

Eduardo Roger Silva Nascimento
Matrícula: 2120586

**Geração de *training datasets* para sistemas
conversacionais de busca sobre *RDF datasets***

Rio de Janeiro - RJ

Dezembro - 2022

Eduardo Roger Silva Nascimento
Matrícula: 2120586

**Geração de *training datasets* para sistemas
conversacionais de busca sobre *RDF datasets***

Projeto de Programação Final apresentada ao
curso de Mestrado em Informática da Pontifí-
cia Universidade Católica do Rio de Janeiro,
para aprovação no componente curricular.

Rio de Janeiro - RJ
Dezembro - 2022

Sumário

| | | |
|------------|---|-----------|
| 1 | ESPECIFICAÇÃO DO PROJETO | 3 |
| 1.1 | Finalidade | 3 |
| 1.2 | Escopo | 4 |
| 1.3 | Requisitos | 6 |
| 1.3.1 | Requisitos Funcionais | 6 |
| 1.3.2 | Requisitos não funcionais | 7 |
| 1.4 | Casos de Uso | 7 |
| 2 | DESENVOLVIMENTO | 12 |
| 2.1 | Plataformas e tecnologias | 12 |
| 2.2 | Arquitetura | 12 |
| 2.3 | Diagrama de Classe | 13 |
| 2.4 | Diagrama de Sequência | 14 |
| 2.5 | Testes | 15 |
| 2.5.1 | Checação da Aplicação | 15 |
| 2.5.2 | Geração de Datasets | 15 |
| 2.5.3 | Checação de rótulos de classificação para combobox da interface | 15 |
| 3 | MANUAL DO USUÁRIO | 22 |
| 3.1 | Requisitos necessários | 22 |
| 3.2 | Requisitos mínimos | 22 |
| 3.3 | Instalação do Sistema | 22 |
| 3.4 | Apresentação do Sistema | 23 |
| | REFERÊNCIAS | 27 |

1 Especificação do Projeto

Este capítulo apresenta a especificação do projeto, contém a sua finalidade, seu escopo e os requisitos tanto os funcionais quanto os não funcionais e finaliza apresentando o diagrama de casos de uso e suas descrições de forma detalhada.

1.1 Finalidade

Um sistema conversacional de busca é um sistema que se baseia em interfaces de usuário conversacionais que oferecem suporte a interações homem-máquina a fim de dar informações para o usuário (THOMAS et al., 2021).

Dado um enunciado do usuário, o sistema conversacional é encarregado de recuperar um a poucos documentos, de dentro de uma coleção de documentos, que contenham a resposta à necessidade de informação expressa no enunciado.

Para isso, os sistemas conversacionais devem entender o enunciado e sua relação com o contexto expresso na conversa, encontrar informações relevantes em uma base de conhecimento ou coleção de documentos e por fim, selecionar uma lista única ou muito restrita de resultados, de preferência incluindo a resposta específica Mele et al. (2021). Um enunciado como *Qual é a capital da Albânia?* é fácil de processar pelo sistema conversacional, pois há referências explícitas sobre os assuntos em que ele deve processar, nesse caso, é um enunciado autoexplicativo. Entretanto, dando continuidade a conversação, caso a próxima sentença fosse *Qual era a população total em 1950 desse país?*, esse enunciado se caracteriza como não autoexplicativo e o sistema deveria ser capaz de entender o contexto a fim de saber que a pergunta se refere a Albânia. Essa é uma tarefa difícil e muitos enunciados carecem de contexto ou até ambíguos.

Para entender o contexto do enunciado, sistemas de busca conversacionais podem ser treinados com classificadores de enunciados a fim de identificar aqueles que são autoexplicativos e que dependem de contexto. Para aqueles que dependem de contexto, o sistema pode reescrever esse enunciado de forma que coloque referências explícitas sobre os assuntos no enunciado. Foi o que fizeram Mele et al. (2021), adicionaram contexto em enunciados de forma adequada a fim de melhorar a tarefa de recuperação em sistemas conversacionais. Isso foi feito através da classificação de enunciados para identificar contextos. As perguntas eram treinadas graças à taxonomia de enunciados deles baseados em 3 classes: enunciado autoexplicativo, enunciado dependendo do primeiro tópico ou tópico anterior. Assim que identificassem aquelas perguntas que carecem de contexto, utilizavam uma técnica de reescrita para transformar essa pergunta em estado bruto para

um enriquecido, autoexplicativo e de fácil processamento para um sistema automático de recuperação de informação.

A fim de fazer a tarefa de classificação, são necessários datasets de treino que forneçam enunciados suficientes para treinar esses classificadores. Porém, existem problemas a respeito desses datasets, tais como são poucos os disponíveis publicamente, ou não possuem uma grande quantidade de enunciados, ou são criados e rotulados anualmente por especialistas, como foi realizado por [Mele et al. \(2021\)](#).

Desse modo, o objetivo dessa aplicação é gerar de forma automática esses datasets para treinamento e ela se destina a acadêmicos ou companhias que visam implementar seus sistemas conversacionais de busca e treiná-los utilizando o dataset gerado automaticamente em cima de um dataset rdf específico nesse trabalho. No caso, o dataset rdf utilizado foi o Mondial database ([MAY, 1999](#)), mas somente para os dados que se referem a Europa.

1.2 Escopo

O escopo deste projeto é a geração automática de datasets de treinamento sobre o dataset Mondial¹ em RDF apenas para dados de países europeus a fim de treinar sistemas conversacionais de busca. A listagem 1.1 mostra um fragmento desse dataset usando a notação Turtle.

Listing 1.1 – Fragmento do Mondial dataset usando notação turtle

```
1 <countries/AL/>  rdf:type  :Country  ;
2   :name  "Albania"  ;
3   :localname  "Shqiperi"  ;
4   :carCode  "AL"  ;
5   :area  28750  ;
6   :capital  <countries/AL/cities/Tirana/>  ;
7   :population  2821977  ;
8   :hadPopulation  [  a  :PopulationCount;
9       :year  "2001"^^xsd:gYear;
10      :value  3069275]  ;
11  :populationGrowth  0.3  ;
12  :infantMortality  13.19  ;
13  :gdpTotal  12800  ;
14  :gdpInd  12  ;
15  :gdpServ  68.5  ;
16  :gdpAgri  19.5  ;
17  :inflation  1.7  ;
```

