UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JULIANO IANKOSKI

APLICAÇÃO DO CONCEITO DE MICROSSERVIÇOS E WEB APIS. CONSTRUÇÃO DE UM SISTEMA DE COLETA DE OPINIÃO PÚBLICA COMO ESTUDO DE CASO.

TOLEDO

2022

JULIANO IANKOSKI

APLICAÇÃO DO CONCEITO DE MICROSSERVIÇOS E WEB APIS. CONSTRUÇÃO DE UM SISTEMA DE COLETA DE OPINIÃO PÚBLICA COMO ESTUDO DE CASO.

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Prof. Dr. Roberto Milton Scheffel

TOLEDO

2022

JULIANO IANKOSKI

APLICAÇÃO DO CONCEITO DE MICROSSERVIÇOS E WEB APIS. CONSTRUÇÃO DE UM SISTEMA DE COLETA DE OPINIÃO PÚBLICA COMO ESTUDO DE CASO.

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 05/dezembro/2022

Eduardo Pezutti Beletato dos Santos
Mestre em Ciências da Computação e Matemática Computacional
Universidade Tecnológica Federal do Paraná

Rosane Fatima Passarini
Doutora em Engenharia de Automação e Sistemas
Universidade Tecnológica Federal do Paraná

Roberto Milton Scheffel
Doutor em Ciência da Computação
Universidade Tecnológica Federal do Paraná

TOLEDO

2022

Dedico este trabalho a todos os que acreditaram que
eu chegaria até aqui.

AGRADECIMENTOS

Para todos os meus amigos e familiares, mesmo que seu nome não esteja aqui nesta página, saiba que de uma forma ou de outra você contribuiu positivamente para que eu alcançasse este destino.

Agradeço ao meu orientador Prof. Dr. Roberto Milton Scheffel por ter me orientado esse tempo todo.

A minha mãe Clarice Salete Iankoski a qual sempre sonhou com o momento da conclusão deste curso.

A minha namorada Thayene Queiroz pela paciência comigo mesmo em momentos de dificuldades.

E por fim ao meu irmão Vinicius Neyssinger Lourenço e a todos os meus amigos os quais dividimos diversas alegrias durante todos esses anos, obrigado.

A ciência é, portanto, uma perversão de si mesma, a menos que tenha como fim último, melhorar a humanidade.
(Nikola Tesla).

RESUMO

Como alternativa a utilização de arquiteturas monolíticas para a construção de sistemas, a estrutura de microsserviços com *Web APIs* propõe a combinação de diversos serviços trabalhando para compor o sistema. Para a implementação deste conceito será adotado a coleta de opinião pública como estudo de caso. A fim de alcançar seu público alvo com maior eficiência, donos de estabelecimentos, comerciais ou não, podem usar coletas de opiniões para melhorar o atendimento aos seus clientes. Reunir essas informações sobre os gostos e opiniões do público por meios não digitais pode ser custoso, moroso e trabalhoso, e até mesmo inviável, pois isso depende de mão de obra humana, ou se o local é muito frequentado e movimentado. Há ferramentas disponíveis no mercado que facilitam esse trabalho, todavia, a maioria delas não requer que o cliente que está enviando uma resposta ou avaliação tenha frequentado fisicamente o local, podendo assim gerar dados falsos para os proprietários. O objetivo deste trabalho é implementar um sistema que faça uso do conceito de arquitetura de microsserviços e *Web APIs REST*, e que permita um proprietário de qualquer estabelecimento, através de enquetes de múltipla escolha criadas por ele, obter respostas de seu público presente. Esse sistema poderá ser utilizado por qualquer estabelecimento que lide com público, como por exemplo bares, restaurantes, igrejas, clínicas médicas ou repartições públicas.

Palavras-chave: opinião pública; avaliações públicas; *Web APIs;* microsserviços.

ABSTRACT

As an alternative to using monolithic architectures to build systems, the structure of microservices with Web APIs proposes the combination of different services working to compose the system. For the implementation of this concept, the collection of public opinion will be adopted as a case study. In order to reach their target audience more efficiently, business and non-commercial establishment owners can use feedback collections to improve customer service. Gathering this information about the tastes and opinions of the public by non-digital means can be costly, time-consuming and laborious, and even unfeasible, as this depends on human labor, or if the place is very frequented and busy. There are tools available on the market that facilitate this work, however, most of them do not require that the customer who is sending a response or evaluation has physically visited the location, thus being able to generate false data for the owners. The objective of this work is to implement a system that makes use of the concept of architecture of microservices and REST Web APIs, and that allows an owner of any establishment, through multiple choice polls created by him, to obtain answers from his public. This system can be used by any establishment that deals with the public, such as bars, restaurants, churches, medical clinics or public offices.

Keywords: public opinion; public reviews; Web APIs; microservices.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

QRCode	*Quick Response Code*
API	*Application Programming Interface*
NPM	*Node Package Manager*
HTTP	*Hypertext Transfer Protocol*
REST	*Representational State Transfer*
SGBD	Universidade Sistemas Gerenciadores de Banco de Dados
JSON	*JavaScript Object Notation*
UUID	*Universally Unique Identifier*
URL	*Uniform Resource Locator*
JWT	*JSON Web Token*
CORS	*Cross-origin Resource Sharing*
REST	*Representational State Transfer*
AJAX	*Asynchronous JavaScript e XML*
SQL	*Standard Query Language*
ORM	*Object-Relational Mapping*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
UML	*Unified Modeling Language*

SUMÁRIO

1. INTRODUÇÃO

O conceito de arquitetura de microsserviços consiste em diversos serviços independentes trabalhando em seu próprio processo para garantir o funcionamento de um sistema como todo. A comunicação entre estes componentes é feita através de *HTTP (Hypertext Transfer Protocol) REST (Representational State Transfer)* e *Web APIs* (*Application Programming Interface*) (VILLAÇA; PIMENTA JR; AZEVEDO; 2018). Cada serviço destes possui funcionalidades específicas como por exemplo cadastro de perguntas ou gerenciar as respostas.

Para aplicar esses conceitos foi feito uso de um estudo de caso no qual consiste em automatizar a coleta de opinião e satisfação pública.

Ambientes que atendem público como bares, restaurantes, clínicas médicas, igrejas, praças de alimentação de *shoppings* normalmente focam seu atendimento em seu público alvo. Podemos definir público alvo como “um grupo de consumidores com características em comum que a empresa identifica no mercado e para quem direciona suas estratégias e campanhas”, portanto, identificar suas variações de gostos e opiniões podem auxiliar na definição de sua estratégia de negócio[1]. Antes da estratégia de definição do público alvo utilizava-se de *marketing* de massa, o qual consiste em impactar o maior número possível de clientes em potencial. Contudo, a era digital gerou o fortalecimento do consumidor, o qual é mais exigente e espera que as suas preferências, problemas e gostos sejam de conhecimento das empresas e que elas usem isso para atraí-lo[2]. Analisar os problemas e dúvidas de seu público é uma forma de se aproximar de seu consumidor[3].

Saber com precisão as opiniões de diversos clientes em um ambiente com muitas pessoas, de forma não digital, pode tornar-se custoso, laborioso e até mesmo inviável, principalmente quando se está lidando com um lugar muito movimentado, ou com alternância muito rápida do público. Saber qual estilo musical será reproduzido no sistema de som ambiente, qual programação será exibida no telão local, qual prato será vendido na festa da igreja, ou como melhorar o atendimento de uma clínica médica são exemplos de questões a serem levantadas pelos responsáveis do estabelecimento.

---

[1] Disponível em: <https://rockcontent.com/br/blog/publico-alvo/>
[2] Disponível em: <https://rockcontent.com/br/blog/marketing-de-massa/>
[3] Disponível em: <https://raccoon.ag/blog/marketing-digital/publico-alvo/>

Com a coleta de opinião pública como estudo de caso, este trabalho visa a automação deste processo utilizando a estrutura de microsserviços e *Web APIs*. Isso acontecerá através de um sistema onde, após cadastrado o estabelecimento, ele poderá criar enquetes de múltipla escolha as quais poderão ser respondidas através dos dispositivos móveis Android de seus clientes. O proprietário ou responsável pelo local poderá gerar quantas perguntas quiser e delimitar o período que ela ficará disponível para ser respondida. Ele também poderá visualizar a opção mais respondida ou a quantidade de respostas de cada alternativa criada. Este sistema será chamado de Perguntador.

O acesso ao módulo de cadastro e manutenção de perguntas, alternativas e visualização de respostas será feito através de qualquer navegador de internet disponível. A hospedagem desses serviços e de todos os dados será feita em nuvem, dispensando assim a instalação local de *software*. Para acessar as enquetes cadastradas será feita a leitura de um *QRCode (Quick Response Code)* único por estabelecimento, efetuada através de uma aplicação Android, garantido assim que somente frequentadores reais do local gerem respostas únicas.

Outro objetivo aqui buscado é realizar a construção deste sistema utilizando conceitos e tecnologias atuais, e que possuam boa visibilidade e uso no mercado atual de desenvolvimento de software a fim de maior enriquecimento curricular.

1.1 Objetivos

1.1.1 Objetivos Gerais

O objetivo deste trabalho é aplicar a utilização do conceito de arquitetura de microsserviços e *Web APIs* a um estudo de caso. Esse consiste em criar um sistema para auxiliar quaisquer donos e responsáveis por estabelecimentos que trabalham com público. Com este *software* eles poderão coletar a opinião de seus clientes ou visitantes, sobre assuntos diversos através de enquetes criadas com questões de múltipla escolha. Além de poderem colher estas informações, ainda poderão enviar avisos e notícias para seus clientes em um quadro específico para tal.

1.1.2 Objetivos Específicos

Dados os objetivos gerais deste trabalho, foram definidos os específicos como seguem:

- Levantar os requisitos funcionais e não funcionais para a confecção do sistema
- Elaborar o projeto do sistema junto com a modelagem do banco de dados.
- Desenvolver uma aplicação *web* a qual será responsável por cadastrar os estabelecimentos, gerar um *QRCode* para a validação de presença para responder, armazenar os dados do sistema, criar enquetes, receber as respostas, exibir seus resultados e um quadro de notícias.
- Desenvolver um *software* para ser utilizado através de dispositivos móveis que será voltado para o cliente ou frequentador do local. Ele fará a leitura do *QRCode* do estabelecimento por este sistema, receberá as enquetes, as responderá e visualizará o quadro de notícias.

2 **REFERENCIAL TEÓRICO**

Para a tarefa do desenvolvimento de *software* existem diversas tecnologias, *frameworks* e linguagens de programação disponíveis, cada uma com suas particularidades e com possibilidade de atender a demanda. Para o implementar o sistema foram escolhidas ferramentas e conceitos que melhor se aplicam. Este desenvolvimento foi projetado para que os usuários que não precisem manter um servidor de aplicações. Sendo assim, este serviço é hospedado em nuvem para que o banco de dados, servidor, armazenamento seja tudo acessado via internet (SOUSA; MOREIRA; MACHADO, 2009). Além dos usuários não precisarem manter diretamente um servidor de aplicações, um sistema *web* também dispensa instalações locais nas máquinas para a operação. Esta seção aborda as ferramentas e conceitos escolhidos para o desenvolvimento do sistema.

2.1 JavaScript

Para o desenvolvimento principal do Perguntador foi usada a linguagem de programação JavaScript. É uma linguagem de programação de alto nível, interpretada e pode ser utilizada tanto de forma procedural quanto orientada a objetos (FLANAGAN, 2004). Sua utilização acontece principalmente em páginas *web* no navegador do cliente. Apesar de seu uso primário ser dentro dos navegadores *web* a linguagem não é limitada a isso, podendo também ser utilizada tanto no lado cliente quanto no servidor da aplicação (FLANAGAN, 2004).

A popularidade está entre os motivos da escolha do JavaScript para a construção do Perguntador, pois ela se encontra entre as cinco linguagens de programação mais utilizadas (IEEE SPECTRUM, 2022). Além de uma boa popularidade, a linguagem possui uma forte comunidade ativa. Há outras linguagens de programação como PHP, Ruby ou Python disponíveis no mercado para a construção deste sistema, contudo, a escolha do Javascript foi feita para aumentar o conhecimento sobre ele. Esta linguagem de programação está em constante evolução e recebe atualizações frequentes, a cada uma delas tornando a tecnologia mais completa para melhorar a experiência do usuário e seus usos[4].

Também como vantagens no uso desta linguagem de programação, temos a rapidez de leitura e execução, pois ela acontecendo também do lado do cliente e

[4] Disponível em: <https://ilustradev.com.br/6-motivos-para-aprender-javascript/>

processada pelo navegador, utiliza o poder de processamento da máquina do usuário e não precisando de comunicações frequentes com o servidor principal (PRESCOTT, 2016). Além do mais, os *frameworks* que fazem uso de JavaScript possuem uma documentação completa e de frequente atualização e expansão, tendo o investimento de grandes empresas[5]. Dispensando o uso de um compilador, o JavaScript é compatível com a maioria dos navegadores modernos contribuindo assim para que a ferramenta seja uma das mais utilizadas para desenvolvimento *web*[6].

2.1.1 Node.js

O desenvolvimento do *back-end* foi feito no modelo de *APIs* com o Node.js. Node.js pode ser visto como uma plataforma de aplicação, onde os programas são escritos em JavaScript, mas são compilados por uma máquina virtual (MORAES, 2015).

Criado para trabalhar com apenas um *thread*, essa ferramenta consegue atender um grande número de requisições. Trabalhar de forma assíncrona, isto é, não bloqueando a *thread*, faz com que esse *thread* não precise aguardar que operações como acesso ao banco ou leitura de arquivos sejam concluídas para seguir a execução (NODE JS, 2022). Independendo de um navegador *web* para executar um programa, o Node.js pode ser utilizado para criar aplicações que rodam no lado do servidor (MORAES, 2015).

Por ser multiplataforma, o Node.js também permite que as aplicações sejam executadas em qualquer SO, podendo assim reduzir os custos com licenças. Tratando-se de licença com Windows em *datacenters* como a Amazon, a economia nos custos de contratação pode chegar a 50%[7]. O ganho em produtividade alcançado ao usar essa ferramenta também é uma vantagem. O *NPM (Node Package Manager)* é um dos maiores repositórios do mundo, o que permite com que termos um bom aproveitamento e reutilização de códigos[8], otimizando o tempo de desenvolvimento.

---

[5] Disponível em: <https://acervolima.com/vantagens-e-desvantagens-do-javascript/>
[6] Disponível em: <https://www.cpt.com.br/cursos-informatica-desenvolvimentodesoftwares/artigos/linguagem-de-programacao-javascript-as-principais-vantagens>
[7] Disponível em: <https://www.luiztools.com.br/post/o-que-e-nodejs-e-outras-5-duvidas-fundamentais/?utm_source=google&utm_medium=cpc&utm_campaign=902557428>
[8] Disponível em: <https://conteige.cloud/node-js-vantagens-e-desvantagens/>

2.1.1.1 Express

Para auxílio na construção do *back-end* do programa o Express foi adotado. Ele consiste em um *framework* do Node.js, ou seja, a codificação acontece em JavaScript. A ferramenta é de código aberto e está sob a licença MIT, aplicado para otimizar a construção das *APIs* da aplicação *web* e seus recursos. O Express consegue gerenciar requisições de diferentes métodos *HTTP* e possui um sistema de gerenciamento de rotas completo e foi construído para facilitar a rápida criação de aplicações (EXPRESS JS, 2022). Essa ferramenta disponibiliza formas de especificar chamadas de função quando uma requisição *HTTP* é recebida, facilitando assim a formação das respostas (EXPRESS JS, 2022).

2.1.1.2 Sequelize

A ferramenta responsável pelo mapeamento de dados relacionais para objetos JavaScript é o Sequelize. Essa biblioteca é responsável pela conexão ao banco de dados e manipulação dessas informações, mapear dados relacionais vindos do banco para objetos JavaScript e objetos desse tipo para dados relacionais. Com suporte aos SGBD (Sistemas de Gerenciamento de Banco de Dados) PostgreSQL, MariaDB, MySQL, SQLite e MSSQL, essa tecnologia possibilita criar, alterar, excluir e buscar dados do banco utilizando funções escritas em TypeScript (SEQUELIZE, 2022).

2.1.1.3 Joi

Tendo o fato de o sistema ter sido construído em formato de *APIs*, requisições *HTTP* são comuns e frequentes dentro do funcionamento do programa. Para auxiliar na validação dos dados enviados e recebidos foi utilizada a biblioteca do *NPM* Joi. Essa ferramenta é um validador de *schemas*, onde criamos um *schema* e passamos para a validação um objeto *JSON (JavaScript Object Notation)*, verificando assim se este objeto atende a estrutura esperada (JOI, 2022). Nesta aplicação o Joi auxilia validando se as requisições recebidas pelo *back-end* possuem o *schema* correto.

