice]
65             indice_filho = self._escolher_filho(indice)
66
67             while elemento < self._elementos[indice_filho]:
68                 self._troca(indice, indice_filho)
69                 indice = indice_filho
70                 indice_filho = self._escolher_filho(indice)

```

```

71         except NoSonException:
72             pass
73
74         # Subir elemento para posição correta
75         def _subida(self, indice_elemento):
76             try:
77                 indice = indice_elemento
78                 elemento = self._elementos[indice]
79                 indice_pai = Fila._get_pai(indice)
80
81                 # Troca o elemento de posição com seu pai caso seja maior
que ele
82                 # Repete até estar na posição certa
83                 while elemento > self._elementos[indice_pai]:
84                     self._troca(indice, indice_pai)
85                     indice = indice_pai
86                     indice_pai = Fila._get_pai(indice)
87
88             except NoFatherException:
89                 pass
90
91         # Criar fila a partir de lista
92         def _criar(self, elementos:list):
93             self._elementos = elementos
94
95         # Executa a descida para as n/2 primeiras posições na lista
96         for i in range(len(self._elementos)//2-1, -1, -1):
97             self._descida(i)
98
99         # Inserir elemento
100        def inserir(self, elemento):
101            # Adiciona novo elemento no fim da árvore
102            self._elementos.append(elemento)
103
104            # Organiza a árvore com novo elementos
105            self._subida(len(self._elementos)-1)
106
107        # Remover elemento de maior prioridades
108        def remover(self):
109            try:
110                self._troca(0, -1)
111            except IndexError:
112                raise EmptyQueueException()
113
114            # Armazenar elemento
115            removido = self._elementos.pop()
116
117            # Ajustar árvore descendo elemento em 0 até sua posição correta
118            try:
119                self._descida(0)
120            except IndexError:
121                pass
122
123            # Retornar elemento retirado da fila
124            return removido
125

```

```

126     # Seleção do próximo elemento da fila (não remove)
127     def ver_proximo(self):
128         # Caso a fila esteja vazia, retorna None
129         try:
130             return self._elementos[0]
131         except IndexError:
132             raise EmptyQueueException()
133
134     def fila_completa(self):
135         return self._elementos
136
137     def __len__(self):
138         return len(self._elementos)

```

Pode ser observado que as comparações na fila são feitas diretamente entre os objetos, sem a necessidade de informar que tais objetos são especificamente do tipo Pessoa. Tal característica é possível graças a implementação dos métodos de comparação da classe Pessoa. Portanto, a fila se torna genérica, podendo ser usada para qualquer classe em que se tenha claro quando um objeto é maior que outro.

5.3. User Interface

5.3.1. Cadastro de Pessoas

Para cadastrar novas pessoas, a aplicação conta com uma tela de cadastro (Figura 10) que solicita os dados da pessoa a ser cadastrada. Como apresentado na seção 5.1.1 que define o modelo conceitual do banco de dados, as informações e-mail e complemento do endereço não são obrigatórias, portanto, todas as outras estão marcadas com "*" para indicar sua obrigatoriedade.

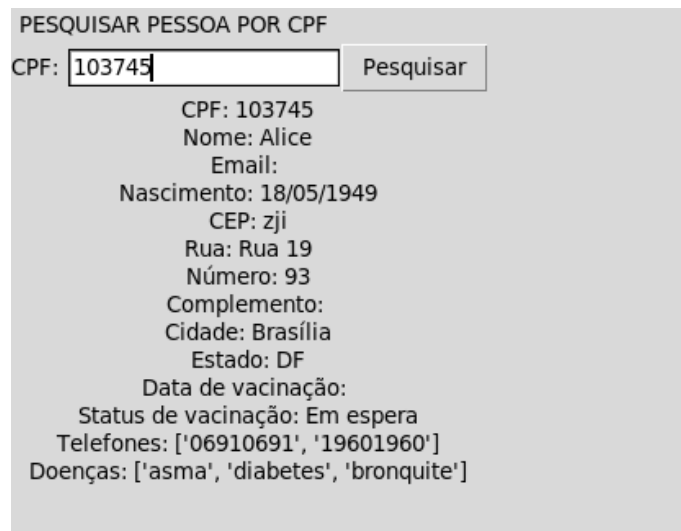
Para cada pessoa podem ser atribuídas quaisquer das doenças já cadastradas.

A interface de cadastro de pessoas é organizada em seções verticais. No topo, o título "DADOS PESSOAIS" precede os campos para CPF e Nome Completo. Abaixo, o campo "Nascimento" é acompanhado por um calendário para a seleção de data. A seção "ENDEREÇO" contém campos para CEP, Rua, Número, Cidade, Estado e um campo opcional para Complemento. Segue a seção "INFORMAÇÕES DE CONTATO" com campos para Telefone, Celular e Email. A seção "INFORMAÇÕES DE SAÚDE" apresenta uma lista de doenças: 7-Aids, 1-asma, 4-bronquite, 3-câncer, 2-diabetes, 8-HIV e 6-Paralisia Cerebral. No rodapé, há dois botões: "Confirmar" e "Cancelar".