¹ Disponível em <<https://www.dbis.informatik.uni-goettingen.de/Mondial/>>

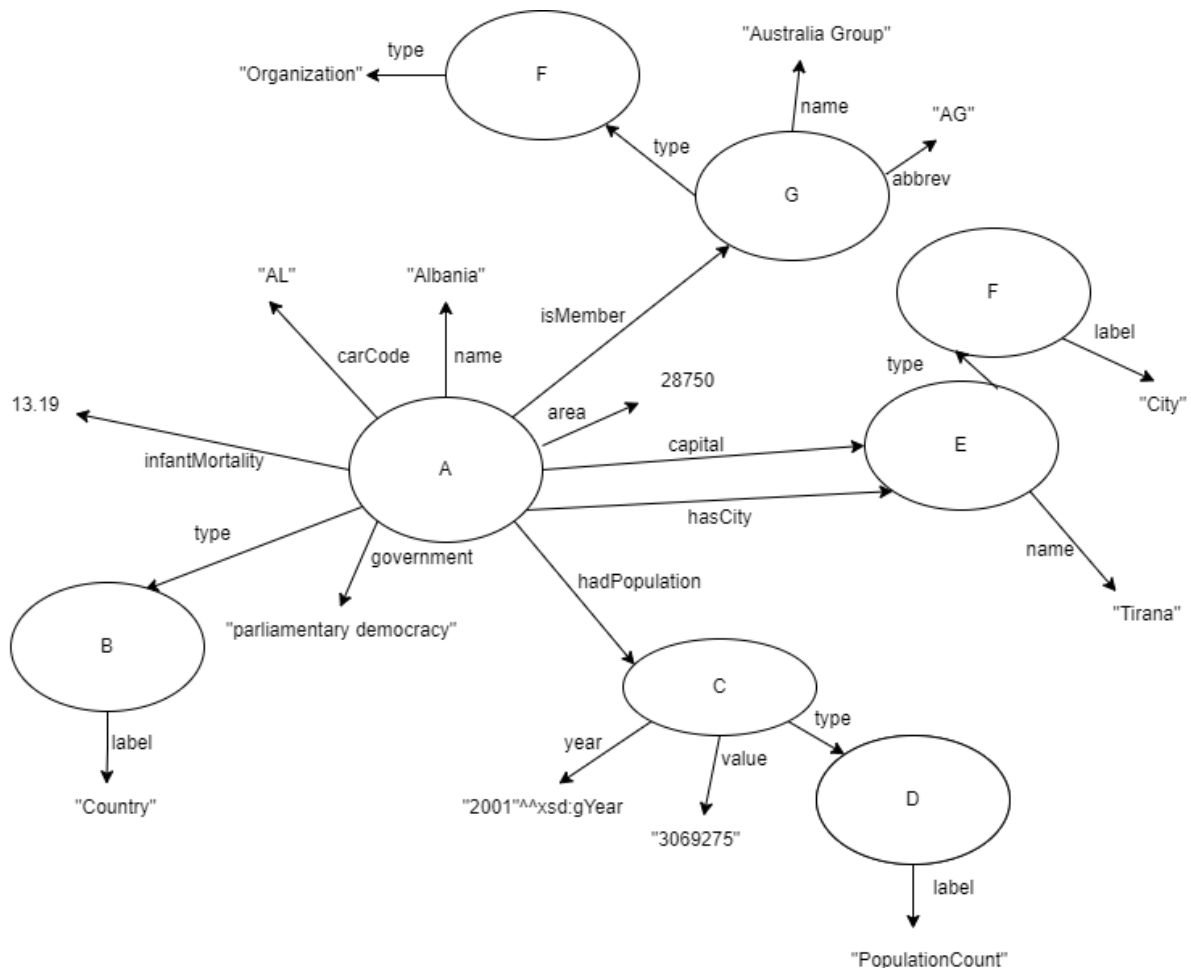
```

18 :unemployment 16.9 ;
19 :government "parliamentary democracy" ;
20 :independenceDate '1912-11-28'^^xsd:date ;
21 :wasDependentOf
22     <politicalbodies/Ottoman+Empire/> .
23 <politicalbodies/Ottoman+Empire/> rdf:type :PoliticalBody .

```

No backend da aplicação, foram considerados apenas três domínios: *Country*, *City* e *Organization* e as propriedades *government*, *area*, *capital*, *carCode*, *infantMortality*, *hasCity*, *hadPopulation*, *isMember*, *locatedIn*, *isCapital*, *checkCapital*, *abbrev*, *isCompost*, *checkOrganization*. A Figura 1 mostra exemplos de representação em grafo para esse domínio e com as propriedades escolhidas.

Figura 1 – Representação em Grafo no domínio da aplicação sobre o RDF dataset



Fonte: Próprio Autor

Já no frontend, o usuário deve informar quantas instâncias gostaria de gerar. Os resultados incluem os dados apresentados em uma tabela com três colunas, o código

do enunciado, o enunciado e o seu rótulo, o qual pode ser autoexplicativo, definido por *SELF_EXPLANATORY*, primeiro tópico como *FIRST_TOPIC* ou tópico anterior, *PREVIOUS_TOPIC* (MELE et al., 2021). Com os resultados, o usuário pode filtrar por enunciado ou por rótulo e por fim, pode exportá-los em formato ".csv".

1.3 Requisitos

1.3.1 Requisitos Funcionais

O sistema é constituído dos seguintes requisitos funcionais descritos abaixo:

1. Leitura da quantidade de instâncias a serem gerados.

| | |
|------------|--|
| Descrição | O usuário poderá informar a quantidade de instâncias a serem gerados pela aplicação. |
| Prioridade | Essencial |
| Esforço | Baixo |

2. Visualização dos dados gerados.

| | |
|------------|---|
| Descrição | O usuário pode visualizar os dados gerados pela aplicação em uma tabela composta por três colunas: código, enunciado e rótulo do enunciado. . |
| Prioridade | Essencial |
| Esforço | Baixo |

3. Filtrar os dados gerados.

| | |
|------------|--|
| Descrição | O usuário poderá filtrar os dados por enunciado, informando no campo de texto, ou por rótulo, selecionando uma entre as três opções disponíveis. |
| Prioridade | Desejável |
| Esforço | Médio |

4. Limpar o formulário que contém os filtros.

| | |
|------------|---|
| Descrição | O usuário após filtrar os dados, poderá limpar os filtros para que os dados voltem ao estado inicial. |
| Prioridade | Desejável |
| Esforço | Baixo |

5. Exportar para CSV

| | |
|-------------------|--|
| Descrição | O usuário poderá exportar os dados para csv. |
| Prioridade | Desejável |
| Esforço | Baixo |

1.3.2 Requisitos não funcionais

O sistema é constituído dos seguintes requisitos não funcionais descritos abaixo:

1. Multiplataforma

| | |
|-------------------|--|
| Descrição | O sistema poderá ser executado em diferentes sistemas operacionais, tais como Linux e Windows, necessitando somente de um browser. |
| Prioridade | Desejável |
| Esforço | Médio |

2. Robustez

| | |
|-------------------|---|
| Descrição | O sistema deverá ser estável, ou seja, não deverá travar ou fechar inesperadamente. |
| Prioridade | Essencial |
| Esforço | Alto |

