UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO POLITÉCNICO DA UFSM
CURSO TÉCNICO EM INFORMÁTICA

Anderson Dotto Padoin

ANÁLISE DAS CARACTERÍSTICAS DO FRAMEWORK ANGULAR E SUA APLICAÇÃO NO DESENVOLVIMENTO DE UMA APLICAÇÃO WEB

Santa Maria, RS
2019

Anderson Dotto Padoin

ANÁLISE DAS CARACTERÍSTICAS DO FRAMEWORK ANGULAR E SUA APLICAÇÃO NO DESENVOLVIMENTO DE UMA APLICAÇÃO WEB

Trabalho de Conclusão apresentado ao Curso Técnico em Informática, do Colégio Politécnico da Universidade Federal de Santa Maria (CPOL-UFSM, RS), como requisito parcial para obtenção do título de **Técnico em Informática**.

Orientador: Prof. Giani Petri

Santa Maria, RS
2019

Anderson Dotto Padoin

ANÁLISE DAS CARACTERÍSTICAS DO FRAMEWORK ANGULAR E SUA APLICAÇÃO NO DESENVOLVIMENTO DE UMA APLICAÇÃO WEB

Trabalho de Conclusão apresentado ao Curso Técnico em Informática, do Colégio Politécnico da Universidade Federal de Santa Maria (CPOL-UFSM, RS), como requisito parcial para obtenção do título de **Técnico em Informática**.

Aprovado em 11 de dezembro de 2019:

_____

Giani Petri, Dr. (UFSM)
(Presidente/Orientador)

_____

Fernando Emilio Puntel, Me. (UFSM)

_____

Bruno Augusti Mozzaquatro, Me. (UFSM)

Santa Maria, RS
2019

RESUMO

ANÁLISE DAS CARACTERÍSTICAS DO FRAMEWORK ANGULAR E SUA APLICAÇÃO NO DESENVOLVIMENTO DE UMA APLICAÇÃO WEB

AUTOR: Anderson Dotto Padoin
ORIENTADOR: Prof. Giani Petri

Esse trabalho apresenta um estudo e análise das características e ferramentas básicas oferecidas pelo framework Angular no desenvolvimento de sistemas web através da demonstração e implementação de uma aplicação *front-end* personalizada, independente e sem vínculo com uma tecnologia ou sistema em *back-end*. Esse trabalho tem início com uma revisão da literatura dos conceitos básicos que tangem o framework, bem como uma pesquisa a respeito de seu surgimento, definição e estrutura básica para sua implementação. O desenvolvimento da aplicação web, para demonstrar as características do framework, inicia na etapa de especificação da aplicação e definição dos requisitos para realizar o desenvolvimento. Em seguida é realizada a iniciação do projeto e explicação a respeito da sua estrutura e árvore de diretórios. O trabalho da continuidade com a demonstração passo a passo da implementação da aplicação, apresentando as dependências necessárias e sintaxe básica da linguagem de programação TypeScript. A última etapa do desenvolvimento é a mais abrangente, onde o foco é realizar a implementação e exibição, por meio de capturas de telas, de todos os elementos que compõem a aplicação, entre eles os *services,* módulos, componentes HTML e TypeScript, sistemas de rotas e também justificar a utilização e facilidades das tecnologias que o Angular oferece como diretivas e *data bindind*. Como conclusão do trabalho, observa-se que o framework exige bastante do desenvolvedor durante a fase de iniciação e organização do projeto, mas que após completa essa fase, as ferramentas e tecnologias presentes no framework facilitam na elaboração de uma aplicação *front-end* completa, robusta e de grande fator de evolução e manutenibilidade.

Palavras-chave: Angular. Front-end. Implementação. TypeScript.

SUMÁRIO

1 INTRODUÇÃO

O número de aplicações web vem aumentando conforme a tecnologia se difunde nas mais variadas áreas de conhecimento da atualidade (NASCIMENTO, 2014). Esse crescimento da tecnologia, por sua vez, demanda frequentes pesquisas e atualizações no conhecimento tecnológico por parte dos usuários e desenvolvedores que desejam ter seu negócio ou empreendimento disponível no meio digital. Essas necessidades e preocupações refletem no aumento das demandas no desenvolvimento de aplicações web focadas no interesse do cliente e na sua regra de negócio.

Grande parte das aplicações seguem rumos tradicionais, integrando o seu desenvolvimento lógico, a regra de negócio e a abstração e exibição do sistema ao usuário (interface) em um único e grande sistema unificado. Com o objetivo de facilitar a implementação da lógica do sistema, bem como a sua interface, surgem os frameworks de desenvolvimento back-end e front-end.

O termo *back-end* é utilizado para especificar os módulos de uma aplicação que executam no servidor e realizam operações de controle e de acesso à base de dados enquanto o *front-end* especifica a aplicação que é exibida ao usuário e que preferencialmente execute no navegador.

Os frameworks integram outras tecnologias e são configurados para realizar a abstração de muitas partes do desenvolvimento. Os frameworks *front-end* mais especificamente separam a lógica de negócio menos complexa da lógica de negócio mais complexa (*back-end*) e também fornecem mais formas e tecnologias para desenvolver um *front-end* de maior manutenibilidade e dinamicidade. Dessa forma os frameworks *front-end* tendem a ser aplicações que permitem um desenvolvimento mais independente do *back-end*, embora necessitem do mesmo para realizar efetivamente grande parte das funcionalidades como a comunicação e troca de informação com uma base de dados.

O Spring[1], Django[2] e Laravel[3] são exemplos de frameworks *back-end* para diferentes linguagens de programação sendo elas Java[4], Python[5] e PHP[6] consecutivamente. React[7] e Vue[8] são exemplos de frameworks *front-end* JavaScript e também são alguns dos mais populares no mercado atualmente.

O framework Angular[9] é um exemplo de framework e também uma plataforma JavaScript que permite criar mais distância entre o *front-end* e *back-end* de aplicações web. Com uma estrutura de projeto mais robusta e organizada, e a utilização de uma linguagem de programação diferenciada, o angular permite uma maior independência do *back-end*.

O Angular é um framework completo, ou seja, fornece todas as ferramentas e dependências básicas para desenvolver uma aplicação completa, além de um amplo repositório de pacotes desenvolvidos pela comunidade, assim facilitando na elaboração e estruturação de novos projetos de diferentes tipos e objetivos.

O estudo e desenvolvimento de aplicações web por meio do Angular é justificado pela dificuldade muitas vezes encontrada durante a implementação, manutenção e evolução da interface gráfica de um software que integra o *front-end* junto ao *back-end*. A utilização do angular também isola o sistema em aplicações menos acopladas assim gerando diferentes domínios de desenvolvimento que facilitam na refatoração e identificação e correção de erros.

1.1 PROBLEMA

O crescimento no surgimento de novas aplicações web pode implicar no aumento de aplicações de *back-end* e *front-end* integrados, resultando em interfaces

[1] https://spring.io/
[2] https://www.djangoproject.com/
[3] https://laravel.com/
[4] https://www.java.com/pt_BR/
[5] https://www.python.org/
[6] https://www.php.net/
[7] https://pt-br.reactjs.org/
[8] https://vuejs.org/
[9] https://angular.io/

pouco elaboradas e menos personalizadas devido às limitações da tecnologia do *back-end*.

Para reduzir as limitações tecnológicas e melhorar a implementação e apresentação da interface é utilizado um framework *front-end*. Neste contexto, as perguntas que norteiam o desenvolvimento desse trabalho são: Quais as ferramentas e técnicas são oferecidas pelo framework Angular? Como a independência do *front-end* colabora em uma melhor implementação da interface?

1.2 OBJETIVOS

1.2.1 Objetivo geral

Implementar uma aplicação web onde seu *front-end* execute de forma independente e isolada do *back-end* por meio do framework Angular com o objetivo de demonstrar durante o desenvolvimento a estrutura, técnicas e ferramentas que o framework oferece para a elaboração de uma interface menos integrada e mais customizada.

1.2.2 Objetivos específicos

- Implementar o sistema seguindo os padrões estruturais de projetos e características do Angular sendo elas orientação a objetos, rotas, módulos, *services* e componentes.
- Aplicar o conceito de *binding* entre as páginas HTML e componentes TypeScript.
- Implementar componentes que utilizem as diretivas do Angular.

2 REVISÃO DE LITERATURA

2.1 ANGULAR: HISTÓRIA E DEFINIÇÃO

O Angular é a versão 2 do seu predecessor AngularJS (versão 1.x) e que foi desenvolvido em grande parte pela equipe principal da Google e liberado ao público em sua versão estável em meados de 2016. O nome Angular se refere à todas as versões desenvolvidas a partir da versão 2.x. O AngularJS também é um framework JavaScript *front-end* desenvolvido pela Google mas que devido a diversos problemas lógicos e detalhes técnicos foi reescrito dando origem ao atual Angular, que trouxe uma nova abordagem, estrutura de projeto e que manteve as funcionalidades do AngularJS.

De acordo com DAYLEY, Brad; DAYLEY, Brendan; DAYLEY (2018)

> Toda a ideologia por trás do Angular é prover um framework que torne fácil o desenvolvimento de páginas web e aplicações bem desenhadas e estruturadas utilizando um framework Model View Controller (MVC) ou Model View View Model (MVVM).

Essa ideologia seguida pelo angular deixa claro a necessidade de uma abordagem mais independente e completa no que se refere ao fron-end das aplicações. O angular busca minimizar ao máximo a sensação de dependência tecnológica entre os componentes gráficos de uma aplicação web e sua lógica de negócio mais técnica, introduzindo uma gama de API's e tecnologias exclusivas para dar mais liberdade na criação do *front-end*.

O Angular oferece um conjunto de tecnologias que permitem a criação de *Single Page Aplications* (Aplicativos de Página Única). Monteiro (2014) reforça que "O objetivo principal sobre esse tipo de aplicação é ser capaz de atualizar partes de uma interface, sem enviar ou receber a requisição de uma página inteira".

Essa característica implica na economia de recursos, principalmente no que se refere a banda e tráfego de informações na rede, visto que uma requisição de uma página completa requer o reenvio de informação, as quais são mantidas e reutilizadas no lado do cliente em aplicações de página única.

2.2 A RELAÇÃO DO ANGULAR E O TYPESCRIPT

O Angular faz uso da linguagem de programação TypeScript[10], desenvolvida e mantida pela Microsoft, que é utilizada para desenvolver grande parte da lógica do *front-end* de aplicações angular. O TypeScript embora seja uma linguagem de programação é também um super conjunto ECMAScript, uma especificação de linguagem de programação para o JavaScript. De forma simplificada, os códigos desenvolvidos em TypeScript serão traduzi em códigos JavaScript seguindo a especificação da ECMAScript.

A utilização do typeScript no desenvolvimento da aplicação é opcional uma vez que o Angular pode ser desenvolvido utilizando diversas linguagens de programação como ECMAScript 5/6/7[11] ou Dart[12] que é uma linguagem de programação que busca substituir o javaScript no desenvolvimento de aplicações web. O TypeScript é comumente utilizado como linguagem de desenvolvimento de aplicações Angular pois o desenvolvimento do framework em si é feito em TypeScript, dessa forma “Ser proficiente no TypeScript dará aos desenvolvedores uma enorme vantagem quando se trata de entender as entranhas do framework” (DEELEMAN, p 4).

GAVIN, O; ABADI, Martín; TORGERSEN, Mads (2014) ressaltam que “A intenção do TypeScript não é ser uma nova linguagem de programação em si, mas sim aprimorar e dar suporte no desenvolvimento em JavaScript”.

O entendimento da utilização e importância do TypeScript no desenvolvimento de aplicações *front-end* utilizando o Angular é indispensável dado ao fato de que a maior parte do desenvolvimento em Angular é realizada por intermédio dos módulos, interfaces, classes e expressões que somente se fazem possíveis através da utilização do TypeScript.

A abstração que é criada com a utilização do TypeScript permite que o desenvolvedor construa e evolua a aplicação web sem se preocupar com muitos detalhes internos do JavaScript que se usados da forma pura dificultam a organização

[10] https://www.typescriptlang.org/
[11] https://www.ecma-international.org/
[12] https://dart.dev/

do sistema e a legibilidade da aplicação para manutenção principalmente em aplicações amplas e complexas. Para o desenvolvedor essas dificuldades são abstraídas pelo TypeScript e o ECMAScript, apresentando um ambiente de trabalho muito mais organizado e semelhante a linguagens de programação orientadas a objetos através de uma forma mais conhecida de trabalhar com objetos, classes, interfaces. O TypeScript surge como uma linguagem tipada, contradizendo uma das características básicas do javaScript que é ser uma linguagem fracamente tipada.

2.3 ESTRUTURA E REQUISITOS

A implementação de sistemas em Angular exige um conjunto de atividades e implementações complementares para efetivamente ser executada. Um projeto em Angular necessita ter um esquema de módulos, rotas, componentes e diversas outras configurações que permitam a integração e comunicação efetiva entre o modelo e views.

Durante o desenvolvimento o Angular requer outras tecnologias como o Node.js, o Gerenciador de Pacotes do Note (NPM) e o próprio TypeScript como requisitos para desenvolver uma aplicação de forma mais elaborada. Os gerenciadores de pacote como NPM contribuem na aceleração do processo de implementação e codificação realizando as tarefas de busca, inclusão e exclusão dos pacotes e dependências no projeto, assim como o Node.js entrega um interpretador de códigos JavaScript para realizar efetivamente a execução dos comandos TypeScript. Essas tecnologias serão abordadas de forma mais detalhadas no capítulo de modelagem e desenvolvimento do sistema. Também serão abordados os arquivos de configuração angular.json, package.json e arquivos de módulos durante a demonstração da implementação das interfaces.