2.1.1.4 Helmet

Para aumentar a segurança da aplicação, o protocolo *HTTP* possui vários *Headers* que podem ser úteis e prevenir ataques. Configurar esses cabeçalhos

corretamente pode ser auxiliado pela ferramenta do Node.js chamada Helmet. Ela consiste em uma coleção de *middlewares* menores os quais configuram os cabeçalhos das requisições feitas pelas *Web APIs* melhorando sua segurança (HELMET, 2022). O Helmet é aplicado ao Perguntador para aumentar a segurança das requisições *HTTP* que acontecem entre as APIs.

2.1.2 TypeScript

As *APIs* do *back-end* foram criadas usando TypeScript. Criado pela Microsoft para melhorar as características da linguagem, ele consiste em um conjunto de funcionalidades e estruturas que incrementam o JavaScript, adicionado a ele recursos da Programação Orientada a Objetos, possibilidade de criação de interfaces claras e tipagem forte de dados (TYPESCRIPT, 2022).

O uso do JavaScript em sua forma nativa no *back-end* apresenta algumas dificuldades. Os quatro conceitos fundamentais da Programação Orientada a Objetos são encapsulamento, herança, polimorfismo e abstração. Estes conceitos básicos são prejudicados pela tipagem fraca de dados e pelo fato de a linguagem não permitir definir classes de forma tão clara. O TypeScript é uma solução a esses problemas. Sendo um super conjunto do JavaScript, qualquer código dele pode ser incluído em arquivos TypeScript, os quais possuem a extensão .ts, e usado diretamente neles (TYPESCRIPT, 2022).

O uso do TypeScript no desenvolvimento do Perguntador trouxe a possibilidade de encontrar erros em tempo de desenvolvimento do projeto. Além disso, agiliza o processo de desenvolvimento e traz segurança ao seu ambiente durante esta etapa (TYPESCRIPT, 2022).

2.2 BCrypt

Os dados dos usuários são essenciais para os negócios de qualquer organização. Junto com a performance e escalabilidade do programa, garantir a segurança é uma das principais missões dos desenvolvedores (NARDELLI, 2021). A criptografia é uma ferramenta que contribui para a proteção dessas informações valiosas, auxiliando para que não ocorram acessos indevidos.

Utilizando o algoritmo de *hash*, o BCrypt foi criado para esconder senhas no banco de dados de uma forma confiável e segura. Além disso, qualquer linguagem popular como C, C ++, C#, Java, PHP, Python e JavaScript podem utilizá-lo[9].

O BCrypt possui a vantagem de podermos adicionar aleatoriedade a uma operação *hash*, este recurso é chamado de *salt*. Isto faz com que mesmo que houverem senhas iguais, os *hashs* sempre serão diferentes. Outra medida que torna os dados mais seguros a ataques de força bruta é uma configuração que permite aumentar o custo de processamento do *hash*, este exercendo influência em ataques deste tipo (PROVOS; MAZIÈRES, 1999). Nesta aplicação o BCrypt é responsável por realizar o *hash* das senhas dos usuários que são armazenadas no banco de dados.

2.3 *UUID (Universal Unique Identifier)*

Para o melhor desempenho do Perguntador, as questões de múltiplas escolhas cadastradas pelos estabelecimentos serão acessadas através da leitura de um *QRCode* exclusivo por empresa. Esse código poderá ser exibido em um telão ou até mesmo impresso, de acordo com a preferência dos usuários do sistema. A fim de garantir a geração desses *QRCodes* únicos, o uso de um *UUID* foi aplicado ao sistema. Isso garante que a leitura de um *QRCode* em um local não concederá indevidamente acesso a perguntas e notícias de outro.

Este identificador universalmente único consiste em um número inteiro de 128 *bits* gerado sem uma coordenação central (RFC 4122, 2005). Assim é possível identificar unicamente as empresas cadastradoras de perguntas.

2.4 Dotenv

O comportamento de um programa pode ser personalizado sem a definição de valores diretamente no código fonte utilizando as variáveis de ambiente. Existindo somente onde elas foram instanciadas, essas variáveis irão armazenar configurações importantes do sistema que não se alteram com frequência[10].

---

[9] Disponível em: <https://medium.com/reprogramabr/uma-breve-introdu%C3%A7%C3%A3o-sobre-bcrypt-f2fad91a7420>

[10] Disponível em: <https://www.freecodecamp.org/portuguese/news/como-usar-variaveis-de-ambiente-do-node-com-um-arquivo-dotenv-para-node-js-e-npm/#:~:text=O%20DotEnv%20%C3%A9%20um%20pacote,o%20comando%3A%20npm%20i%20dotenv%20>

O Dotenv é uma biblioteca do Node.js gerenciadora de variáveis de ambiente. Esta carregará todas as variáveis definidas no arquivo .env do projeto, e poderão ser acessadas através do objeto chamado process.env. Esse objeto pode ser acessado em qualquer lugar do ambiente do projeto. Na produção deste *software* o Dotenv é aplicado para carregar as variáveis de ambiente que contém configurações vitais ao projeto conforme o seguinte:

- *URL (Uniform Resource Locator)* de todas as *APIs* do sistema
- Nome de usuário do banco de dados, porta e senha dele.
- Tempo em que o *JWT (JSON Web Token)* irá expirar.

Aumentando a segurança do sistema, adicionamos este o arquivo .env ao .gitignore, fazendo assim com que essas informações sensíveis não sejam disponibilizadas em um repositório git indevidamente[11].

2.5 *CORS (Cross-origin Resource Sharing)*

O Perguntador foi construído fazendo o uso de *Web APIs*, sendo necessário que frequentemente o *front-end* faça requisições para obter dados do *back-end.* Por motivos de segurança nos navegadores *Web* só podemos acessar recursos que vem da mesma origem de uma solicitação. Esse fato pode ser definido como política de mesma origem (W3C, 2020).

Apesar de o navegador Web não permitir que recursos de origens cruzadas sejam acessados, podemos configurar o *CORS* para habilitar determinados recursos nos navegadores, acessando assim esses dados. O W3C especificou o CORS para isso. Quando uma requisição é enviada para uma origem cruzada, o navegador define um *header* chamado *Access-Control-Allow-Origin*. Este campo deve ser corretamente preenchido pelo servidor para que a requisição não seja negada pelo navegador[12]. Caso o navegador verifique que o valor do *header Access-Control-Allow-Origin* é diferente do *Origin* enviado na requisição, ela é negada, apresentando erro de *CORS*. Esta tecnologia permite que o *front-end* e as *APIs* do *back-end* realizem requisições *HTTP* entre si.

---

[11] Disponível em: <https://blog.flutterando.com.br/dotenv-configurando-suas-vari%C3%A1veis-de-ambiente-e7fea2270020>

[12] Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-cors-e-como-resolver-os-principais-erros>

2.6 Microsserviços

O modelo tradicional de arquitetura de *software* utiliza uma abordagem monolítica do sistema. A documentação da Amazon Web Services (2022) afirma que dessa forma, todos os recursos do sistema são fortemente agrupados executando como um serviço só. Conforme o programa evolui e precisa ser incrementado, a adição de recursos a ele se torna complexa. O grande número de dependências de um sistema monolítico também é uma forte desvantagem no seu uso, pois uma única falha em cascata pode ocasionar em uma indisponibilidade do aplicativo (NEWMAN, 2020).

O uso de microsserviços é uma alternativa ao modelo monolítico de arquitetura. Esse modelo consiste em que um aplicativo seja composto por diversos outros componentes independentes e de acoplamento fraco. A comunicação destes serviços acontece através da união entre *APIs*, e a grande vantagem do seu uso é que os componentes são independentes, podem ter linguagens de programação diferentes e podem até estar hospedados em servidores distintos (IBM, 2021). A manutenção de um serviço desta arquitetura não deve causar grande impacto no sistema como um todo.

Na construção do Perguntador cada entidade do banco de dados será mantida pelo próprio microsserviço. Isso significa que internamente na construção do sistema elas serão independentes, apesar da conexão que a regra de negócio exige que elas possuam. Por exemplo, mesmo que uma resposta seja diretamente vinculada a uma pergunta, na construção do sistema essas entidades são representadas por microsserviços diferentes. Desta forma, cada microsserviço é responsável pela manutenção de seus dados, e eles não possuem acesso às informações de responsabilidade de outros serviços.

2.7 *API REST*

Para que aconteça a comunicação entre os diversos serviços do sistema foi utilizado o conceito de *API*. Um grupo de definições e protocolos possibilita a conexão entre cliente e servidor. Existem outros tipos de *APIs*, mas a *API REST* foi escolhida pela popularidade na *web*, flexibilidade e agilidade no envio e retorno dos dados (RED HAT, 2020).

Neste sistema cada entidade irá possuir a sua própria *API*. Elas fornecem serviços para consulta e manutenção dos dados. O *front-end* e as outras *APIs* consomem estes serviços para todas as atividades realizadas pelo sistema. Criação de uma nova pergunta, alteração dela ou responder essa pergunta são exemplos destas ações.

2.8 Axios

Para realizar as requisições *HTTP* necessárias para a comunicação dos serviços do sistema foi feito uso do Axios. Ele consiste em um cliente *HTTP* baseado em *promises* que faz essas requisições. *Promises* consistem em objetos os quais inicialmente possuem valor desconhecido, pois a obtenção deste valor pode não estar concluída no momento em que a linha executa a instrução no sistema. Temos como exemplo do uso das *promises* requisições *HTTP* ou acessos a banco de dados, processamentos os quais podem levar mais tempo para serem concluídos (MORAES, 2021). O Axios transforma dados em *JSON* automaticamente e em seu *back-end* faz requisições *AJAX (Asynchronous JavaScript e XML)* no navegador através de *XMLHttpRequests* (AXIOS, 2022). No Perguntador essa ferramenta é responsável por realizar as requisições *HTTP* entre o *front-end* e as *APIs*.

2.9 JSON

Assim como o restante do sistema que é construído em JavaScript, o formato de arquivo para a transferência de dados entre as *APIs* é feito em *JSON*. A Notação de Objeto em JavaScript é um formato leve, de sintaxe simples e possui multiutilidade para essa troca de informações, podendo ser usada também para requisições *AJAX* nos sites (W3C, 2020). Todas as requisições *HTTP* que acontecem entre o *front-end* e as *APIs* do sistema utilizam o formato *JSON*.

2.10 *JWT*

A fim de aumentar a segurança das requisições trocadas entre as APIs do sistema, foi aplicado o uso de *JWT*. O padrão utiliza como base o *JSON* para compartilhar informações de autenticação de usuário de uma forma extremamente segura (RCF 7519, 2015). Primeiramente o cliente envia os dados de *login* do usuário para o servidor, em seguida o servidor assina um *JWT* e devolve ao cliente. Enquanto

este *token* durar todas as requisições feitas pelo cliente serão validadas pelo servidor através dele (MONTANHEIRO; CARVALHO; RODRIGUES, 2017). No perguntador o uso do *JWT* serve para controlar o *login* de usuários.

2.11 MySQL

O SGBD utilizado para o armazenamento dos dados é o MySQL. Ele é um sistema gerenciador de banco de dados relacional de código aberto que utiliza a linguagem *SQL (Standard Query Language)*. A escolha de utilizá-lo foi por ser de fácil utilização, suporta grandes volumes de dados, interface simples, é compatível com a maioria dos sistemas operacionais e é um dos SGBD mais utilizados no mundo[13]. Há outros SGBD disponíveis como por exemplo PostgreSQL, porém a escolha do MySQL ocorreu pela maior familiaridade com a linguagem e sua interação com o *ORM (Object-Relational Mapping)* Sequelize.

2.12 React Js

Para aumentar a agilidade no desenvolvimento do *front-end* foi optado por utilizar *frameworks* a fim de construir as páginas. Há diversas opções de bibliotecas e ferramentas JavaScript que facilitam e ganham tempo de programação para o *client-side* do programa. Alguns fatores fizeram com que o React JS fosse a melhor opção. Sua popularidade superou a do *JQuery* no ano de 2022. Ele é uma biblioteca de código aberto desenvolvido pelo Facebook e é principalmente utilizado para criar interfaces de usuárias complexas com pouco tempo. Outro grande benefício desta tecnologia é baseado em componentes e eles podem ser reutilizados de acordo com a necessidade do programador (REACT, 2022).

O React foi criado principalmente para auxiliar a conexão entre diferentes partes de uma página. Ele cria componentes encapsulados que gerenciam seu próprio estado e podem ser combinados para construir interfaces completas. Assim sendo, a página não é tratada de forma monolítica, mas sim fragmentada com cada componente podendo agir de forma independente. Estes componentes são renderizados na tela do usuário somente de acordo com a mudança dos dados (REACT, 2022).

[13] Disponível em: <https://www.devmedia.com.br/introducao-ao-mysql/27799>

2.12.1 React Bootstrap

O Bootstrap é amplamente utilizado para facilitar as tarefas de desenvolvimento do *front-end* de sites responsivos. Esse *framework* simplifica a criação das páginas, fornecendo estruturas *CSS (Cascading Style Sheets)* prontas (REACT BOOTSTRAP, 2022). Como novas tecnologias surgiram alterando a forma como são escritos os programas, a utilização do Bootstrap também teve que ser adequada. O React Bootstrap foi aplicado então ao desenvolvimento deste trabalho.

Essa ferramenta disponibiliza componentes do Bootstrap original criados para o React. Sendo ele um projeto de código aberto e de comunidade participativa, as bibliotecas podem ser alteradas de acordo com a necessidade do programa (REACT BOOTSTRAP, 2022).

Outra grande vantagem da utilização do React Bootstrap é que ele não possui dependência alguma com o Bootstrap.js ou o JQuery, podendo ser usado assim de forma pura no projeto[14].

2.12.2 React DOM

O conteúdo de uma página *web* pode ser manipulado por navegadores e scripts através de uma interface padronizada chamada *DOM (Document Object Model)*. O *DOM* fica embutido dentro dos navegadores e é usado por ele para apresentar as páginas *web* para os usuários. Fazendo uso do React no desenvolvimento do *front-end*, também foi aplicado o uso da biblioteca React DOM. Ele contém métodos que são usados na parte superior da aplicação, renderizando os componentes React no *DOM* do navegador (REACTDOM, 2022).

Após criadas diversas páginas de um *site*, definimos sua navegação através das rotas, cada uma espelhando uma página *web*. O React Router DOM é o pacote responsável por esse roteamento dinâmico. Uma lista com todas as rotas da aplicação foi desenvolvida, e ela é responsável por direcionar o usuário para cada uma delas[15].

---

[14] Disponível em: <https://imasters.com.br/back-end/react-bootstrap-fusao-perfeita-entre-o-react-e-o-bootstrap#:~:text=O%20React%2DBootstrap%20%C3%A9%20uma%20reimplementa%C3%A7%C3%A3o%20completa%20dos%20componentes%20do,como%20um%20script%20JS%20global>

[15] Disponível em: <https://ateliware.com/blog/react-router#:~:text=O%20que%20%C3%A9%20react%2Drouter,sua%20aplica%C3%A7%C3%A3o%20de%20forma%20din%C3%A2mica.&text=Os%20componentes%20utilizados%20at%C3%A9%20antes,a%20fun%C3%A7%C3%A3o%20raiz%20da%20lib>

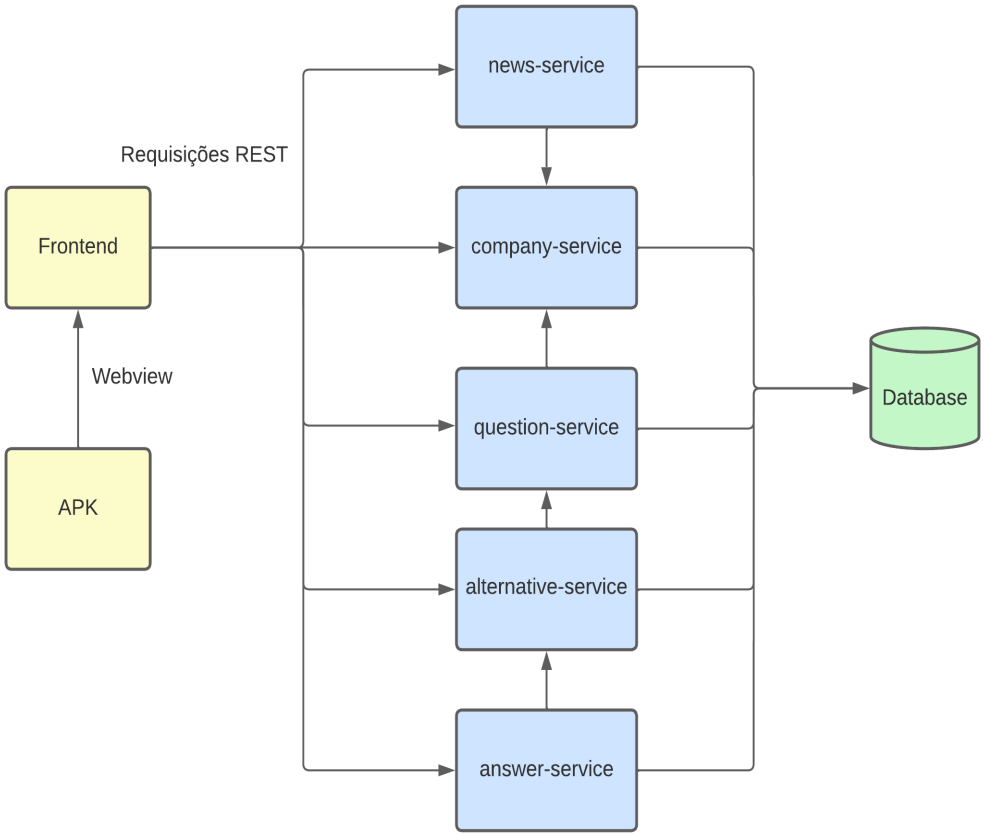
2.13 Arquitetura do sistema

A arquitetura do sistema pode ser definida como um modelo o qual o sistema pode ser desenvolvido. Ela demonstra como as partes são organizadas e como os componentes se comportam. Essa estrutura facilita melhorar a flexibilidade, organização dos componentes, antecipar tomadas de decisões importantes e minimizar riscos (VAROTO, 2002).

