ASSISTENTE DE ATUALIZAÇÃO DE VERSÃO DE PROJETOS ANGULAR

Nathan Reikdal Cervieri, Luciana Pereira de Araújo Kohler – Orientadora

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

nathan@furb.br, lpa@furb.br

Resumo: *Este trabalho apresenta o desenvolvimento de uma ferramenta de conversão automática de código para aplicações Angular com o intuito de adaptar os projetos da versão 15 para 16. A ferramenta foi desenvolvida em C# para o servidor e HTML e typescript para a interface de usuário. O servidor utiliza uma estratégia de separação do código em linhas e utiliza regex para buscar e categorizar ações necessárias, assim como substituir parte das linhas quando possível. O trabalho apresenta o desenvolvimento da ferramenta, assim como suas funcionalidades e funcionamento. O resultado foi uma página feita em Angular que permite a conversão de código singular ou de pastas.*

Palavras-chave*: Angular. Conversão automática. Regex. C#. Typescript.*

1 INTRODUÇÃO

No cenário da internet é difícil determinar o impacto que páginas da web tem na experiência dos usuários. Tudo pode ser encontrado em páginas da web, desde conhecimentos diversos como a Wikipédia, ferramentas de pesquisa como o Google até websites de compra online como o Mercado livre e a Amazon. Websites são uma interface de comunicação com as várias facetas do mundo online e com isso se tem uma variedade de sistemas e projetos relevantes a seus domínios.

Para facilitar o desenvolvimento de páginas da web há vários *frameworks* de desenvolvimento. Um deles é o Angular, um *framework* orientado a componente que provê várias funcionalidades que auxiliam a criação de websites interativos, dinâmicos e escaláveis com esforço mínimo (Angular, 2022). De acordo com uma pesquisa feita pela plataforma StackOverflow (2022), Angular está entre as cinco tecnologias mais utilizadas em desenvolvimento web, com aproximadamente 20% das quase 60 mil repostas ao formulário.

O Angular, assim como outros *frameworks*, funciona em um sistema de versão no qual novas versões trazem novas funcionalidades e correção de inconsistências ou problemas de segurança (Angular, 2023b). A mais recente versão do Angular é a 17.0.0 no momento da escrita (Angular, 2023b). Essas atualizações também trazem alterações que as tornam parcialmente incompatíveis com código escrito previamente. Assim, se o desenvolvedor que utiliza o *framework* não realizar o esforço para atualizar versões do sistema com frequência pode aumentar consideravelmente o custo da correção (PMBOK, 2017).

A partir destas afirmações, este trabalho apresenta o desenvolvimento de uma ferramenta que auxilia na atualização de aplicações Angular de modo a diminuir o custo e tempo de alteração. O propósito é diminuir a propensão a erros que refatoração de larga escala pode trazer a um projeto. Por meio da categorização e alteração automática do código, como pode-se encontrar no site que cataloga alterações necessárias entre versões do Angular (Angular, 2023a), busca-se garantir que o sistema funcione entre a versão de origem e a versão de destino.

1.1 OBJETIVOS

O objetivo deste trabalho é simplificar o processo de atualização de código escrito na versão 15 para a 16 do *framework* Angular para diminuir o trabalho necessário através da automatização da análise do projeto e alterações a serem feitas.

Os objetivos específicos são:

a) classificar alterações necessárias para converter Angular da versão 15 para 16;
b) converter código através da divisão de linhas e *regex*;
c) orientar o desenvolvedor sobre mudanças apontadas que não foram possíveis de serem realizadas.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão discutidos os conceitos que fundamentarão o trabalho como um todo, começando pelo Angular (Angular, 2022) e seu guia de atualização de versão (Angular, 2023a); uma análise de ferramentas de atualização de código; e finalmente uma visão de testes de software.

2.1 ANGULAR

Framework de desenvolvimento é uma estrutura que pode ser usada para construir software ao agir como uma fundação para o desenvolvedor não ser obrigado a escrever código desnecessário do zero (Olawanle, 2022). O Angular é um *framework* baseado na linguagem de programação TypeScript (Angular, 2022) criado em 2010 que auxilia na construção de projetos web através de um conjunto robusto de ferramentas e de funcionalidades importantes para o desenvolvimento de aplicações como: roteamento; injeção de dependência; e vinculação de dados bidirecional (Miller, 2021). O Angular permite a construção de aplicações de pequeno e grande porte e facilita a criação de projetos escaláveis de forma rápida e consistente ao remover a responsabilidade de criação de componentes básicos e estrutura de comunicação entre esses componentes dos desenvolvedores (Angular, 2022; Miller, 2021).

De acordo com uma entrevista com o criador do Angular, o projeto começou como um trabalho alternativo na Google para facilitar o desenvolvimento interno em plataformas web, eventualmente sendo liberado para todos, não apenas desenvolvedores Google (Columbus, 2016). Atualmente, o Angular é utilizado por milhares de usuários, como indicado por uma pesquisa feita pelo website Stackoverflow (2022), Angular é a quinta tecnologia ou *framework* mais utilizado para desenvolvimento web, totalizando aproximadamente onze mil seleções no questionário. Entre as companhias que utilizam o Angular pode-se encontrar a Google, Microsoft, Netflix e Paypal. (Quinn, 2022).

Entre as informações disponíveis para o *framework* Angular se encontra o Angular Update Guide (Angular, 2023a). Essa página da web permite ao usuário indicar quaisquer duas versões do Angular para receber um guia listando as alterações gerais necessárias para adequar da versão mais antiga para mais nova. O guia também permite escolher entre uma de três complexidades (Angular, 2023a). Também é possível escolher entre algumas dependências que interferem na adequação entre versões, sendo elas: o uso de `ngUpgrade` para combinar AngularJS e Angular, o uso de Angular Material e se o usuário utiliza sistema operacional Windows (Angular, 2023a) para assim permitir atualizar projetos que estão em situações diversas.

2.2 FERRAMENTAS DE ATUALIZAÇÃO DE VERSÕES DE CÓDIGO

Uma prática da programação é a utilização de versionamento ou a ideia de se ter diversas versões do app ordenadas de maneira cronológica. Uma forma é o Semantic Versioning (semver) que apresenta a maneira mais comum de versionamento, seguindo o formato Major.Minor.Patch. Um exemplo para esse formato é a própia versão do semver que é atualmente a versão 2.0.0, liberada em 2013. Há três partes na formulação da versão, sendo elas: *major*, *minor* e *patch*. A alteração de versão *major* é feita quando se tem uma alteração que quebra funcionalidades na aplicação ou biblioteca. *Minor* é quando uma função é adicionada de maneira retrocompatível e *patch* para quando se tem uma alteração que para correção de problemas, ainda retrocompatível (Semver, [2013]).

Versões *major*, que causam quebras, são o que torna necessário a alteração do código entre versões de linguagens ou *frameworks* de código. Em alguns casos é necessário realizar essas mudanças de maneira manual, seguindo um guia de instruções ou uma lista de mudanças de quebra, como pode ser encontrando no Angular Update Guide (Angular, 2023a) ou o ngMigration Assistant (Olson, 2019). Para algumas linguagens de programação, existem ferramentas para auxiliar nesse processo, realizando alterações automatizadas como o .NET Upgrade Assistant (.Net, 2021) para a linguagem .NET ou o Python2to3 (Python, 2012) para a linguagem Python.