A aplicação tem como foco demonstrar como as tecnologias e tópicos citados acima colaboram na elaboração de interfaces mais dinâmicas e customizáveis através da implementação de páginas que façam uso dessas tecnologias.

3 ESPECIFICAÇÃO, MODELAGEM E DESENVOLVIMENTO DO SISTEMA

3.1 ESPECIFICAÇÃO E REQUISITOS DO SISTEMA

Com o objetivo de demonstrar as características principais e facilidades oferecidas pelo Angular, é desenvolvido, no escopo desse trabalho, um sistema semelhante a um fórum de discussões simplificado. A aplicação tem como público alvo os usuários interessados por assuntos de tecnologia e informação.

A aplicação contará com um cadastro de usuários. Cada usuário cadastrado terá acesso aos fóruns assim podendo criar novos tópicos de discussão ou responder a tópicos abertos por outros usuários.

O sistema de tópicos funcionará semelhante aos fóruns existentes na web atualmente, permitindo a inclusão de um tópico sobre uma determinada categoria (notícias e novidades, discussões, dúvidas e perguntas ou tutorias), contendo um título e uma descrição para o tópico. Um tópico irá possui um estado de ‘aberto’ ou ‘fechado’, dessa forma permitindo que usuários respondam apenas a tópicos de estado ‘aberto’.

O campo de descrição do tópico irá permitir a formatação do texto, possibilitando que o usuário personalize as definições da fonte e arranjo do texto do seu tópico. A formatação também está disponível para as respostas dos textos. O título do tópico deverá aceitar a formatação padrão do navegador, em texto plano.

O sistema será desenvolvido na plataforma Windows através do Ambiente de desenvolvimento integrado (IDE) WebStorms. A tabela 1 exibe os requisitos funcionais e não funcionais desejados no sistema.

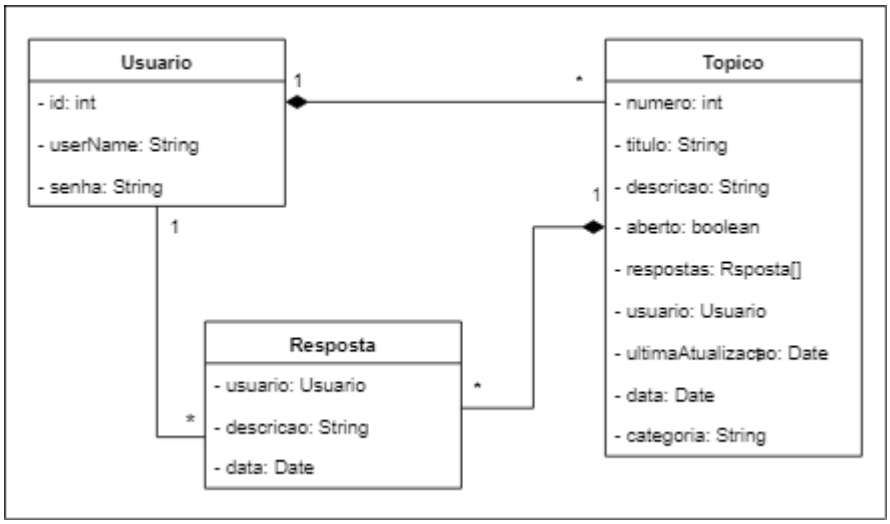
Tabela 1 – Requisitos funcionais e não funcionais do sistema

Requisitos Funcionais	
Requisito	**Descrição**
Cadastro de usuários	O usuário poderá realizar o seu cadastro para acesso ao sistema.
Login no sistema	O usuário cadastrado poderá realizar o login no sistema.
Submissão de tópicos	Um usuário cadastrado poderá submeter tópicos no fórum.
Formatação de tópicos	Os tópicos podem ser formatados com uma ferramenta de formatação no momento da submissão.
Responder aos tópicos	Os usuários podem responder aos tópicos de status aberto se desejado.
Controle do tópico	O usuário poderá fechar seu tópico para respostas ou abri-lo novamente a qualquer momento.
Requisitos não funcionais	
Requisito	**Descrição**
Plataforma de execução	O sistema deverá funcionar nos navegadores web Google Chrome e Firefox.
Plataforma de Desenvolvimento	O sistema será desenvolvido na plataforma Microsoft Windows.

Fonte: Autoria Própria

Na figura 1 é elaborado o diagrama de classes conceitual do sistema, possuindo as classes que compõem os modelos da aplicação.

Figura 1 – Diagrama de classes conceitual do sistema.



Fonte: Autoria própria.

3.2 CRIAÇÃO E ESTRUTURAÇÃO DO PROJETO

3.2.1 Criação do projeto

Para iniciar a implementação de projetos em Angular é preciso estar ciente dos requisitos básicos para sua criação. O Angular faz uso do interpretador JavaScript Node.js para dar início às atividades de codificação e sem ele não é possível desenvolver ou executar localmente a aplicação.

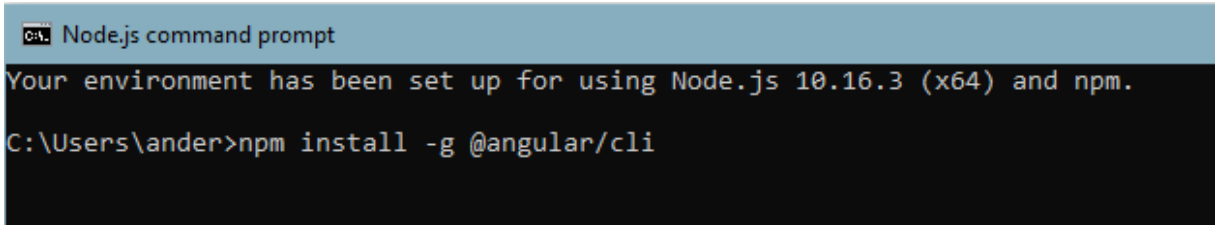
Como interpretador JavaScript o Node realiza todas as operações que dizem respeito à aplicações JavaScript, sendo elas compilação, tradução, execução de comandos internos e dos pacotes que são instalados via NPM. O NPM é o gerenciador de pacotes do node e que já está incluso na instalação do Node.js. Sua principal função é realizar a leitura e busca de pacotes no repositório online. Este repositório é comunitário e permite o compartilhamento de pacotes, componentes e demais ferramentas desenvolvidas pelos usuários.

O Node.js puramente instalado na máquina não é o suficiente para iniciar os trabalhos em Angular, como mencionado acima, o Node apenas irá interpretar os comandos dos pacotes instalados, sendo necessário realizar o download dos pacotes do Angular.

Os desenvolvedores do Angular implementaram a interface de linha de comandos AngularCli (ou somente CLI) que é responsável por gerenciar os projetos em Angular, dando suporte à iniciação, desenvolvimento e manutenção de aplicações. O AngularCLI é facilmente obtido através do NPM utilizando o terminal do Node.JS.

Através do comando ‘npm install -g @angular/cli’ (figura 2) o angularCLI será instalado pelo NPM e através do parâmetro ‘-g’ o pacote do irá ser instalado globalmente, assim permitindo o uso de suas funcionalidades em todos os projetos na máquina.

Figura 2 – Captura de tela do comando de instalação do angularCli.

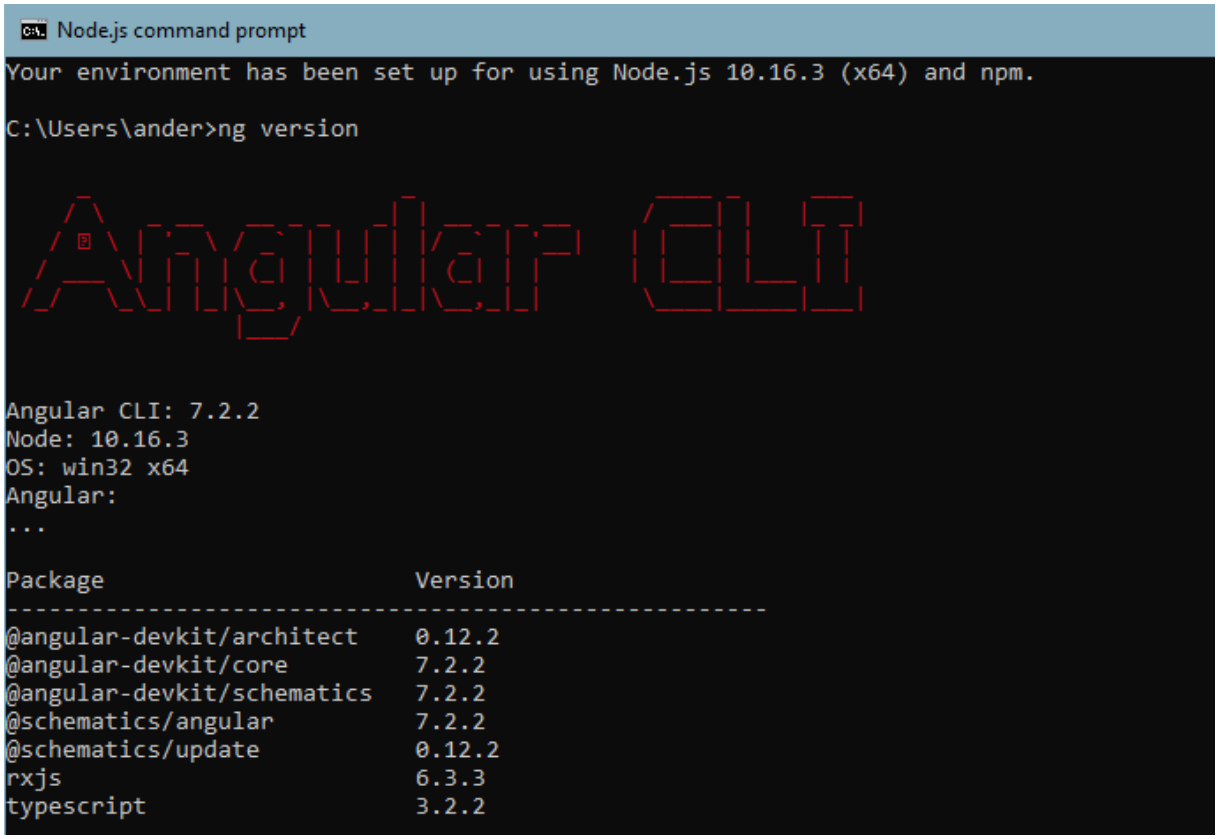


Fonte: Autoria própria.

Uma vez instalado, as operações do AngularCLI são iniciadas pelo comando ‘ng’. Para verificar a instalação e a versão instalada é inserido o comando ‘ng version’ (figura 3). Os dados exibidos mostram a versão no Angular instalada, assim como a versão do Node.JS. Vale ressaltar que versões mais novas do Angular comumente requerem uma versão mais recente do Node.JS e possivelmente uma versão mais recente do NPM.

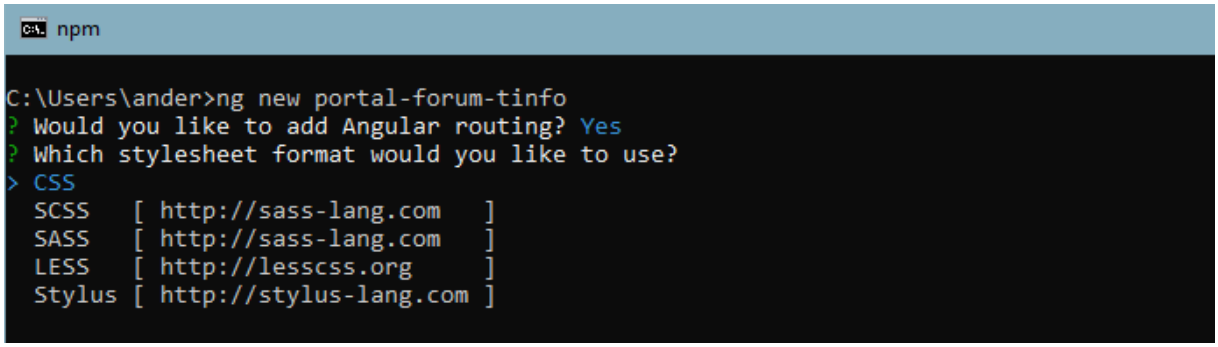
Uma vez que o angularCli esteja instalado, já será possível dar início ao desenvolvimento de projetos em Angular. O CLI permite a criação de projetos Angular básicos diretamente pela linha de comando. Através do comando ‘ng new’ é possível criar novos projetos especificando um nome do projeto como o próximo parâmetro do comando (figura 4).

Figura 3 – Captura de tela da verificação da versão do angularCli.



Fonte: Autoria própria.

Figura 4 – Captura de tela da criação de um projeto Angular através do angularCli.



Fonte: Autoria própria.