A aplicação do dispositivo móvel acessa as páginas do *front-end* através de uma *Webview.* O *front-end* comunica-se com todos os serviços do *back-end* por requisições *REST API*. Os serviços do *back-end* também interagem entre si por meio deste tipo de requisição quando necessário. Cada serviço é responsável por manter a sua entidade no banco de dados na tabela apropriada.

A Figura 1 a seguir demonstra como o *software* foi desenvolvido.

Figura 1 – Arquitetura do sistema



Fonte: o autor

2.14 Requisitos funcionais

Todos os requisitos funcionais e não funcionais que envolvem o sistema são descritos nesta seção.

Requisitos de sistema são funções, objetivos, propriedades e restrições os quais o sistema deve ter para cumprir seu propósito perante seus usuários, ou seja, tudo o que o *software* precisa ter para atingir o seu objetivo. Estes requisitos também servem para haver uma concordância com clientes e quaisquer envolvidos sobre o papel que ele precisa desempenhar e os seus limites de atuação (GUEDES, 2009).

Os requisitos de sistema são divididos entre os requisitos funcionais e não funcionais. Um requisito funcional demonstra uma ação que deve ser realizada pelo *software*, ou seja, o que o sistema deve fazer. Ele também expressa suas necessidades, características e funcionalidades. São chamados assim pois todos eles são funcionalidades cumpridas por ações do *software*[16]. O quadro 1 a seguir apresenta os requisitos funcionais do sistema.

[16] Disponível em: <https://analisederequisitos.com.br/requisitos-funcionais-e-nao-funcionais/>

Quadro 1 – Requisitos funcionais

ID	Descrição
RF01	Manter a empresa, responsável por cadastrar as perguntas, alternativas, notícias e visualizar as respostas.
RF02	Visualizar as perguntas cadastradas, junto com a alternativa mais respondida caso houver.
RF03	Visualizar a quantidade de respostas que cada alternativa obteve.
RF04	Manter as perguntas de múltipla escolha, podendo ter entre duas e cinco alternativas.
RF05	Manter o quadro de notícias.
RF06	A notícias terão uma data início e data fim para serem visualizadas.
RF07	As perguntas terão uma data de início e fim para serem respondidas.
RF08	Somente perguntas que não possuem respostas podem ser alteradas ou excluídas.
RF09	Gerar o *QRCode* para ser lido pelo aplicativo no dispositivo móvel.
RF10	Apresentar as perguntas e o quadro de notícias para os clientes, respeitando as datas de início e fim de cada uma. Esse acesso é permitido somente através da leitura do *QRCode*.
RF11	Permitir que um dispositivo responda uma pergunta somente uma vez.

Fonte: o autor

Os requisitos não funcionais explicam como o sistema irá cumprir os seus requisitos funcionais, todas as necessidades que não serão atendidas pelas funcionalidades. Também são chamados de atributos de qualidade, eles têm uma grande importância na produção do *software*[17]. O quadro 2 seguinte apresenta os requisitos não funcionais do sistema.

[17] Disponível em: <https://codificar.com.br/requisitos-funcionais-nao-funcionais/>

Quadro 2 – Requisitos não funcionais

ID	Descrição
RNF01	O sistema deve ser construído com JavaScript
RNF02	As *APIs* do *back-end* devem ser construídas com a estrutura de microsserviços
RNF03	O sistema deve autenticar o usuário com *JWT*
RNF04	O sistema deve recusar requisições que possuem *token* ausente ou inválido
RNF05	As senhas dos usuários devem ser criptografadas no banco de dados
RNF06	Os serviços tanto do *back-end* quanto do *front-end* devem comunicar-se através de *REST*
RNF07	As perguntas a serem respondidas e o quadro de notícias deve ser visualizados somente através da leitura do *QRCode* pela aplicação móvel.
RNF08	Todas as exclusões do sistema são tratadas como um *soft delete.* Todos os registros excluídos poderão ser auditados através de consultas ao banco de dados.

Fonte: o autor

Com estes requisitos foram criados os diagramas de classe, casos de uso e sequência. Esses diagramas definem os relacionamentos, entidades e comportamentos do sistema.

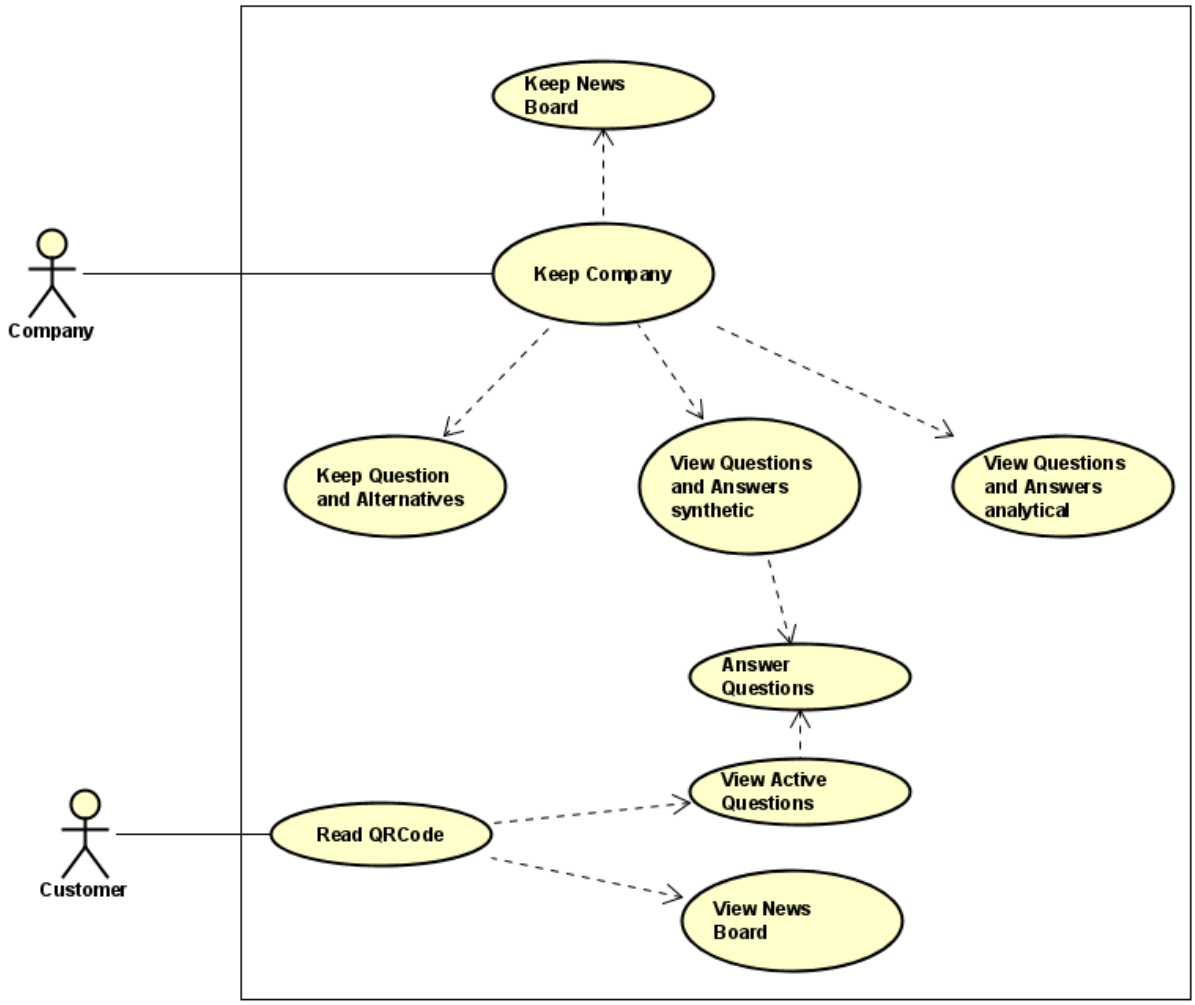
2.15 Diagrama de casos de uso

O diagrama de casos de uso do *UML (Unified Modeling Language)* consiste em uma modelagem do comportamento do sistema, e auxilia a mapear os seus requisitos. Esses diagramas retratam o escopo do programa e funções de alto nível. Também mapeiam as interações entre os agentes e o *software*. Como os agentes usam e o que o sistema faz são descritos no diagrama de casos de uso, contudo, a operação interna dele não é abordada aqui (IBM, 2021). Ele tem sua importância no projeto pois facilita de forma antecipada que os usuários entendam como o sistema funcionará, e auxilia assim no entendimento comum entre equipe de programação e cliente[18].

A Figura 2 a seguir representa o diagrama de casos de uso, nele são apresentadas as integrações entre os agentes e o sistema.

[18] Disponível em: <https://www.lucidchart.com/pages/pt/diagrama-de-caso-de-uso-uml>

Figura 2 – Diagrama de casos de uso



Fonte: o autor

O ator responsável por manter as perguntas, quadro de notícias, visualizar a quantidade de respostas por alternativa e disponibilizar o *QRCode* para acesso a essas perguntas é chamado de *Company*. Já a responsabilidade de ler o *QRCode* para visualizar o quadro de notícias, perguntas a serem respondidas e responder uma pergunta fica para o ator *Customer*.

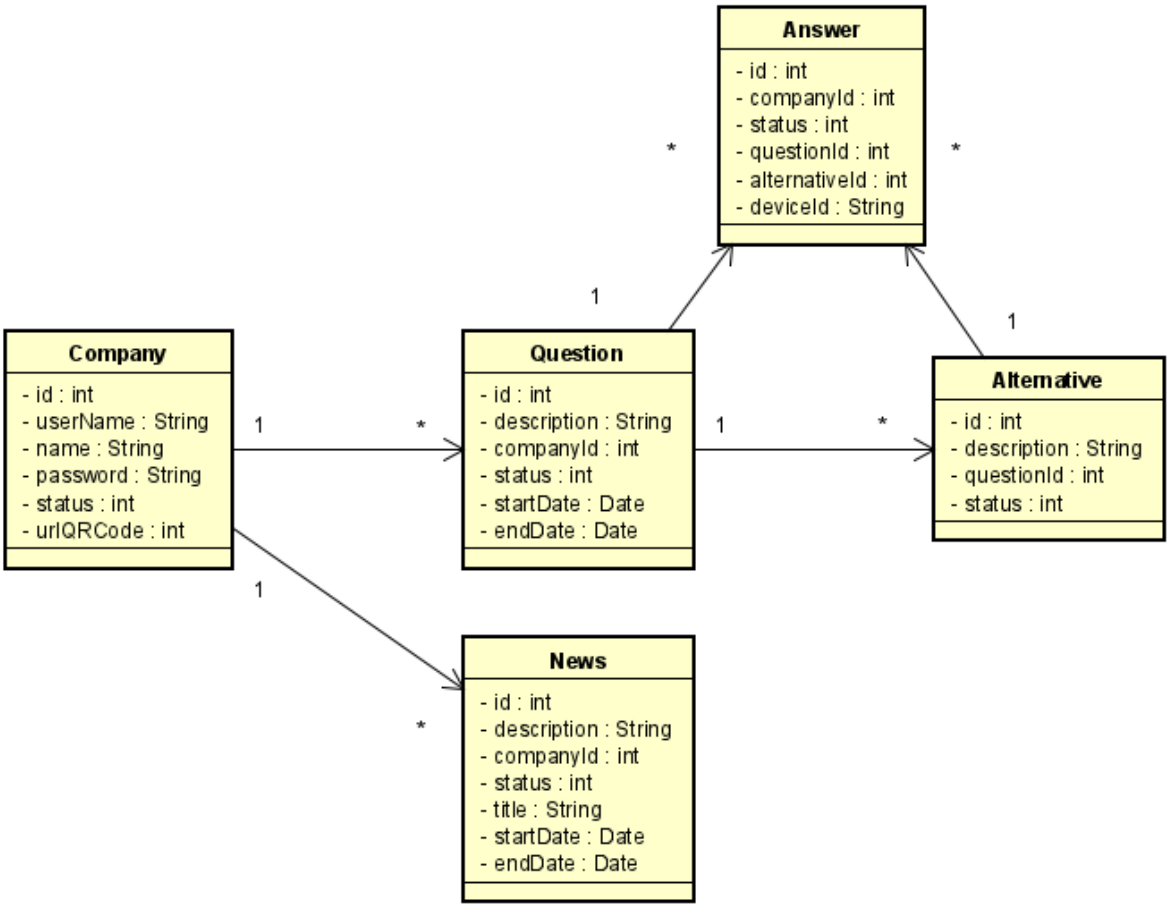
2.16 Diagrama de classes

Dentro do paradigma de programação orientada a objetos, podemos entender uma classe como uma abstração de um objeto da vida real para dentro do sistema. A fim de realizar a modelagem dessas estruturas foi utilizado o diagrama de classes, este é um diagrama estrutural da *UML*. Esse diagrama pode ser utilizado tanto para modelos simples quanto para modelos mais complexos como com padrões de projetos diversos. Também é possível a sua aplicação para estruturas de

microsserviços com *APIs*[19]. Ele também exibe o relacionamento entre as entidades e o seu papel dentro do sistema. Usado no estágio de análise para a compreensão do problema e identificação dos seus componentes, esse diagrama descreve com exatidão como o sistema funciona (IBM, 2021).

A seguir a Figura 3 representa as entidades do sistema enquanto a Figura 4 apresentada na sequência é o diagrama de classe do microsserviço news-serivce.