Para as ferramentas que atuam diretamente em código é feito um processo de busca no código-fonte com o intuito de buscar locais que devem ser alterados e como fazer essas alterações. Todas as ferramentas citadas no parágrafo anterior (Olson, 2019; .Net, 2021; Python, 2012) utilizam um tipo de expressão regular (*regex*) para encontrar as partes do código inválidas e depois aplicam um código de substituição simples, ou criam um tipo de relatório para informar que uma alteração é necessária. Expressões regulares são padrões usados para buscar um pedaço de texto em um texto maior (MDN, 2023) em várias linguagens de programação. Friedl (2006) diz que expressões regulares são a chave para processamento de texto flexível, potente e eficiente.

2.3 TESTES DE SOFTWARE

De acordo com Dias Neto (2007), “teste de software é o processo de execução de um produto para determinar se ele atingiu suas especificações...”. Testes no desenvolvimento em cascata são feitos de maneira manual, mas no desenvolvimento ágil são feitos de maneira automatizada (Valente, 2022).

Métodos como o Test-Driven Development (TDD) aplicam uma prática em que o teste automatizado é escrito “... pelo desenvolvedor, mesmo antes que as funcionalidades sejam implementadas.” (Campopiano, 2022). Crispin e Gregory (2009) afirmam que esses testes unitários permitem ao programador escrever código com segurança que não causará impactos inesperados ao sistema. Em uma apresentação feita em Kirkland, Chirila (2014) apresentou que na empresa Google eles buscam ter uma “cobertura de teste” (quantidade de linhas testadas por teste unitário) de aproximadamente 85% de todo código, mas isso nem sempre é possível, com a quantidade sendo variável com base na linguagem de programação do sistema.

Existem cinco tipos de teste de software definidos por Dias Neto (2007, Apud ROCHA et al 2001): unidade; integração; sistema; aceitação e regressão. Teste de unidade ou teste unitário "tem por objetivo explorar a menor unidade de projeto, [...] os métodos dos objetos ou mesmo pequenos trechos de código" (Dias Neto, 2007). Teste de integração "visa provocar falhas associadas às interfaces entre os módulos..." (Dias Neto, 2007). Teste de sistema executa e "avalia o software em busca de falhas por meio da utilização do mesmo..." (Dias Neto, 2007). Teste de aceitação "são realizados geralmente por um restrito grupo de usuários finais do sistema" (Dias Neto, 2007) dessa maneira agindo como um teste de sistema em geral. Teste de regressão é relevante após a entrega e durante o tempo de atualização ao "aplicar a cada nova versão do software [...] todos os testes que já foram aplicados nas versões [...] anteriores do sistema" (Dias Neto, 2007).

2.4 TRABALHOS CORRELATOS

Esta subseção apresenta algumas ferramentas comercias com finalidades semelhantes ao apresentado neste trabalho. Entre as ferramentas apresentadas estão o .NET Upgrade Assistant por .NET (2021) escrito no Quadro 1. O Quadro 2 apresenta a ferramenta de NgMigration Assistant de Olson (2018). Finalmente, o Quadro 3 descreve a ferramenta comercial para a linguagem Python: 2to3 (PYTHON, 2012).

Quadro 1 – Trabalho Correlato 1

Referência	.NET (2021)
Objetivos	Converter projetos .NET entre versões e *frameworks* .NET diferentes
Principais funcionalidades	Analisar projetos para definir compatibilidade de atualização incluindo: dependência de bibliotecas, referências, *framework*, pacote e suporte para APIs. Atualiza código de maneira automatizada e se não for possível é necessário ação do usuário.
Ferramentas de desenvolvimento	Ferramenta comercial
Resultados e conclusões	Ferramenta comercial

Fonte: elaborado pelo autor, 2023.

Quadro 2 – Trabalho Correlato 2

Referência	Olson (2018)
Objetivos	Demonstrar alterações necessárias para converter código de AngularJS para Angular 2
Principais funcionalidades	Escanear e criar uma lista de alterações para converter uma aplicação Angular JS. Contar linhas de código para dar uma estimativa de tempo para aplicar alterações.
Ferramentas de desenvolvimento	Ferramenta comercial
Resultados e conclusões	Ferramenta comercial

Fonte: elaborado pelo autor, 2023.

Quadro 3 – Trabalho Correlato 3

Referência	Python (2012)
Objetivos	Ler código em Python 2 e aplicar atualizadores para alterar código para Python 3
Principais funcionalidades	Converter arquivos individuais utilizando linha de comando e permite adicionar opções e configurações ao comando para alterar comportamento.
Ferramentas de desenvolvimento	Ferramenta comercial
Resultados e conclusões	Ferramenta comercial

Fonte: elaborado pelo autor, 2023.

3 DESCRIÇÃO DA FERRAMENTA

Esta seção está dividida em duas subseções. A subseção 3.1 apresenta a arquitetura e estrutura do servidor. A subseção 3.2 apresenta casos de uso e funcionalidades da interface de usuário.

O Quadro 4 apresenta os Requisitos Funcionais (RF) da ferramenta e o Quadro 5 apresenta os Requisitos Não Funcionais (RNF).

Quadro 4 – Requisitos funcionais da ferramenta.

Requisitos funcionais da ferramenta.
RF01: permitir escrever código para converter;
RF02: inserir vários arquivos para conversão.

RF03: alterar arquivos de maneira automatizada.
RF04: demonstrar arquivo após alteração do servidor.
RF05: mostrar as alterações manuais necessárias independente de arquivo.
RF06: mostrar as alterações manuais necessárias dependente de arquivo.
RF07: apontar mudanças necessárias para atualização entre versões 15 e 16 do Angular.

Fonte: Elaborado pelo autor, 2023.

Quadro 5 – Requisitos não funcionais da ferramenta

Requisitos não funcionais da ferramenta.
RNF01: a interface com o usuário deve ser desenvolvida em Angular (Requisito Não Funcional - RNF);
RNF02: o servidor de análise e substituição deve ser desenvolvido em C# (RNF);
RNF03: guardar informação para conversão como objeto estático (RNF);
RNF04: utilizar regex para buscar pontos de alteração (RNF);
RNF05: utilizar Angular Update Guide (Angular, 2023a) como base para alterações (RNF).

Fonte: Elaborado pelo autor, 2023.

A ferramenta é dividida em duas partes necessárias para o funcionamento. A primeira parte é um servidor C# que processa e categoriza o código, assim como realiza alterações automáticas e gera a lista de alterações manuais necessárias. A segunda parte é um projeto Angular que server como interface de usuário e permite escrever e converter códigos singulares ou pastas inteiras.