Nas versões mais novas do angularCLI é solicitado ao usuário se ele deseja adicionar um arquivo de rotas do Angular na criação do projeto. Este arquivo será responsável por realizar a relação entre URL e componentes HTML e TypeScript. Em

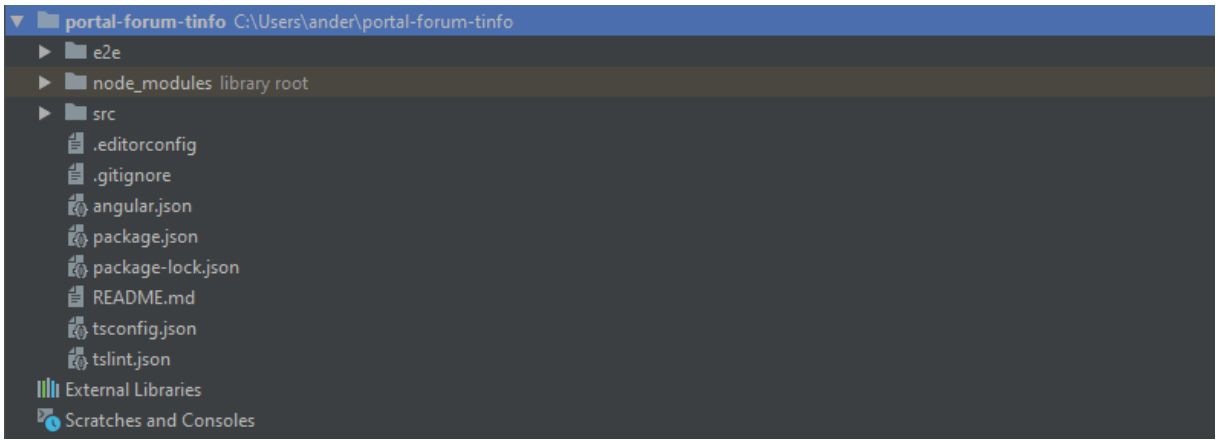
seguida o CLI irá solicitar que o usuário defina um padrão de formato para os estilos dos componentes HTML. Para este projeto foi definido o padrão CSS (Cascading Style Sheets).

O CLI é o responsável por criar toda a estrutura de diretórios padrão do projeto, assim como realizar a criação e definição da configuração básica de todos os arquivos de configuração do projeto. Em conjunto ao angularCLI, o NPM também realiza o download de todos os módulos JavaScript necessários para desenvolver e executar a aplicação.

3.2.2 Estrutura de diretórios

A estrutura de diretórios criada pelo angularCLI segue um padrão e de acordo com a versão do Angular pode conter algumas diferenças. No projeto trabalhado a árvore de diretórios gerada é condizente com a versão 7.2.2 do angular (figura 5).

Figura 5 – Captura de tela da estrutura do projeto Angular gerada pelo angularCli.



Fonte: Autoria própria.

A estrutura básica criada pelo CLI é composta pelas três pastas principais:

- e2e – Pasta que contém os arquivos e ferramentas responsáveis por realizar testes que simulam ações de um usuário. Gerado por padrão porém não é obrigatório para testes em ambiente de desenvolvimento.

- node_modules – Pasta que contém todos os módulos necessários para a execução do projeto. Pacotes individuais instalados pelo NPM também irão ser colocados nessa pasta.
- src – Pasta raiz dos componentes do projeto. Componentes HTML e TypeScript (classes, interfaces, etc), CSS e demais arquivos de configuração dos componentes devem estar alocados em alguma de suas subpastas de acordo com a hierarquia estabelecida pelo usuário.

Na raiz do projeto também são gerados os arquivos de configuração angular.json e package.json e a configuração de ambos os arquivos afetam o projeto como um todo.

O angular.json é o arquivo responsável por definir e especificar toda e qualquer configuração relacionada ao projeto Angular, assim como realizar a inclusão dos arquivos de estilo (CSS) e JavaScript adicionais ao projeto (figura 6).

Figura 6 – Captura de tela parcial do arquivo angular.json.

Fonte: Autoria própria.

Todos os arquivos CSS e JS necessários para a exibição ou comportamento correto de algum componente ou elemento na interface deve estar presente no angular.json ou importados em algum arquivo primário, o qual precisa estar presente nesse arquivo.

O arquivo package.json contém as informações de compilação e dependências do projeto. Também estão presentes nesse arquivo as dependências de desenvolvimento (figura 7), as quais são necessárias para executar o projeto em ambiente de desenvolvimento para testes.

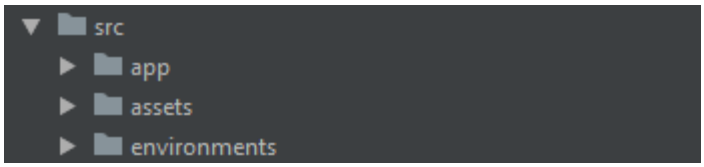
Figura 7 – Captura de tela parcial do arquivo package.json

Fonte: Autoria própria..

Semelhante ao angular.json, o arquivo de configuração package.json exige a inclusão de todas as dependências que são utilizados para a implementação e implantação da aplicação. As informações presentes no arquivo são geradas automaticamente quando pacotes são instalados por meio do comando ‘npm install *nome_do_pacote*’. Todo componente instalado através do NPM é incluído neste arquivo e toda modificação feita no arquivo requer a execução do comando ‘npm install’ para atualizar as dependências do projeto.

A estrutura de subpastas da pasta src tem como objetivo diferenciar a aplicação e seus arquivos adicionais que complementam a aplicação, como imagens, arquivos CSS e JavaScript por meio da pasta assets (figura 8). A pasta ‘app’ contém por padrão os componentes básicos e que são o suficiente para executar a aplicação. Novos componentes, classes, e demais arquivos que tenham relação à lógica de negócio da aplicação devem estar contidos nesta pasta.

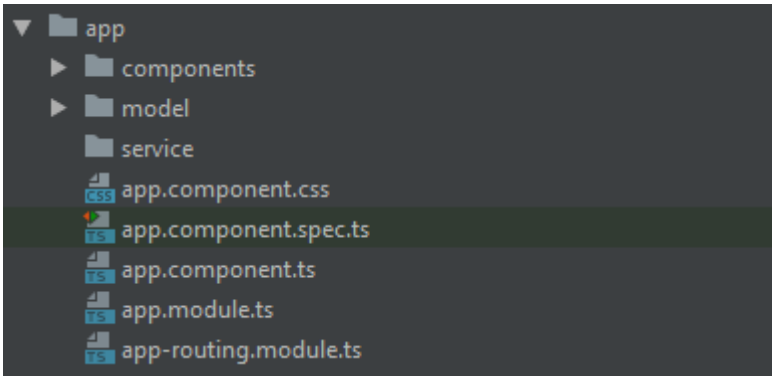
Figura 8 – Captura de tela da estrutura de diretórios do diretório ‘src’



Fonte: Autoria própria.

No projeto atual foi definido um padrão de estrutura semelhante ao MVC, padrão seguido por diversos sistemas web e desktop. Devido ao suporte a orientação a objetos pelo typeScript o padrão MVC consegue se adequar de maneira satisfatória em projetos Angular. Na figura 9 percebe-se a existência da pasta componentes, representando a View do modelo MVC, a pasta model e a pasta service, representando o Controller do modelo MVC.

Figura 9 – Captura de tela da estrutura gerada no diretório ‘src/app’ do projeto



Fonte: Autoria própria.

Não é comum encontrar projetos em Angular com uma estrutura MVC perfeita visto que cada componente possui sua classe TypeScript que trabalha semelhante a um controller. O pacote service tem como tarefa agrupar as classes responsáveis pela troca de dados entre os componentes e bases de dados, sejam elas oriundas de uma API ou da própria aplicação Angular.

3.3 DESENVOLVIMENTO E DEMONSTRAÇÃO DO SISTEMA

3.3.1 TypeScript - sintaxe e orientação a objetos

O Angular faz uso do TypeScript para realizar a programação orientada a objetos na aplicação *front-end*. Por se tratar de uma linguagem orientada a objetos o

typeScript provê suporte a muitas características presentes em outras linguagens de programação orientadas a objetos como classes, interfaces, enum e herança.

A implementação de classes segue o mesmo padrão encontrado em outras linguagens orientadas a objetos, começando pela declaração da classe através da expressão ‘export class’ seguida do nome da classe a ser desenvolvida. A indicação de herança ou implementação de interfaces é realizada através das expressões ‘extends’ e ‘implements’ respectivamente.

Assim como outras linguagens orientadas a objetos, os atributos possuem visibilidade, tipo e nomes, embora o tipo do atributo não seja obrigatório, e a sua não utilização será entendida como um atributo do tipo *any*, ou seja, irá comportar qualquer valor atribuído em tempo de execução. A organização dos atributos é feita pela definição da visibilidade, seguida do nome da do atributo. A definição do tipo, caso exista, é feita por meio do operador ‘ **:** ’ (dois pontos) seguido do nome do tipo.

Na figura 10 é demonstrado o exemplo de implementação de uma classe que utiliza herança, implementa interfaces e declara atributos, seguindo a sintaxe definida no typeScript.

Figura 10 – Exemplo de implementação de uma classe contendo herança e interfaces.

```
export class Usuario extends Pessoa implements Autenticavel{

  id: any;
  userName: string;
  senha: string;

  constructor(id?: number, userName?: string, senha?: string) {
    super();
    this.senha = senha;
    this.id = id;
    this.userName = userName;
  }
}
```

Fonte: Autoria própria.

Para o funcionamento efetivo do sistema, atendendo as especificações definidas, é necessário a criação de três classes, sendo elas:

- Usuario – Classe que representa o usuário do mundo real, e que irá acessar o sistema (figura 11).
- Topico – Representa o tópico submetido pelos usuários no fórum (figura 12).
- Resposta – Representa as respostas enviadas a um tópico em especifico. As respostas estão presentes na classe Topico, sendo nesta classe um conjunto de respostas (figura 13).

Figura 11 – Captura de tela da classe Usuario.

```
export class Usuario {

  id: any;
  userName: string;
  senha: string;

  constructor(id?: number, userName?: string, senha?: string) {
    this.senha = senha;
    this.id = id;
    this.userName = userName;
  }
}
```

Fonte: Autoria própria.

Figura 12 – Captura de tela da classe Topico.

```
export class Topico {
  titulo: any;
  descricao: any;
  numero: any;
  data: Date;
  ultimaAtualizacao: Date;
  aberto: boolean;
  respostas: Resposta[] = [];
  categoria: any;
  usuario: Usuario;

  constructor(titulo?: any, descricao?: any, numero?: any) {
    this.titulo = titulo;
    this.descricao = descricao;
    this.numero = numero;
  }
}
```

Fonte: Autoria própria.

Figura 13 – Captura de tela da classe Resposta.

```
export class Resposta {

  usuario: Usuario;
  descricao: any;
  data: Date;

  constructor(usuario?: Usuario, descricao?: any, data?: Date) {
    this.data = data;
    this.descricao = descricao;
    this.usuario = usuario;
  }
}
```

Fonte: Autoria própria.

A injeção de dependências é a capacidade dos componentes typeScript de terem acesso a objetos no momento de sua criação. No projeto atual a injeção de dependências é realizada no momento que o construtor da classe é executado, dessa forma os objetos são injetados através dos parâmetros do construtor (figura 14).

Figura 14 – Captura de tela da injeção de dependências através do construtor.

```
export class LoginComponent {

  senha: any = '';
  login: any = ''

  constructor(private router: Router){

  }
```

Fonte: Autoria própria.

Como visto na figura 14, o construtor do componente LoginComponent cria uma instancia da classe Router no momento de criação do componente, ou seja, no instante que a página Login for acessada no navegador web. A injeção de dependências é uma característica muito importante pois em muitos momentos é preciso acessar atributos e métodos de objetos já populados, como é o caso da classe Router, que irá possuir os dados referentes à url e métodos de navegação.

3.3.2 Instalando pacotes

Embora os pacotes básicos do Angular permitam o desenvolvimento de uma aplicação completa, é comum ser necessária a utilização de algum pacote não incluso na instalação básica ou também pacotes desenvolvidos pela comunidade. Nesse projeto são utilizados quatro pacotes adicionais, o bootstrap, ngx-webstorage, quill e ngx-quill;

O bootstrap será utilizado para facilitar na criação da interface da aplicação. O pacote do bootstrap possui todos os arquivos necessários para utilizar os estilos do framework. O pacote ngx-webstorage é utilizado para realizar o armazenamento local das informações visto que não é utilizada nenhuma API para fornecer e armazenar os dados dos usuários e tópicos da aplicação. Por último são incluídos os pacotes quill e ngx-quill que possuem componentes que permitem a formatação do texto inserido nos tópicos e respostas.

A instalação de pacotes adicionais no projeto é realizada através do terminal do Node.js, ou qualquer outro terminal do sistema desde que as variáveis de ambiente tenham sido configuradas (no caso da plataforma Windows). Através do comando 'npm install' seguido do nome do pacote é possível realizar o download do pacote desejado do repositório online (figura 15).

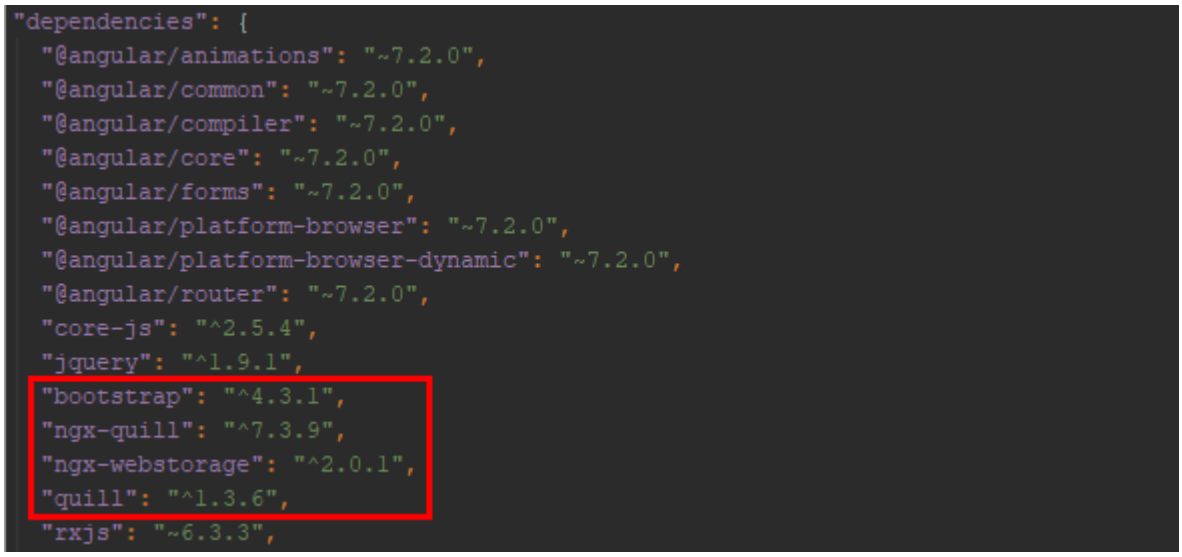
Conforme a figura 15, todos os pacotes foram instalados em sua ultima versão estável disponível por meio da inclusão da expressão '@latest'. A especificação da versão é realizada através do simbolo '@' (arroba) seguido da numeração da versão desejada, por exemplo 'npm install bootstrap@3.0.1'.