Figura 3 – Entidades do sistema



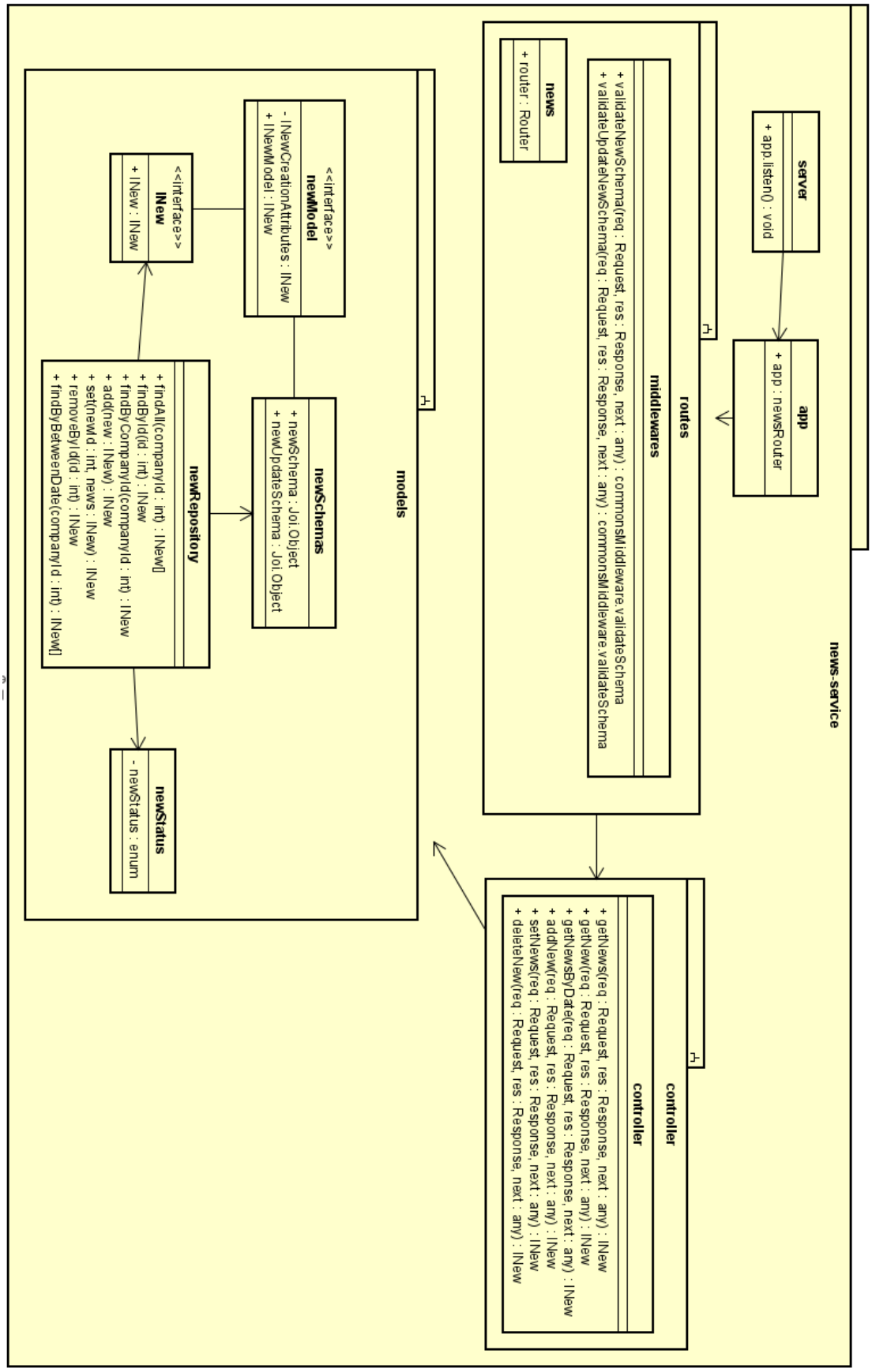
Fonte: o autor

Cada entidade representa um microsserviço. Cada um deles possui suas interfaces, atributos e funções os quais garantem o funcionamento do sistema, como todos os mirosserviços do Perguntador possuem estrutura semelhante, a seguir será apresentada a estrutura interna apenas do serviço responsável por manter o quadro de notícias. A modelagem do banco de dados, com as tabelas e relacionamentos,

[19] Disponível em: <https://www.ateomomento.com.br/uml-diagrama-de-classes/>

segue nessa estrutura onde cada entidade representa uma tabela do banco relacional, e cada atributo desta representa uma coluna.

Figura 4 – Diagrama de classe news-service



Fonte: o autor

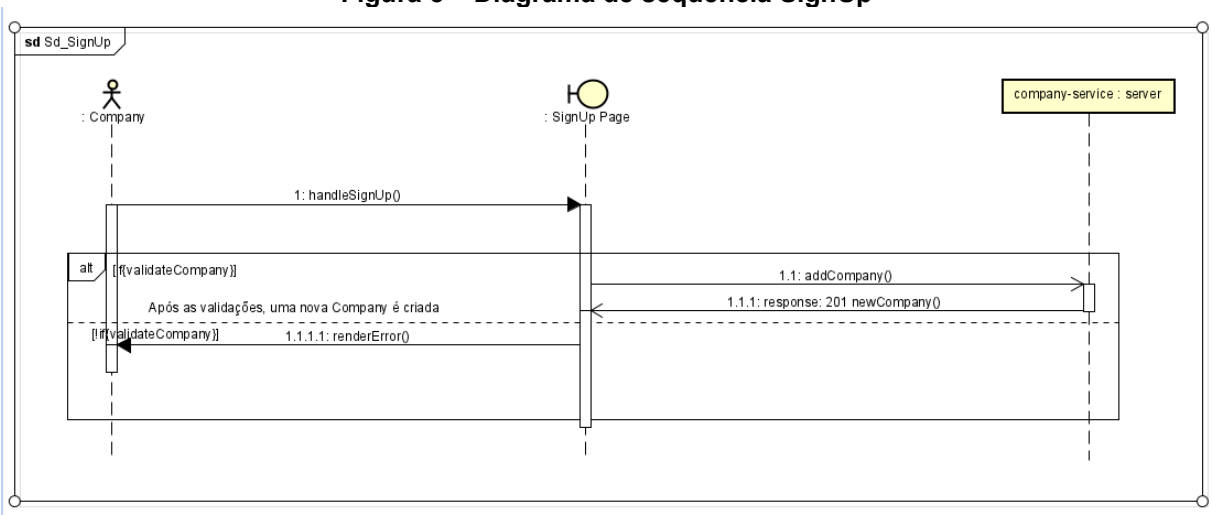
Server é o responsável por subir o servidor da API, ele instancia app o qual é uma instância do *framework* Express. O pacote routes armazena todas as *URLs* que a *API* deste serviço poderá receber e responder requisições. Os middlewares são validações de esquema que antecedem a chamada do *controller,* pacote responsável pelos métodos que processam e respondem essas requisições. Models possui a definição da entidade e todas as funções que acessam e manipulam os dados do banco de dados.

2.17 Diagrama de sequência

Para representar a sequência de mensagens trocadas entre as partes do sistema durante o seu uso pelos atores foi utilizado o diagrama de sequência da *UML*. Além dessa sucessão de mensagens, o diagrama de sequência também exibe a estrutura de controle entre os objetos. Em uma linha do tempo que começa na parte superior e desce gradativamente e expressa a sequência de mensagens e eventos (IBM, 2021).

A Figura 5 mostra a sequência de mensagens entre o sistema para o cadastro da *Company*, processo responsável por atender o ao requisito funcional RF01.

Figura 5 – Diagrama de sequência SignUp



Fonte: o autor

No processo de cadastro da *Company* serão validados os campos da seguinte forma:

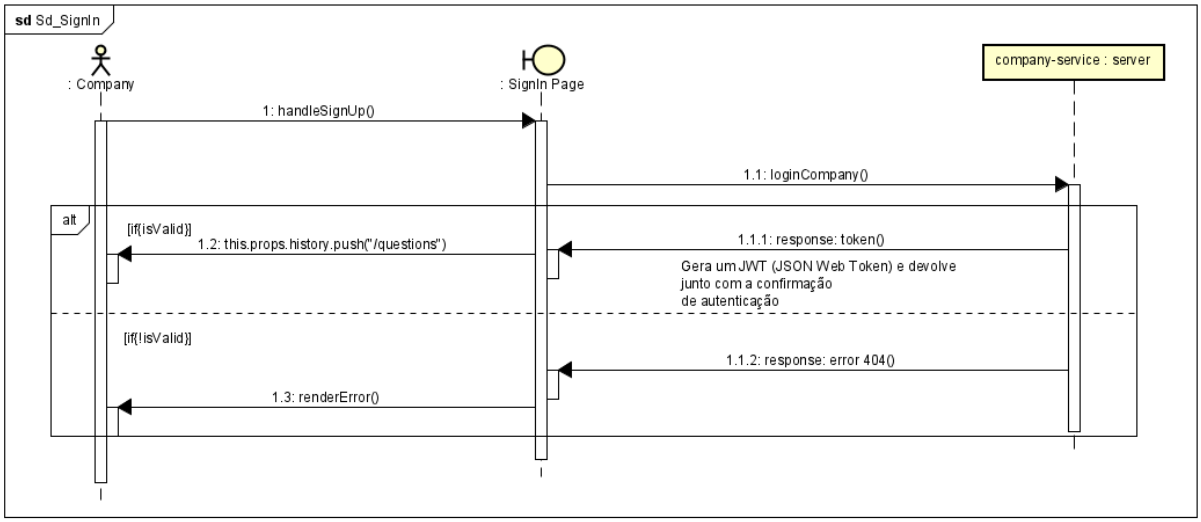
- Todos os campos são obrigatórios
- O nome da *Company* deve ter entre 3 e 150 caracteres
- O nome de usuário da *Company* deve ter entre 3 e 150 caracteres

- A senha deve ter entre 6 e 200 caracteres

Atendidos todos esses requisitos o *front-end* comunica-se com o *back-end* através de uma requisição *HTTP REST*, e caso o retorno dela seja 201 o cadastro é concluído.

A figura 6 exibe o diagrama que representa o processo de *login* do sistema.

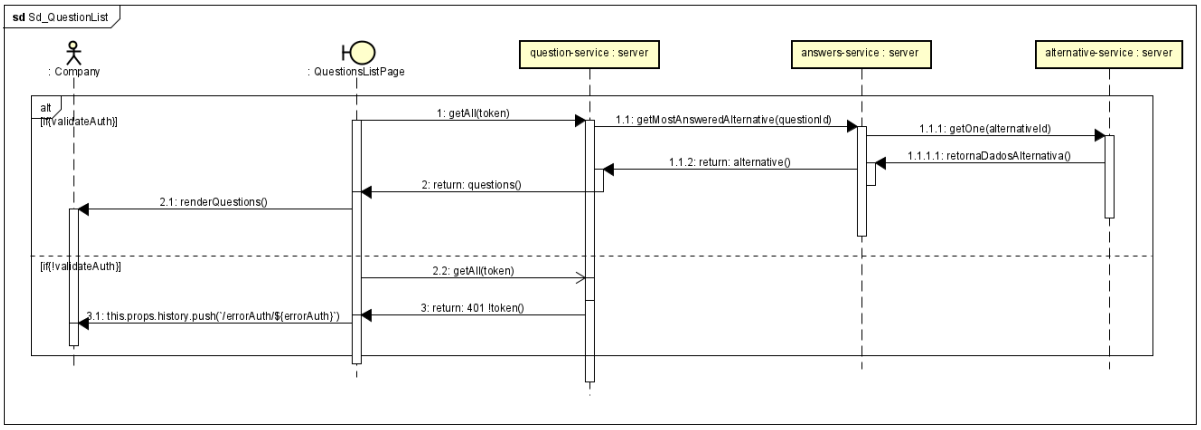
Figura 6 – Diagrama de sequência *SignIn*



Fonte: o autor

No processo de login de usuário serviço *company-service* recebe uma requisição *HTTP POST* com os dados enviados pelo *front-end*. Caso a combinação de usuário e senha seja válida um *JWT* será gerado e devolvido na requisição. Este *token* permitirá ao portador acessar as páginas e dados privados relativos a aquela *Company*, e será enviado pelo *front-end* no cabeçalho de todas as requisições que este usuário fizer. Após esse retorno, o sistema redirecionará o usuário para a página que exibe as perguntas cadastradas caso houver (RF02). Este processo é representado pela Figura 7 adiante.

Figura 7 – Diagrama de sequência *QuestionsList*



Fonte: o autor

O processo de listagem de perguntas valida o *token* do cabeçalho da requisição. Este token também contém em sua composição o *ID* da *Company* que está acessando os dados. Caso o token seja válido, a página do *front-end* aciona o serviço *question-service* buscando todas as perguntas que não foram excluídas.

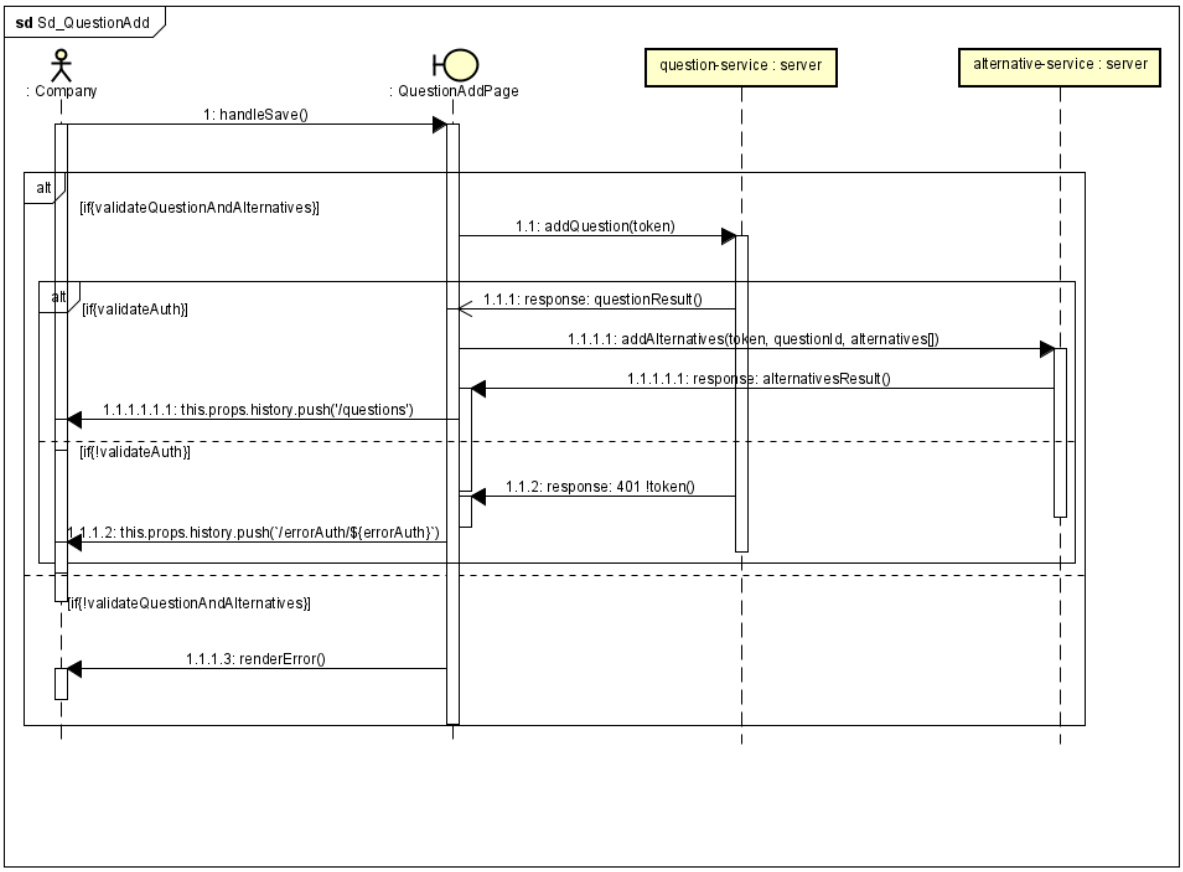
A exclusão de dados no sistema é tratada com um *soft delete*. Isso consiste em ao invés de realizar a exclusão de um registro do banco de dados, simplesmente alterar o seu *status* com alguma coluna de controle[20]. Durante o desenvolvimento o status de número 200 foi adotado para registros ativos e 400 para registros inativos.

Após retornada a lista de perguntas, o serviço *answers-service* retorna a alternativa mais respondida caso houver. Na sequência *alternative-service* recebe outra requisição *HTTP REST* e retorna os dados desta alternativa com maior número de respostas.

A Figura 8 a seguir demonstra o processo de adicionar uma nova pergunta (RF04).