3.1 SERVIDOR

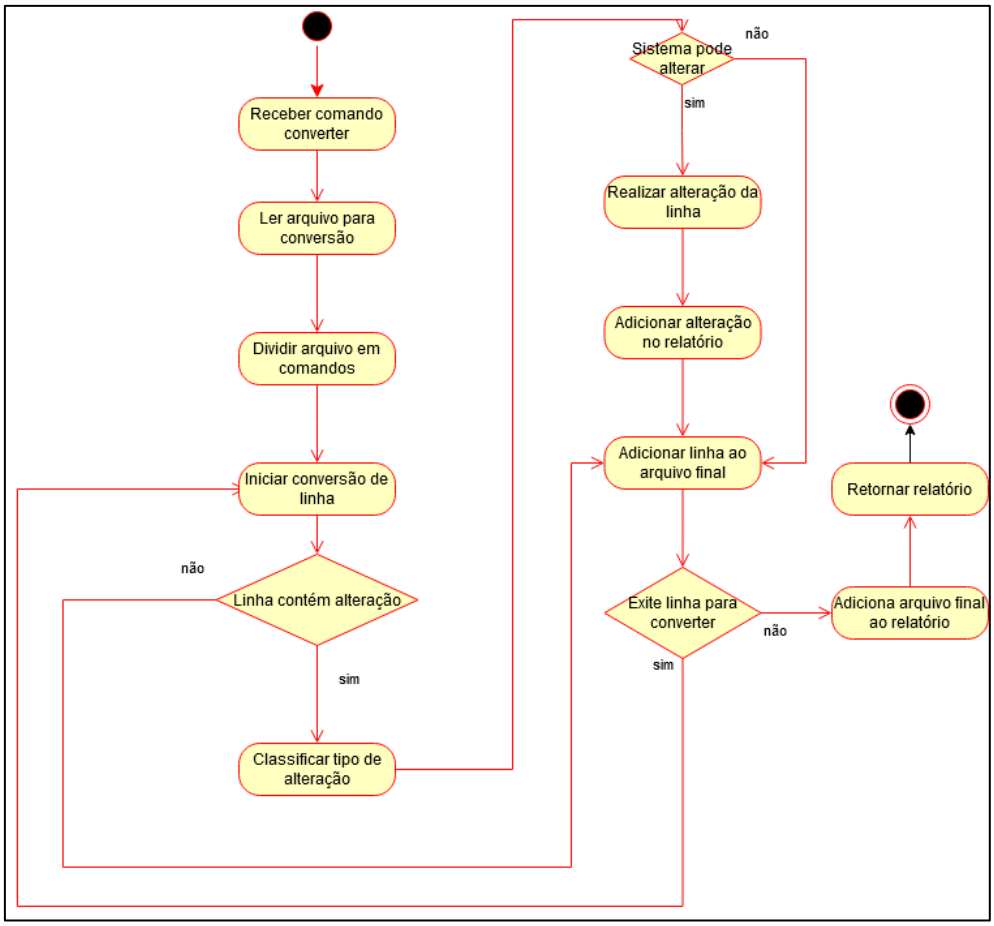
O servidor da ferramenta foi desenvolvido utilizando o Microsoft Visual Studio 2022 com a linguagem C# no *framework* .NET 6.0 e AspNetCore (.Net, 2023). A arquitetura do projeto C# usa como base o conceito de Domain Driven Design (DDD). DDD é um *framework* de desenvolvimento de software que junta a área de design de código e o domínio de negócio que o código controla e representa.

Seguindo o padrão do DDD, a arquitetura de arquivos está dividida em 5 projetos C#. O projeto de API que contém as classes *controller*, cuja função é prover uma interface de comunicação HTTP utilizando a ferramenta ASP.NET Core. O projeto de *Application* para lidar com a classe de serviço que orquestra o processamento requisitado pelo *controller*. O projeto de *Domain* que especifica os modelos a serem utilizados pelos outros projetos. O projeto de *Infra*, que manipula as informações estáticas necessárias para o funcionamento da ferramenta e finalmente um projeto de teste.

O projeto de teste foi integral para o desenvolvimento, pois foi utilizada a metodologia de Test Driven Development (TDD). O TDD é uma prática de desenvolvimento baseado na ideia contraditória do desenvolvedor escrever testes antes de escrever o código da aplicação (BECK, 2022). Essa forma de desenvolvimento garante que ao criar novas funcionalidades ao sistema, já serão escritos casos de teste que auxiliarão na manutenção futura, pois ao alterar uma classe ou método a funcionalidade esperada do mesmo é confirmada pelos testes.

O processo de conversão de código em si é composto por nove passos diferentes que podem ser vistos na Figura 1. Primeiramente é recebido o objeto para conversão, depois o arquivo recebido é interpretado para uma `string`, após a interpretação é feito a conversão do arquivo em si e a geração do relatório e finalmente o servidor retorna o arquivo convertido junto ao relatório. A conversão, que é a seção principal, está dividida em três passos notáveis: “Dividir arquivo em comandos”, “Classificar tipo de alteração” e “Realizar alteração da linha”.

Figura 1 – Diagrama de fluxo do servidor



Fonte: Elaborado pelo autor, 2023.

No passo `Dividir arquivo em comandos` é determinado como comando qualquer linha de texto antes de uma quebra de linha. Para esse trabalho utiliza-se a quebra de linha devido a um comportamento da linguagem JavaScript e por consequência typescript de não necessitar um ponto e vírgula para executar um comando, apenas necessita uma quebra de linha. O código que pode ser visto na `linha 46` no Quadro 6Quadro 6 demonstra a quebra das partes, `Regex.Split(fileString, "\r\n|\n|\r")`, assim como o *loop* de conversão na `linha 52` e a adição da linha ao arquivo na `linha 54`.

Quadro 6 – Métododo de separação no `ConverterService`

```
43  public Report ConvertAngularFile(Stream fileToConvert, AngularVersionEnum
    versionFrom = AngularVersionEnum.Angular14, AngularVersionEnum versionTo =
    AngularVersionEnum.Angular15)
44  {
45      var fileString = fileToConvert.ReadStreamToEnd();
46      var separatedTsFileInLines =  Regex.Split(fileString, "\r\n|\n|\r");
47      var reportBuilder = new ReportBuilder(versionFrom, versionTo);
48
49      var versionChanges =
    versionChangeRepository.GetDynamicChangesFromTo(versionFrom, versionTo);
50
51      var finalFile = new StringBuilder();
52      foreach (var line in separatedTsFileInLines)
53      {
54          finalFile.AppendLine(ConvertAngularLine(line, versionChanges,
    reportBuilder));
55          reportBuilder.IncrementLine();
56      }
57
58      reportBuilder.AddFinishedFile(finalFile.ToString());
59      return reportBuilder.Build();
60  }
```

Fonte: Elaborado pelo autor, 2023.

Para o próximo passo, `Classificar tipo de alteração`, é iterado pela lista de alterações cadastradas entre versão original e final, verificando se a linha contém algum dos termos chave para classificar o tipo de alteração. O código que pode ser visto na `linha 80` no Quadro 7, `line.IsNotMatchFor(versionChange.ChangeFinder)`, executa uma busca por *regex* na linha, buscando palavras chave que verificam se a alteração é relevante.

Quadro 7 – Parte do método de conversão de versão

```
62  public string ConvertAngularLine(string lineToConvert, IEnumerable<VersionChange>
    versionChangeList, ReportBuilder reportBuilder)
63  {
64      if (string.IsNullOrWhiteSpace(lineToConvert))
65      {
66          return string.Empty;
67      }
68
69      var finalLine = lineToConvert;
70      foreach (var versionChange in versionChangeList)
71      {
72          finalLine = HandleVersionChange(finalLine, versionChange, reportBuilder);
73      }
74
75      return finalLine;
76  }
77
78  private static string HandleVersionChange(string line, VersionChange versionChange,
    ReportBuilder reportBuilder)
79  {
80      if (line.IsNotMatchFor(versionChange.ChangeFinderRegexString))
81      {
82          return line;
83      }
84
85      return ApplyVersionChange(line, versionChange, reportBuilder);
86  }
```

Fonte: Elaborado pelo autor, 2023.

Após a classificação no passo `Realizar alteração da linha` é feita uma divisão no fluxo dependendo do tipo da alteração. No momento existem 2 fluxos, um quando não é possível fazer a alteração encontrada, em que o sistema adiciona no relatório e outra na qual o sistema faz correção de local de importação de dependência. No Quadro 8 é possível ver dois métodos de substituição, o primeiro na `linha 48` que demonstra apenas uma busca e substituição por *regex* e o segundo na `linha 55` que demonstra uma busca e substituição com adição de uma nova linha de código.