Figura 15 – Captura de tela dos comandos para a instalação dos pacotes adicionais.

Fonte: Autoria própria.

Uma vez instalados, os pacotes adicionais estarão listados no arquivo package.json, na raíz do projeto. Os pacotes serão listados dentro dos atriburos ‘dependencies’ e ‘devDependencies’, contendo seu nome e versão instalada (figura 16). As informações contidas no arquivo são livres para edição, sendo assim uma vez instalado um pacote é possivel alterar a sua versão, porém todas as alterações efetuadas requerem a execução do comando ‘npm install’, o qual irá fazer o npm verificar se todos os pacotes inclusos no package.json estão presentes no projeto, e caso contrario realizar o seu download.

Figura 16 – Captura de tela das dependencias instaladas do arquivo package.json.



Fonte: Autoria própria

3.3.3 Services

No Angular existe o conceito de *service*, uma classe que tem contato muito próximo com os dados que trafegam na aplicação, comumente realizando a comunicação com alguma API ou com variáveis de tempo de execução. Um *service* é um exemplo de classe que faz uso da injeção de dependências, ou seja, sendo injetado em um componente no momento de sua criação ou iniciação, já contendo dados e métodos prontos para uso pelo componente.

Na aplicação desenvolvida no escopo desse trabalho, os *services* são utilizados para exercer o papel da base de dados, armazenando a recuperando os dados da

aplicação em “bases de dados” locais e temporárias que são suficientes para simular uma aplicação que consome alguma API. Os *services* tendem a realizar operações em comum a um determinado contexto.

A aplicação em desenvolvimento possui dois contextos, sendo eles contexto de tópicos e respostas e contexto de usuários. Dessa forma cada *service* atende as necessidades de um dos contextos em específico. Um *service* não está limitado a executar somente operações de um determinado contexto, mas sim a possuir sua lógica focada em um deles. Assim caso seja necessário um *service* pode utilizar as operações de outros *services* para efetivamente realizar uma tarefa que envolve vários contextos ao invés de implementar ou duplicar as operações do outro *service*.

Devido ao fato de não existir uma API ou base de dados para consumir os dados um novo *service* é criado, o ControlsService. Ele é o responsável por fazer o controle da geração de números identificadores dos usuários e tópicos. O ControlsService possui atributos e métodos que retornam a salvam o último id gerado para cada tópico e usuário (figura 17).

Como apresentado na figura 17, o *service* possui um atributo do tipo numérico para os números identificadores dos usuários e tópicos, possuindo também métodos para incrementar este número e atualizar os seus valores no armazenamento local do navegador. O armazenamento é realizado através do Objeto localStorage, uma instância da classe LocalStorageService, injetado no construtor e que possui os métodos para armazenamento, busca e demais operações no armazenamento local dos navegadores.

A classe LocalStorageService faz parte do pacote ngx-webstorage, importado no item 3.3.2 e possui um conjunto de classes para realizar o armazenamento de dados. Pensando nos testes e na persistência de dados foi decidido a utilização do LocalStorageService ao invés do SessionStorageService, cujos dados são armazenados na sessão do navegador, possuindo um período de existência muito curto prejudicando a eficiência dos testes e se distanciando de uma aplicação com dados oriundos de uma API.

Figura 17 – Captura de tela do service ControlsService.

```
export class ControlsService {

  constructor(private localStorage: LocalStorageService) {
    if (localStorage.retrieve( raw: 'lastUserId') != null) {
      this.lastId = localStorage.retrieve( raw: 'lastUserId');
    }
    if (localStorage.retrieve( raw: 'lastIdTopic') != null) {
      this.lastIdTopic = localStorage.retrieve( raw: 'lastIdTopic');
    }
  }

  private lastId: number = 0;
  private lastIdTopic: number = 0;

  incIdUser(): number {
    this.lastId++;
    this.localStorage.store( raw: 'lastUserId', this.lastId);
    return this.lastId;
  }

  incIdTopic(): number {
    this.lastIdTopic++;
    this.localStorage.store( raw: 'lastIdTopic', this.lastIdTopic);
    return this.lastIdTopic;
  }
```

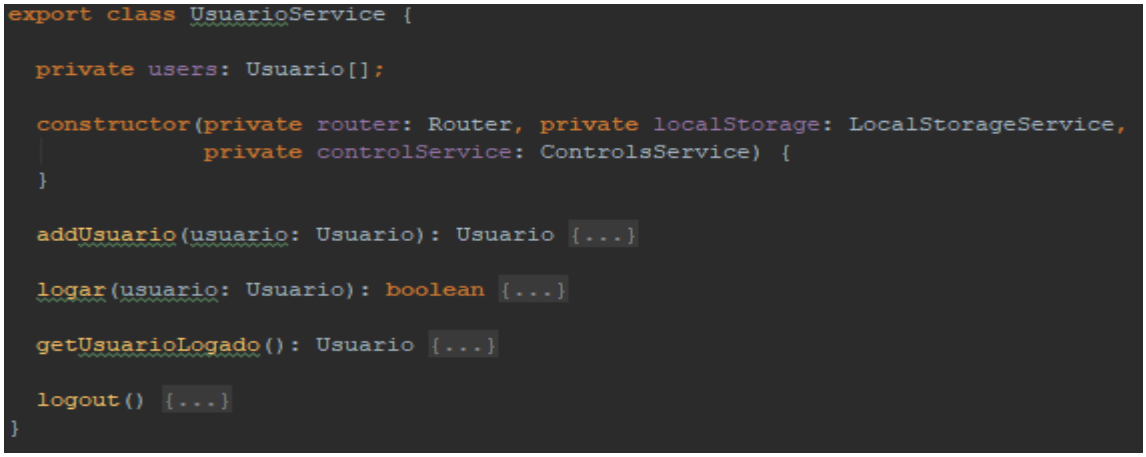
Fonte: Autoria própria.

Com a implementação do controlsService já é possível simular a geração de números identificadores semelhante à geração serial e única de diversos banco de dados, assim garantindo a identificação única de cada usuário de tópicos. Como o sistema não dará suporte à edição de respostas, as mesmas não irão possuir um identificador pois serão armazenadas dentro do array de respostas de cada tópico cadastrado no LocalStorage.

Para permitir o cadastro e busca dos usuários cadastrados, o *service* UsuarioService é criado. Seu objetivo é realizar todas as operações que dizem respeito a usuários como mostra na figura 18.

O UsuarioService possui os métodos necessários para realizar o cadastro de um novo usuário no sistema, bem como realizar o seu login e inclui-lo no armazenamento local como o usuário logado na sessão do navegador, sendo assim o único *service* que faz uso do SessionStorageService.

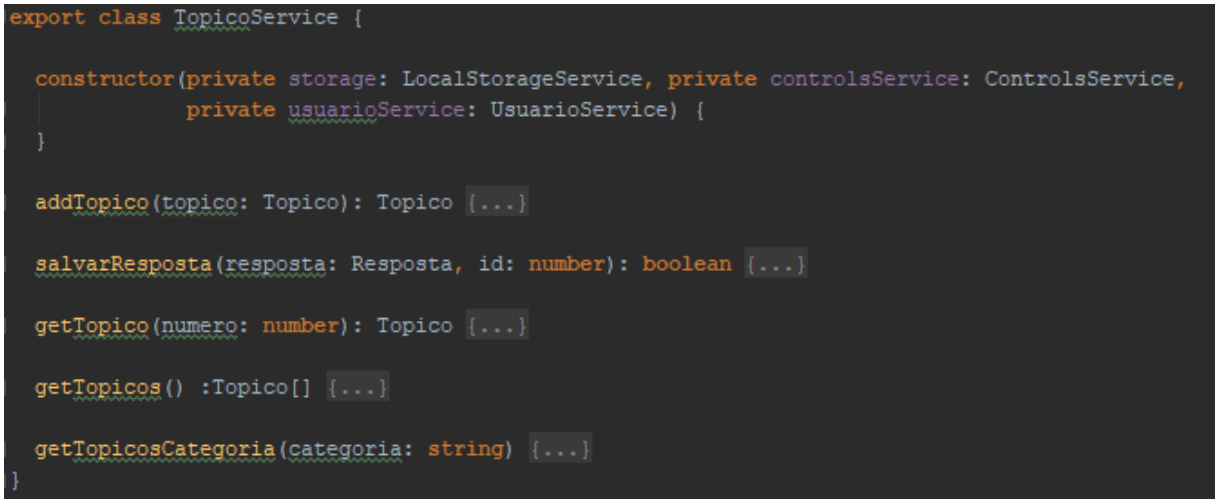
Figura 18 – Captura de tela do service UsarioService e seus atributos e métodos implementados



Fonte: Autoria própria.

Um último service é criado para atender as demandas referentes aos tópicos do fórum. O TopicoService é desenvolvido para realizar as operações referentes aos tópicos e respostas submetidas pelos usuários no sistema, assim garantindo o armazenamento dos tópicos e respostas, e também a busca dos mesmos para serem exibidos aos usuários (figura 19).

Figura 19 – Captura de tela do service TopicoService.



Fonte: Autoria própria.

3.3.4 Componentes

No Angular as páginas HTML são entendidas como partes de um componente. Toda página HTML deve possuir um componente typeScript associado a ela. Componentes typeScripts são classes que possuem a anotação @Component. Uma classe que possui essa anotação passará a carregar um template HTML sempre que a mesma tiver sido carregada em algum momento no sistema.

A inclusão da anotação não basta para vincular um template HTML a um componente, sendo necessário atribuir valores a alguns atributos presentes na anotação @Component. A atribuição de um template HTML a um componente pode ser feito através do atributo ‘template’ ou ‘templateUrl’. O primeiro irá receber como valor os próprios elementos HTML enquanto o segundo irá receber como valor uma string contendo o endereço relativo do arquivo HTML a ser carregado (figura 20).

Figura 20 – Captura de tela dos atributos básicos da anotação @Component.

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html'
})
export class LoginComponent {
```

Fonte: Autoria própria.

Para o projeto atual, apenas seis componentes principais são necessários, sendo eles cadastro, login, homepage, forum, novo-topico e tópico. O termo ‘principais’ utilizado na frase anterior significa que apenas aqueles componentes irão representar páginas completas e não somente uma porção da página.

3.3.4.1 Componente Login

O componente Login é responsável por exibir a interface de login, necessária para acessar o fórum. O componente typeScript da página de login possui a lógica necessária para verificar os campos do formulário e realizar a solicitação de login para

o objeto da classe UsuarioService, cujo é injetado no momento de acesso à página (figura 21).

Figura 21 – Captura de tela do componente typeScript Login.

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html'
})
export class LoginComponent {

  senha: any = '';
  login: any = ''

  constructor(private router: Router, private usuarioService: UsuarioService){

  }

  logar() {...}

}
```

Fonte: Autoria própria.

O componente typeScript Login é uma classe que assim como as demais classes do sistema, possui atributos. Os atributos nele existentes representam os campos do formulário existente no componente HTML (figura 22).

O Angular possui um sistema de ligação de dados (Data Binding). Esse sistema permite que o desenvolvedor realize uma ligação entre as variáveis presentes do componente typeScript da página e elementos HTML como por exemplo o elemento <input>.

O data binding pode ser realizado de forma uni-direcional ou bi-direcional, significando que no segundo caso, a alteração do valor em um componente irá afetar o valor da variável presente no outro, e vice-versa. A figura 23 mostra a utilização do *data binding* bidirecional entre o componente TypeScript e HTML da página de login. Através do atributo [(ngModel)] é feita a ligação entre os campos do formulário a seus respectivos atributos do componente typeScript.

Figura 22 – Captura de tela parcial da tela de Login da aplicação.

Fonte: Autoria própria.

Figura 23 – Captura parcial do código HTML do componente Login.

```
<form (submit)="logar()">
  <label>Usuário</label>
  <input class="form-control" type="text" [(ngModel)]="login" placeholder="Informe o usuário" name="login">
  <label>Senha</label>
  <input class="form-control" type="password" [(ngModel)]="senha" placeholder="Insira a senha" name="senha">
  <br>
  <button class="form-control btn btn-primary">Logar</button>
</form>
```

Fonte: Autoria própria

Com a utilização do [(ngModel)], todos os valores inseridos nos campos dos formulários serão enviados automaticamente aos atributos do componente typeScript cujos nomes forem os mesmos dos valores informados no HTML.

Todos os métodos implementados no componente typeScript, e que possuírem visibilidade pública, poderão ser acessados pelo componente HTML, como é mostrado na figura 23 através do atributo (submit). O atributo (submit) é uma reimplementação feita pelo Angular do atributo onSubmit nativo do HTML. Esse

atributo irá executar um comando typeScript quando o formulário for enviado através do botão ‘Logar’.

A figura 24 apresenta a implementação do método logar(). Esse método irá realizar as validações desejadas no formulário com base nos atributos ‘senha’ e ‘userName’ e então realizar a autenticação do usuário.