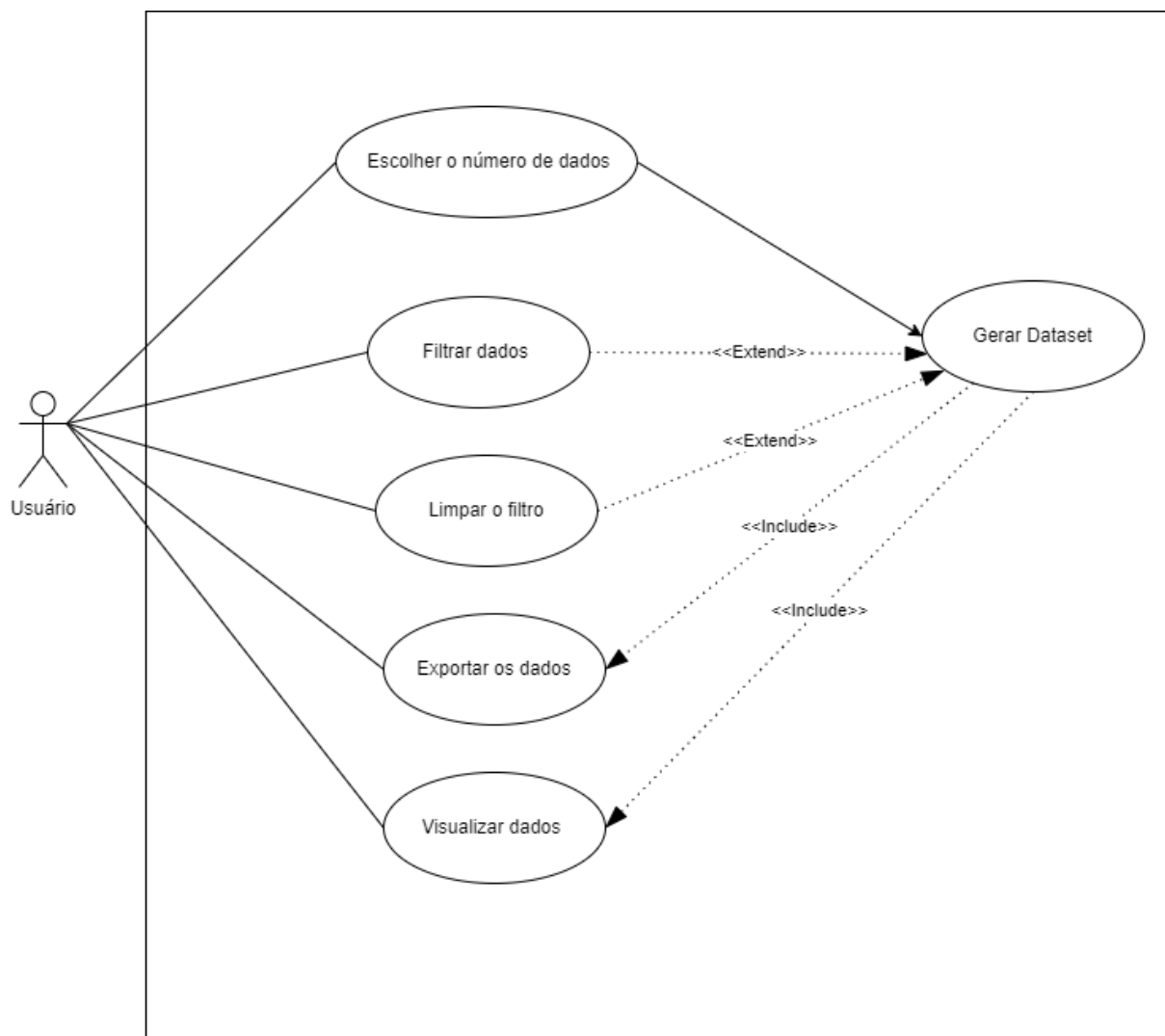
3. Performance

| | |
|-------------------|---|
| Descrição | O sistema deve responder à solicitação de um serviço específico do usuário dentro de um intervalo de tempo aceitável. |
| Prioridade | Essencial |
| Esforço | Alto |

1.4 Casos de Uso

Nessa subseção é apresentado o diagrama de casos de uso, exibida na Figura 2. Os casos de uso são baseadas nos requisitos do sistema e suas descrições detalhadas estão nas tabelas abaixo.

Figura 2 – Diagrama de Casos de Uso



Fonte: Próprio Autor

1. Escolher o número de dados

| Descrição | |
|-----------------------------|--|
| Ator: | O usuário |
| Descrição sucinta: | Escolher a quantidade de instâncias a serem gerados para formar o dataset de treino |
| Pré-condições: | Estar com o sistema aberto |
| Pós-condições: | Redireciona o usuário para o Caso de Uso "Gerar Dataset" |
| Cenário principal: | <ol style="list-style-type: none"> 1. O usuário acessa a página da aplicação; 2. O usuário informa a quantidade de dados a serem gerados no campo "Total samples to generate"; 3. O usuário clica no botão "Go"; 4. O sistema inicia o processo de geração de datasets. |
| Cenário alternativo: | <ol style="list-style-type: none"> 1. Se a conexão com o servidor backend estiver instável não será realizado a geração do dataset. 2. O sistema só gera de 5 a 1000 instâncias, caso o usuário informe uma quantidade fora desse intervalo. 3. Se o usuário deixar o campo vazio, o sistema emite um alerta. |

2. Gerar dataset

| Descrição | |
|-----------------------------|--|
| Ator: | O usuário |
| Descrição sucinta: | Criação de um dataset de treino automatizado |
| Pré-condições: | Ter escolhido o número de amostras. (Veja o Caso de Uso "Escolher o número de dados") |
| Pós-condições: | Exibe o dataset na tela. |
| Cenário principal: | <ol style="list-style-type: none"> 1. O sistema gera um dataset para treinamento com o número de instâncias informados pelo usuário; 2. O sistema faz rola a página para baixo automaticamente; 3. O sistema exibe o dataset na tela em uma tabela. |
| Cenário alternativo: | Se a conexão com o servidor backend estiver instável não será realizado a geração do dataset e o sistema exibe um alerta. |

3. Filtrar dados gerados

| Descrição | |
|-----------------------------|---|
| Ator: | O usuário |
| Descrição sucinta: | Filtra os dados gerados que aparecem na tabela pela condição colocada pelo usuário |
| Pré-condições: | Estar com o sistema aberto |
| Pós-condições: | Altera a exibição dos dados na tela |
| Cenário principal: | <ol style="list-style-type: none"> 1. O usuário seleciona uma opção para o rótulo ou preenche o campo de texto para buscar por enunciado; 2. O usuário clica no botão roxo "Search"; 3. O sistema altera a exibição dos dados de acordo com as condições do usuário. |
| Cenário alternativo: | Nenhum. |

4. Limpar o filtro dos dados

| Descrição | |
|-----------------------------|--|
| Ator: | O usuário |
| Descrição sucinta: | Limpa os campos no formulário de filtro e os dados voltam para o estado inicial |
| Pré-condições: | Estar com o sistema aberto |
| Pós-condições: | Os campos no formulário de filtro são limpos e os dados voltam ao estado inicial |
| Cenário principal: | <ol style="list-style-type: none"> 1. O usuário clica no botão de cor vermelha "Clear"; 2. O sistema limpa os campos do formulário e os dados na tabela voltam para o estado quando foram gerados. |
| Cenário alternativo: | Nenhum |

5. Exportar os dados

| Descrição | |
|-----------------------------|---|
| Ator: | O usuário |
| Descrição sucinta: | Exporta os dados para um arquivo csv. |
| Pré-condições: | Ter gerado o dataset. (Veja o Caso de Uso "Gerar Dataset") |
| Pós-condições: | Alterna entre as câmeras criadas, mostrando a visualização da câmera selecionada; |
| Cenário principal: | <ol style="list-style-type: none"> 1. O usuário clica no botão laranja "Export CSV"; 2. O sistema faz o download dos dados em um arquivo csv. |
| Cenário alternativo: | Se nenhum dado for gerado, o sistema exibe um alerta |

6. Visualizar dados gerados

| Descrição | |
|-----------------------------|--|
| Ator: | O usuário |
| Descrição sucinta: | Exibe as amostras do dataset |
| Pré-condições: | Ter gerado o dataset. (Veja o Caso de Uso "Gerar Dataset") |
| Pós-condições: | Exibe as amostras que compõe o dataset |
| Cenário principal: | 1. O usuário gera o dataset; 2. O sistema exibe as os dados em uma tabela com três colunas. |
| Cenário alternativo: | Se a conexão com o servidor backend estiver instável, os dados não serão exibidos. |

2 Desenvolvimento

Nesta capítulo é explanado o processo de desenvolvimento da aplicação. Apresentando deste as tecnologias usadas até sua arquitetura.

2.1 Plataformas e tecnologias

A aplicação foi desenvolvida usando o conceito de cliente-servidor. No servidor, a linguagem escolhida para desenvolvimento foi Python 3.10.7. Foi implementado uma api com a biblioteca Flask e para checagem dos endpoints utilizou o Postman. Para geração de datasets utilizou a biblioteca RDFLib e para testes, o Pytest. A instalação do Python foi feita localmente na máquina. Para a construção da visualização do cliente, foi usando o ambiente de execução Node.js, responsável por rodar o framework Angular. A Interface Gráfica de Usuário (GUI), foi montada usando HTML5, CSS, Typerscript e o Material Dashboard Angular construído com auxílio do framework Bootstrap. Foram usados a IDE Visual Studio Code da empresa Microsoft para suporte ao desenvolvimento.