Figura 10. Tela para cadastro de pessoas

5.3.2. Pesquisar Pessoas

Os dados de uma pessoa podem ser acessados usando o identificador (CPF) da mesma na tela de pesquisa (Figura 11).

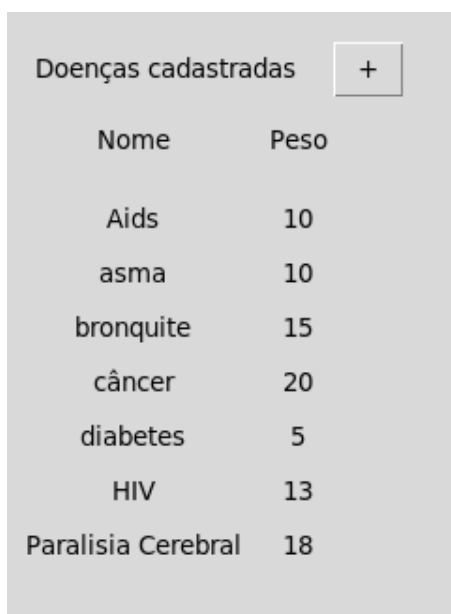


A interface de pesquisa de pessoas apresenta um formulário com o título "PESQUISAR PESSOA POR CPF". Abaixo do título, há um campo de entrada rotulado "CPF:" com o valor "103745" preenchido e um botão "Pesquisar" ao lado. Abaixo do formulário, os dados da pessoa são exibidos em texto centralizado: CPF: 103745, Nome: Alice, Email: (vazio), Nascimento: 18/05/1949, CEP: zji, Rua: Rua 19, Número: 93, Complemento: (vazio), Cidade: Brasília, Estado: DF, Data de vacinação: (vazio), Status de vacinação: Em espera, Telefones: ['06910691', '19601960'], Doenças: ['asma', 'diabetes', 'bronquite']

Figura 11. Tela para pesquisa de pessoas

5.3.3. Editar/Cadastrar Doenças

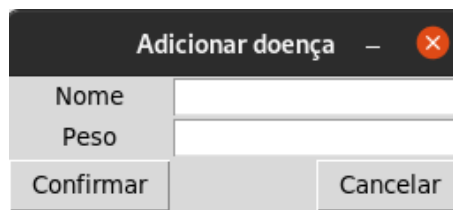
Na tela de doenças (Figura 12), é possível adicionar novas doenças clicando na figura "+" (Figura 13) ou editar as já existentes clicando sobre seus nomes. Editar permite mudar o nome e até mesmo o peso de uma doença.



A interface de exibição de doenças mostra o título "Doenças cadastradas" e um botão "+" para adicionar novas doenças. Abaixo, há uma tabela com duas colunas: "Nome" e "Peso".

Nome	Peso
Aids	10
asma	10
bronquite	15
câncer	20
diabetes	5
HIV	13
Paralisia Cerebral	18

Figura 12. Tela para exibição de doenças



Adicionar doença

Nome	<input type="text"/>
Peso	<input type="text"/>
Confirmar	Cancelar

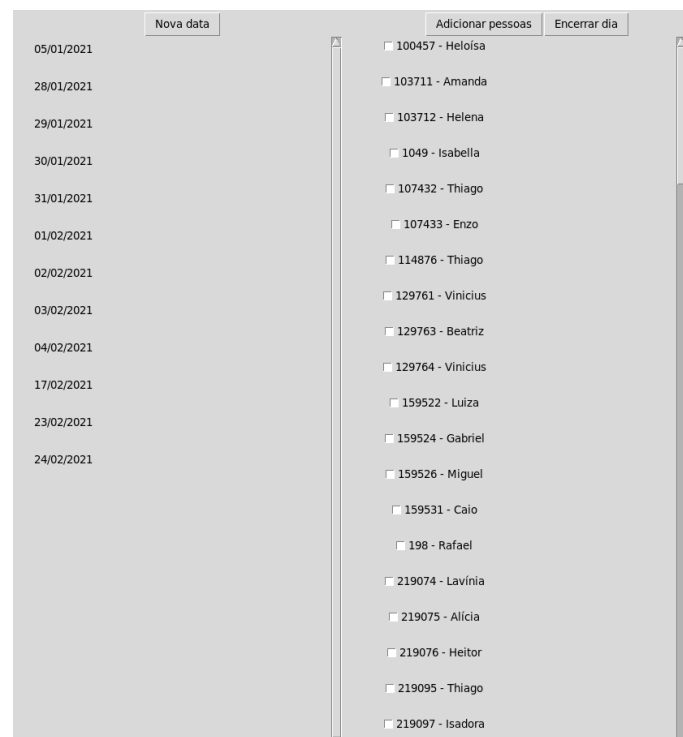
Figura 13. Tela para cadastro de doenças

5.3.4. Datas de Vacinação

Na tela de datas de vacinação (Figura 14), é possível visualizar cada data cadastrada. Ao clicar em uma data, é exibida na parte esquerda da tela uma lista com todas as pessoas agendadas para o dia específico. Caso seja necessário, é possível adicionar mais pessoas a esse dia clicando em "Adicionar pessoas".