Figura 24 – Captura de tela do método de login do componente typeScript Login.

```
logar() {
  if (this.senha.length > 0 && this.login.length > 0) {
    let usuario: Usuario = new Usuario();
    usuario.userName = this.login;
    usuario.senha = this.senha;
    if (this.usuarioService.logar(usuario)) {
      this.router.navigate( commands: ['forum']);
    }
  }
}
```

Fonte: Autoria própria.

O método de login faz uso do método ‘logar’ do service UsuárioService, o qual irá retornar um valor booleano, indicando sucesso ou falha na tentativa de login.

3.3.4.2 Componente Cadastro

Se desejado os usuários podem realizar o cadastro na aplicação para ter acesso aos fóruns. O cadastro é realizado através do link indicativo abaixo do botão de login. A tela de cadastro conta com um formulário com os campos semelhantes aos campos da tela de login (figura 25).

Conforme a figura 25 exibe, o formulário solicita que o usuário informe um nome para sua conta, assim como uma senha e uma confirmação da senha. Da mesma forma que o formulário de login, o formulário de cadastro faz uso do data binding entre os campos do formulário e os atributos de mesmo nome no componente typeScript (figura 26).

Figura 25 – Captura de tela da interface de cadastro de usuário.

Fonte: Autoria própria.

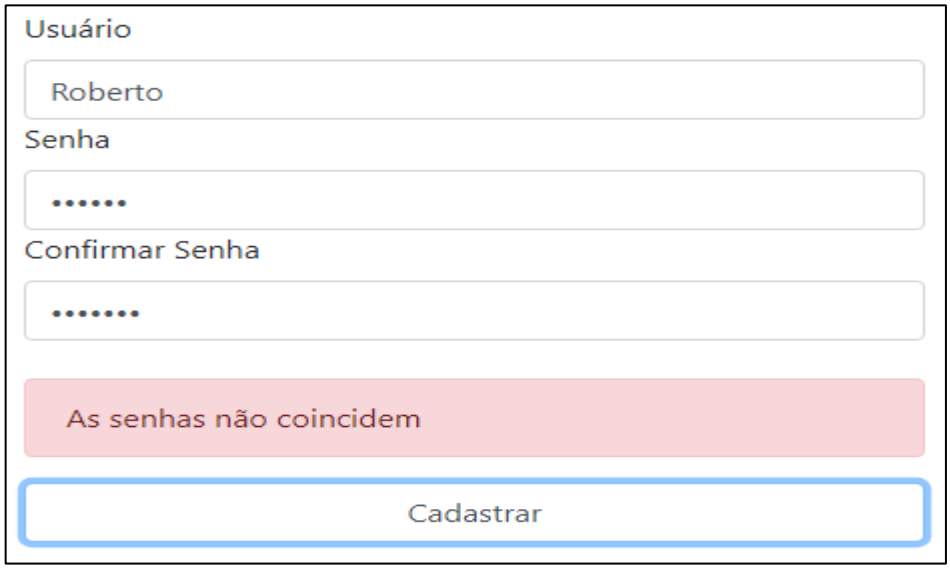
Figura 26 – Captura de tela do código do formulário de cadastro do componente HTML.

```
<form (submit)="cadastrar()">
  <label>Usuário</label>
  <input class="form-control" [(ngModel)]="userName" type="text" name="userName"
  placeholder="Informe o nome do usuário">
  <label>Senha</label>
  <input class="form-control" [(ngModel)]="senha" type="password" name="senha"
  placeholder="Insira uma senha">
  <label>Confirmar Senha</label>
  <input class="form-control" [(ngModel)]="confirmarSenha" type="password" name="confirmarSenha"
  placeholder="Digite a senha novamente">
  <br>
  <div *ngIf="mensagem != null" class="alert alert-danger">{{mensagem}}</div>
  <button class="form-control btn btn-primary">Cadastrar</button>
</form>
```

Fonte: Autoria própria.

O formulário de cadastro é validado pelos atributos do componente typeScript e em caso de erros, senhas inválidas ou nomes repetidos é exibida uma mensagem para o usuário. Essa mensagem contém a diretiva *ngIf e somente é exibida quando o atributo ‘mensagem’ do componente typeScript não for nulo. O atributo ‘mensagem’ irá receber um valor em string em caso de algum erro no cadastro, deixando de ser nulo, e consequentemente sendo exibido na interface para o usuário, contendo a descrição do erro (figura 27).

Figura 27 – Captura de tela do formulário de cadastro contendo erros de inserção.



Fonte: Autoria própria.

Quando o usuário pressionar o botão ‘Cadastrar’ será executado o método cadastrar() no componente typeScript. Se os campos do formulário forem válidos o método irá criar uma instância de usuário e atribuir os valores de acordo com os dados inseridos no formulário. Em seguida esse usuário é enviado como parâmetro no método addUsuario() do UsuarioService (figura 28). Ao final do processo de cadastro o usuário será redirecionado para a tela inicial (HomePage) da aplicação.

Figura 28 – Captura de tela do método cadastrar().

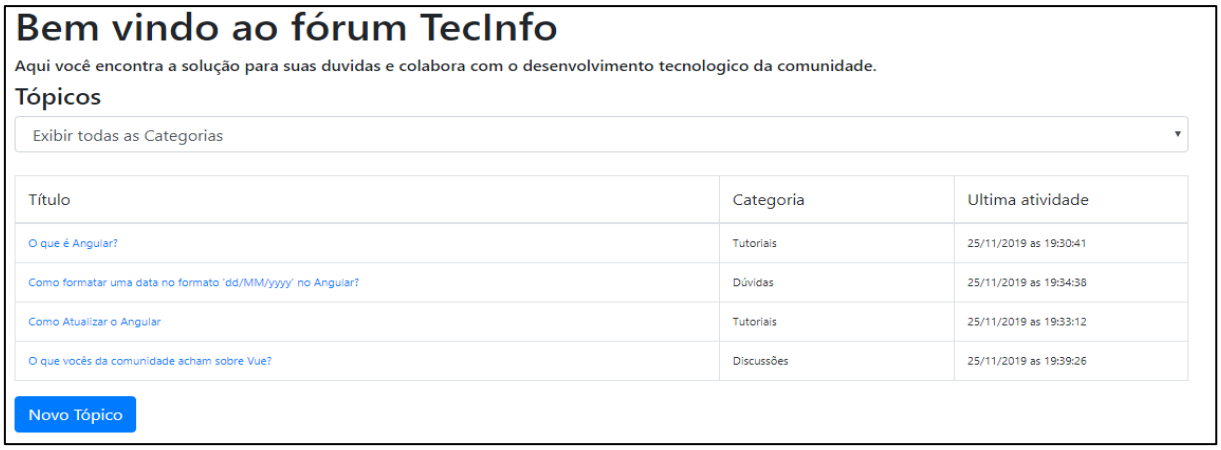
```
cadastrar() {
  if (this.senha.length > 3 && this.userName.length > 5 && this.confirmarSenha.length > 5) {
    if (this.confirmarSenha == this.senha) {
      let usuario: Usuario = new Usuario();
      usuario.senha = this.senha;
      usuario.userName = this.userName;
      let usr = this.usuarioService.addUsuario(usuario);
      if (usr != null) {
        if (this.usuarioService.logar(usr)) {
          this.router.navigate( commands: ['/forum']);
        }
      } else {
        this.mensagem = "Usuário já cadastrado. Escolha outro nome de usuário";
      }
    } else {
      this.mensagem = "As senhas não coincidem";
    }
  } else {
    this.mensagem = "Preencha o formulário corretamente";
  }
}
```

Fonte: Autoria própria.

3.3.4.3 Componente HomePage

Após finalizar o cadastro ou efetuar o login, o usuário será redirecionado para a página principal do fórum (HomePage). Nessa página o usuário irá visualizar todos os tópicos já submetidos pelos usuários. Uma opção de filtragem por categoria é disponibilizada para o usuário, e por padrão é definida como ‘Exibir todas as categorias’ (figura 29).

Figura 29 – Captura de tela da tela inicial do fórum.

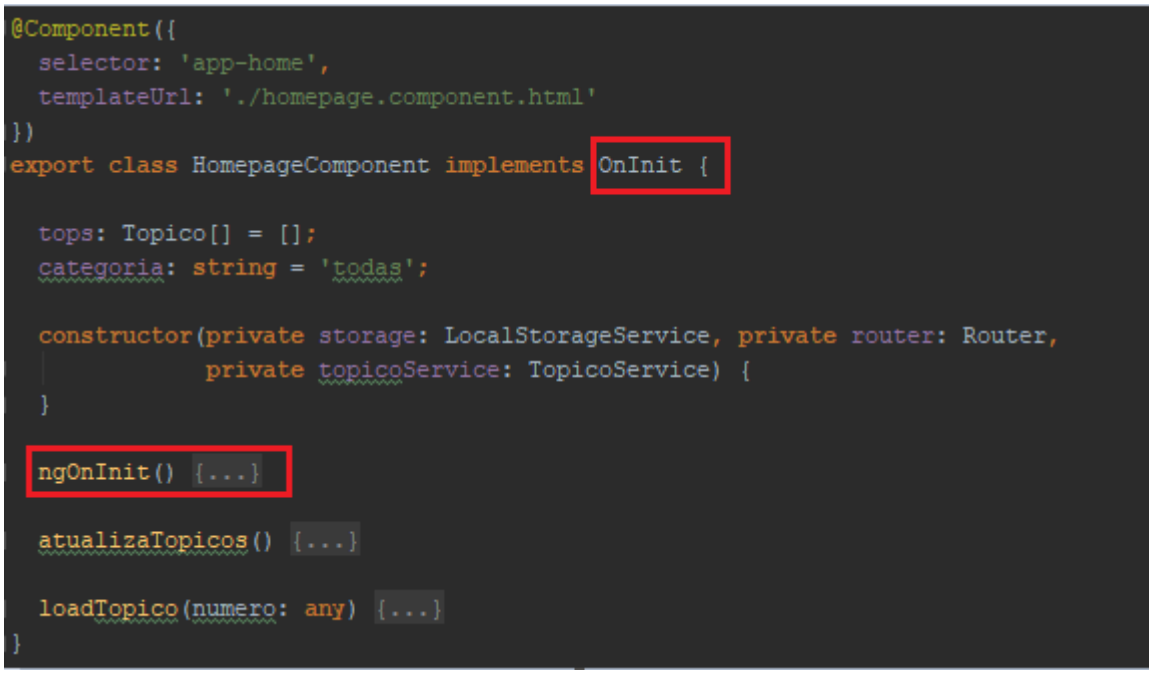


Fonte: Autoria própria.

No momento que a página principal é carregada, o método ngOnInit é executado. Esse método é a implementação concreta do método declarado na interface OnInit, a qual deve ser implementada no componente (figura 30).

A implementação da interface OnInit é uma maneira mais personalizada e correta de trabalhar com a iniciação de componentes, uma vez que o construtor deve (por convenção) ficar responsável pelas injeções de dependências e na instanciação do próprio componente. O método ngOnInit sempre é executado no momento que um componente for acessado, e pode inclusive ser executado por outro método.

Figura 30 – Captura de tela do componente typeScript HomePage.



Fonte: Autoria própria.

O método ngOnInit() é responsável por buscar no armazenamento local todos os tópicos submetidos pelos usuários, e em todas as categorias. Em seu escopo há apenas a execução do método getTopicos() do TopicoService, o qual foi injetado como dependência no construtor do componente (figura 31).

Figura 31 – Captura de tela do construtor e método ngOnInit do componente Homepage.

```
constructor(private storage: LocalStorageService, private router: Router,
            private topicoService: TopicoService) {
}

ngOnInit() {
  this.tops = this.topicoService.getTopicos();
}
```

Fonte: Autoria própria.

Conforme a figura 31, os tópicos são armazenados no atributo ‘tops’, um array de objetos da classe Topico. Esses tópicos são utilizados no elemento HTML e iterados através da diretiva estrutural do Angular *ngFor (figura 32). As diretivas são marcadores em um elemento DOM (Document Object Model) que comunicam o Angular que um determinado elemento deve realizar um determinado comportamento.

Figura 32 – Captura de tela da utilização da diretiva *ngFor.

```
<table class="table table-bordered">
  <thead>
  <tr><td width="60%">Titulo</td><td width="20%">Categoria</td><td width="20%">Ultima atividade</td></tr>
  </thead>
  <tbody style="...">
  <tr *ngFor="let t of tops">
    <td><a routerLink (click)="loadTopico(t.numero)">{{t.titulo}}</a></td>
    <td>{{t.categoria}}</td><td>{{t.ultimaAtualizacao | date:'dd/MM/yyyy'}} as {{t.ultimaAtualizacao | date: 'HH:mm:ss'}}</td>
  </tr>
  </tbody>
</table>
```

Fonte: Autoria própria.

A diretiva *ngFor é uma diretiva estrutural que é utilizada para realizar a iteração em elementos de um array ou de um objeto no HTML. Através da declaração e utilização de uma variável local o *ngFor percorre o array e reproduz o conteúdo do elemento e elementos filhos que está vinculado para cada iteração. Na figura 32 a diretiva está utilizada no elemento <tr> da tabela e anexando um novo elemento <tr> à tabela de forma dinâmica. O elemento <tr> representa uma linha da tabela, sendo assim a cada iteração uma linha da tabela, contendo os dados de cada tópico será

gerada. Os valores dos atributos do tópico iterado são anexados ao HTML como texto plano por meio do uso da interpolação (figura 33).