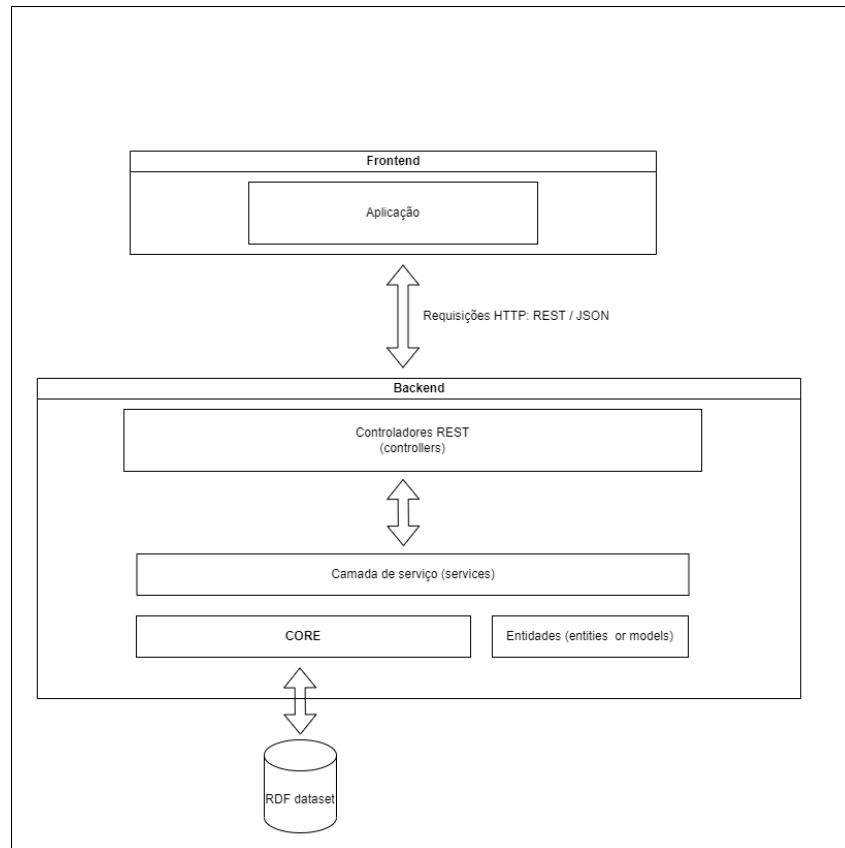
A tabela abaixo resume as tecnologias e plataformas utilizadas para o desenvolvimento do programa:

| | |
|-------------------------------------|---|
| Linguagem de Programação | Python 3.10.7 e Typescript |
| IDE | Visual Studio Code |
| Ambientes de desenvolvimento | Node.js |
| Bibliotecas no Servidor | Flask, RDFLib e Pytest |
| Bibliotecas no Cliente | Angular, Material Dashboard Angular |
| Plataformas | Windows, Linux, Mac OS X |
| Navegadores Suportados | Firefox 4+, Safari 5.1+, Opera 15+, Google Chrome 9+ e Microsoft Edge |

2.2 Arquitetura

A arquitetura do programa usa o padrão de Camadas e assim pode se fazer o desenvolvimento dessa aplicação em etapas. Esse padrão tem como vantagens, além da organização, ganha-se tempo e performance no desenvolvimento da aplicação, o código tende a ser mais limpos, facilitando o entendimento futuro do software e em sua manutenção. Caso algum bug surgir, neste tipo de organização de software é mais fácil encontrá-lo e eliminá-lo, pois como as camadas são bem "isoladas"(focadas em funcionalidades específicas), fica relativamente simples descobrir em qual camada o bug está. A figura 3 mostra a arquitetura do sistema.

Figura 3 – Arquitetura do sistema utilizando o padrão de camadas



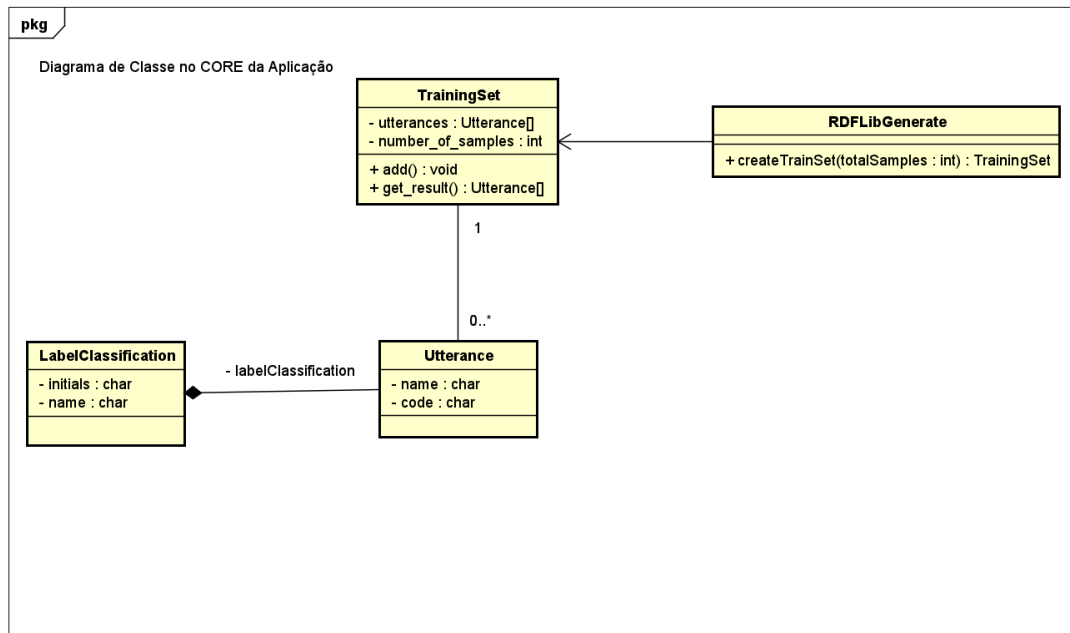
Fonte: Próprio Autor

Existe a camada de aplicação no frontend que se comunica com o backend através de requisições HTTP/REST JSON. Dentro do backend tem a camada de controller da aplicação para processar as requisições e gerar respostas. Ele se comunica com a camada de service, responsável pela lógica de negócio da aplicação e por se comunicar com as camadas mais internas do sistema, no caso o core e camada de dados.

2.3 Diagrama de Classe

A lógica de negócio fica localizado na parte central do Sistema (Core). As classes contidas nesse módulo são responsáveis por criar e determinar toda a estrutura de dataset gerado em rdf datasets para treinamento. A Figura 4 mostra o diagrama de classes com todas as classes que compõem esse módulo. O diagrama apresenta a estrutura lógica entre as classes, seus atributos e métodos e como as classes se comunicam entre si.

