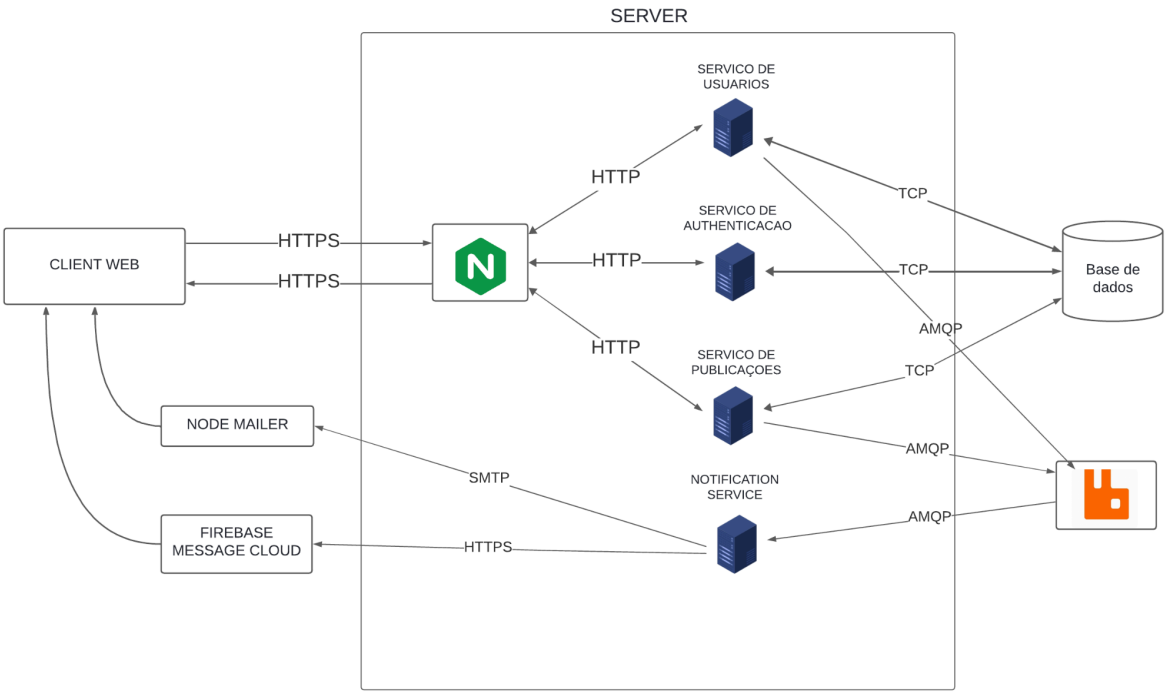
Descrição Arquitetural e de Interfaces do Sistema Distribuído CAROLINA News

Desenvolvedores:
Christofer Daniel Rodrigues Santos
Diogo Rodrigues dos Santos
Gustavo Zanzin Guerreiro Martins

1. Arquitetura do sistema



2. Principais componentes:

- **WEB Client**: Parte do sistema em que o usuário interage, é desenvolvido para ser acessado como um portal *web* usando a tecnologia ReactJS. O **WEB Client** interage com o sistema a partir de chamadas HTTPS para o serviço do NGINX.
- **NGINX**: Nesse contexto, o **NGINX** atua como um proxy reverso, lidando com as comunicações com o cliente via mensagens HTTPS, ele também está servindo como um balanceador de carga encaminhando as chamadas para o serviço apropriado, como o **SERVIÇO DE AUTENTICAÇÃO**, **SERVIÇO DE USUÁRIO**, e **SERVIÇO DE PUBLICAÇÃO**, auxiliando a não sobrecarregar os serviços. O **NGINX** também traz uma camada de segurança, isolando os principais serviços do servidor da comunicação com o cliente.
- **Base de Dados**: Sistema gerenciador de banco de dados utilizado para armazenar de forma persistente as informações sobre usuários e publicações. O banco de dados usado é o PostgreSQL.
- **Serviço de Autenticação**: Serviço responsável pela autenticação do usuário, implementado utilizando NestJS Web Framework. Ele recebe as requisições pelo

NGINX e se comunica com o **BASE DE DADOS** para checar as credenciais e outras informações do usuário. Caso o *login* seja realizado, é enfileirada uma tarefa para o **RabbitMQ**. Essa tarefa é consumida pelo **NOTIFICATION SERVICE** que irá enviar um e-mail para o usuário notificando sobre o novo *login*.