Figura 33 – Captura de tela do uso da interpolação no componente HTML HomePage.

```
<td><a routerLink (click)="loadTopico(t.numero)">{{t.titulo}}</a></td>
<td>{{t.categoria}}</td><td>{{t.ultimaAtualizacao | date:'dd/MM/yyyy'}}
```

Fonte: Autoria própria.

A interpolação é uma forma de acessar os dados de variáveis typeScript no documento HTML e é realizada através dos símbolos ‘{{‘ (duplo abre chaves) e ‘}}’ (duplo fecha chaves). Quando um valor se encontrar contido entre as chaves duplas significa que está sendo utilizado interpolação, informando o angular que os valores da variável informada devem ser carregados.

A interpolação irá localizar variáveis no componente HTML e TypeScript que sejam compatíveis com o nome ou objeto informado e em caso de inexistência de variáveis compatíveis será exibido um erro no console do navegador, informando que não foi encontrado objeto e/ou atributos com o nome informado na interpolação.

O filtro de categorias é realizado no momento da mudança da categoria selecionada. A detecção da mudança é feita pelo atributo (change), e que assim como (submit) é uma reimplementação do onChange nativo do HTML para ser utilizado com expressões do typeScript. O atributo (change) recebe como valor a chamada do método atualizaTopicos() (figura 34).

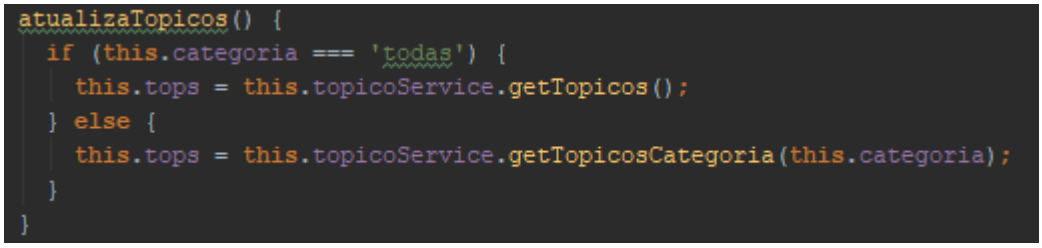
Figura 34 – Captura de tela do elemento HTML de seleção de categorias do componente HomePage.

```
<select [(ngModel)]="categoria" (change)="atualizaTopicos()" class="form-control">
  <option selected value="todas">Exibir todas as Categorias</option>
  <option value="Anúncios">Anúncios</option>
  <option value="Discussões">Discussões</option>
  <option value="Dúvidas">Dúvidas</option>
  <option value="Tutoriais">Tutoriais</option>
</select>
```

Fonte: Autoria própria.

A figura 34 também mostra a utilização de *data binding* do valor selecionado com o atributo ‘categoria’ presente no componente typeScript HomePage, alterando o valor da categoria no momento da seleção. No momento que o usuário realizar a mudança da categoria o método atualizaTopicos() será executado e de acordo com a categoria selecionada irá realizar a busca filtrada dos tópicos utilizando o TopicoService (figura 35).

Figura 35 – Captura de tela do método atualizaTopicos() do componente TypeScript HomePage.



Fonte: Autoria própria.

Como mostra a figura 35, o método irá realizar a filtragem dos tópicos de acordo com o valor do atributo ‘categoria’ e através do método getTopicosCategoria() presente no TopicoService e receberá um novo array de tópicos da categoria escolhida. Devido ao *data binding*, toda alteração no array de tópicos ‘tops’ resultará na atualização imediata nos valores da tabela de tópicos na interface.

3.3.4.4 Componente Novo Topico.

Por meio do botão ‘Novo Tópico’, na Página Inicial, o usuário será redirecionado para a página de submissão de um tópico. Essa página contém um formulário para inserir um novo tópico, possuindo uma seleção de categoria, um campo para o título e outro campo para a descrição do tópico (figura 36).

Figura 36 – Captura de tela da interface do formulário de submissão de um novo tópico.

Fonte: Autoria própria.

A implementação do formulário segue os mesmos princípios apresentados nos itens anteriores, fazendo o uso do data binding. Pensando na praticidade e na facilidade de implementação, o data binding é feito diretamente nos atributos de um objeto da classe Topico (figura 37).

Figura 37 – Captura de tela do uso do data binding em atributos de um objeto.

```
<div class="col-md-9">
  <label>Informe o título do tópico</label>
  <input class="form-control" [(ngModel)]="topico.titulo" placeholder="Informe o título do tópico" type="text">
</div>
<div class="col-md-3">
  <label>Selecione a categoria</label>
  <select [(ngModel)]="topico.categoria" class="form-control">
    <option selected value="Anuncios">Anúncios</option>
    <option value="Discussões">Discussões</option>
    <option value="Dúvidas">Dúvidas</option>
    <option value="Tutoriais">Tutoriais</option>
  </select>
</div>
```

Fonte: Autoria própria.

O objeto da classe Topico é declarado no componente TypeScript e por meio da implementação da interface OnInit ele é instanciado e inicializado no método ngOnInit() (figura 38). Para o correto funcionamento do data binding em atributos de um objeto é preciso instanciar o objeto para que seus atributos, embora muitas vezes nulo, possam ser acessados. Na tentativa de acesso a um atributo de um objeto nulo ocorrerá uma exceção.

Figura 38 – Captura de tela da declaração e inicialização do objeto tópico.

Fonte: Autoria própria.

A utilização do data binding associado diretamente a um objeto ou atributos de um objeto é utilizada para agilizar o processo de submissão do tópico bem como também reduzir os gastos em memória na declaração de atributos que futuramente serão inseridos em um objeto Topico para ser cadastrado.

O elemento que compõem a descrição do tópico faz parte dos pacotes adicionais ngx-quill e quill instalados no item 3.3.2. Tais pacotes entregam um componente desenvolvido para dar suporte a formatações de texto, inclusão de imagens, listas e demais ferramentas de formatação existentes em blogs ou redes sociais. Os componentes que são utilizados do pacote são o quill-editor e o quill-view.

O componente quill-editor possui uma configuração interna e que permite a personalização das opções de formatações disponíveis ao usuário. Assim como os demais pacotes, ele precisa estar incluso no arquivo de módulos, e através do arquivo de módulo que sua configuração é feita (figura 39).

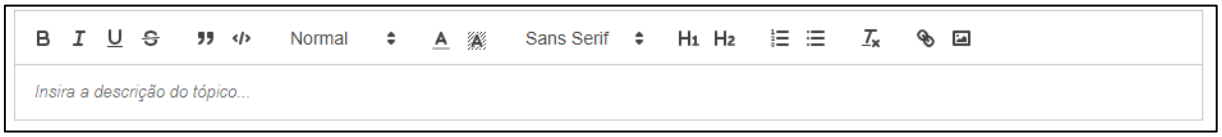
A configuração mostrada na figura 39 define as opções de formatações disponíveis no editor HTML, sendo elas formatação da cor, tamanho, estilo e fonte da letra, inclusão de listas, anexo de imagens, inserção de bloco de código, atalhos para títulos e subtítulos, cor de fundo do texto, links e bloco de citação. Essas configurações são acessadas por todos os componentes quill-editor em todo o projeto, assim uma configuração diferenciada. A figura 40 mostra todas as opções de personalização disponíveis ao usuário baseada na configuração realizada.

Figura 39 – Inclusão e configuração do editor do componente quill-editor no arquivo de módulo forum.component.module.ts.

```
imports: [
  BrowserModule,
  RouterModule.forRoot(AppRoutes),
  FormsModule,
  QuillModule.forRoot( config: {
    modules: {
      toolbar: [
        ['bold', 'italic', 'underline', 'strike'],
        ['blockquote', 'code-block'],
        [{'size': ['small', false, 'large', 'huge']}],
        [{'color': []}, {'background': []}],
        [{'font': []}],
        [{'header': 1}, {'header': 2}],
        [{'list': 'ordered'}, {'list': 'bullet'}],
        ['clean'],
        ['link', 'image']
      ]
    }
  })
],
```

Fonte: Autoria própria.

Figura 40 – Captura de tela das opções de formatação do componente quill-editor.



Fonte: Autoria própria.

O componente quill-editor é incluído na página HTML por meio do elemento <quill-editor> e possui alguns atributos para ser configurado. Um deles é o atributo ‘format’ que define o formato que as informações serão salvas (figura 41). O formato definido afeta a maneira que serão exibidas as informações. Os formatos disponíveis são: texto plano, html e objeto.

Figura 41 – Captura de tela do elemento quill-editor no componente NovoTopico.

```
<label>Informe a descrição do tópico</label>
<quill-editor placeholder="Insira a descrição do tópico..."
              class="text-light" [(ngModel)]="topico.descricao" format="object"></quill-editor>
```

Fonte: Autoria própria.

Para o desenvolvimento desse trabalho foi definido o formato objeto. Esse formato armazena o texto inserido, assim como imagens e formatações em um objeto complexo, armazenando quando, onde e qual formatação foi aplicada. Pensando nisso o atributo ‘descrição’ da classe Topico é do tipo ‘any’, podendo comportar qualquer tipo de dado, inclusive objetos. Assim como os demais campos da página os dados inseridos e formatados são automaticamente atribuídos à variável ‘descricao’ do objeto ‘topico’ através do data binding.

O tópico é submetido por meio do botão ‘Submeter tópico’. O botão possui um atributo (click) cuja instrução presente como valor será executada no momento uma interação de clique for executada sobre o botão. O (click) é outro atributo reimplementado pelo Angular para utilizar em elementos nativos ou não do HTML.

Uma vez que os campos estejam corretamente preenchidos e o botão de ‘submeter tópico’ for pressionado o método ‘novoTopico()’ será executado. Esse método é o responsável por enviar o novo tópico para o armazenamento local por meio dos métodos do TopicoService (figura 42).

Figura 42 – Captura de tela do método ‘novoTopico()’ do componente NovoTopico.

```
novoTopico() {
  if (this.topico.titulo == '' || this.topico.titulo == null) {
  } else {
    this.topico.aberto = true;
    let t = this.topicoService.addTopico(this.topico);
    if (t != null) {
      this.router.navigate( commands: ['/forum/topico'], extras: {queryParams:{id: t.numero}});
    }
  }
}
```

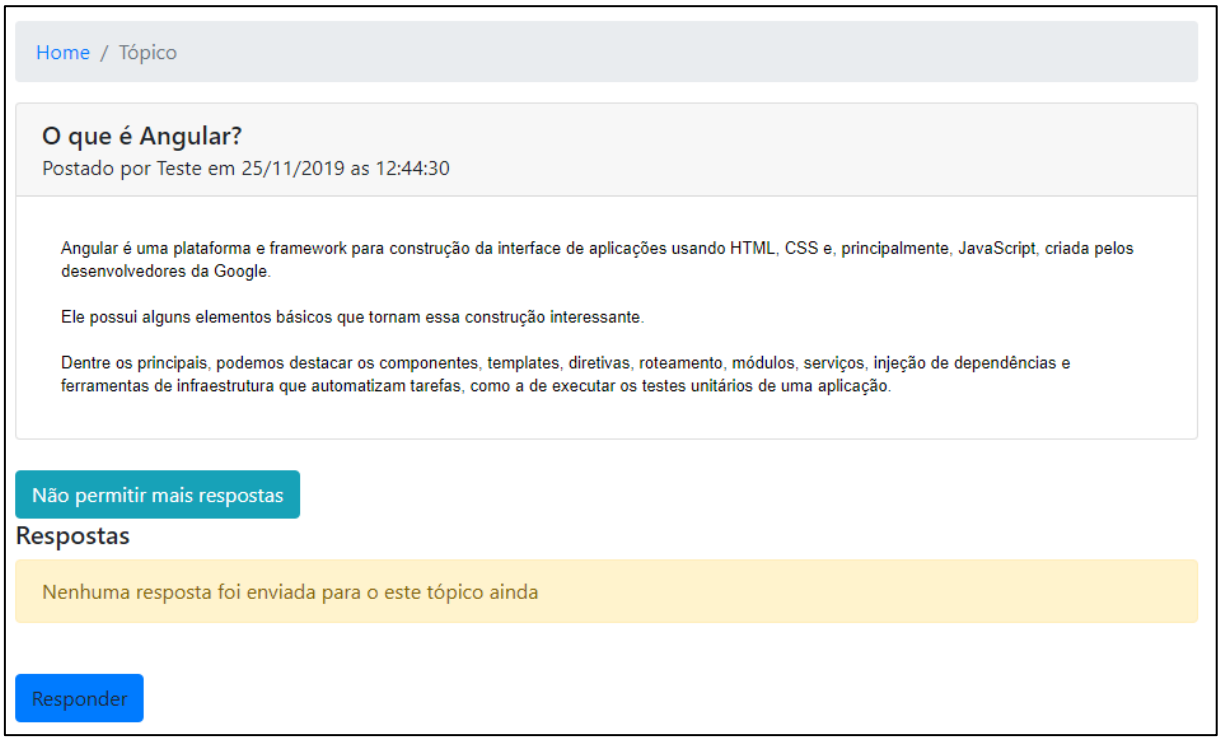
Fonte: Autoria própria.

Como apresentado na figura 42, a submissão de um novo tópico é feita após o TopicoService retornar um valor não nulo de volta ao componente. O valor retornado pelo *service* é o próprio tópico armazenado, porém contendo seu número gerado pelo ControlsService. Esse número é utilizado para reencaminhar o usuário para a página da visualização do tópico recém submetido.

3.3.4.5 Componente Topico

O usuário tem acesso a esse componente através da página inicial, clicando em algum tópico listado, ou após o termino da submissão de um novo tópico, que irá resultar em um reencaminhamento para página do tópico recém inserido. A página do tópico é composta basicamente pelo tópico inserido pelo usuário, sua respectiva descrição e pelas respostas enviadas pelos usuários, se existentes (figura 43).