Figura 4 – Diagrama de classe da aplicação

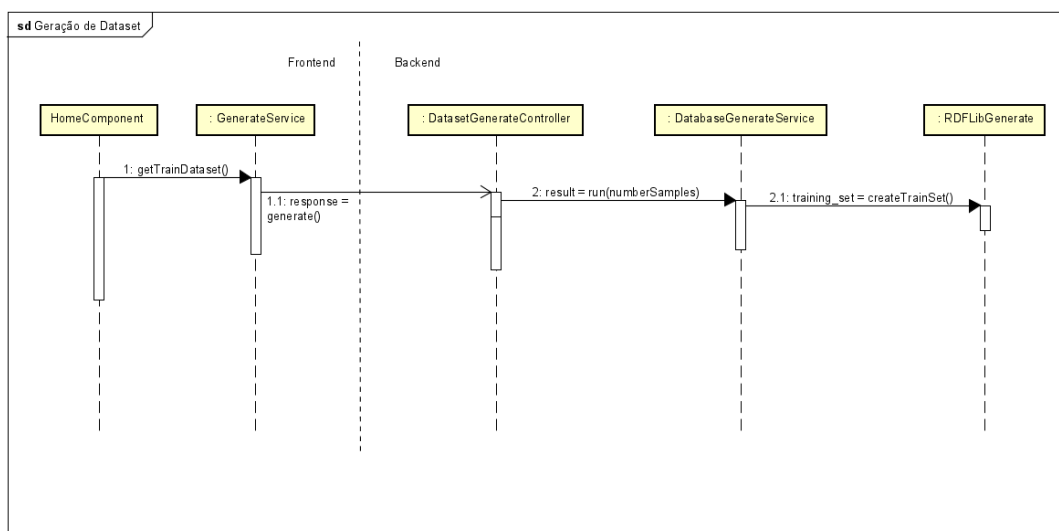


Fonte: Próprio Autor

2.4 Diagrama de Sequência

A figura 5 mostra um diagrama de sequência que ilustra a seqüência das mensagens entre objetos tanto do frontend quanto do backend na geração do dataset de forma automatizada.

Figura 5 – Diagrama de sequência na geração de datasets



Fonte: Próprio Autor

2.5 Testes

Os testes foram divididos em duas partes. No backend, foram implementados os testes automáticos na linguagem Python utilizando a biblioteca **unittest**. No backend se em três *endpoints*, então, para cada um, existe uma teste, abaixo descreve-se cada uma deles.

2.5.1 Checagem da Aplicação

Esse teste tem apenas o objetivo de verificar se a aplicação está rodando. Ele verifica o *endpoint*, a resposta e a mensagem de boas-vindas. A figura 6 ilustra como esse teste foi feito.

Figura 6 – Resultados dos testes automatizados de checagem da aplicação

```
# Importa a biblioteca de testes
import unittest

...
Test if apps is running
...
class TestHome(unittest.TestCase):
    ...
    Como todos os 3 casos de teste fazem um get na home "/"
    da aplicacao, defini-se a funcao setup. Ela e executada
    automaticamente sempre que o Pytest instancia a classe TestHome.
    A funcao setup e semelhante a um metodo construtor.
    ...

    def setUp(self):
        myApp = app.test_client()
        self.response = myApp.get('/')

    # Testa se a resposta e 200 ("ok")
    def test_get(self):
        self.assertEqual(200, self.response.status_code)

    # Testa se a nossa home retorna a string esperada
    def test_html_string_response(self):
        self.assertEqual("Hello, World! - This is application is running with <b>Flask Version: 2.2.2</b>",
                           self.response.data.decode('utf-8'))

    # Testa se o content_type da resposta da home esta correto
    def test_content_type(self):
        self.assertIn('text/html', self.response.content_type)
```

Fonte: Próprio Autor

2.5.2 Geração de Datasets

Esse teste verifica se os datasets com os enunciados estão sendo gerados. A figura 7 ilustra esse teste, o qual são três no total, o primeiro para verificar se o *endpoint* está funcionando e retornando o status 200, o segundo, verifica se gera datasets quando não se passa o número de amostras para serem geradas, no caso o valor padrão é 5, o qual é a resposta esperada e por último, é a geração de datasets com o parâmetro sendo passado.

2.5.3 Checagem de rótulos de classificação para combobox da interface

. Esse teste verifica se os rótulos de classificação estão sendo gerados pelo *endpoint*. Ele é necessário para utilização do filtro de rótulos na interface. A figura 8 ilustra esse

teste, tem-se que o primeiro verifica o *endpoint*, o segundo se o número de rótulos gerados é sempre três, o qual é o número de rótulos possíveis. E o último, verifica se é de fato cada um dos rótulos esperados estão sendo retornados.

Figura 7 – Resultados dos testes automatizados de checagem de rótulos de classificação

```
class TestGenerate(unittest.TestCase):
    """
    Como todos os 3 casos de teste fazem um get no generate "/generate"
    da aplicacao, define-se a funcao setUp. Ela e executada
    automaticamente sempre que o Pytest instancia a classe TestGenerate.
    A funcao setUp e semelhante a um metodo construtor.
    """

    def setUp(self):
        self.myApp = app.test_client()
        self.response = None

    # Testa se a resposta e 200
    def test_get(self):
        self.response = self.myApp.get('/generate')
        self.assertEqual(200, self.response.status_code)

    # Testa a requisicao sem passar os parâmetros, deve ser esperado a quantidade de exemplos gerados igual a
    # 5 (default)
    def test_get_without_parameter(self):
        self.response = self.myApp.get('/generate')
        data = json.loads(self.response.data.decode('utf-8'))
        self.assertEqual(5, data['total'])

    # Testa a requisicao passando algum parâmetro, deve ser esperado a quantidade de exemplos gerados igual ao
    # número passado na requisicao
    def test_get_with_parameter(self):
        total_samples = 25
        self.response = self.myApp.get('/generate/'+str(total_samples))
        data = json.loads(self.response.data.decode('utf-8'))
        self.assertEqual(total_samples, data['total'])
```

Fonte: Próprio Autor

Figura 8 – Resultados dos testes automatizados de checagem de rótulos de classificação

```
class TestLabelClassificationService(unittest.TestCase):
    """
    Como todos os 3 casos de teste fazem um get no labels "/labels"
    da aplicacao, define-se a funcao setUp. Ela e executada
    automaticamente sempre que o Pytest instancia a classe TestLabelClassificationService.
    A funcao setUp e semelhante a um metodo construtor.
    """

    def setUp(self):
        myApp = app.test_client()
        self.response = myApp.get('/labels')

    # Testa se a resposta e 200
    def test_get(self):
        self.assertEqual(200, self.response.status_code)

    # Testa a requisicao, deve ser esperado a quantidade de labels retornados igual a 3
    def test_get_result(self):
        data = json.loads(self.response.data.decode('utf-8'))
        self.assertEqual(3, len(data['data']))

    def test_get_result(self):
        data = json.loads(self.response.data.decode('utf-8'))
        expected = [LabelClassification('SELF EXPLANATORY', 'SE').__dict__(),
                    LabelClassification('FIRST TOPIC', 'FT').__dict__(),
                    LabelClassification('PREVIOUS TOPIC', 'PT').__dict__()]
        self.assertEqual(expected, data['data'])
```

Fonte: Próprio Autor

A figura 9 mostra os resultados dos testes automatizados.

Figura 9 – Resultados dos testes automatizados no backend

```
plugins: cov-4.0.0
collected 7 items

tests\test_generate_service.py ..
tests\test_home.py ...
tests\test_label_classification_service.py ..

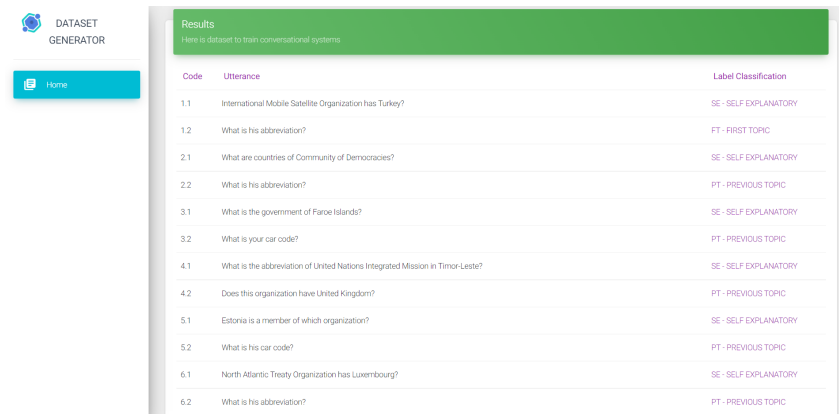
===== 7 passed in 10.65s =====
```

Fonte: Próprio Autor

Já no frontend, foi implementado um roteiro que explorasse todos os casos de uso da aplicação.