Quadro 8 – Código de substituição de linha

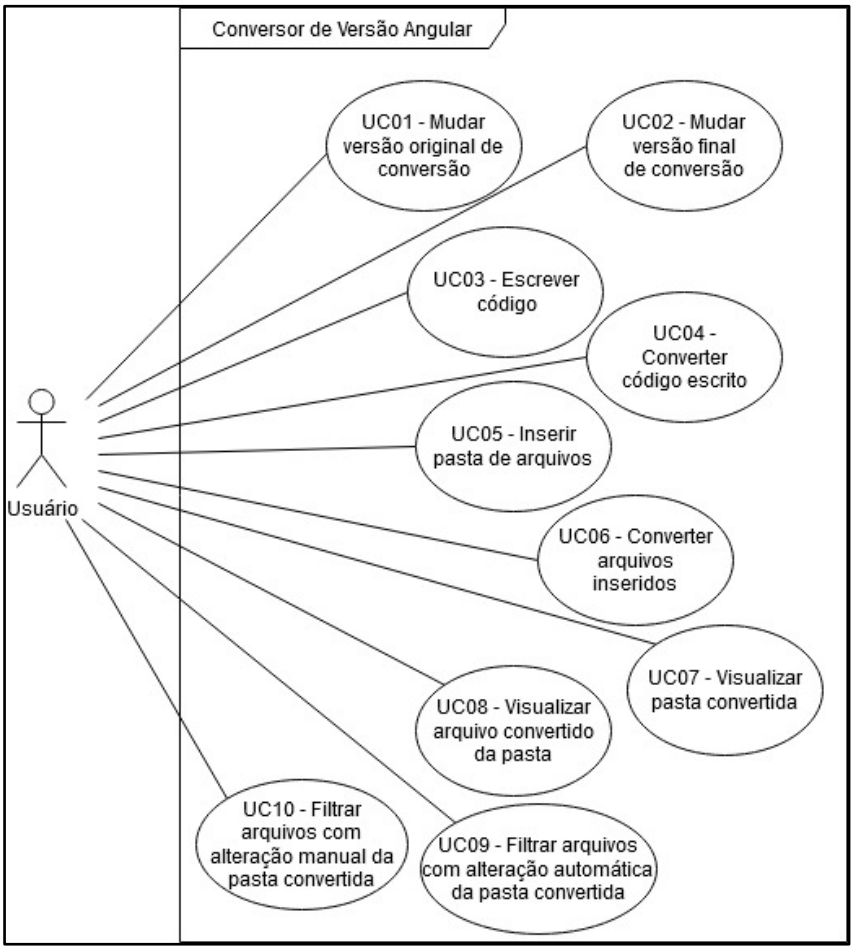
```
46  private static string HandleFindReplaceImportTypeReplace(string line, FindReplace
    findReplace)
47  {
48      var regex = new Regex(findReplace.WhatReplaceRegex);
49      return regex.Replace(line, findReplace.ReplaceText);
50  }
51
52  private static string HandleFindReplaceImportTypeReplaceAndNewLine(string line,
    FindReplace findReplace)
53  {
54      var regex = new Regex(findReplace.WhatReplaceRegex);
55      var replacedLine = regex.Replace(line, findReplace.ReplaceText);
56
57      var finalLineBuilder = new StringBuilder();
58      finalLineBuilder.AppendLine(replacedLine);
69      finalLineBuilder.Append(findReplace.NewLine);
70
71      return finalLineBuilder.ToString();
72  }
```

Fonte: Elaborado pelo autor, 2023.

3.2 INTERFACE

A interface de usuário da ferramenta foi desenvolvida utilizando o Visual Studio Code com a linguagem Typescript junto com HTML utilizando o *framework* Angular. Na Figura 2 é possível ver um diagrama de casos de uso que detalha as ações possíveis do usuário do conversor a partir da tela do projeto.

Figura 2 – Diagrama de casos de uso

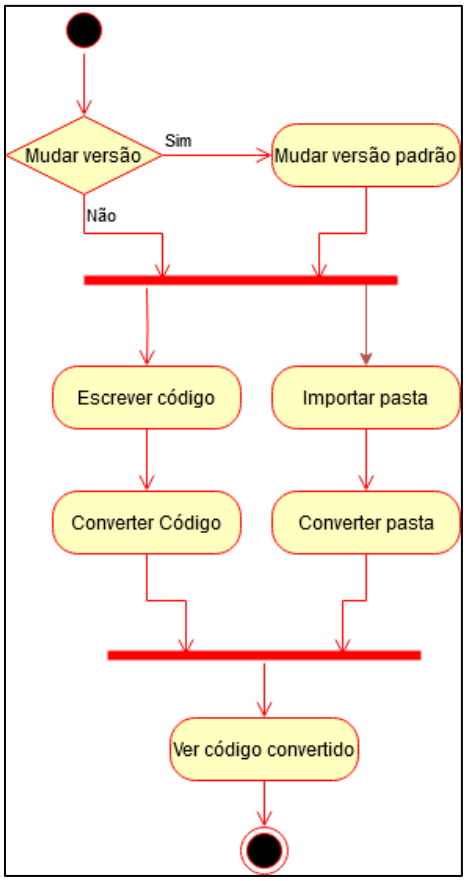


Fonte: Elaborado pelo autor, 2023.

O caso de uso `UC01 – Mudar versão original de conversão` permite ao usuário alterar a versão de origem da conversão do código da versão padrão, que atualmente é a versão 14, enquanto o caso de uso `UC02 – Mudar versão final de conversão` permite alterar a versão final da conversão, que atualmente é a versão 15, porém, como o sistema não está preparado para aceitar conversão entre outras versões, essa função é preparação para versões futuras da ferramenta. O `UC03` se refere a parte que permite escrever código de maneira manual para conversão, enquanto o `UC04` é a ação de converter o código escrito para a versão mais atual. O `UC05` permite ao usuário selecionar uma pasta de arquivos para converter componentes específicos ou projetos. Ao selecionar o `UC05` o sistema filtra quais arquivos usam a extensão `.ts` para apenas converter arquivos *typescript*. Após o `UC05` é possível iniciar a conversão dos arquivos através da `UC06` e visualizar o progresso através da `UC07`. Após a conversão de cada arquivo pode-se ver o novo arquivo a partir do `UC08`, assim como filtrar a pasta pelas alterações automáticas ou manual na `UC09` ou `UC10`.

Na Figura 3 é possível ver o fluxo mapeado para as ações do usuário acessando a interface. O passo `mudar versão padrão` se refere a ambos o `UC01` e o `UC02` e é opcional se o usuário tiver interesse em alterar a versão de conversão. Após esse primeiro passo, existem dois fluxos principais, um a partir da conversão de código escrito, que pode ser copiado de um arquivo existente e um que se inicia a partir de inserir uma pasta para conversão. Após ambos os fluxos é possível ver o código final.