A parte da tela que contém o identificador e nome das pessoas, funciona como uma espécie de lista de presença. Para cada pessoa agendada que comparecer e receber a vacina, a caixa ao lado deve ser marcada para que, ao final do dia, o botão "Encerrar dia" seja pressionado e todas as pessoas que não compareceram retornem para um estado em que são elegíveis para a fila.

Cada caixa marcada ou desmarcada afeta imediatamente o banco de dados, portanto, caso a aplicação seja reiniciada, o estado da caixa será mantido.



Nova data

Adicionar pessoas

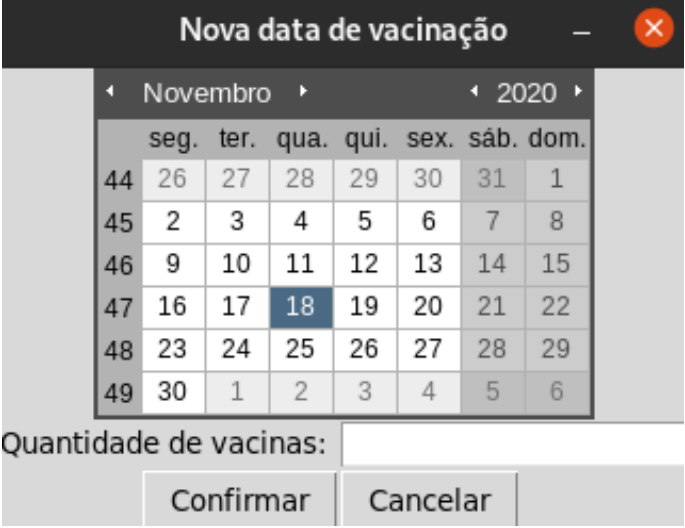
Encerrar dia

Data	Pessoas
05/01/2021	<input type="checkbox"/> 100457 - Heloisa
28/01/2021	<input type="checkbox"/> 103711 - Amanda
29/01/2021	<input type="checkbox"/> 103712 - Helena
30/01/2021	<input type="checkbox"/> 1049 - Isabella
31/01/2021	<input type="checkbox"/> 107432 - Thiago
01/02/2021	<input type="checkbox"/> 107433 - Enzo
02/02/2021	<input type="checkbox"/> 114876 - Thiago
03/02/2021	<input type="checkbox"/> 129761 - Vinicius
04/02/2021	<input type="checkbox"/> 129763 - Beatriz
17/02/2021	<input type="checkbox"/> 129764 - Vinicius
23/02/2021	<input type="checkbox"/> 159522 - Luiza
24/02/2021	<input type="checkbox"/> 159524 - Gabriel
	<input type="checkbox"/> 159526 - Miguel
	<input type="checkbox"/> 159531 - Calo
	<input type="checkbox"/> 198 - Rafael
	<input type="checkbox"/> 219074 - Lavínia
	<input type="checkbox"/> 219075 - Alícia
	<input type="checkbox"/> 219076 - Heitor
	<input type="checkbox"/> 219095 - Thiago
	<input type="checkbox"/> 219097 - Isadora

Figura 14. Tela de exibição das datas de vacinação

Para adicionar datas de vacinação, basta clicar em "Nova data" e escolher uma válida (que não está cadastrada) (Figura 15). No campo "Quantidade de vacinas" pode ser

colocado qualquer número maior ou igual a 0, referente a quantidade de pessoas que se deseja agendar para o dia selecionado. É possível cadastrar o dia sem agendar nenhuma pessoa a ele, para isso, a o campo de quantidade de vacinas pode ser 0 ou simplesmente não preenchido. É possível adicionar pessoas ao dia mais tarde na aplicação, como mostrado anteriormente.



Nova data de vacinação

◀ Novembro ▶ 2020 ▶

	seg.	ter.	qua.	qui.	sex.	sáb.	dom.