- **Serviço de Usuário**: Serviço responsável por gerenciar os dados dos usuários. Ele recebe as requisições do **NGINX** e faz a comunicação com o **BASE DE DADOS** tanto para persistir, quanto para obter os dados dos usuários.
- **Publication Service**: Serviço responsável pelo recebimento e gerenciamento das publicações feitas no sistema. Ele recebe as requisições do **NGINX** e se comunica com o **BASE DE DADOS** para persistir os dados referentes às publicações. Além disso, tem como papel enviar uma mensagem para o **RabbitMQ** usando o protocolo **AMQP** sinalizando sobre novas publicações.
- **Notification Service**: Serviço responsável por gerenciar as notificações do sistema para o usuário. Utiliza duas formas de notificação, sendo elas: **FIREBASE MESSAGE CLOUD** e **NODEMAILER**(Serviço de email).
- **Firebase Message Cloud**: Serviço que será usado para notificar os usuários sobre novas publicações feitas pelos usuários sinalizados como interesse.
- **RabbitMQ**: Ele enfileira tarefas do **SERVIÇO DE PUBLICAÇÃO** e do **SERVIÇO DE AUTENTICAÇÃO,** o **NOTIFICATION SERVICE** consome as tarefas dessa fila e ou envia uma *push notification* sobre uma nova publicação, ou envia um e-mail notificando sobre um novo *login*.

3. Principais fluxos do sistema:

- **Fluxo de Postagem de uma Publicação**: Iniciada no **Web Client** um usuário logado realiza uma publicação. A postagem é encaminhada para o servidor do sistema via uma mensagem HTTP, essa chega no **NGINX** que irá encaminhar a requisição para o serviço correto, no caso o **SERVIÇO DE PUBLICAÇÃO**. Nesse momento, esse serviço se comunica com o **BASE DE DADOS** para persistir as informações da publicação e também envia a notificação do evento para o **RabbitMQ**. Esse repassa as informações para o **NOTIFICATION SERVICE**, que aciona um evento no **FIREBASE MESSAGE CLOUD** que é usado para notificar usuários que tenham marcado como interessados nas postagens do usuário que fez a postagem.
- **Fluxo de Cadastro de um Usuário**: Iniciada no **WEB CLIENT**, um novo usuário do sistema faz seu cadastro colocando suas informações. Uma requisição é enviada para o **NGINX** que irá encaminhar a requisição para o **SERVIÇO DE USUÁRIO**, esse faz as verificações necessárias e persiste os dados do novo usuário no **BASE DE DADOS**.
- **Fluxo de Login**: Iniciada no **WEB CLIENT**, um usuário do sistema faz um *login* com seus dados informados previamente na etapa de cadastro. Uma requisição é enviada para o **NGINX** que irá encaminhar a requisição para o **SERVIÇO DE AUTENTICAÇÃO**, esse faz as verificações checando a existência e a veracidade dos dados por meio da obtenção desses dados no **BASE DE DADOS** e confirma ou rejeita o *login*. Esse também anexará na fila do **RabbitMQ** uma tarefa para notificar o usuário sobre um novo *login*. Essa é consumida pelo **NOTIFICATION SERVICE** que irá enviar um e-mail para o usuário sobre o novo *login*.
- **Fluxo de Listagem de Publicadores**: Iniciada no **WEB CLIENT**, um usuário do sistema requisita a listagem de publicadores. Uma requisição é enviada para o

NGINX que irá encaminhar a requisição para o **SERVIÇO DE USUÁRIO**, esse faz uma requisição para a **BASE DE DADOS** buscando os usuários cadastrados e devolve a listagem.