[20] Disponível em: <https://albuquerque53.medium.com/entendendo-o-soft-delete-do-laravel-ce097c41214>

Figura 8 – Diagrama de sequência *QuestionAdd*



Fonte: o autor

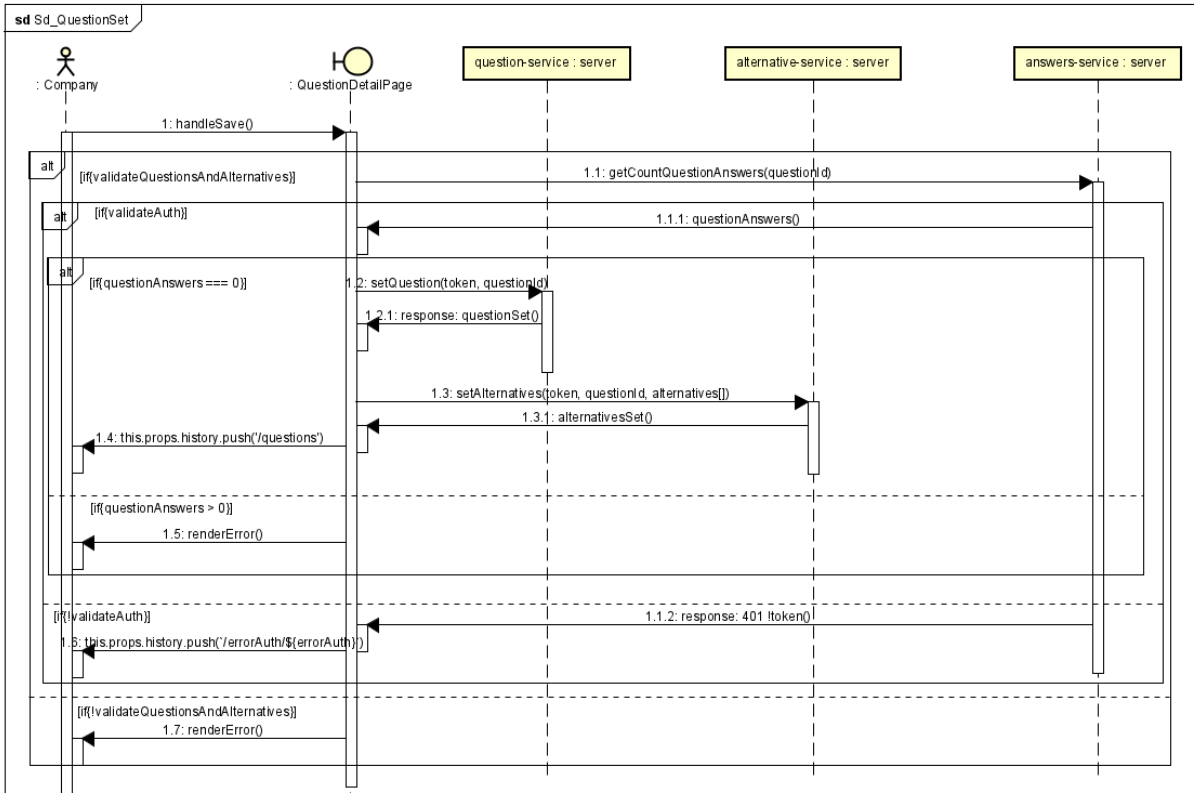
Para cadastrar uma pergunta as seguintes validações são necessárias:

- A quantidade de alternativas deve ser entre 2 e 5.
- A pergunta deve ter entre 10 e 150 caracteres.
- Todos os campos são obrigatórios
- A data de início em que ela pode ser respondida deve ser menor ou igual a data final.
- A descrição da alternativa deve ter entre 5 e 150 caracteres.

O *token* é validado em todo o processo deste sistema, caso ele tenha expirado o usuário será redirecionado a página de *login*.

Caso o responsável pela *Company* queira visualizar a quantidade de respostas que cada alternativa obteve, o diagrama a seguir representado pela Figura 9 explana o processo. Esse mesmo processo também mostra as mensagens trocadas entre as entidades do sistema para alterar uma pergunta ou alternativa.

Figura 9 – Diagrama de sequência *QuestionSet*



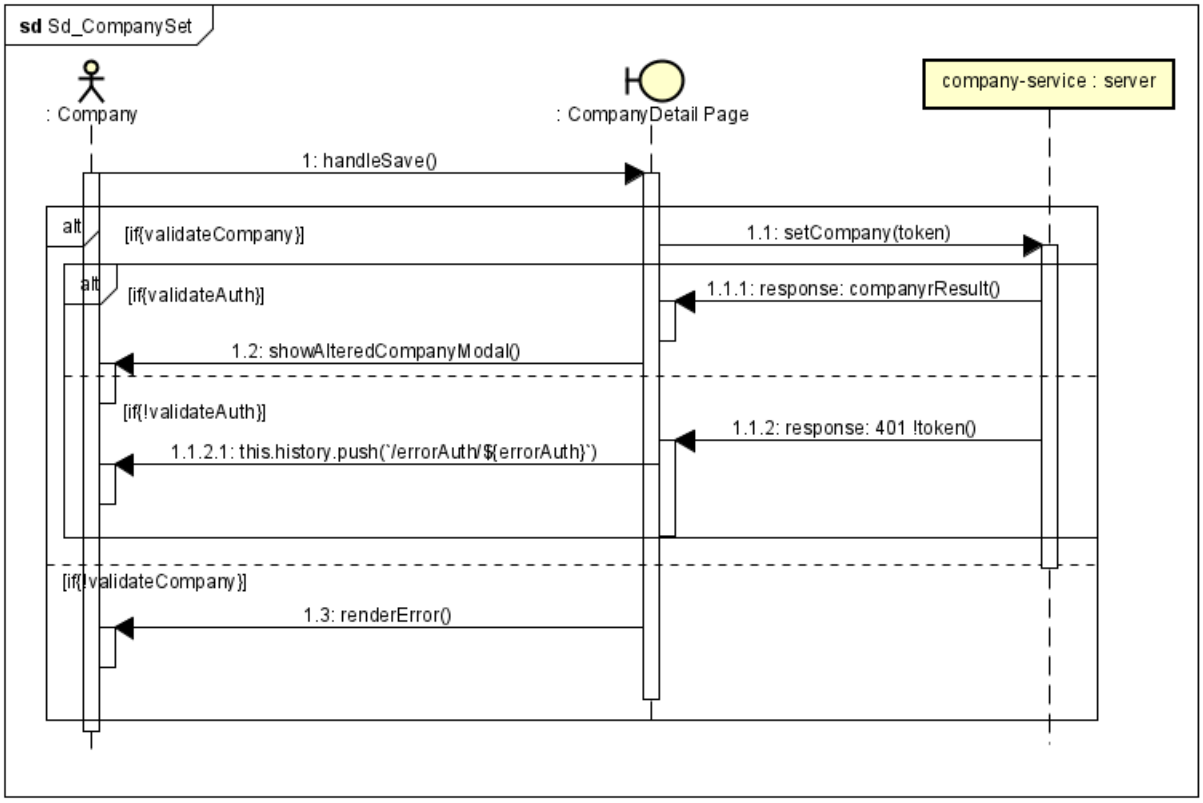
Fonte: o autor

Devido ao fato da página do *front-end* já possuir armazenado em seu *state* todos os dados da pergunta, a requisição para obtê-los não é necessária. Então o serviço *alternative-service* é acionado e retorna as alternativas daquela pergunta em específico. Logo após, *answers-service* fornece a informação da quantidade de respostas que cada alternativa possui (RF03).

Para alterar uma pergunta (RF04) além das validações feitas durante o seu cadastro anteriormente citado, também é validado se ela possui respostas. Em caso positivo ela não poderá ser alterada (RF08).

Além das perguntas, os dados da própria *Company* podem ser alterados. A seguir a Figura 10 apresenta o diagrama que demonstra as mensagens trocadas entre as entidades.

Figura 10 – Diagrama de sequência *CompanySet*

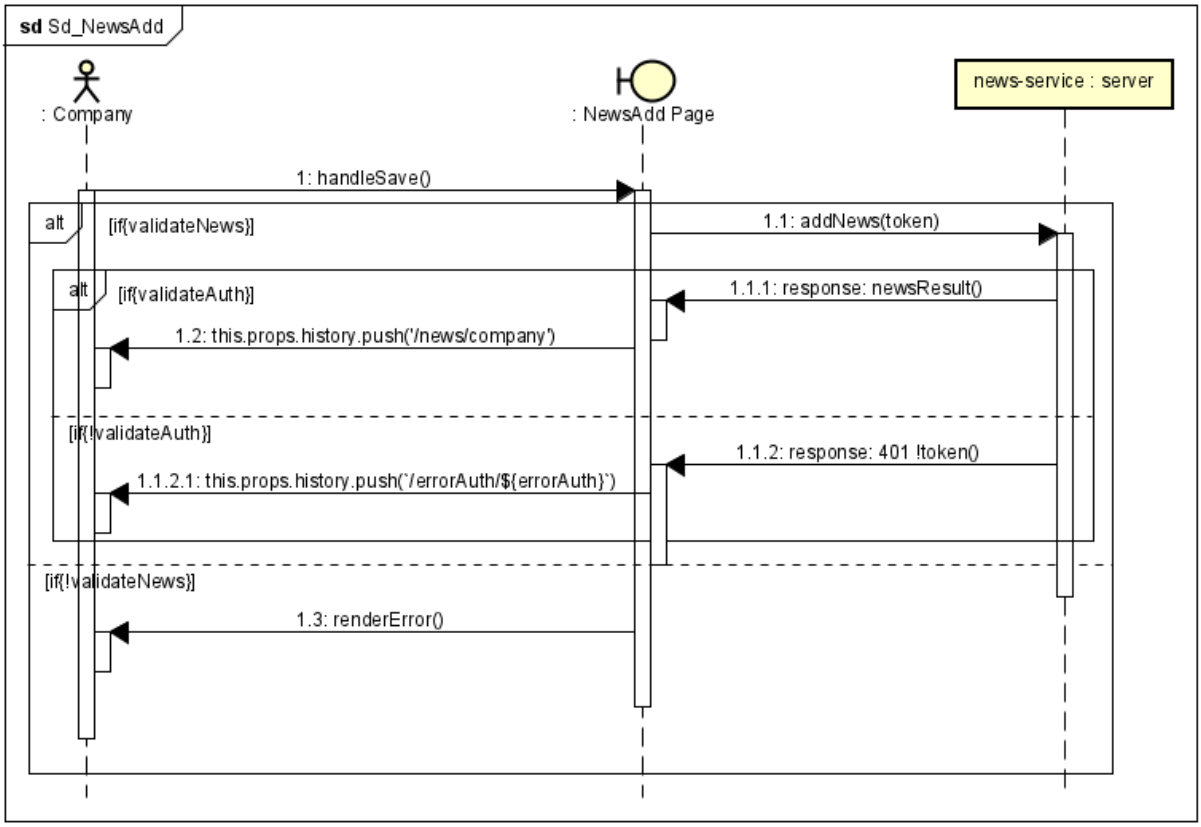


Fonte: o autor

Além de atender ao requisito funcional RF01, essa sequência também atende o requisito funcional RF09. Quando uma nova *Company* é criada, como foi demonstrado na Figura 4, o microsserviço *company-service* cria automaticamente um *QRCode* para esta *Company*. Este código pode ser alterado, gerando um novo *QRCode* único para o acesso das perguntas a serem respondidas e quadro de notícias. Para isso também são feitas as mesmas validações de tamanhos de campos e obrigatórios que no processo de criação de uma nova *Company*. Após ajustados os dados o sistema faz uma requisição *HTTP POST* para o serviço *company-service*. Processada essa requisição esta *Company* é alterada.

Além de cadastrar perguntas, o sistema também permite ao ator *Company* criar notícias e exibi-las em um quadro (RF05). Essas podem ser acessadas através da leitura do mesmo *QRCode* o qual concede acesso as perguntas. Assim como as enquetes, as notícias também possuem data de início e fim. A troca de mensagens do *software* para isso é mostrado a seguir na Figura 11.

Figura 11 – Diagrama de sequência *NewsAdd*



Fonte: o autor

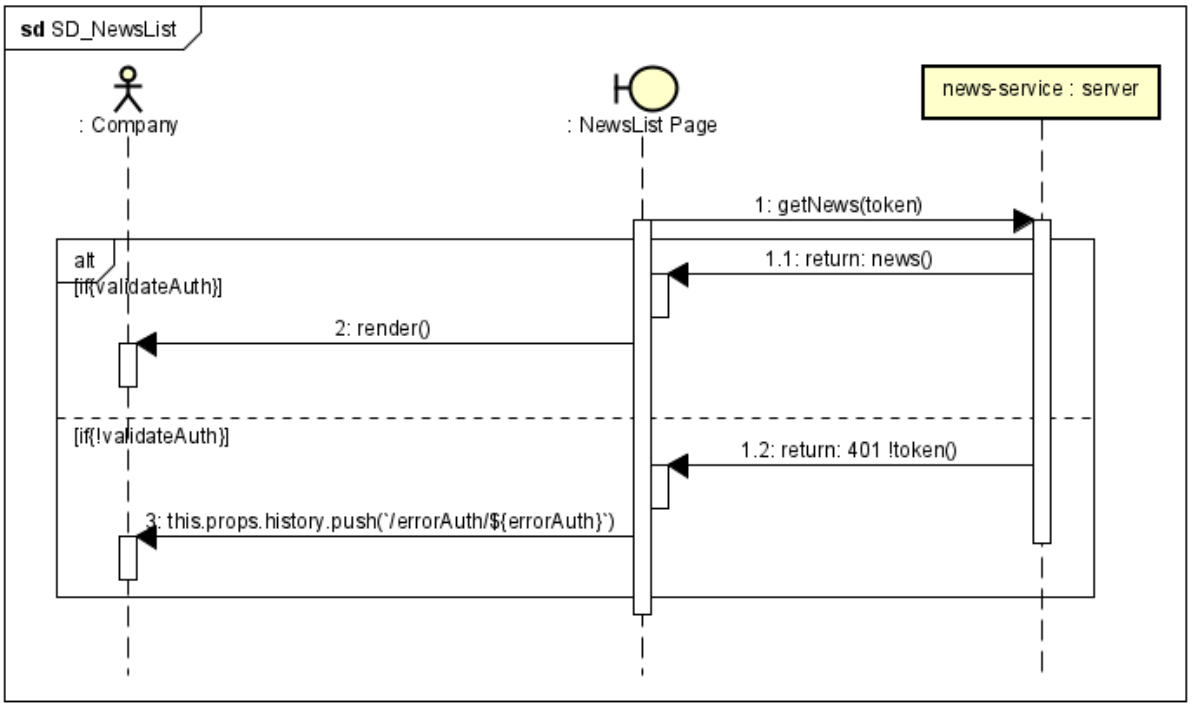
Essas notícias são validadas conforme o seguinte:

- A data inicial que essa notícia será exibida deverá ser menor ou igual a data final ou data término.
- Todos os campos são obrigatórios.
- O seu título deve ter entre 5 e 60 caracteres
- A descrição deve ter entre 5 e 2000 caracteres.

O intuito do quadro de notícias é que ele seja genérico. O ator *Company* pode divulgar resultados de enquetes ou qualquer outro tipo de avisos. Ao salvar essa notícia, a página do *front-end* envia ao microsserviço *news-service* uma requisição solicitando a alteração. Em caso de sucesso uma mensagem JSON com código 201 é enviada ao *front-end*, então a página informa ao usuário que a notícia foi alterada.

O sistema oferece uma lista de todas as notícias cadastradas de acordo com o diagrama da Figura 12 abaixo.

Figura 12 – Diagrama de sequência *NewsList*

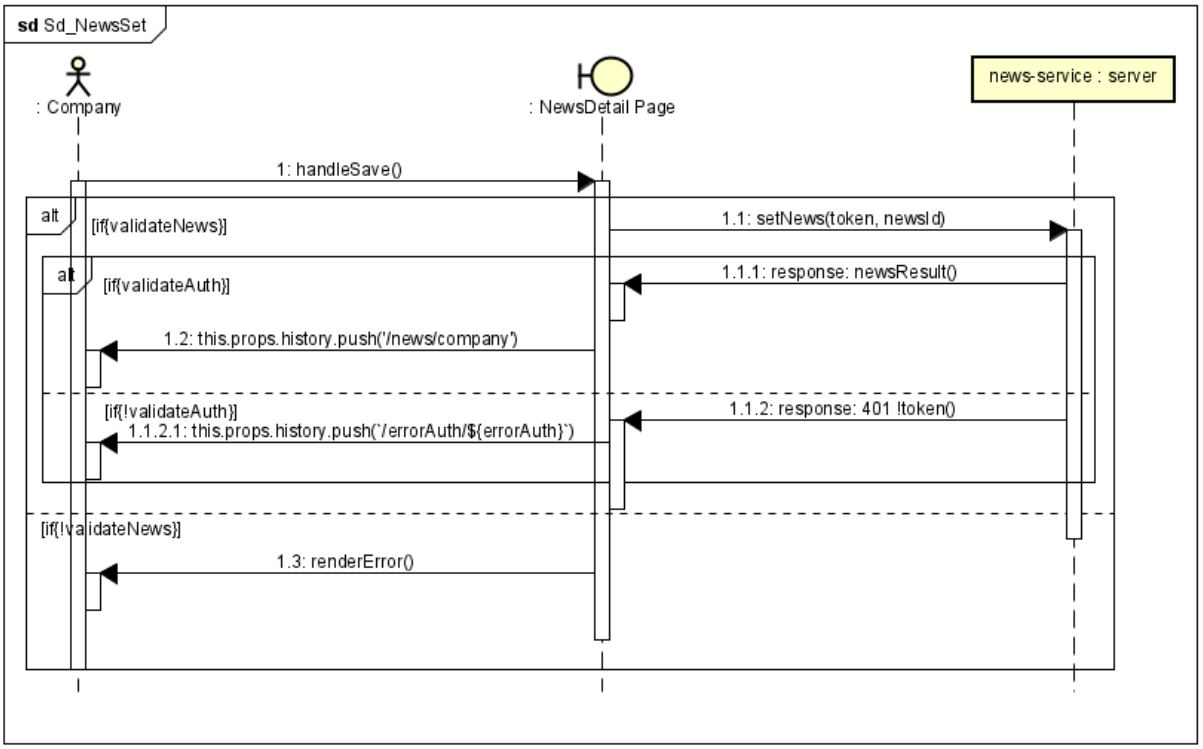


Fonte: o autor

A troca de mensagens para listagem de notícias é semelhante à listagem de perguntas. A página do *front-end* envia ao serviço *news-service* uma requisição *HTTP* com o *token* no cabeçalho dela. Esse *token* contém o *id* da *Company*, e caso ele seja válido, todas as notícias daquela *Company* são buscadas pelo *back-end*, retornadas ao *front-end* e renderizadas em uma lista. Caso o *token* não seja válido o usuário é redirecionado a página de *login*.