1. Gerar dataset informando o número de instâncias (Figura 10)
2. Gerar dataset sem informar o número de instâncias (Figura 11)
3. Gerar dataset quando o servidor não está respondendo (Figura 12)
4. Filtrar por enunciado (Figura 13)
5. Filtrar por rótulo do enunciado (Figura 14)
6. Filtrar por enunciado e rótulo (Figura 15)
7. Limpar Filtro (Figura 16)
8. Exportar para csv quando dataset é gerado (Figura 17)
9. Exportar para csv quando dataset não é gerado (Figura 18)

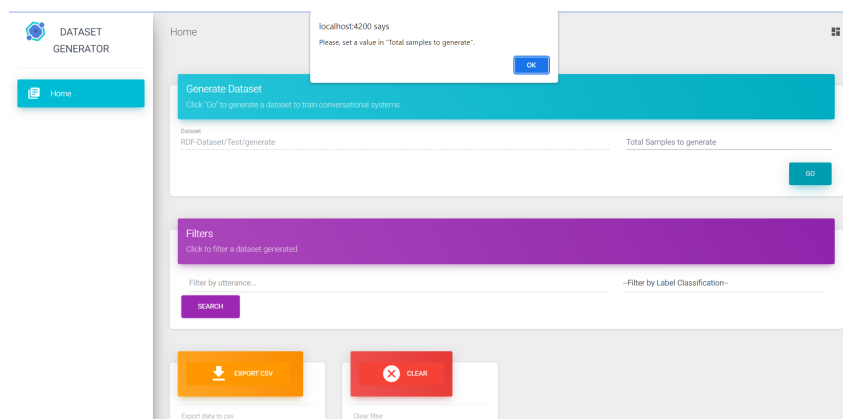
Figura 10 – Geração de Dataset informando o número de instâncias



| Code | Utterance | Label Classification |
|------|---|-----------------------|
| 1.1 | International Mobile Satellite Organization has Turkey? | SE - SELF EXPLANATORY |
| 1.2 | What is his abbreviation? | FT - FIRST TOPIC |
| 2.1 | What are countries of Community of Democracies? | SE - SELF EXPLANATORY |
| 2.2 | What is his abbreviation? | PT - PREVIOUS TOPIC |
| 3.1 | What is the government of Faroe Islands? | SE - SELF EXPLANATORY |
| 3.2 | What is your car code? | PT - PREVIOUS TOPIC |
| 4.1 | What is the abbreviation of United Nations Integrated Mission in Timor-Leste? | SE - SELF EXPLANATORY |
| 4.2 | Does this organization have United Kingdom? | PT - PREVIOUS TOPIC |
| 5.1 | Estonia is a member of which organization? | SE - SELF EXPLANATORY |
| 5.2 | What is his car code? | PT - PREVIOUS TOPIC |
| 6.1 | North Atlantic Treaty Organization has Luxembourg? | SE - SELF EXPLANATORY |
| 6.2 | What is his abbreviation? | PT - PREVIOUS TOPIC |