- **Fluxo de Listagem de Publicadores**: Iniciada no **WEB CLIENT**, um usuário do sistema requisita a listagem de publicações. Uma requisição é enviada para o **NGINX** que irá encaminhar a requisição para o **SERVIÇO DE PUBLICAÇÕES**, esse faz uma requisição para a **BASE DE DADOS** buscando as publicações e retorna a listagem delas com o título da publicação e o conteúdo.

4. Definição das Interfaces de Serviço

- **NGINX**
  - **Interface HTTPS**
    - **Parâmetros de entrada:**
      - `HTTP Request: requisição HTTPS.`
    - **Saída:** `HTTP Response`
    - **Mensagens de erro:**
      - Http Status, e mensagens customizadas por cada serviço.
- **Serviço de Usuário**
  - **Interface de cadastro de usuário** (`register`)
    - **Parâmetros de entrada:**
      - `name`: `string`. Nome do usuário;
      - `email`: `string`. Correio eletrônico do usuário;
      - `password`: `string`. Senha do usuário.
    - **Resposta:** `ResponseDTO`. Objeto que contém o código de *status* HTTP do resultado da operação (em caso de sucesso, `201`) e uma mensagem descritiva.
    - **Mensagem de erro:** `ResponseDTO`. Objeto que contém o código de *status* HTTP do resultado da operação, neste caso, correspondendo a um erro, como `500`. O objeto também contém uma mensagem descritiva do erro.
- **Serviço de Autenticação**
  - **Interface de *login*** (`authenticate`)
    - **Parâmetros de entrada:**
      - `email`: `string`. Correio eletrônico do usuário;
      - `password`: `string`. Senha do usuário.
    - **Resposta:** `ResponseDTO`. Objeto que contém o código de *status* HTTP do resultado da operação (em caso de sucesso, `200`), uma mensagem descritiva e o *token* de acesso.
    - **Mensagens de erro:** Objeto que contém o código de *status* HTTP do resultado da operação, neste caso, `500`, correspondendo a um erro. Além disso, contém uma mensagem descritiva do erro.
- **Serviço de Publicação**
  - **Interface de listagem de publicações** (list)
    - **Parâmetros de entrada:** nenhum parâmetro necessário

- **Resposta:** `ResponseDTO`. Objeto que contém o código de *status* HTTP do resultado da operação (em caso de sucesso, `200`), uma mensagem descritiva e um array de publicações.
- **Mensagens de erro:** `ResponseDTO`. Objeto que contém o código de *status* HTTP do resultado da operação, neste caso, `400`, correspondendo a um erro. Além disso, o objeto contém uma mensagem descritiva do erro.

➢ **Interface de criação de publicação** (`create`)

- **Parâmetros de entrada:**
  - `userID: int`. Identificador único do usuário que está criando a publicação;
  - `title: string`. Título da publicação que está sendo criada;
  - `content: string`. Conteúdo da publicação que está sendo criada.
- **Resposta:** `ResponseDTO`. Objeto que contém o código de *status* HTTP do resultado da operação (em caso de sucesso, `201`) e uma mensagem descritiva.
- **Mensagens de erro:** `ResponseDTO`. Objeto que contém o código de *status* HTTP do resultado da operação, neste caso, `500`, correspondendo a um erro. Além disso, o objeto contém uma mensagem descritiva do erro.

❖ **Notification Service**

➢ **Interface para notificar um novo *login***

- **Parâmetros de entrada:**
  - `name`: `string`. Nome do usuário;
  - `email`: `string`. Correio eletrônico do usuário;
  - `password`: `string`. Senha do usuário.
- **Resposta:** O método não produz saída para quem o invoca.
- **Mensagem de erro:** O método também não produz nenhuma possível mensagem de saída.