Para alterar uma notícia as mesmas validações que ocorrem em sua criação acontecem novamente. A seguir a Figura 13 representa a troca de mensagens entre o sistema para essa alteração.

Figura 13 – Diagrama de sequência *NewsSet*

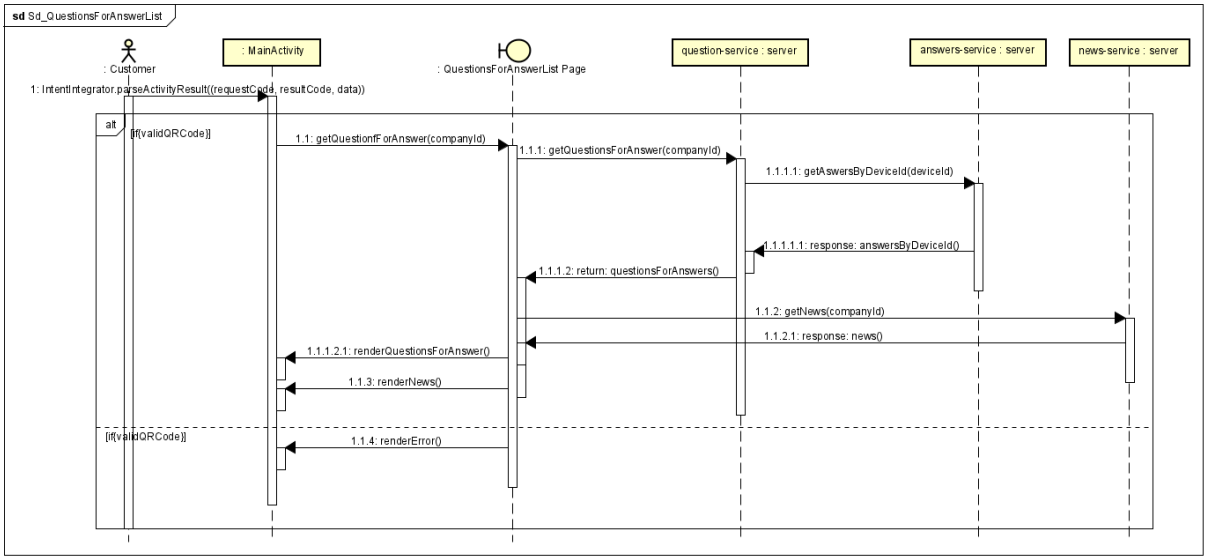


Fonte: o autor

Como em todas as requisições do sistema, a validade do *token* também é verificada. Caso ele e todos os campos que compõem uma notícia estejam válidos, o *back-end* realiza a alteração e retorna ao *front-end* uma resposta 201 na requisição *POST*.

Através da leitura do *QRCode* pelo aplicativo móvel do sistema, é possível visualizar as perguntas a serem respondidas. Estas são todas as perguntas que ainda não foram respondidas pelo dispositivo em questão (RF10), e que se encontram no período de respostas, ou seja, a data atual está entre a data inicial e a data final (RF07). O quadro de notícias também segue essa regra (RF06), são exibidas somente as notícias disponíveis para o período. A Figura 14 abaixo demonstra o diagrama de sequência *UML* responsável pela visualização das enquetes abertas e as notícias.

Figura 14 – Diagrama de sequência *QuestionsForAnswerList*

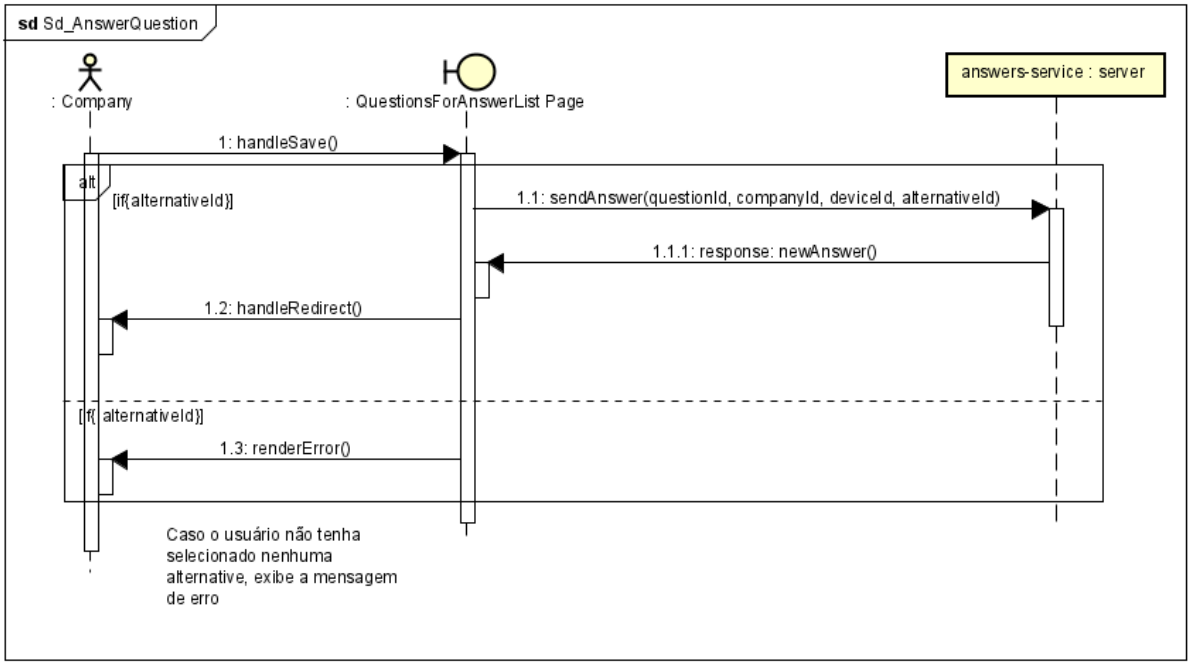


Fonte: o autor

O *QRCode* pode ser disponibilizado para a leitura impresso em locais estratégicos do estabelecimento, ou então exibido em um painel ou telão, sendo cada *QRCode* único por *Company*. O aplicativo móvel do Perguntador faz a leitura deste código, e caso ele for válido, abre uma *Web View* com a página de perguntas a serem respondidas e o quadro de notícias. Essa página do *front-end* busca no *back-end* através de requisições *HTTP REST* todas as perguntas e notícias às quais a data atual está entre a data inicial e final. O *id* do dispositivo móvel que está lendo o *QRCode* também é enviado na requisição. Os serviços *question-service* e *answers-service* somente retornarão enquetes que ainda não foram respondidas por este dispositivo (RF11).

Após selecionada uma pergunta a sua resposta poderá ser enviada. A seguir, a Figura 15 mostra as mensagens trocadas pelo sistema para que essa resposta seja gravada.

Figura 15 – Diagrama de sequência *AnswerQuestion*



Fonte: o autor

A única validação que acontece neste processo é se foi selecionada uma alternativa. Após isso a página do *front-end* envia uma requisição *HTTP POST* para o microsserviço *answers-service*. A resposta então é vinculada a pergunta, e o serviço retorna um código de resposta 201 para a página do *front-end*.

3 **DESENVOLVIMENTO**

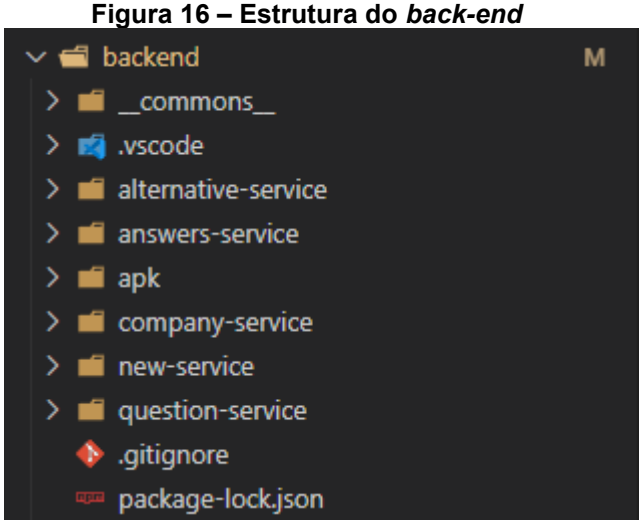
A construção deste sistema foi feita com a divisão entre o *front-end* e *back-end*. O *back-end* possui todos os microsserviços e *Web APIs* que recebem as requisições *HTTP REST* e mantém as informações no banco de dados. O *front-end* apresenta páginas web as quais o usuário interage, realiza as validações básicas do negócio e consomem as *APIs* do *back-end*. E também há o desenvolvimento da aplicação móvel Android, a qual é responsável pela leitura do *QRCode* e acessar as perguntas a serem respondidas e o quadro de notícias. Porém, esta somente utiliza a câmera do dispositivo e captura o seu *id*, deixando a página em si sob responsabilidade do *front-end*, processo feito através de uma *Web View*.

3.1 *Back-end*

Como todos os microsserviços possuem funcionamento semelhante, esta seção apresentará como foi construído o serviço do quadro de notícias, chamado dentro do sistema de *news-service*.

A raiz do *back-end* demonstrada na Figura 16 possui todos os microsserviços necessários para o funcionamento do sistema, e cada um deles deve ser instanciado para isso. Dentro da pasta destes serviços há os arquivos de configurações do programa:

Figura 16 – Estrutura do *back-end*



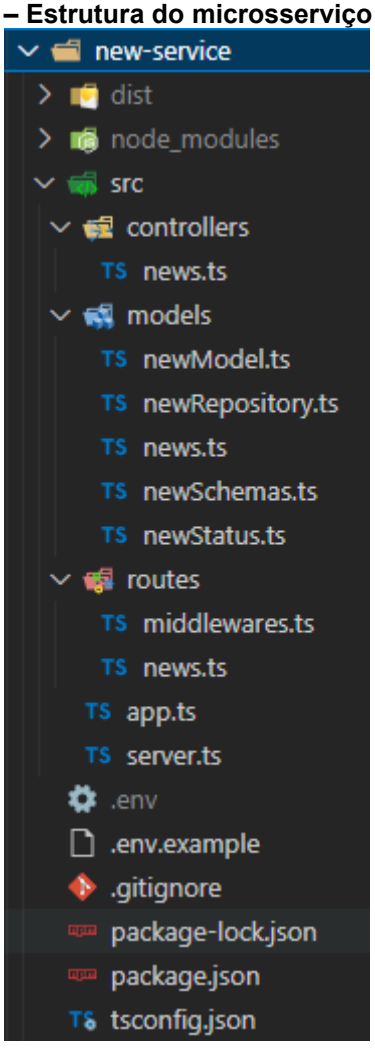
Fonte: o autor

- O arquivo .env.example contém a definição das variáveis de ambiente.

- Os diretórios que não serão enviados para o github ficam no .gitignore.
- Todas as dependências do sistema são definidas em package-lock.json e package.json, junto com a versão de cada biblioteca node utilizada.
- O TypeScript é configurado em tsconfig.json.

A Figura 17 seguinte exibe toda a estrutura interna de um microsserviço do sistema. O arquivo app.js contém as definições da *Web API* vindas do *framework* ExpressJS, ele é quem irá subir o servidor e responder as requisições. Já o arquivo server instancia o app juntamente com o *framework* Sequelize, o *ORM* responsável pela conexão ao banco de dados. Os diretórios principais do serviço são o *controller*, *models* e *routes*.

Figura 17 – Estrutura do microsserviço *news-service*



Fonte: o autor

O diretório *routes* possui um arquivo de rotas e outro de *middlewares*, este contém validações feitas imediatamente no recebimento da requisição e antes da chamada do *controller* para o seu processamento. O arquivo de rotas instancia a classe *Router* do ExpressJS e possui todos os *end-points* necessários para o funcionamento do serviço. Essas rotas executam os *middlewares* que validam, quando possível, autenticação do usuário através do *token*, e *schemas* de requisição.

O diretório *Models* contém os arquivos necessários para a conexão e manutenção das informações no banco de dados. O arquivo news.ts é a interface a qual define todos os atributos dessa entidade e os seus tipos. O arquivo newModel.ts contém implementa esta interface e define os tipos de acordo com os do *ORM* Sequelize. Ele também permite outras configurações como por exemplo se o atributo permite valores nulos ou se ele incrementa seu valor automaticamente, esta configuração é utilizada para o *id* da entidade no banco de dados. O ENUM com os status da notícia, criado, ativo, excluído ou suspenso encontra-se no arquivo newStatus.ts. Por fim newRepository.ts armazena todas as funções que efetivamente acessam o banco de dados para manutenção deles através do Sequelize.

O *controller* deste microsserviço possui um único arquivo, suas funções são as que atendem as requisições *HTTP POST* e invocando as funções do model quando necessário.

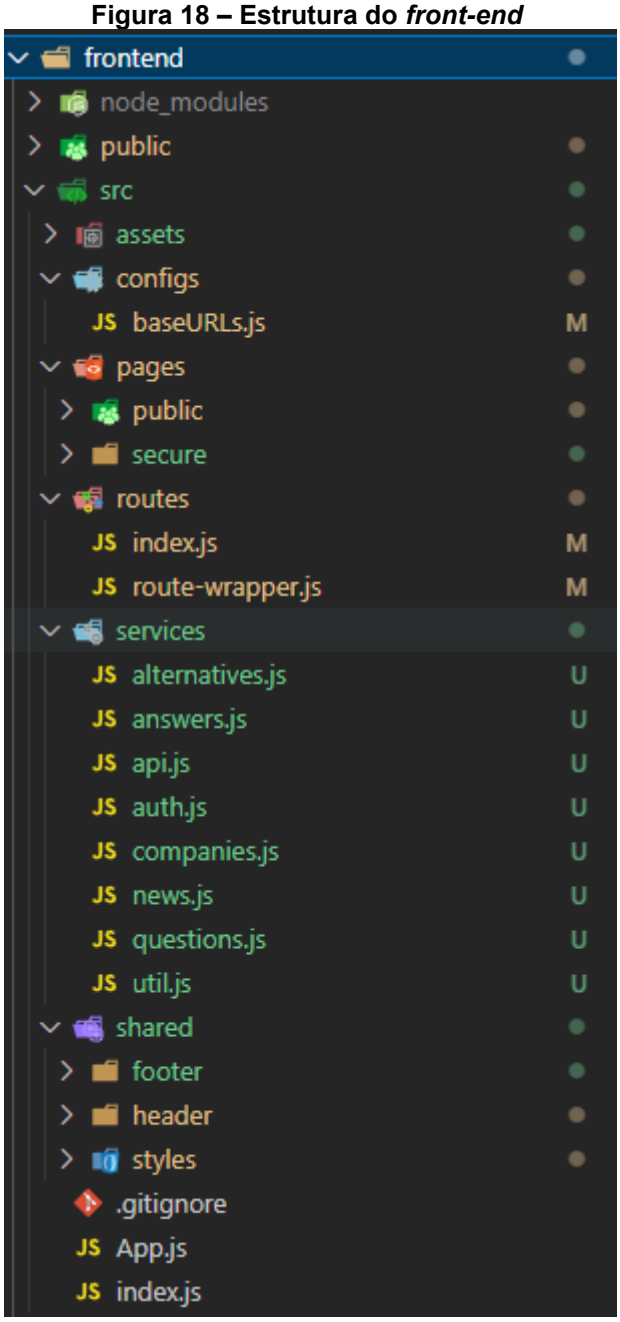
O projeto é de código aberto e encontra-se no seguinte repositório do Github.

- https://github.com/iankoski/question-sender-backend

3.2 *Front-end*

A Figura 18 a seguir exibe todas as pastas e arquivos utilizados no *front-end*.