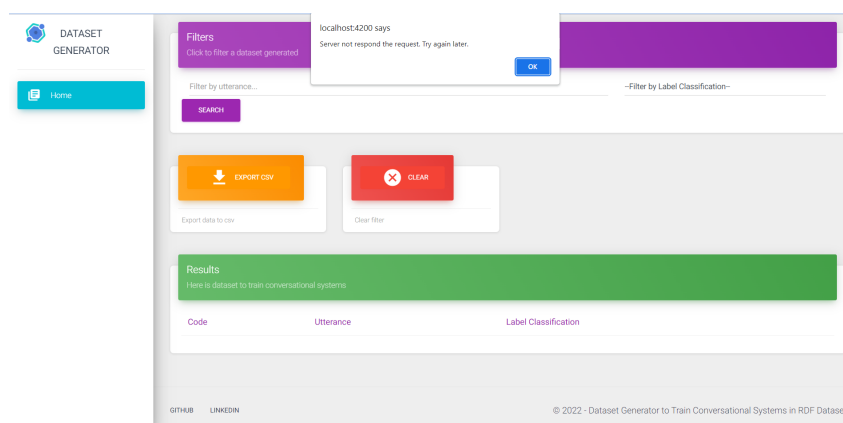
Fonte: Próprio Autor

Figura 11 – Geração de Dataset não informando o número de instâncias



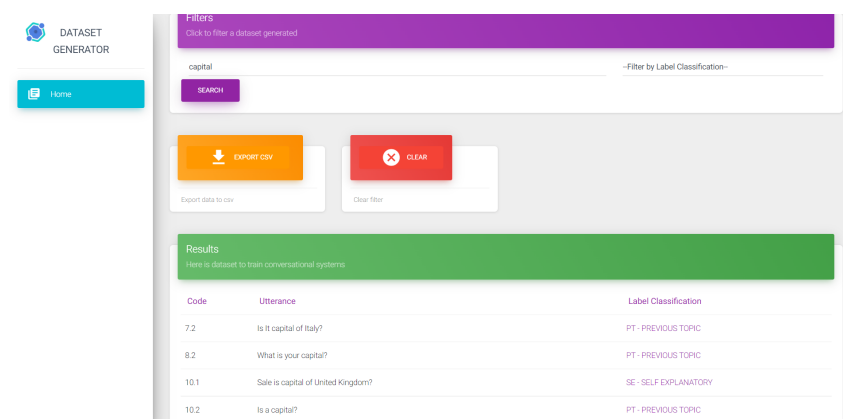
Fonte: Próprio Autor

Figura 12 – Geração de Dataset quando o servidor não responde



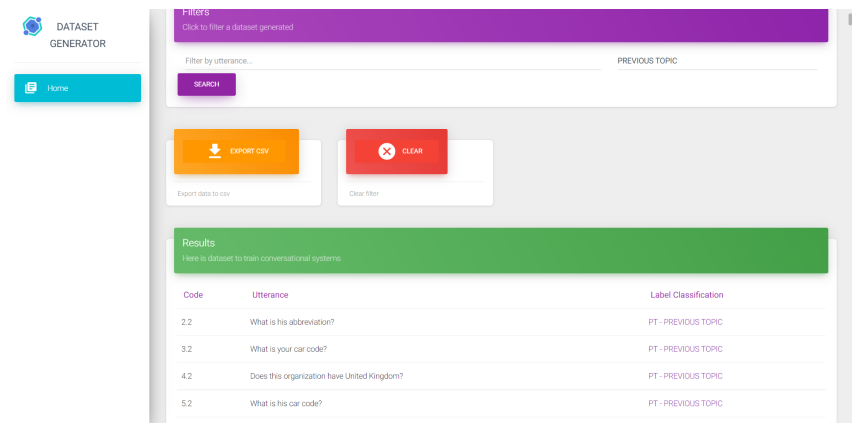
Fonte: Próprio Autor

Figura 13 – Filtrar por enunciado



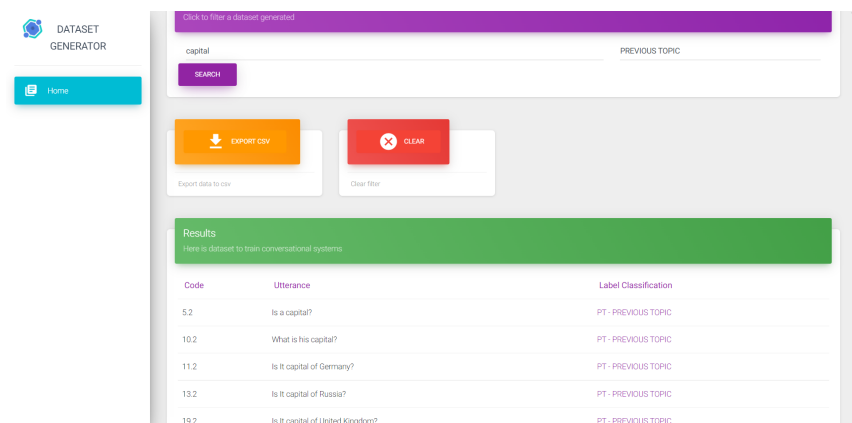
Fonte: Próprio Autor

Figura 14 – Filtrar por rótulo do enunciado



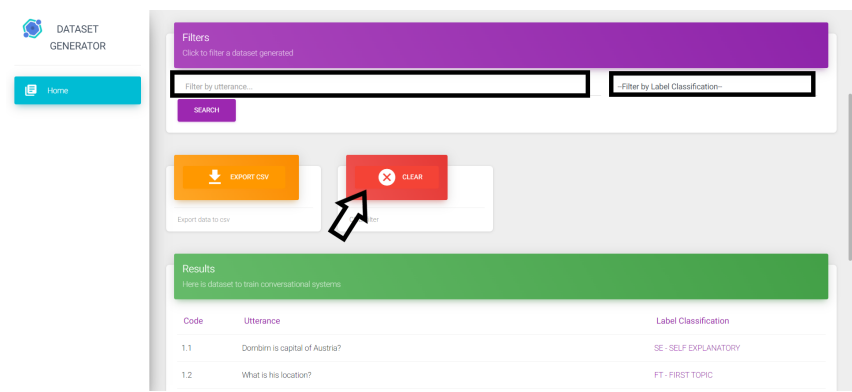
Fonte: Próprio Autor

Figura 15 – Filtrar por enunciado e rótulo



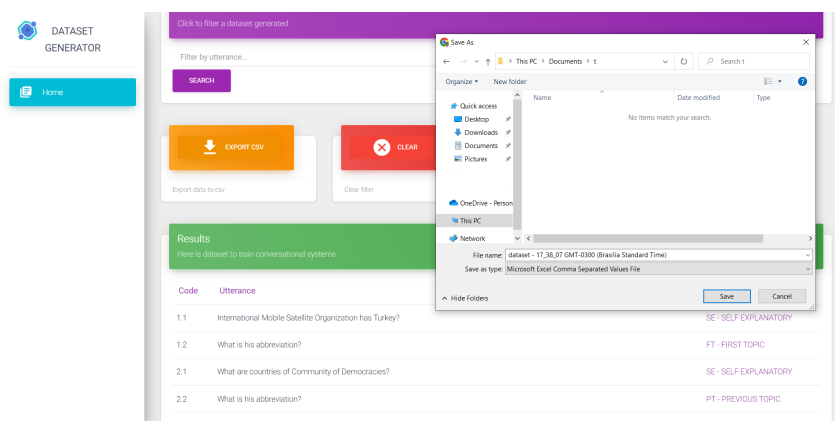
Fonte: Próprio Autor

Figura 16 – Limpar Filtro



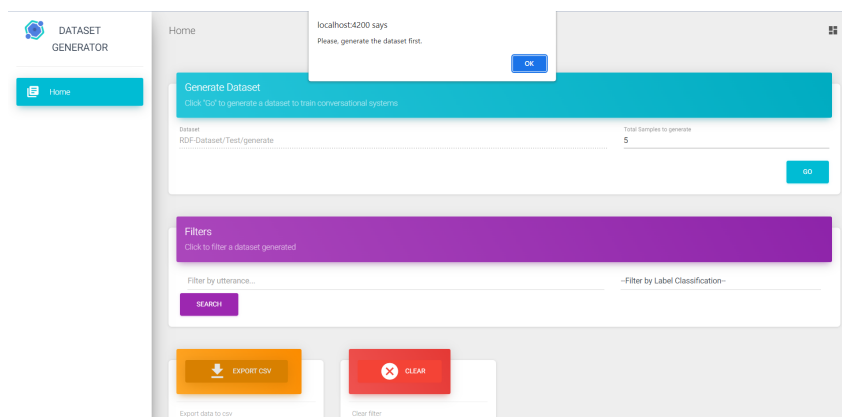
Fonte: Próprio Autor

Figura 17 – Exportar para CSV quando dataset é gerado



Fonte: Próprio Autor

Figura 18 – Exportar para CSV quando dataset não é gerado



Fonte: Próprio Autor

3 Manual do usuário

Este capítulo apresenta o manual do usuário. É demonstrado como utilizar o programa para gerar datasets automatizados sobre RDF datasets para treinar sistemas conversacionais de busca.

3.1 Requisitos necessários

Para instalação e execução é necessário possuir um ambiente com python 3.8 ou superior e o Nodejs. Todas as dependências do backend estão no arquivo *requirements.txt*.

3.2 Requisitos mínimos

Após a configuração, o frontend da aplicação deve ser carregada no endereço <http://localhost:4200/> e o backend, <http://localhost:5000/>. Para visualização e controle da interface recomenda-se possuir ao menos um mouse e teclado.

3.3 Instalação do Sistema

O usuário tem que baixar ou clonar o projeto do Github². Depois configurar o backend primeiramente, abrindo a pasta específica do projeto e instalar as dependências necessárias. Por fim, rodar o servidor da aplicação, caso queira rodar no modo *debug*, também é possível. Os comandos são apresentados na figura 19.

Figura 19 – Passos com os comandos para configurar o backend da aplicação

```
> git clone https://github.com/dudursn/dataset-generator.git
> cd dataset-generator/backend
> pip install -r requirements.txt
-- Executar sem debug
> flask run
-- Executar com debug
> flask --debug run
```

Fonte: Próprio Autor

Já no frontend, o usuário deve ir para pasta específica e rodar o comando para instalar as dependências. E por fim, rodar o comando para abrir a interface da aplicação. Os comandos são apresentados na figura 20.

² <https://github.com/dudursn/dataset-generator>

Figura 20 – Passos com os comandos para configurar o frontend da aplicação

```
> git clone https://github.com/dudursn/dataset-generator.git  
> cd dataset-generator/frontend  
> npm install  
> ng serve
```

Fonte: Próprio Autor

3.4 Apresentação do Sistema

A figura 21 apresenta a interface inicial do sistema. Para gerar o dataset, o usuário tem que informar um número entre 5 a 1000 de amostras que devem ser geradas pelo sistema. Depois, o usuário tem que clicar no botão "GO".

Figura 21 – Tela inicial do Sistema

Fonte: Próprio Autor

O sistema rola a página automaticamente para baixo onde aparece na interface os filtros e após alguns segundos os dados gerados são exibidos em uma tabela com três colunas (*Code*, *Utterance*, *Label Classification*). A figura 22 abaixo apresenta a interface com os dados gerados.