Figura 3 – Diagrama de atividade da interface do usuário

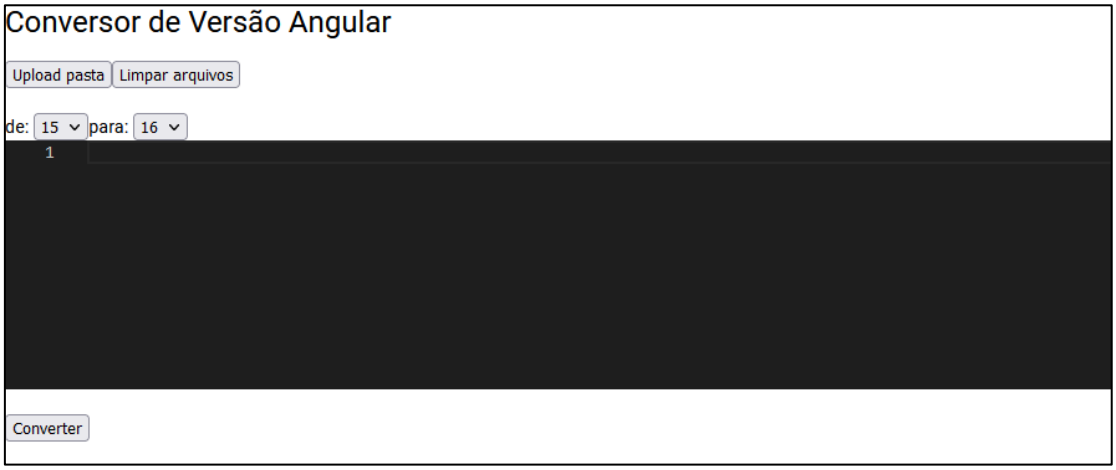


Fonte: Elaborado pelo autor, 2023.

3.3 UTILIZAÇÃO DA FERRAMENTA

A **Erro! Fonte de referência não encontrada.** demonstra a interface de usuário no estado inicial. Para realizar a conversão de código existem dois fluxos principais que podem ser vistos na Figura 3 da seção **Erro! Fonte de referência não encontrada.Erro! Fonte de referência não encontrada.**, o início dos fluxos são representados pelo botão de `Converter` na parte inferior da **Erro! Fonte de referência não encontrada.** para a conversão de arquivo individual e o botão `Upload pasta` na parte superior esquerda da mesma figura para conversão de pasta.

Figura 4 – Interface de usuário do conversor

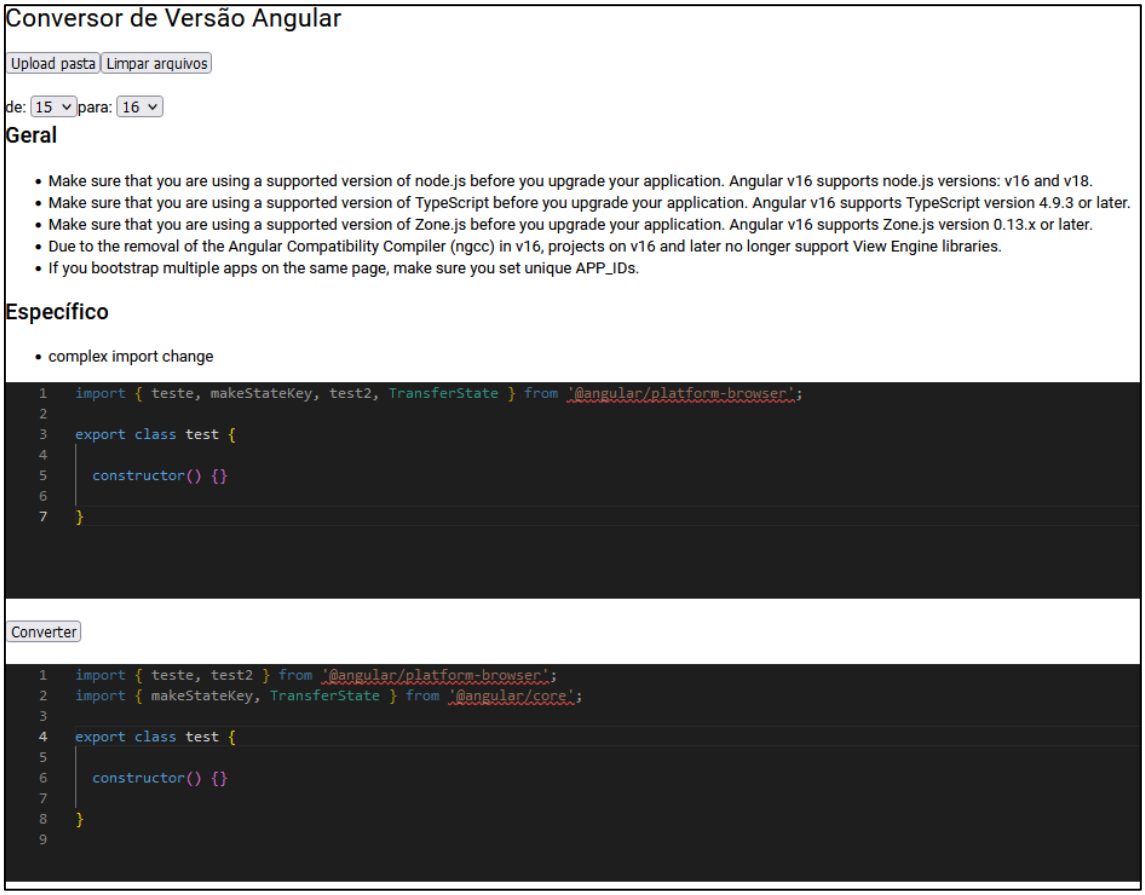


Fonte: Elaborado pelo Autor, 2023.

Seguindo o fluxo de conversão de arquivo individual, a **Erro! Fonte de referência não encontrada.** demonstra a conversão de um arquivo com as alterações gerais indicando as alterações a serem aplicadas antes da conversão do projeto para a versão de destino, simbolizado pelo rótulo `geral` e a partir do rótulo `Específicos` se refere ao que foi feito especificamente para o arquivo selecionado. Na conversão é possível ver na primeira linha da figura a diferença

entre o código escrito e o alterado, a ação da conversão automática nas importações de dependência, que o sistema chama de *Complex import change*.

Figura 5 – Fluxo de conversão de arquivo singular



Fonte: Elaborado pelo autor, 2023.

Outro exemplo tem-se na **Erro! Fonte de referência não encontrada.** que demonstra o processo de conversão após converter a pasta como um todo. Após o usuário selecionar a pasta, o sistema lista a quantidade de arquivos com a extensão do typescript e disponibiliza o botão de converter esses arquivos. A listagem de arquivos é mostrada no lado direito da interface onde se encontram os arquivos que foram convertidos e um botão para visualizar as alterações, assim como botões para filtrar a partir do tipo da alteração.

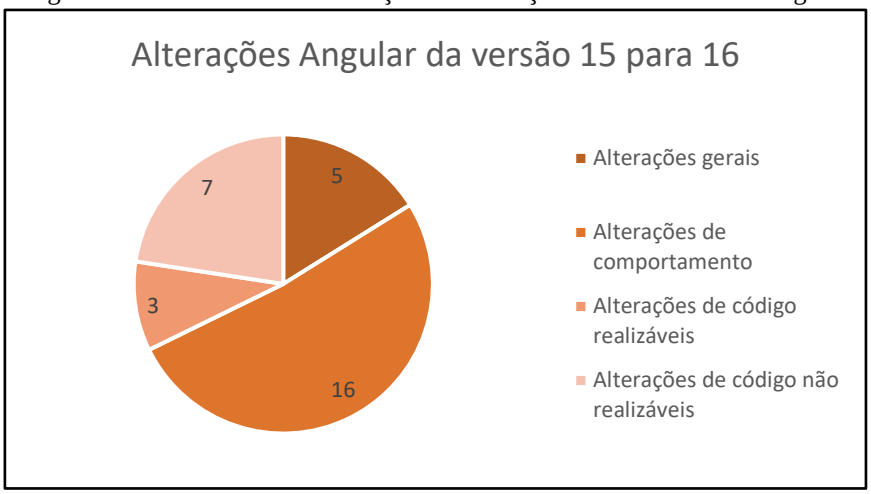
Figura 6 – Fluxo de conversão de pasta

Fonte: Elaborado pelo autor, 2023.