Figura 18 – Estrutura do *front-end*



Fonte: o autor

Assets é o diretório onde todos os ícones e logos apresentados nas páginas são armazenados. *Configs* possui um único arquivo com um ENUM retornando todos os *end-points* do *back-end*. *Shared* é a pasta dos componentes compartilhados cabeçalho e rodapé da página e estilos *CSS*. O diretório *services* possui os arquivos e funções que realizam as requisições *HTTP REST* para o *back-end*, cada arquivo correspondendo a um microsserviço. Os arquivos app.js e index.js carregam o React e o ReactDOM para renderizar os componentes nas páginas. Essas páginas em si

situam-se na pasta *pages*, que é dividida entre páginas públicas e seguras e cada uma delas possui apenas um arquivo index.js, o qual é responsável pela renderização do componente React. As únicas páginas públicas do sistema são a de *login* e a de cadastro, as quais podem ser acessadas sem nenhuma autenticação. As páginas seguras exigem autenticação de usuário ou leitura do *QRCode* para esse acesso. A pasta *routes* possui todas as rotas do *front-end*, ou seja, todas as *URLs* que ele responderá. Ela define também qual componente React será renderizado no navegador ao acessar essa rota.

A criação de todos os componentes do sistema é semelhante, a Figura 19 adiante exibe o componente responsável por adicionar uma nova notícia ao quadro de notícias.

Figura 19 – Componente *NewsAdd*

```
1   import React from 'react';
2   import Header from '../../../shared/header';
3   import Footer from '../../../shared/footer';
4   import { PageContent, BoxForm } from '../../../shared/styles';
5   import { Container, Button, Form, Alert, Row, Col, Modal } from 'react-bootstrap';
6   import NewsService from '../../../services/news';
7   import { validateNews } from '../../../services/util';
8   import { withRouter, Link } from 'react-router-dom';
9
10  class NewsAdd extends React.Component {
11      constructor(props) {···
22      }
23
24      handleAddedNewsModal() {···
26      }
27
28      handleSave = async (event) => {···
49      }
50
51      handleRedirectModal = async (event) => {···
54      }
55
56      renderError = () => {···
63      }
64
65      render() {
66          return (
67              <>
68                  <Header />
69                  <PageContent>
```

Fonte: o autor

A classe NewsAdd estende React.Component, ou seja, a página é um componente React. Ela armazena no *state* de página todos os dados que são

renderizados em tela. Dentro desta classe estão todos os métodos responsáveis pela geração de uma nova notícia, validações necessárias, funções de redirecionar o usuário para a página anterior e renderização de mensagens de erro quando houverem. Por fim, essa classe retorna o componente em si, e renderiza todos os elementos da tela, neste caso, um formulário *web* para criação de uma nova notícia ao quadro.

Assim como o *back-end*, esta parte do projeto encontra-se no seguinte repositório público do Github.

- https://github.com/iankoski/question-sender-frontend

4 **RESULTADOS**

Este trabalho teve como objetivo utilizar o modelo de arquitetura de microsserviços com *Web APIs*. Aplicado ao estudo de caso da construção de um sistema que auxilie os responsáveis por quaisquer tipos de estabelecimentos que atendam público a coletar a opinião de seus frequentadores, através de enquetes de múltipla escolha. Não há restrição quanto ao tipo de estabelecimento, o programa faz essa abordagem de forma genérica e pode ser utilizado, por exemplo, para bares, restaurantes, repartições públicas, clínicas médicas ou até mesmo igrejas. Contudo, devido a necessidade de se armazenar o id do aparelho móvel que efetua uma resposta para garantir apenas uma resposta por dispositivo, o uso desta plataforma é restrito a dispositivos *Android*.

O *back-end* do Perguntador foi construído com o *framework* ExpressJS e a linguagem TypeScript. Essa combinação beneficia a codificação pois combina a agilidade do uso de um *framework* com a forte tipagem que o TypeScript traz, ele acusa diversos erros ainda em tempo de compilação e agiliza a construção do programa. Cada *API* do sistema é independente das demais, tanto nas classes e funções como nas tabelas do banco de dados. Quando houve necessidade de interação entre as entidades, requisições *HTTP* foram feitas entre elas para atender essa demanda.

O *front-end* foi criado utilizando ReactJS que também contribui bastante para reduzir o tempo de codificação. Isso porque essa ferramenta possibilita a criação de componentes que consistem em partes independentes e reutilizados no site (REACT, 2022). Desta forma, uma vez criado um componente, como por exemplo um botão, este pode ser aplicado em qualquer página do *front-end*.

O aplicativo móvel do sistema foi desenvolvido com Java Android, visto que essa parte do programa é responsável somente por ler o *QRCode*, guardar o *id* do dispositivo do usuário, e redirecioná-lo para a página apropriada do *front-end*.

As páginas criadas são utilizadas para:

- Manter o estabelecimento no sistema.
- Realizar o *login* do estabelecimento.
- Visualizar todas as perguntas cadastradas, assim como a alternativa mais respondida.

- Visualizar uma pergunta e a quantidade de respostas que cada alternativa obteve.
- Manter as perguntas.
- Visualizar as notícias que foram criadas no quadro de notícias.
- Manter as notícias.
- Visualizar o *QRCode*, para exibição em uma tela ou impressão.
- Ler o *QRCode* e visualizar o quadro de notícias e a lista de perguntas disponível para serem respondidas.
- Responder uma pergunta através de seleção de uma alternativa.

Todas as páginas exibem informações sobre suas finalidades no rodapé. A seguir a Figura 20 apresenta a página inicial da aplicação *web*, a tela de *login* do sistema.

Figura 20 – *SignIn Page*

Fonte: o autor

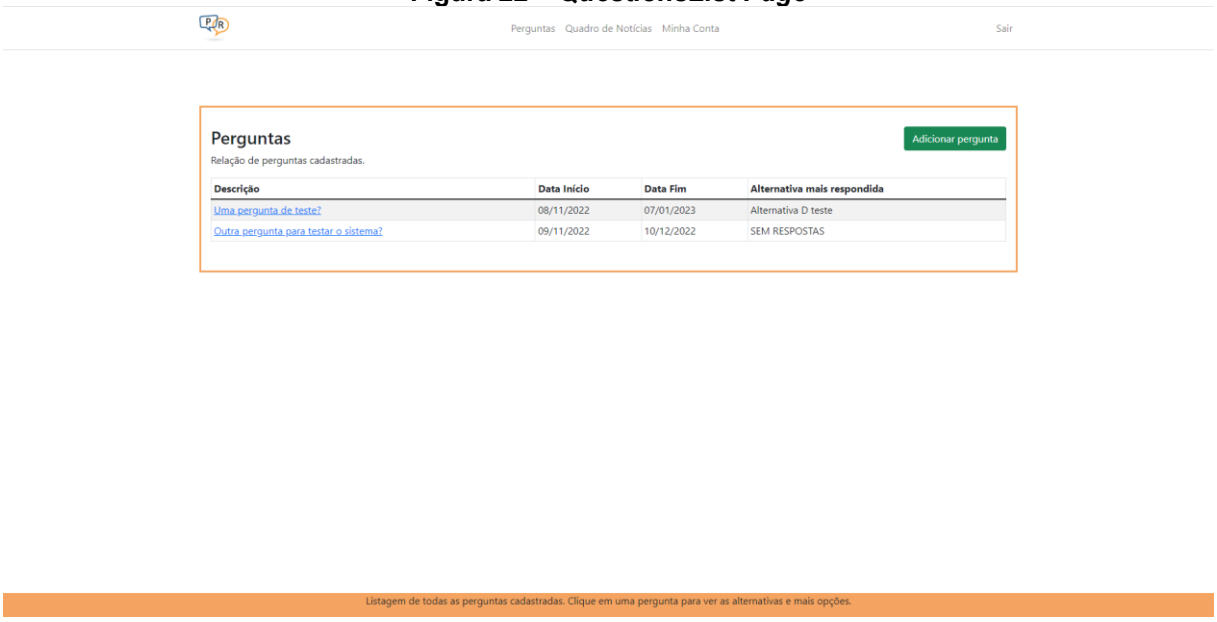
A partir desta página o responsável pelo estabelecimento pode realizar o *login* no sistema ou cadastrá-lo, este o direciona para a página a seguir representada pela figura 21.

Figura 20 – *SignUp Page*

Fonte: o autor

A Figura 22 na sequência representa a primeira página exibida ao usuário após o *login*.

Figura 22 – *QuestionsList Page*

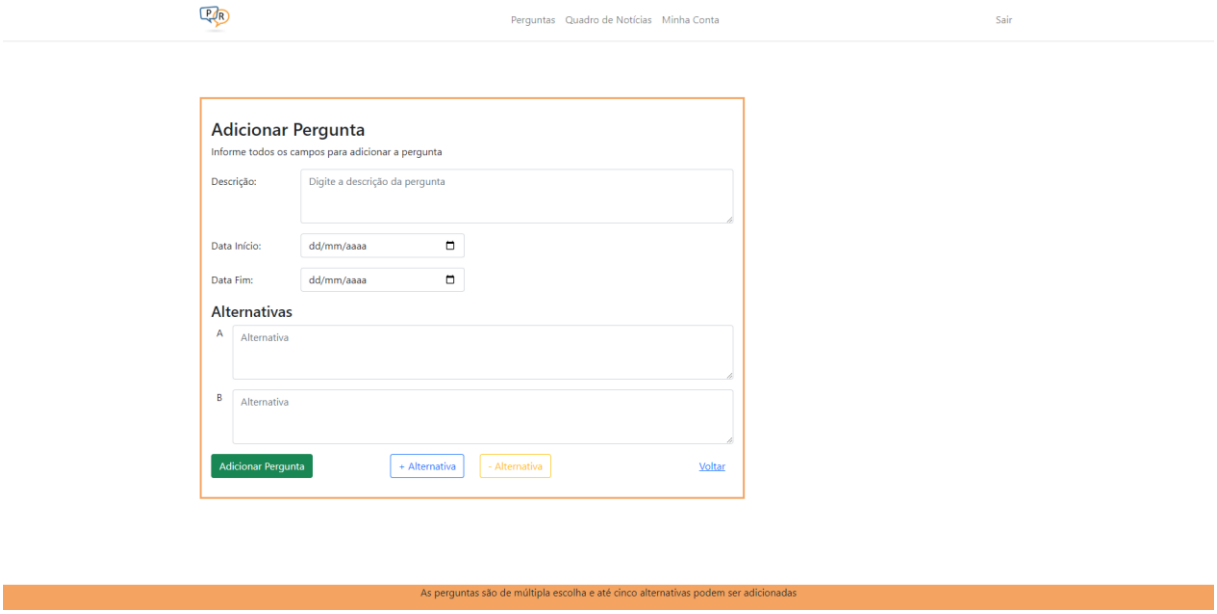


Fonte: o autor

Essa página contém um menu de navegação e a lista de todas as perguntas cadastradas juntamente com a alternativa mais respondida, caso houver respostas.

Há também um botão para adição de uma nova pergunta, este processo é exibido na Figura 23 a seguir.

Figura 23 – *QuestionAdd Page*

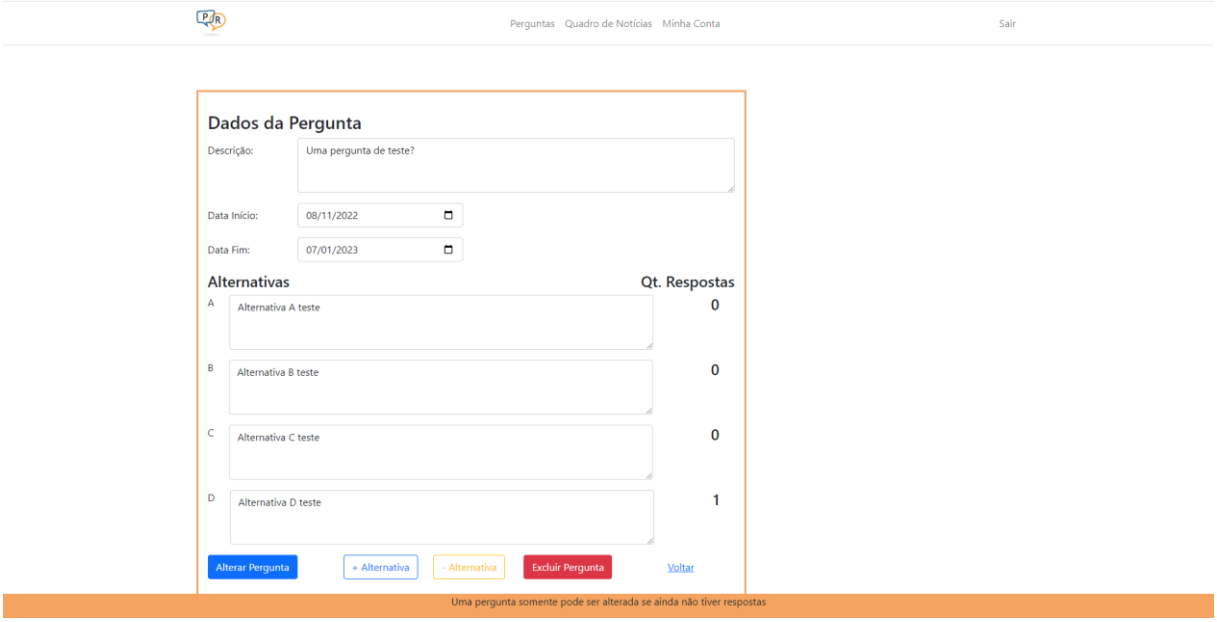


Fonte: o autor

Além dos campos do formulário para o cadastro de uma nova pergunta, a página também permite incrementar o decrementar o número de alternativas.

Clicando em uma pergunta na página que as lista, o sistema direciona o usuário a uma página com os detalhes dela e a quantidade de respostas obtidas por cada alternativa, conforme a Figura 24 adiante.

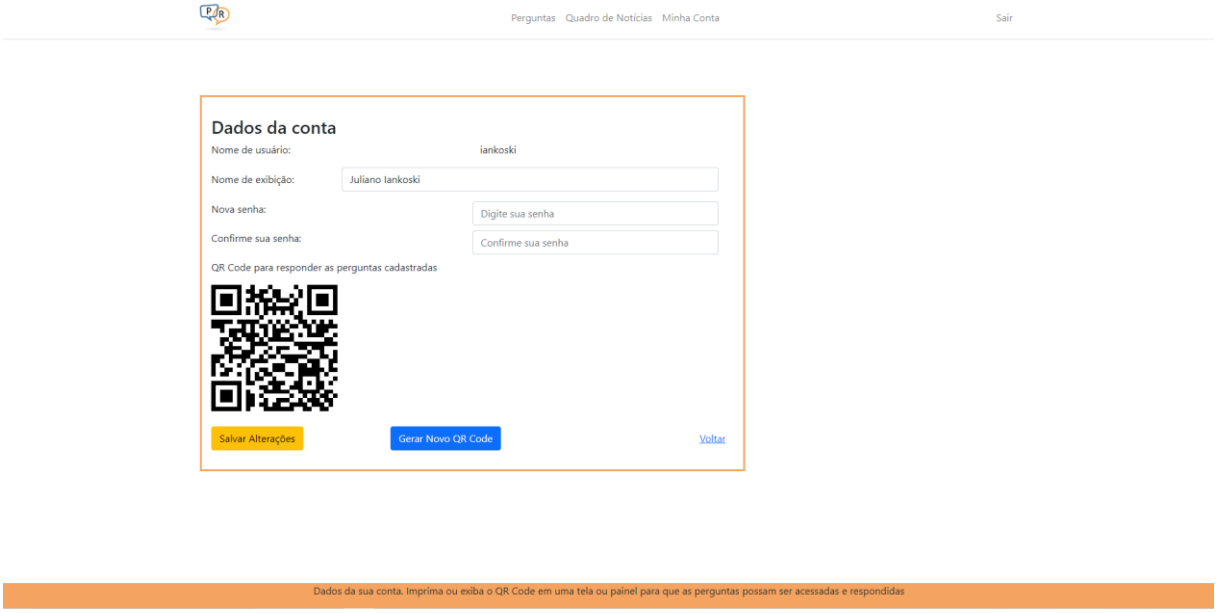
Figura 24 – *QuestionDetail Page*



Fonte: o autor

A opção “Minha Conta” no menu superior de todas as páginas exibe os dados dela e o *QRCode* gerado, conforme a próxima Figura 25.