44	26	27	28	29	30	31	1
45	2	3	4	5	6	7	8
46	9	10	11	12	13	14	15
47	16	17	18	19	20	21	22
48	23	24	25	26	27	28	29
49	30	1	2	3	4	5	6

Quantidade de vacinas:

Figura 15. Tela para cadastro de data de vacinação

5.4. Desempenho

Para avaliar o desempenho da aplicação em relação a performance, foram criadas diferentes bases de dados com 5 mil, 10 mil, 100 mil (Figura 16), 1 milhão, 3 milhões e 5 milhões (Figura 17) de pessoas fictícias contendo diferentes doenças.

Foram avaliados os tempos de execução das operações mais importantes da aplicação: recuperação de todas as pessoas com seus pesos totais do banco de dados, construção da fila de prioridade e retirada da pessoa com maior prioridade da fila.

Os resultados obtidos foram os seguintes:

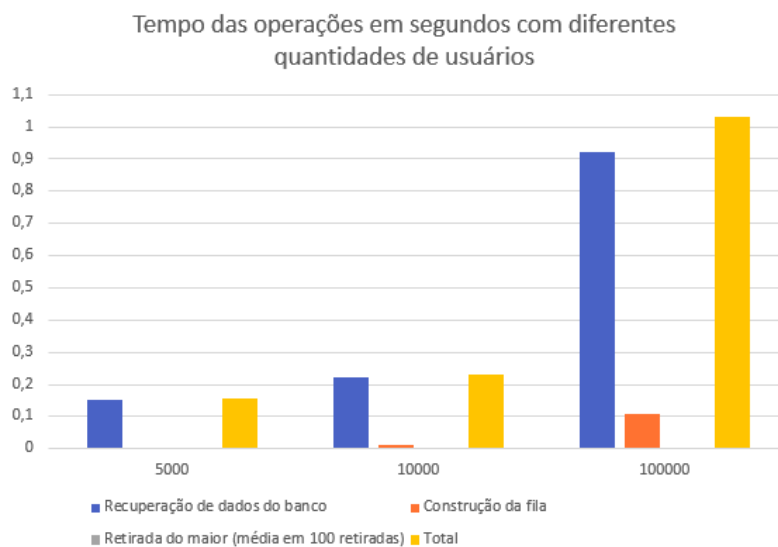


Figura 16. Performance para 5 mil, 10 mil e 100 mil pessoas



Figura 17. Performance para 1 milhão, 3 milhões e 5 milhões de pessoas

Vale ressaltar que as operações apresentadas acima são executadas apenas no momento de cadastro de uma nova data de vacinação ou de adição de pessoas a uma data já existente. Para consultar pessoas agendadas para uma data, não é necessário recuperar dados de todas as pessoas do banco e nem construir uma fila.

Os testes foram feitos partindo do princípio de que nenhuma pessoa no banco está agendada/vacinada. É importante que isso seja levado em consideração, pois quanto maior a quantidade de pessoas não agendadas/vacinadas, maior a quantidade de pessoas na construção da fila.

6. Conclusão

A disponibilização das vacinas contra o COVID-19 precisa ser feita com extrema cautela, já que existem pessoas com maior risco de complicações graves de saúde ou até mesmo óbito por conta da doença.

Nesse contexto, a aplicação desenvolvida no presente trabalho fornece maneiras de organizar as pessoas de uma região de acordo com sua prioridade, definida pelo estado de saúde de cada indivíduo. A aplicação permite criar datas de vacinação e agendar pessoas nelas, oferecendo também um bom nível de controle e organização para os pontos de vacinação.

Se tratando de performance, os algoritmos usados para criação da fila se provaram eficientes, funcionando em tempo linear e dependendo apenas da quantidade de pessoas que formam a fila. Contudo, o desempenho das consultas ao banco de dados ficou abaixo do desejado, podendo chegar a mais de 130 segundos numa base de dados com 5 milhões de pessoas.

Possíveis linhas de estudo a partir desse trabalho podem incluir uma aplicação web para cadastro de pessoas e um sistema integrado ao invés de local.

Referências

- Abdala, D. (2017). *Fundamentos da Estrutura da Informacao*. 1st edition.
- Brunet, J. (2019). *Estrutura de Dados e Algoritmos*. 1st edition.
- Farias, R. (2009). *Estrutura de Dados e Algoritmos*. 1st edition.
- Feofiloff, P. (2015). *Analise de Algoritmos*. 1st edition.
- Gomes, J. (2020). *OPAS*. 1st edition.
- Rocha, J. (2010). *Introdução à Linguagem Python*. 1st edition.
- Schouery, R. (2018). *Filas de Prioridade e Heap*. 1st edition.
- Silva, I. (2019). *Linguagem de Programação python*. 1st edition.
- Tanenbaum, A. A. (1995). *Estruturas de Dados Usando C*. MAKRON Books, 1st edition.