Entre as alterações indicadas por Angular (2023a) para conversão entre as versões 15 e 16 em uma aplicação avançada, podem ser encontradas 31 alterações. Essas 31 alterações foram classificadas em 4 categorias: alterações gerias; alterações de comportamento; alterações de código realizáveis; e alterações de código não realizáveis. Alterações gerais

se refere às alterações que devem ser aplicadas para todos os projetos para qual se tem interesse de atualizar, como rodar o comando de atualização de dependências e garantir a versão de node e typescript, contendo 5 das 31 alterações ou aproximadamente 16% das alterações. Alterações de comportamento são componentes cujo comportamento padrão foi alterado, logo, é necessário a ação do usuário e refatoração do código como um todo, como remoção de objetos ou métodos obsoletos, mudança de comportamento de métodos ou alteração em modelos, que compõem um total de 16, ou aproximadamente 51% das alterações. Alterações de código realizáveis são as que o projeto consegue realizar de maneira automatizada, sem requerer ação do usuário de qualquer maneira, sendo elas alterações no local de importação de classes, dando um total de 3 alterações na listagem ou aproximadamente 10%. Finalmente, alterações de código não realizáveis são alterações de código que não envolvem refatoração ou alteração do comportamento, mas não podem ser feitas de maneira automatizada, como utilização de uma classe *factory* ao invés de um construtor ou adição e/ou remoção de parâmetros de um método, resultando em 7 alterações ou aproximadamente 23% das alterações. O gráfico indicado na **Erro! Fonte de referência não encontrada.** demonstra a proporção de cada categoria de alteração.

Figura 7 – Gráfico com classificação das alterações entre versões do Angular



Fonte: Elaborado pelo autor, 2023.

4 RESULTADOS

O desenvolvimento deste trabalho permitiu a criação de uma ferramenta de conversão de código que converte arquivos individuais e pastas, também indica ao usuário quais alterações necessitam de alterações manuais ou refatoração. Ao comparar a ferramenta com os trabalhos apresentados na seção 2.4 é possível perceber resultados semelhantes e algumas diferenças. A ferramenta desenvolvida é única por ser a única capaz de fazer conversão automática de sistemas Angular e apresentar uma interface de usuário ao não realizar interações através de linha de comando. Contudo, a ferramenta por Olson (2018) é a única que demonstra uma estimativa de quantidade de tempo para resolver dependências, entretanto, não realiza nenhuma conversão de maneira automatizada, diferente das ferramentas de .NET (2021) e Python (2012). Todas as ferramentas demonstram as alterações que necessitam ser feitas de maneira manual após processar o projeto ou arquivos. O Quadro 9 demonstra essas informações resumidamente.

Quadro 9 – Comparação entre trabalhos correlatos.

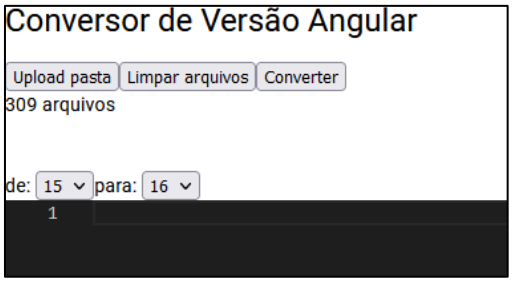
Características	**Este**	**.NET (2021)**	**Olson (2018)**	**Python (2021)**
Tipo de código	Angular	.NET	AngularJS	Python 2
Converte código automaticamente	Sim	Sim	Não	Sim
Relata alterações	Sim	Sim	Sim	Sim
Interface de usuário	Sim	Não	Não	Não

Fonte: Elaborado pelo autor, 2023.

4.1 APLICAÇÃO DA FERRAMENTA

Com a intenção de testar a ferramenta foi utilizado um repositório público no Github da biblioteca AngularFire (AngularFire, 2023), que é uma biblioteca de integração entre o *framework* Angular e o *Software Development Kit* (SDK) Firebase para Javascript. Para iniciar o processo de conversão, primeiramente é utilizada a função de upload de pasta da ferramenta, como pode ser visto na Figura 8, com isso a ferramenta identifica 309 arquivos `.ts` para conversão.

Figura 8 – Inserção de pasta na ferramenta



Fonte: Elaborado pelo autor, 2023.

Após iniciar o processo de conversão, todos os 309 arquivos são enviados individualmente para o servidor para categorizar e convertê-los. Este processo de conversão dura um total de 2.425 segundos, como pode ser visto entre a diferença de tempo indicada na Figura 9, indicando o início da conversão do primeiro arquivo e a Figura 10, indicando o final da conversão do último arquivo.

Figura 9 – Horário de início da conversão do primeiro arquivo

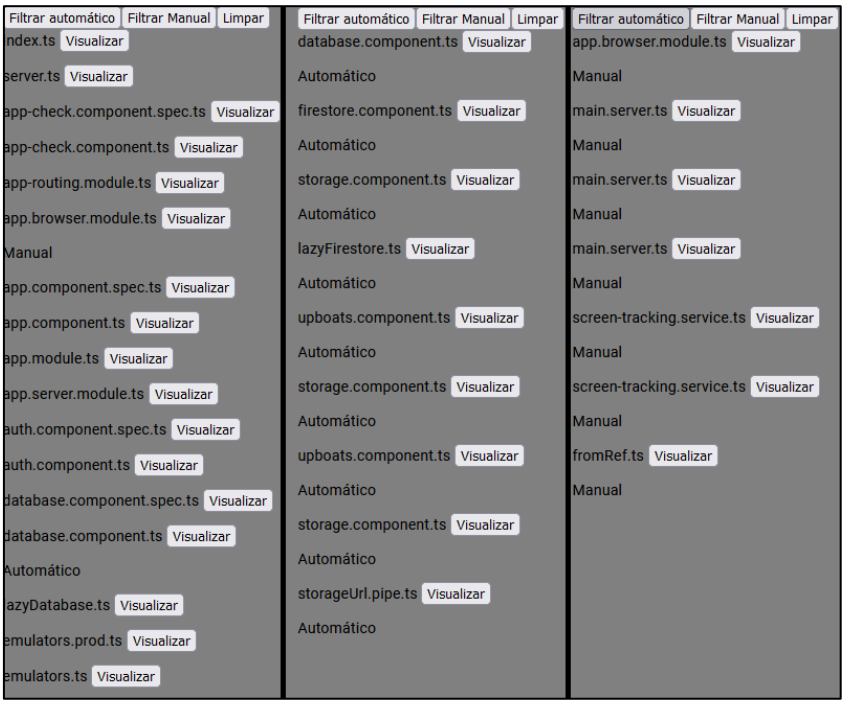
Fonte: Elaborado pelo autor, 2023.

Figura 10 – Horário de fim da conversão do último arquivo

Fonte: Elaborado pelo Autor, 2023.

Durante e após a conversão é possível ver a lista de arquivos, assim como indicado pela Figura 11, representado por 3 versões da lista separadas por uma linha. Na lista da esquerda está a lista sem filtro. Na lista do meio está com o filtro de alterações automáticas e na lista da direita está a lista filtrada por alterações manuais. Na figura existe uma descrição de quais tipos de alterações podem ser encontradas nos arquivos convertidos.