Figura 25 – *CompanyDetail Page*



Fonte: o autor

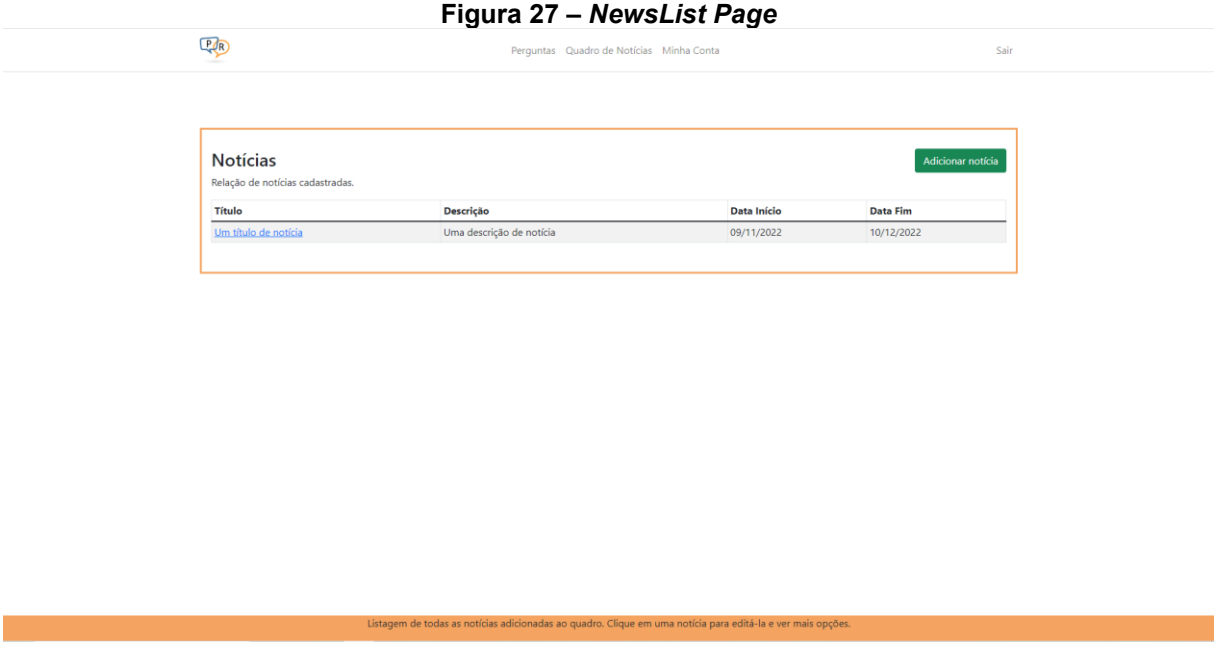
Como este *QRCode* pode ser impresso ou exibido em um telão para sua leitura, ao clicar sobre ele uma visão mais limpa do *QRCode* é apresentada para esta impressão ou exibição, conforme a Figura 26 a seguir.

Figura 26 – *QRCodeDetail Page*

Fonte: o autor

Também no menu do cabeçalho da página há a opção para visualização do quadro de notícias, acordante com a Figura 27 a seguir.

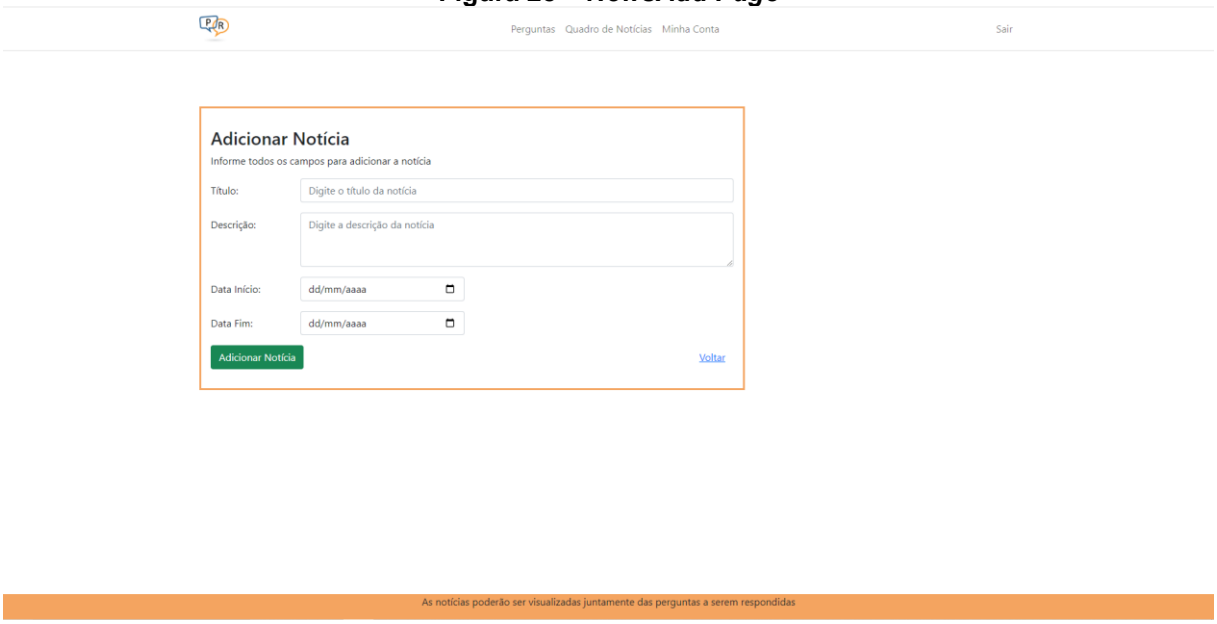
Figura 27 – *NewsList Page*



Fonte: o autor

As notícias do quadro são exibidas em formato de lista, nesta página existem as opções de adicionar uma notícia e alterar uma já cadastrada. A Figura 28 na sequência exibe a página para o cadastro de uma nova notícia.

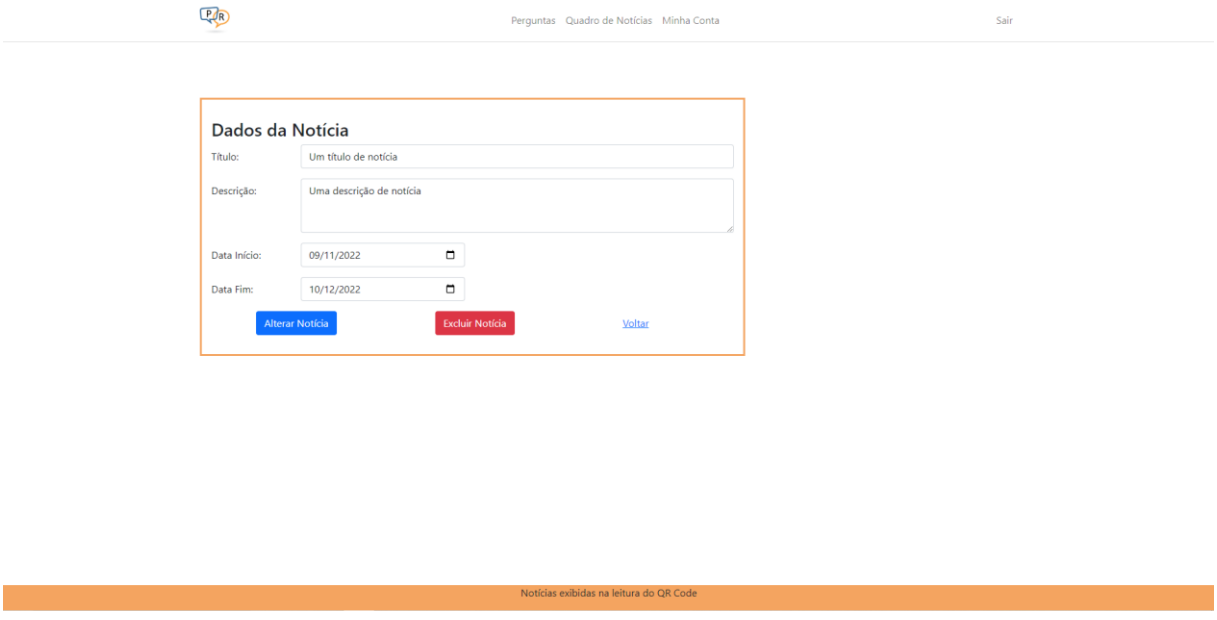
Figura 28 – *NewsAdd Page*



Fonte: o autor

A Figura 29 demonstra a página para alterar uma notícia do quadro.

Figura 29 – *NewsDetail Page*



Fonte: o autor

Essas são todas as páginas acessáveis pelo criador das enquetes. A Figura 30 a seguir exibe a página renderizada no dispositivo móvel de quem ler um *QRCode* do perguntador, com a lista de perguntas a serem respondidas.

Figura 30 – *QuestionsForAnswerList Page:* perguntas

Fonte: o autor

O cabeçalho possui opções para visualização das perguntas e do quadro de notícias. Ao clicar sobre uma pergunta da lista, a seguinte Figura 31 exibe suas alternativas.

Figura 31 – *QuestionsForAnswerDetail Page*

Fonte: o autor

Esta página possui opções de enviar a resposta e voltar à página anterior. A seguir, a Figura 32 exibe o quadro de notícias do sistema.

Figura 32 – *QuestionsForAnswerList Page*: quadro de notícias

Fonte: o autor

5 TRABALHOS FUTUROS

Para aprimorar a experiência de todos os atores envolvidos no uso do Perguntador, as seguintes ações podem ser aplicadas futuramente:

- Eliminar o uso do aplicativo móvel Android e realizar a autenticação de dispositivo através de *APIs* de *login* do Google ou do Facebook, por exemplo.
- Exibir uma lista com todas as respostas já enviadas por um usuário
- Exibir uma lista para os frequentadores dos estabelecimentos com a alternativa mais respondida, caso o proprietário ou responsável assim queira.
- Realizar o *deploy* do sistema em nuvem e torná-lo público.

6 CONSIDERAÇÕES FINAIS

Apresentado modelo de arquitetura de microsserviços com *Web APIs*, foi levantado um estudo de caso sobre as dificuldades que donos de estabelecimentos os quais atendam público possuem em definir os gostos, vontades, opiniões e sugestões de seu público alvo. O Objetivo deste trabalho de conclusão de curso foi automatizar o processo de coleta de opinião pública aplicando estes conceitos.

O sistema desenvolvido é genérico e pode atender qualquer tipo de local, e também permite que o estabelecimento envie informações a seu público através do quadro de notícias. Devido ao fato de o sistema exigir presencialidade de quem for responder uma pergunta, espera-se uma coleta de opinião assertiva. Ao conhecer melhor a opinião de seus frequentadores, é esperada uma satisfação maior deles quanto ao atendimento ou serviço deste lugar que faz uso do Perguntador. Mesmo com a limitação de que somente dispositivos Android podem responder perguntas.

Embora a arquitetura de microsserviços possua suas vantagens, as quais foram descritas neste trabalho, aqui ela não foi testada com um caso de uso que exija grande integridade dos dados. Como por exemplo um sistema financeiro que receba requisições frequentes para atualização de saldo bancário. Logo não é possível afirmar, com os resultados deste trabalho, se ela seria adequada para este uso.

Este trabalho também aprofundou os conhecimentos acadêmicos do autor quanto ao levantamento e definição de requisitos do sistema e no desenvolvimento de aplicações *web* com microsserviços, *Web APIs, NodeJS* e *React*. Ele também teve um enriquecimento curricular, conseguiu aplicar ao desenvolvimento do trabalho grande parte das habilidades e conhecimentos adquiridos ao longo do curso de Tecnologia em Sistemas para Internet.

8 REFERÊNCIAS

IEEE SPECTRUM. **Top Programming Languages 2022.** 2022. Disponível em: <https://spectrum.ieee.org/top-programming-languages-2022>. Acesso em: 16 out. 2022.

PRESCOTT, Preston. **Programação em JavaScript**. Babelcube Inc., 2016.

MORAES, William Bruno. **Construindo aplicações com NodeJS**. Novatec Editora, 2015.

NODE JS. **Sobre Node.js.** 2022. Disponível em: <https://nodejs.org/pt-br/about/>. Acesso em: 26 out. 2022.

TYPESCRIPT. **TypeScript é JavaScript com sintaxe para tipos.** 2022. Disponível em: <https://www.typescriptlang.org/pt/>. Acesso em: 26 out. 2022.

EXPRESS JS. **Express Framework web rápido, flexível e minimalista para Node.js.** 2022. Disponível em: <https://expressjs.com/pt-br/>. Acesso em: 06 dez. 2022.

W3C. **Cross-Origin Resource Sharing.** 2022. Disponível em: <https://www.w3.org/TR/2020/SPSD-cors-20200602/ />. Acesso em: 06 dez. 2022.

SEQUELIZE. **Sequelize is a modern TypeScript and Node.js ORM for Oracle, Postgres, MySQL, MariaDB, SQLite and SQL Server, and more. Featuring solid transaction support, relations, eager and lazy loading, read replication and more.** 2022. Disponível em: <https://sequelize.org/>. Acesso em: 06 dez. 2022.

MONTANHEIRO, Lucas Souza; CARVALHO, Ana Maria Martins; RODRIGUES, Jackson Alves. Utilização de json web token na autenticação de usuários em apis rest. **XIII Encontro Anual de Computação. Anais do XIII Encontro Anual de Computação EnAComp**, p. 186-193, 2017.

JOI. **The most powerful schema description language and data validator for JavaScript.** 2022. Disponível em: <https://joi.dev/>. Acesso em: 06 dez. 2022.

RFC 4122. **A Universally Unique IDentifier (UUID) URN Namespace.** 2005. Disponível em: <https://www.rfc-editor.org/rfc/rfc4122/>. Acesso em: 06 dez. 2022.

PROVOS, N.; MAZIÈRES, D. **A future-adaptable password scheme. In: Proc. of the FREENIX track: 1999 USENIX annual technical conference**. Monterey, California, USA: USENIX, 1999.

Amazon Web Services. **Microsserviços.** 2022. Disponível em: <https://aws.amazon.com/pt/microservices/>. Acesso em: 8 nov. 2022.

REACT. **Uma biblioteca JavaScript para criar interfaces de usuário.** 2022. Disponível em: <https://pt-br.reactjs.org/>. Acesso em: 8 nov. 2022.

REACTDOM. **ReactDOM.** 2022. Disponível em: <https://pt-br.reactjs.org/docs/react-dom.html/>. Acesso em: 7 dez. 2022.

SOUSA, Flávio RC; MOREIRA, Leonardo O.; MACHADO, Javam C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. **II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)**, p. 150-175, 2009.

IBM. **Microsserviços.** 2021. Disponível em: <https://www.ibm.com/br-pt/cloud/learn/microservices>. Acesso em: 9 nov. 2022.

RED HAT. **O que é API REST?** 2020. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>. Acesso em: 9 nov. 2022.

AXIOS. **Cliente HTTP baseado em promessas para o navegador e Node.js.** 2022. Disponível em: <https://axios-http.com/ptbr/docs/intro>. Acesso em: 10 nov. 2022.

HELMET. **Express.js security with HTTP headers.** 2022. Disponível em: <https://helmetjs.github.io/>. Acesso em: 06 dez. 2022.

W3C. **A JSON-based Serialization for Linked Data.** 2020. Disponível em: <https://www.w3.org/TR/json-ld11/>. Acesso em: 06 dez. 2022.

NARDELLI, Cleber. Segurança da Informação e LGPD Aplicado no Desenvolvimento de Software. In: **Anais da V Escola Regional de Engenharia de Software**. SBC, 2021. p. 169-178.

GUEDES, Gilleanes TA. UML 2. **Uma Abordagem Prática", São Paulo, Novatec**, 2009.

REACT BOOTSTRAP. **React Bootstrap The most popular front-end framework.** 2022. Disponível em: <https://react-bootstrap.github.io//>. Acesso em: 7 dez. 2022.

VAROTO, Ane Cristina. **Visões em arquitetura de software**. 2002. Tese de Doutorado. Universidade de São Paulo.

IBM. **Diagramas de Caso de Uso.** 2021. Disponível em: <https://www.ibm.com/docs/pt-br/rsm/7.5.0?topic=diagrams-use-case>. Acesso em: 10 nov. 2022.

IBM. **Diagramas de Classes.** 2021. Disponível em: <https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=structure-class-diagrams>. Acesso em: 10 nov. 2022.

IBM. **Diagramas de Sequência.** 2021. Disponível em: <https://www.ibm.com/docs/pt-br/rsm/7.5.0?topic=uml-sequence-diagrams>. Acesso em: 10 nov. 2022.

RCF 7519. **JSON Web Token (JWT).** 2015. Disponível em: <https://www.rfc-editor.org/rfc/rfc7519>. Acesso em: 06 dez. 2022.

VILLAÇA, L. H.; PIMENTA JR, Antônio Francisco; AZEVEDO, Leonardo Guerreiro. Construindo aplicações distribuídas com microsserviços. **Tópicos em Sistemas de Informação: Minicursos XV Simpósio Brasileiro de Sistemas de Informação. SBC**, 2018.

FLANAGAN, David. **JavaScript: o guia definitivo**. Bookman Editora, 2004.

NEWMAN, Sam. **Migrando sistemas monolíticos para microsserviços: Padrões evolutivos para transformar seu sistema monolítico**. Novatec Editora, 2020.

MORAES, William Bruno. **Construindo aplicações com NodeJS-3a edição**. Novatec Editora, 2021.