Figura 43 – Captura de tela da página do tópico inserido.



Fonte: Autoria própria.

O componente Topico é o componente que exibe as informações de um tópico e então permite que os usuários interajam através de respostas. Para ter acesso aos dados do tópico em questão, é implementado o método ngOnInit(), o qual irá verificar a URL da página com o objetivo de encontrar uma variável de nome ‘id’.

O id encontrado na URL será utilizado para buscar os dados do tópico referente àquele id. A busca é realizada pelo TopicoControler, injetado como dependência no construtor do componente. A figura 44 mostra os detalhes da implementação.

Figura 44 – Captura de tela da implementação da busca dos dados de um tópico selecionado

Fonte: Autoria própria.

No momento que o método ngOnInit() é iniciado o método loadRespostas() também será executado, esse de fato realizando a busca pelo ID na url. A busca de informações referentes à URL é realizada pelo objeto actived Route. Através do método ‘queryParams’ é possível procurar por parâmetros presentes na URL e capturar seus valores. Uma vez encontrado o valor do ID o topicoService irá utilizá-lo para buscar o tópico condizendo com o ID e então atribui-lo ao objeto ‘topico’ declarado no componente typeScript. Os valores do atributo ‘topico’ são acessados por meio do data binding no componente HTML.

A fim de poder exibir a descrição do tópico devidamente formatada e customizada, é necessário a utilização do componente quill-view, o qual irá interpretar

a informação formatada da descrição do tópico, a qual foi inserida no quill-editor, e então apresentá-la de forma correta ao usuário (figura 45).

Figura 45 – Captura de tela do componente quill-view do componente Topico.

```
<div class="card-body">
  <p class="card-text">
    <quill-view format="object" [content]="topico.descricao"></quill-view>
  </p>
</div>
```

Fonte: Autoria própria.

Assim como o quill-editor, o quill-view também possui um atributo ‘format’, o qual deve possuir o mesmo valor presente no mesmo atributo do quill-editor. Divergências de formatos podem resultar em uma exibição incorreta dos dados inseridos no quill-editor ou até mesmo não exibir nenhuma informação.

A exibição da descrição do tópico de maneira formatada é realizada através da passagem do objeto gerado pelo quill-editor no atributo ‘[content]’ do componente quill-view. Conforme a figura 45, essa passagem do valor é feita através do *data binding*.

Conforme definido nas especificações, o usuário que enviou o tópico tem a liberdade de permitir ou bloquear o envio de respostas a seu tópico. Através do botão ‘Não permitir mais respostas’ ou ‘Abrir para respostas’ um método irá ser executado para bloquear ou liberar o envio de respostas para o tópico respectivamente (figura 46).

Ambos os métodos implementados na figura 46 efetuam a alteração do status ‘aberto’ do tópico, e então, através do TopicoService, realizam a alteração da informação no armazenamento local. A figura 47 demonstra um tópico fechado para respostas.

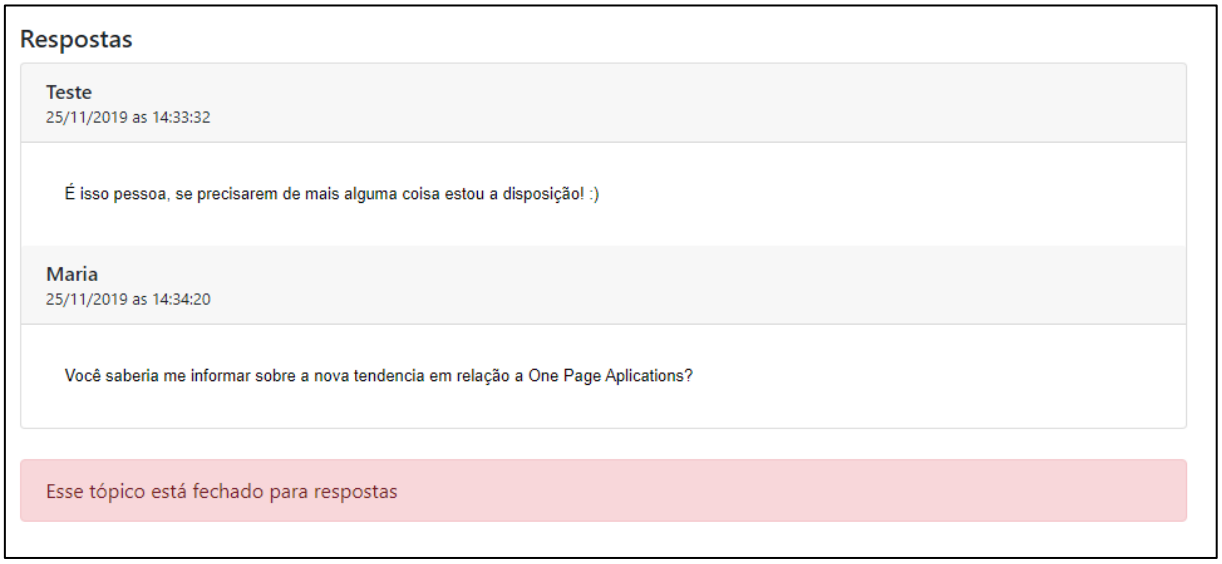
Figura 46 – Captura de tela dos métodos de liberação e bloqueio de envio de respostas em um tópico.

```
abrirTopico() {
  this.topico.aberto = true;
  this.topicoService.atualizaAberto(this.topico);
}

fechaTopico() {
  this.topico.aberto = false;
  this.topicoService.atualizaAberto(this.topico);
}
```

Fonte: Autoria própria.

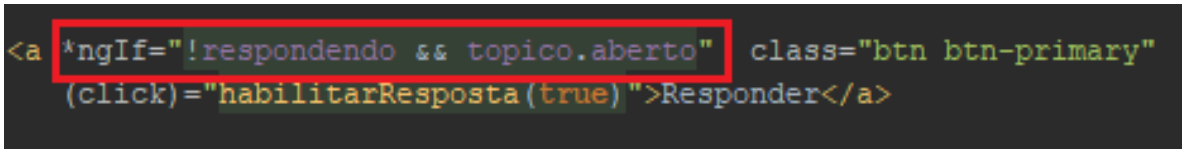
Figura 47 – Captura de tela de um tópico fechado para respostas.



Fonte: Autoria própria.

A habilitação do botão de responder um tópico é realizada através da diretiva estrutural *ngIf, cujo objetivo é adicionar ou remover um componente com base em uma expressão booleana. Os usuários somente terão visibilidade do botão de resposta se o status ‘aberto’ do tópico em questão for verdadeiro, caso contrário o botão não será gerado na página (figura 48).

Figura 48 – Captura de tela da habilitação condicional do botão de responder ao tópico.



Fonte: Autoria própria.

Conforme a figura 48, o botão que habilita o usuário a digitar sua resposta só irá existir na página se o atributo ‘aberto’ do tópico estiver em verdadeiro. A diretiva *ngIf tem como escopo todo o documento HTML e os componentes typeScript, assim como todo e qualquer atributo de visibilidade publica que esteja relacionado ao componente HTML. Ele é completo e da suporte a todas as operações que podem ser realizadas no comando “ IF ” do typeScript.

Durante o período que um tópico estiver com o status de aberto para respostas, será possível enviar novas respostas para o tópico através do botão ‘Responder’. A ação de clique no botão executa um método que habilita ao usuário um componente quill-editor para realizar a digitação da resposta (figura 49).

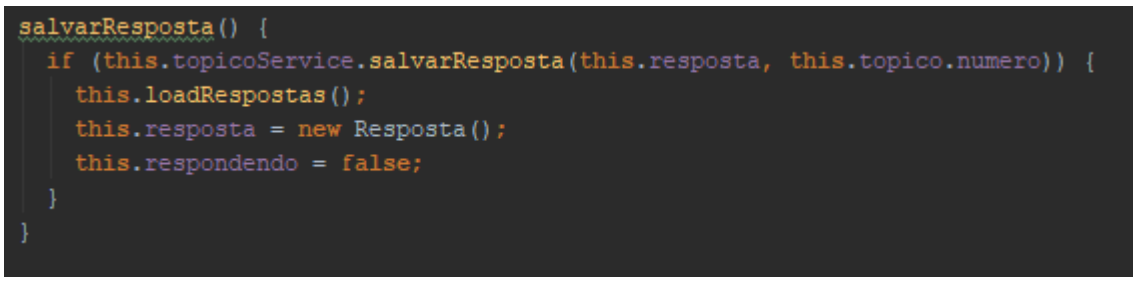
Figura 49 – Captura de tela do componente HTML utilização da diretiva *ngIf em conjunto com diferentes componentes

Fonte: Autoria própria.

Na figura 49 é utilizado a diretiva *ngIf para habilitar ou não o campo de digitação do quill-editor, assim como também os botões de Enviar e Cancelar, os quais não devem ser exibidos se uma resposta não estiver sendo digitada.

O processo de formatação e armazenamento da informação da descrição do quill-editor das respostas é o mesmo dos tópicos, uma resposta possuindo um atributo ‘descricao’ do tipo any. No momento que o usuário clicar em ‘Enviar’ o método salvarResposta() será executado através do atributo (click). Esse método irá enviar como parâmetro os dados do quill-editor para o método salvarResposta() do *service* TopicoService, assim como também enviar o número identificador do tópico (figura 50).

Figura 50 – Captura de tela do método salvarResposta() do componente Topico.



Fonte: Autoria própria.

Sempre que a página for acessada ou após a inserção de uma nova resposta, o método loadRespostas() será executado, e assim através do TopicoService irá obter o tópico contendo todas as suas respostas. Uma vez que existam respostas para um tópico, elas serão exibidas na interface após o título e descrição do tópico.

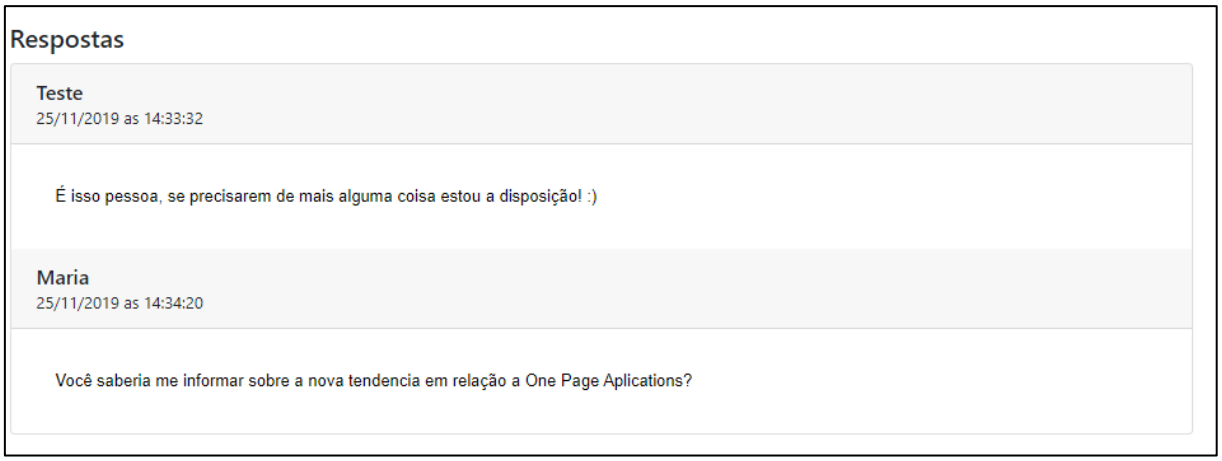
Como um tópico possui um número indeterminado de respostas, é preciso tornar a inclusão das mesmas de forma dinâmica. Para isso é utilizada a diretiva *ngFor. Ela irá iterar sobre todas as respostas presentes no array de respostas do objeto topico no componente, e assim gerar um componente quill-view para cada iteração, completando-o com as informações do usuário, data de publicação e descrição da resposta, em ordem crescente (figuras 51 e 52).

Figura 51 – Captura de tela da iteração *ngFor no componente HTML para as respostas de um tópico.

```
<div *ngFor="let r of topico.respostas">
  <h6 class="card-header">{{r.usuario?.userName}}<br>
    <small>{{r.data | date: 'dd/MM/yyyy'}} as {{r.data | date: 'HH:mm:ss'}}</small>
  </h6>
  <div class="card-body">
    <p class="card-text">
      <quill-view format="object" [content]="r.descricao"></quill-view>
    </p>
  </div>
</div>
```

Fonte: Autoria própria.

Figura 52 – Captura de tela das respostas de um tópico geradas pela diretiva *ngFor.



Fonte: Autoria própria.

3.3.3 Módulos

O Angular exige que todos os componentes que serão carregados na aplicação estejam presentes em algum módulo do projeto. Módulos são arquivos typeScript que definem todos os componentes que estão presentes na aplicação e que em algum momento serão utilizados, sejam por outros componentes ou em alguma injeção de dependência. Um módulo realiza o agrupamento das classes utilizadas no projeto, sendo semelhante às bibliotecas em outras linguagens de programação.

Cada módulo importado em outros módulos representa uma biblioteca de funções, classes e componentes já implementados e que serão utilizados no projeto, como por exemplo o módulo FormsModule (figura 53) que define os componentes e conjunto de funções necessárias para realizar a comunicação entre os componentes nativos dos formulários HTML e as diretivas do Angular.

Em uma aplicação Angular todos os componentes que são utilizados na aplicação devem estar declarados, importados ou exportados em arquivos de módulos do projeto. Para uma aplicação Angular executar corretamente é preciso que pelo menos um arquivo de módulos esteja presente no projeto e que possua a declaração de todos os componentes utilizados. A utilização dos módulos nos projetos Angular pode ser realizada em módulos menores e separados, desde que os mesmos sejam importados em algum módulo do projeto e carregados pelo módulo principal. Por padrão o módulo principal é gerado com o nome app.module.ts (figura 53) e é encontrado dentro do diretório ‘src/app’.