Figura 11 – Demonstração da interface de arquivos



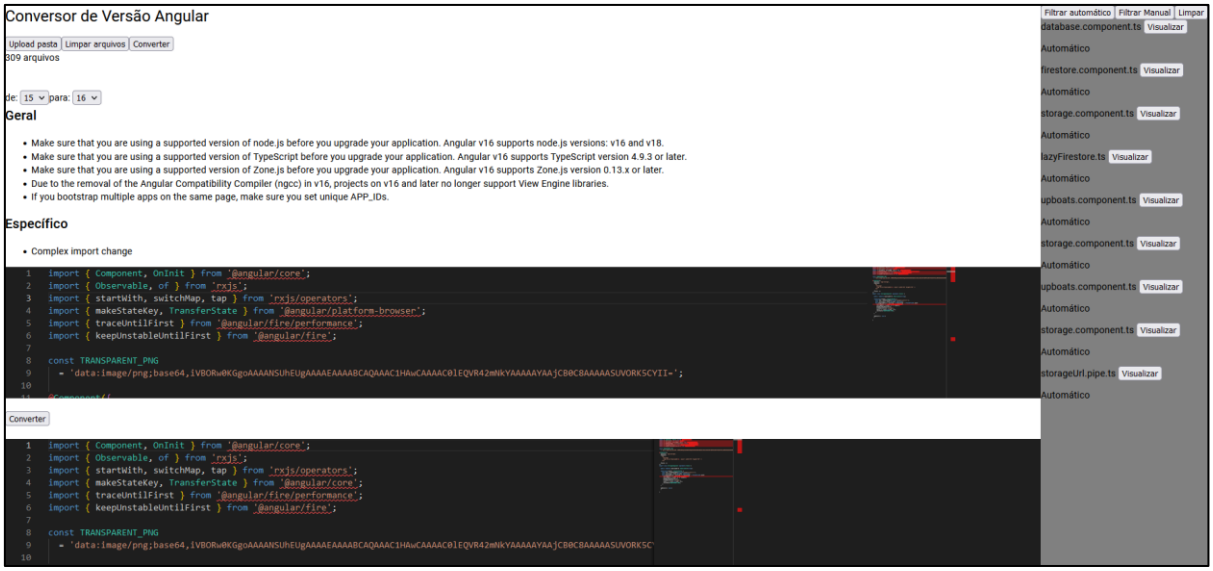
Fonte: Elaborado pelo autor, 2023.

A Figura 11 também demonstra a quantidade de arquivos com alterações manuais ou automáticas, sendo que de 309 arquivos, foi demonstrado nove arquivos com alteração automática e 7 arquivos com alteração manual. Para este caso, é considerado como “Manual” qualquer alteração classificada como “Alterações de comportamento” ou “Alterações

não realizáveis” demonstradas na Figura 7 e alterações do tipo “Automático” como as alterações categorizadas por “Alterações de código realizáveis” na mesma figura.

Finalmente é possível ver o resultado de cada arquivo como demonstrado na Figura 12. Ao visualizar o arquivo é possível ver as alterações gerais como definido na subseção 3.3, que indicam as alterações necessárias para converter qualquer projeto e finalmente as alterações específicas do arquivo, cuja diferença pode ser verificada na linha 4 da origem e destino.

Figura 12 – Demonstração da tela após conversão de pasta



Fonte: Elaborado pelo autor, 2023.

4.2 PERFORMANCE

Para realizar o teste de performance da ferramenta foram utilizados três sistemas Angular que estavam em versões anteriores a versão 16, eles são: o Ecosol, um website feito pelo Laboratório de Desenvolvimento e Transferência de Tecnologia (LDTT) da Universidade Regional de Blumenau (FURB), o próprio repositório do *framework* Angular na versão 15.2.9 (Angular, 2023c) e o repositório do AngularFire (AngularFire, 2023). Os testes foram realizados ao executar tanto o servidor quanto a interface de usuário em um computador utilizando Windows 11 64 bits, possuindo um processador Intel Core i5-9600KF com 3.6GHz e 8GB de RAM dedicada para o funcionamento da ferramenta. A comunicação entre interface de usuário e servidor foi feita através de requisições HTTP locais.

O resultado do teste de performance pode ser encontrado no Quadro 10, para comparação também é demonstrado a aplicação da ferramenta no código da própria ferramenta deste trabalho que está na versão 16 do Angular. O quadro mostra 6 campos. O primeiro indica o projeto que foi utilizado. O segundo o total de arquivos que indica a quantidade de arquivos com a extensão `.ts` presentes na pasta. O terceiro é o “Manuais” que indica a quantidade de arquivos com apenas alterações manuais que a ferramenta encontrou. O quarto campo chamado de “Automáticos” indica a quantidade de arquivos com alterações realizadas de maneiras automáticas pela ferramenta. O quinto demonstra a quantidade de arquivo na pasta que continha tanto alterações do tipo manual quanto automático, sendo que apenas houve um arquivo que entra nessa classificação nos testes realizados. Finalmente o último campo indica o tempo em segundo que a ferramenta necessitou para converter a pasta, sendo determinado pela diferença de tempo entre o início da conversão do primeiro arquivo e o final da conversão do último arquivo.

Quadro 10 – Resultado do teste da ferramenta.

Projeto	**Total de arquivos**	**Manuais**	**Automáticos**	**Ambos**	**Tempo**
Este	26	0	0	0	0,202s
Ecosol	75	0	0	0	0,436s
AngularFire (2023)	309	7	9	0	2,425s
Angular (2023c)	4811	178	10	7	641,506s

Fonte: Elaborado pelo autor, 2023.

A tabela de resultados está organizada de cima para baixo em ordem da quantidade total de arquivos no projeto, ou de maneira mais simples, o tamanho do projeto. Como pode ser visto pelos resultados, a quantidade de arquivos é diretamente relacionada ao tempo de execução da ferramenta, mas não é um indicador direto do tipo, ou até mesmo quantidade de alterações que precisarão ser feitas. O projeto Angular (2023c) é aproximadamente vinte vezes o tamanho

do projeto AngularFire (2023), porém a diferença nas alterações automáticas é pequena. Finalmente, o projeto Ecosol não possui nenhum arquivo que seja necessário alterar para fazer a atualização do Angular 15 para o 16, apenas executar as alterações indicadas pelas "alterações gerais" indicadas na Figura 7. Também é possível ver o impacto que a ferramenta de conversão pode ter ao observar o resultado do teste em Angular (2023c), considerando que foi possível encontrar 195 arquivos dentre 4811, simplificando o processo de busca nestes arquivos ao permitir o desenvolvedor ignorar 4616 arquivos, que totalizam aproximadamente 96% de todos os arquivos no projeto.

5 CONCLUSÕES

Este trabalho apresentou a criação de uma ferramenta que recebe código individual ou uma pasta de arquivos `.ts` e categoriza as alterações necessárias para converter o projeto da versão 15 para 16 do Angular. Para desenvolvimento da ferramenta foi utilizado C# e AspNetCore para criar um servidor que recebe requisições HTTP de uma interface de usuário feita em typescript usando o *framework* Angular.