Figura 22 – Tela com os dados gerados

The screenshot shows the 'DATASET GENERATOR' interface. On the left is a sidebar with a 'Home' button. The main area has a 'Filters' section with a text input 'Filter by utterance...', a 'SEARCH' button, and a dropdown 'Filter by Label Classification'. Below are 'EXPORT CSV' and 'CLEAR' buttons. The 'Results' section shows a table with 3 rows of data.

| Code | Utterance | Label Classification |
|------|---|-----------------------|
| 1.1 | International Mobile Satellite Organization has Turkey? | SE - SELF EXPLANATORY |
| 1.2 | What is his abbreviation? | PT - FIRST TOPIC |
| 2.1 | What are countries of Community of Democracies? | SE - SELF EXPLANATORY |

Fonte: Próprio Autor

Os dados podem ser filtrados a partir dos campos do formulário "Filters", como se pode ver na figura 23. O usuário pode filtrar tanto por enunciado, informando no campo o texto ou selecionando na combobox por rótulo de classificação. Após informar os dados, o usuário clica no botão roxo "SEARCH". A figura 23 mostra os dados após o usuário filtrar por todos os enunciados contendo "capital" e por rótulo "PREVIOUS_TOPIC".

Figura 23 – Tela com os dados filtrados

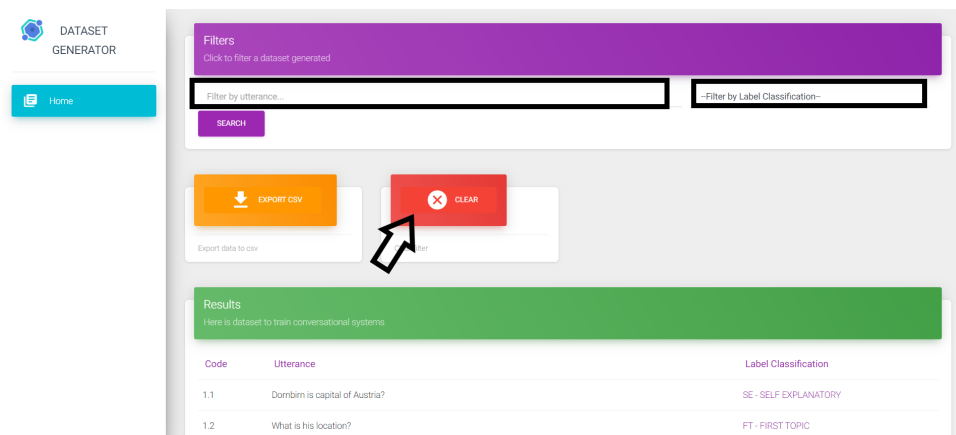
The screenshot shows the same interface as Figure 22, but with filters applied. The 'Filter by utterance' field contains 'capital' and the 'Filter by Label Classification' dropdown is set to 'PREVIOUS TOPIC'. The 'SEARCH' button is highlighted with a red arrow. The 'Results' table now shows 4 rows of data.

| Code | Utterance | Label Classification |
|------|---------------------------|----------------------|
| 5.2 | Is a capital? | PT - PREVIOUS TOPIC |
| 10.2 | What is his capital? | PT - PREVIOUS TOPIC |
| 11.2 | Is it capital of Germany? | PT - PREVIOUS TOPIC |
| 13.2 | Is it capital of Russia? | PT - PREVIOUS TOPIC |

Fonte: Próprio Autor

Se o usuário quiser voltar para o estado anterior ao aplicar os filtros, ele deve clicar no botão vermelho "CLEAR". A figura 24 mostra essa funcionalidade, onde os campos do filtro são reiniciados e os dados gerados são exibidos da forma original.

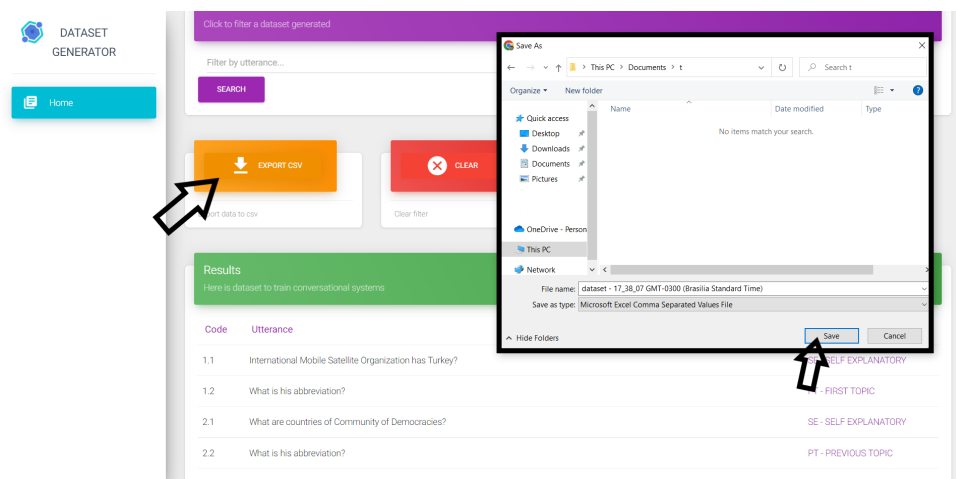
Figura 24 – Tela após clicar no botão CLEAR



Fonte: Próprio Autor

Por fim, o usuário tem a opção de exportar os resultados de cada dataset gerado, basta clicar no botão laranja "EXPORT CSV". Se o usuário estiver com o filtro aplicado, os dados que serão exportados são os dados filtrados. A figura 25 ilustra a funcionalidade de exportar o dataset de treino para csv.

Figura 25 – Tela para exportar os dados para csv



Fonte: Próprio Autor

A figura 26 ilustra o arquivo csv gerado.

Figura 26 – Arquivo CSV gerado do dataset de treinamento

```
code,utterance,labelClassification
"1.1","What are countries of United Nations Peacekeeping Force in Cyprus?","SE"
"1.2","Does this organization have Ukraine?","PT"
"2.1","What is location of Zonguldak?","SE"
"2.2","Is It capital of Ukraine?","PT"
"3.1","What is the abbreviation of United Nations High Commissioner for Refugees?","SE"
"3.2","Does this organization have Finland?","PT"
"4.1","What is the capital of Bulgaria?","SE"
"4.2","What is your car code?","PT"
"5.1","Bergisch Gladbach is capital of Germany?","SE"
"5.2","What is his location?","PT"
"6.1","What is location of Craiova?","SE"
"6.2","Is a capital?","PT"
"7.1","Isle of Man is a member of which organization?","SE"
"7.2","What is your government?","PT"
"8.1","What is the abbreviation of International Organization for Standardization?","SE"
"8.2","Does this organization have Vatican City?","PT"
"9.1","What is car code of Faroe Islands?","SE"
"9.2","What is his organization?","PT"
"10.1","What is location of Helsinki?","SE"
"10.2","Is a capital?","PT"
"11.1","What is the government of Croatia?","SE"
"11.2","What is your car code?","PT"
"12.1","What is car code of Lithuania?","SE"
```

Fonte: Próprio Autor

As principais funcionalidades da aplicação foram apresentadas nesta seção e o código presente no Github³ está bem comentado. Entretanto, caso tenha alguma dúvida, entre em contato pelo e-mail: roger.rsn@outlook.com.

³ <https://github.com/dudursn/dataset-generator>

Referências

MAY, W. *Information Extraction and Integration with Florid: The Mondial Case Study*. 1999. Disponível em: <<<http://dbis.informatik.uni-goettingen.de/Mondial>>>.

MELE, I. et al. Adaptive utterance rewriting for conversational search. *Information Processing & Management*, v. 58, n. 6, p. 102682, 2021. ISSN 0306-4573. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0306457321001679>>. Citado 3 vezes nas páginas 3, 4 e 6.

THOMAS, P. et al. Theories of conversation for conversational ir. *ACM Transactions on Information Systems*, v. 39, n. 39, 2021.