Figura 53 – Captura de tela do arquivo app.module.ts.

```
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot(AppRoutes),
    FormsModule,
    ForumComponentModule,

  ],
  providers: [LocalStorageService, AuthGuard, {provide: LocationStrategy, useClass: HashLocationStrategy}],
  bootstrap: [AppComponent],
  //exports: [MenuSuperiorComponent]
})

export class AppModule {
}
```

Fonte: Autoria própria.

Um arquivo módulo é uma classe que possui a anotação @NgModule e todos os componentes utilizados na aplicação devem ser inseridos no array do atributo ‘declarations’ da anotação.

Como mostra a figura 53, o arquivo de módulo principal do projeto tem conhecimento apenas as declarações dos componentes LoginComponent e AppComponent. Por ser o módulo principal da aplicação, espera-se que o AppComponent esteja contido dentro do mesmo. No projeto atual foi decidido, para fins de organização, separar os módulos, assim criando o módulo ForumComponentModule que irá conter todos os componentes referentes as páginas do fórum (figura 54) e importando-o no módulo principal.

A declaração dos componentes deve ser feita uma única vez, não sendo permitido a declaração do mesmo componente em diferentes módulos. Dessa forma os módulos devem ser importados no módulo principal obrigatoriamente dentro do array ‘imports’ para que todos os componentes sejam conhecidos no projeto, seja em suas declarações ou através das importações.

Figura 54 – Captura de tela do arquivo forum.component.module.ts.

```
@NgModule({
  declarations: [
    HomepageComponent,
    NovoTopicoComponent,
    TopicoComponent,
    MenuSuperiorComponent,
    ForumComponent
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot(AppRoutes),
    ],
  providers: [],
  bootstrap: [AppComponent],
  //exports: [MenuSuperiorComponent]

})
export class ForumComponentModule {
}
```

Fonte: Autoria própria.

3.3.4 Rotas

3.3.4.1 Rotas e Componentes

Uma característica importante em aplicações Angular é a presença de um arquivo de rotas. Esse arquivo é responsável por realizar o vínculo dos componentes a um endereço de url, bem como definir redirecionamentos e condições de acesso aos *templates* com base em uma regra ou hierarquia de navegação. O arquivo de rotas gerado por padrão no seção Criação do projeto é gerado automaticamente no diretório ‘src/app’ do projeto com nome app-routing.module.ts.

O arquivo de rotas pode ser configurado de diferentes formas. Uma dessas formas é definir uma anotação @NgModule no arquivo e dentro desta anotação importar o arquivo RouterModule informando como parâmetro de seu método ‘forRoot’ uma lista de rotas para acesso em todo o projeto. Porém, para o projeto atual a configuração é realizada de outra forma, através da remoção da anotação @NgModule do arquivo de rotas e importando o módulo RouterModule nas importações do appModule, como apresentado na figura 54.

O projeto atual requer poucas rotas visto que seu objetivo é dar uma visão geral do framework e suas ferramentas, sendo assim as páginas presentes no sistema são login, cadastro, fórum, novo-topico e tópico (figura 55).

Figura 55 – Captura de tela do arquivo app-routing.module.ts

```
export const AppRoutes: Routes = [
  {path: '', redirectTo: 'login', pathMatch: 'full'},
  {path: 'login', component: LoginComponent},
  {path: 'cadastro', component: CadastroComponent},
  {
    path: 'forum', component: ForumComponent,
    children: [
      {path: '', component: HomepageComponent},
      {path: 'novo-topico', component: NovoTopicoComponent},
      {path: 'topico', component: TopicoComponent}
    ]
  }
];
```

Fonte: Autoria própria

De acordo com a figura 55, as rotas são configuradas através de um array de objetos da classe Route. Cada objeto possui um atributo path que define a URL que deve ser acessada e o atributo ‘component’ que atribui o componente que será carregado ao endereço navegado. Como mostra a figura 55, se o usuário inserir somente o endereço da aplicação o arquivo de rotas irá redirecionar para a rota ‘login’ através do atributo ‘redirectTo'.

A configuração das rotas de endereços filhos de outros endereços é realizada por meio do atribuo ‘children’, o qual possui um array de Routes. A utilização de rotas através do atributo childen torna os endereços definidos nos atributos ‘path’ relativos ao endereço pai. Em caso da não utilização do atributo children, as rotas dos componentes filhos devem ser realizadas de maneira absoluta, como mostra o exemplo na figura 56.

Figura 56 – Captura de tela de exemplo da estrutura de rotas sem atributo ‘children’.

```
export const AppRoutes: Routes = [
    {path: '', redirectTo: 'login', pathMatch: 'full'},
    {path: 'login', component: LoginComponent},

    {path: 'forum', component: ForumComponent},
    {path: 'forum/novo-topico', component: NovoTopicoComponent},
    {path: 'forum/topico', component: TopicoComponent}
  ]
;
```

Fonte: Autoria própria

3.3.4.2 Guardas de rotas

Em toda a aplicação web existem páginas e ambientes que não devem ser acessados por usuários não logados ou que não possuam algum privilégio. No Angular esse controle pode ser realizado através das classes de guarda. Estas classes implementam interfaces que as permitem interceptar as rotas inseridas no navegador e com base nelas conceder ou não o acesso. A figura 57 mostra a implementação da classe de guarda do sistema.

Figura 57 – Captura de tela da classe de guarda AuthGuard.

```
export class AuthGuard implements CanActivate {
  constructor(private router: Router, private usuarioService: UsuarioService) {
  }

  canActivate(activeRoute: ActivatedRouteSnapshot, routerState: RouterStateSnapshot): boolean {
    console.log();
    if (routerState.url.includes( searchString: 'forum')) {
      if (routerState.url.includes( searchString: 'forum') && this.usuarioService.getUsuarioLogado() != null) {
        return true;
      } else {
        this.router.navigate( commands: ['/login']);
      }
    } else {
      if (routerState.url.includes( searchString: 'login') || routerState.url.includes( searchString: 'cadastro')) {
        if (this.usuarioService.getUsuarioLogado() == null) {
          return true;
        } else {
          this.router.navigate( commands: ['/forum']);
        }
      }
    }
  }
```

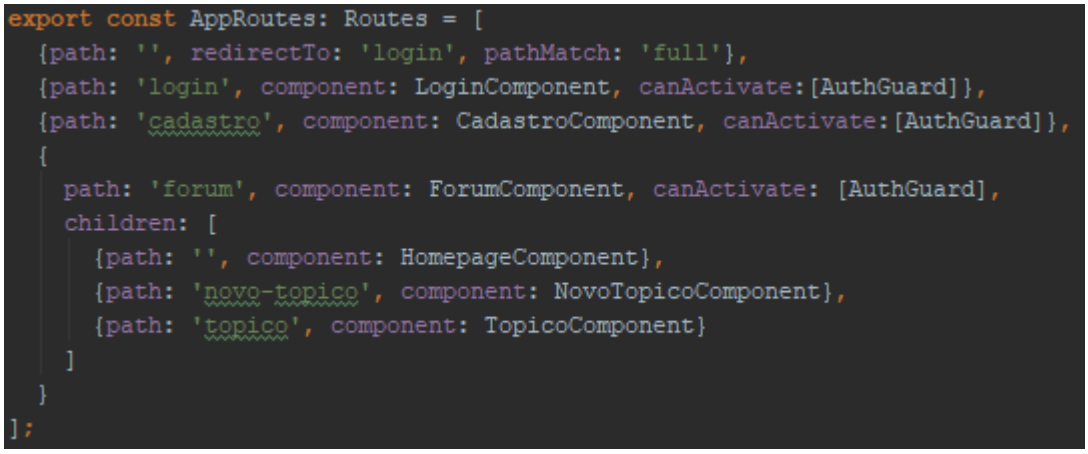
Fonte: Autoria Própria.

As classes responsáveis por realizar a guarda das rotas implementam a interface CanActivate, assim também sendo necessário a implementação do método canActivate. Este método é executado no momento que uma requisição para um componente web for feita, e através da sua lógica a rota será liberada para o acesso. No momento da requisição para a componente o sistema de rotas do Angular irá executar o método canActivate() e passar como parâmetro um objeto da classe ActivatedRouteSnapshot e outro objeto da classe RouterStateSnapshot. Esses objetos já contem as informações das rotas ativas e das rotas a serem acessadas, bem como métodos para buscar determinadas informações.

Nesse projeto o AuthGuard faz uma busca na URL a ser acessada com o objetivo de descobrir o endereço que a requisição busca acessar. Se a palavra ‘login’ for encontrada na URL o algoritmo irá verificar se o usuário está logado, impedindo que o usuário acesse a página de login caso o mesmo já esteja logado, assim redirecionando-o para a página principal do sistema. Caso o usuário não esteja logado e a URL conter a palavra ‘forum’ o AuthGuard irá redirecioná-lo para a página de login, assim impedindo que usuários não logados acessem o fórum de notícias.

Os guardas de rotas precisam ser especificados no arquivo de rota para todas as rotas desejadas através do atributo ‘canActivate’, como mostra na figura 58, lembrando que todas as classes de guarda precisam implementar a interface CanActivate.

Figura 58 – Captura de tela do arquivo de rotas app-routing.module.ts utilizando guardas de rotas



```
export const AppRoutes: Routes = [
  {path: '', redirectTo: 'login', pathMatch: 'full'},
  {path: 'login', component: LoginComponent, canActivate:[AuthGuard]},
  {path: 'cadastro', component: CadastroComponent, canActivate:[AuthGuard]},
  {
    path: 'forum', component: ForumComponent, canActivate: [AuthGuard],
    children: [
      {path: '', component: HomepageComponent},
      {path: 'novo-topico', component: NovoTopicoComponent},
      {path: 'topico', component: TopicoComponent}
    ]
  }
];
```

Fonte: Autoria própria.

Na figura 58 percebe-se que em todas as rotas da aplicação são atribuídos guardas de rotas. Sendo assim, todas as URLs válidas do sistema serão analisadas pela classe AuthGuard e, de acordo com o estado do sistema, conceder o acesso ou não aos componentes requisitados.

A implementação de um padrão de controle de acesso aos componentes é de extrema importância, visto que os dados e o sistema em si podem ser comprometidos por comportamento inadequado ou errôneo de um usuário ou também por usuários mal intencionados.

4 CONSIDERAÇÕES E OBSERVAÇÕES FINAIS

O processo de implementação de uma aplicação web que faz uso de frameworks requer aprendizado. O Angular, principalmente, exige do desenvolvedor uma visão modular da aplicação, uma visão bem definida e organizada de como o seu sistema irá funcionar e do que ele precisa para ser completamente finalizado e atender a todas as especificações.

Semelhante ao processo de implementação de um sistema integrado, onde o *front-end* executa junto ou bem próximo do *back-and*, o desenvolvimento de uma aplicação web isolada e independente se aproxima do desenvolvimento do *back-end* de um sistema. A definição e escolha de um padrão de projeto para adotar, a adoção de um sistema de pacotes, o conceito de orientação a objetos e a implementação das interfaces torna a implementação front-end tão complexa quanto detalhada quando o *back-end*.

Essa característica possui o seu ponto positivo e negativo. Por um lado, a sensação de estar desenvolvendo um sistema *back-end* ajuda no planejamento lógico e estrutural do projeto, dando mais liberdade na implementação lógica da interface e no design da interface em si. Por outro lado, a mesma sensação causada pode dificultar o entendimento de independência.

Apesar da curva de aprendizagem presente quando é iniciado o trabalho com o Angular, diversas vantagens foram percebidas. A presença de uma linguagem de programação quase exclusiva dá suporte no desenvolvimento focado em *front-end* e facilita no pensamento lógico da interface, assim permite que diversas operações e tarefas possam ser feitas de formas menos engessadas. O desenvolvimento da aplicação através de compiladores JavaScript/ECMAScript traz muito ganho no desenvolvimento e manipulação dos dados e informações em aplicações web. O amplo repositório de pacotes entrega ao desenvolvedor soluções muitas vezes prontas e com grande fator de customização.

Por fim, o desenvolvimento web utilizando o Framework Angular requer do desenvolvedor planejamento e organização, tornando o início da implementação mais lento e que demanda um maior pensamento estratégico e lógico do desenvolvedor.

Porém, uma vez que o sistema está organizado e com as suas características bem definidas, as ferramentas disponíveis, a linguagem e a comunidade proporcionam bases sólidas para tornar a aplicação completa, robusta e de fácil evolução e manutenibilidade.

REFERENCIAS

DAYLEY, Brad; DAYLEY, Brendan; DAYLEY, C. **Learning Angular: A Hands-On Guide to Angular 2 and Angular 4**. Addison-Wesley Professional, 2017.

FREEMAN, Adam. **Pro Angular**. Apress, 2017.

GAVIN, B.; ABADI, M.; TORGERSEN, M. **Understanding typescript**. *European Conference on Object-Oriented Programming*. Springer, Berlin, Heidelberg, 2014.

DEELEMAN, Pablo. **Learning Angular 2**. Packt Publishing Ltd, 2016.

RIBEIRO, O. SOUSA **Processos de apoio ao desenvolvimento de aplicações Web,** 2005. 131 p. Dissertação – Universidade de São Paulo, São Carlos, 2005.

NASCIMENTO, L. **O Crescimento da Web e as Tendências de Mercado**. Tableless, 2014. Disponível em: <https://tableless.com.br/o-crescimento-da-web-e-as-tendencias-de-mercado>. Acessado em 20 de novembro de 2019.