O objetivo específico de classificar alterações necessárias para converter Angular da versão 15 para 16 foi alcançado, pois foram definidas quatro categorias que as alterações podem ser encontradas e a ferramenta demonstra para o usuário as alterações através das denominações de "automática" e "manual". O objetivo específico de converter código através de divisão de linhas e regex foi alcançado, pois o servidor separa os arquivos de código recebido em comandos e aplica validações e alterações utilizando regex. O objetivo específico de orientar o desenvolvedor sobre mudanças apontadas que não foram possíveis de serem realizadas também foi alcançado. Após a classificação em alterações automáticas ou manuais é possível que o desenvolvedor verifique as alterações manuais que necessitam ser realizadas através da interface. Dessa forma, o objetivo principal do trabalho de simplificar o processo de atualização do código escrito na versão 15 para a 16 do *framework* Angular também foi alcançado através da execução dos objetivos específicos e pode ser observado principalmente no resultado da aplicação da ferramenta sobre o projeto de Angular (2023c), em que a quantidade de arquivos que seria preciso considerar para a atualização é diminuída e os arquivos que necessitam alterações são facilmente encontrados.

Em comparação aos trabalhos correlatos, é possível perceber que a ferramenta traz funcionalidades semelhantes, ao converter código Angular de maneira automatizada, relatar alterações realizadas e não realizadas para o usuário, mas também traz um diferencial por mostrar as alterações através de uma interface de usuário compreensiva feita em HTML e utilizando o *framework* Angular, ao invés de linha de comando.

Analisando as ferramentas, o Visual Studio juntamente com a linguagem de programação C# e a biblioteca APT.NET Core se mostraram adequadas para o desenvolvimento de um servidor para processamento de requisições assim como categorização e conversão de arquivos. A ferramenta do Visual Studio Code juntamente com typescript e o *framework* Angular também se mostraram adequadas para o desenvolvimento de uma interface de usuário responsiva e de fácil desenvolvimento.

Como contribuições, este artigo e esta ferramenta servem como a descrição de um assistente de conversão de código e seu funcionamento de uma maneira que não foi possível encontrar no ambiente acadêmico. Como contribuição prática este trabalho também demonstra uma ferramenta para simplificar o processo de atualização de versão do Angular, tendo o potencial de auxiliar na manutenção de outras aplicações, além de criar um modelo que pode ser utilizado para estender a capacidade de conversão das versões 15 e 16.

Por mais que os objetivos tenham sido alcançados, existem algumas limitações no processo de conversão. Primeiramente, o método de divisão do código em linhas não aceita a possibilidade de um mesmo comando estar dividido em múltiplas linhas, inviabilizando a conversão automática de algumas formas de escrita do código. Também pelo escopo do artigo, só é possível converter entre as versões 15 e 16 do Angular, o que torna a funcionalidade limitada para essa versão de origem e destino específicas. Finalmente, não existe uma maneira do usuário obter um relatório das alterações além da interface do usuário, sendo necessário visualizar os arquivos individualmente para verificar alterações a serem realizadas.

Em conclusão, os objetivos deste trabalho foram alcançados e o mesmo disponibiliza uma base que pode ser utilizada para o desenvolvimento de outras ferramentas de conversão de código, assim como apresentou a criação de uma interface do usuário que não existia em outras ferramentas do mesmo tipo. Contudo, ainda existe a possibilidade de extensão do trabalho, como por exemplo:

a) adicionar mais versões do Angular para conversão;
b) buscar método para permitir a análise de comandos multilinha;
c) aumentar a quantidade de alterações que podem ser realizadas de maneira automática;
d) implementar um relatório completo que o usuário possa fazer *download*.

REFERÊNCIAS

ANGULAR. **What is Angular**. 2022. Disponível em: https://Angular.io/guide/what-is-Angular. Acesso em: 16 abr. 2023.

ANGULAR. **Angular Upgrade Guide.** 2023a. Disponível em: https://update.Angular.io/. Acesso em: 20 abr. 2023.

ANGULAR. **Angular versioning and releases**. 2023b. Disponível em: https://Angular.io/guide/releases. Acesso em: 30 dez. 2023.

ANGULAR. **Angular**. 2023c. Disponível em: https://github.com/Angular/Angular/releases/tag/15.2.9. Acesso em: 02 dez. 2023.

ANGULARFIRE. **AngularFire**. 2023. Disponível em: https://github.com/Angular/Angularfire. Acesso em: 02 dez. 2023.

BECK, Kent. **Test-Driven Development**: by example. Boston: Addison Wesley, 2022. 240 p.

CAMPOPIANO, Giovanni Teixeira. **Importância de testes unitários no desenvolvimento web**. 2022. Trabalho de Conclusão de Curso (Tecnólogo em Análise e Desenvolvimento de Sistemas) – Faculdade de Tecnologia de São Paulo, São Paulo, 2022.

CHIRILA, Andrei. **Test coverage at Google**. 2014. Disponível em: https://developers.google.com/google-test-automation-conference/2014/presentations. Acesso em: 17 jun. 2023.

COLUMBUS, Louis. **Miško Hevery, Inventor of Angular And How Open Source Languages Are Redefining Enterprise Software**. 2016. Disponível em: https://www.forbes.com/sites/louiscolumbus/2016/11/14/misko-hevery-inventor-of-Angular-and-how-open-source-languages-are-redefining-enterprise-software/?sh=3d699b85270d. Acesso em: 18 jun. 2023.

DIAS NETO, Arilo Cláudio. Introdução a Teste de Software. **Engenharia de Software Magazine**, São Paulo, v. 1, n. 7, p. 54-59, 2007.

.NET. **.NET Upgrade Assistant**. 2021. Disponível em: https://github.com/dotnet/upgrade-assistant. Acesso em: 20 abr. 2023.

.NET. **Visão geral do ASP.NET Core.** 2023. Disponível em: https://learn.microsoft.com/pt-br/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0. Acesso em: 02 dez. 2023.

FRIEDL, Jeffrey E. F. **Mastering Regular Expressions**. 3. ed. Sebastopol: O'reilly Media, Inc., 2006.

OLAWANLE, Joel. What is a Framework? Software Frameworks Definition. 2022. Disponível em: https://www.freecodecamp.org/news/what-is-a-framework-software-frameworks-definition/. Acesso em: 01 dez. 2023.

MDN. **Regular expressions**. 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions. Acesso em: 18 jun. 2023.

MILLER, Stephan. **What Is Angular?** 2021. Disponível em: https://www.codecademy.com/resources/blog/what-is-Angular/. Acesso em: 20 abr. 2023.

OLSON, Ella. **NgMigration Assistant**. 2018. Disponível em: https://github.com/ellamaolson/ngMigration-Assistant. Acesso em: 17 jun. 2023.

PMBOK. Project Management Institute (ed.). **Um Guia do Conhecimento em Gerenciamento de Projetos**: guia pmbok. 6. ed. Newtown Square, Pensilvânia, Eua: Project Management Institute, 2017. 756 p.

PYTHON. **2to3**: automated python 2 to 3 code translation. Automated Python 2 to 3 code translation. 2012. Disponível em: https://docs.python.org/3/library/2to3.html. Acesso em: 12 jun. 2023.

QUINN, Nicola. **Biggest Companies Keeping Angular Popular**. 2022. Disponível em: https://pangea.ai/blog/tech-stack/biggest-companies-keeping-Angular-popular. Acesso em: 17 jun. 2023.

SEMVER. **Semantic Versioning 2.0.0**. [2013]. Disponível em: https://semver.org/spec/v2.0.0.html. Acesso em: 18 jun. 2023.

STACKOVERFLOW. **2022 Developer Survey**. 2022. Disponível em: https://survey.stackoverflow.co/2022/. Acesso em: 20 abr. 2023.

VALENTE, Marco Tulio. **Engenharia de Software Moderna**: princípios e práticas para desenvolvimento de software com produtividade. Belo Horizonte: Independente, 2020. 408 p.