**Briefly – Encurtador de links**

Este é um servidor de encurtador de links desenvolvido em Kotlin utilizando o Ktor. O servidor permite criar URLs curtas que redirecionam para URLs longas, oferecendo funcionalidades adicionais como contagem de cliques únicos e gerenciamento de links.

Será um projeto aberto, destinado a ser incluído em um portfólio e oferecerá um nível gratuito onde todos os links gerados estarão disponíveis para acompanhamento.

## ***Informações Coletadas das Conexões***

* IP: O endereço IP do usuário será registrado para fins de análise e segurança.
* Geolocalização: Com base no endereço IP, a localização geográfica do usuário será determinada.
* Horário: O *timestamp* de cada conexão será registrado.
* Dispositivo: Informações sobre o dispositivo do usuário, incluindo tipo (desktop, mobile), sistema operacional (Windows, iOS, Android, etc.) e navegador (Chrome, Firefox, Safari, etc.).
* URL de origem: A URL de origem da qual o usuário foi redirecionado para o link encurtado.

## ***Informações dos Links***

* Cliques: Número total de cliques em cada link encurtado.
* Filtros de Análise:
  + Por Horário: Quantidade de cliques por hora, dia, semana e mês.
  + Por Região: Distribuição geográfica dos cliques.
  + Por Dispositivo: Análise dos cliques com base no tipo de dispositivo, sistema operacional e navegador.

## ***Funcionalidades dos Links***

* Possibilidade de criar links curtos.
* QR Code: Possibilidade de compartilhar um QR Code que contém o link encurtado.
* Proteção de links: Os links terão a possibilidade de serem protegidos por senha.
* Tempo de vida: Os links podem ser criados com um tempo de expiração.
* Um código personalizado para o link pode ser criado.

## ***Funcionalidades do Sistema***

* Possibilidade de encurtar links (Estando ou não logado).
* Possibilidade de acompanhar informações coletadas de clicks dos links.
* Notificações via E-mail aos usuários cadastrados sobre seus links.
* Gerenciar links como atualizar status ou removê-lo.
* Há uma verificação periódica de links expirados, ou que não recebem mais acessos para serem desativados.
* Funcionalidades para um moderador gerenciar o estado da aplicação.

***Requisitos do Projeto***

* Backend: **Ktor (Kotlin)**
* Geolocalização: API para coleta de informações de IP (GeoIp)
* Para distribuição será utilizado o Docker.
* Postman para os testes requisições e construção da documentação
* Banco de Dados:
  + **MongoDB** - Para armazenamento de dados sensíveis.
  + **Redis** - Para armazenamento de dados que precisam de acesso mais rápido.

***Para instalar o projeto***

Para a fácil distribuição da aplicação, será utilizado o **Docker**.

* Instale o **Mongodb** (Verifique na página oficial, ou monte uma imagem com o Docker).
* Instale o **Redis** na sua máquina (Verifique a página oficial, ou monte uma imagem com o Docker).
* Crie um arquivo **.env** na pasta raiz do projeto, ou adicione um arquivo **.env** manualmente na construção da imagem **Docker**.

#URL do banco de dados MONGODB  
MONGODB\_SERVER\_URL=mongodb://host.docker.internal:27017  
#Nome do banco de dados do MONGODB  
NAME\_DATABASE=Briefly  
#URL do banco de dados do REDIS  
REDIS\_SERVER\_URL=host.docker.internal  
#Porta do banco de dados do REDIS  
REDIS\_SERVER\_PORT=6379  
#URl no qual a aplicação será instalada / Domain  
SELF\_URL=http://localhost:9098  
  
#Audience do token, quem deve processar o token  
JWT\_AUDIENCE=  
#Dominio do token, quem foi o emissor  
JWT\_DOMAIN=  
#Secret Token do usuário  
USER\_SECRET=  
#Secret Token do moderador  
MODERATOR\_SECRET=  
  
#Configuração para envios de e-mail  
SMTP\_HOST=smtp.gmail.com  
SMTP\_STARTTLS\_ENABLE=true  
SMTP\_SSL\_PROTOCOLS=TLSv1.2  
SMTP\_SOCKETFACTORY\_PORT=465  
SMTP\_SOCKETFACTORY\_CLASS=javax.net.ssl.SSLSocketFactory  
SMTP\_AUTH=true  
SMTP\_PORT=465  
SMTP\_SSL\_TRUST=\*  
  
#Email que irá ser associado a aplicação para autenticação do serviço de e-mail  
EMAIL\_SYSTEM=  
#Senha do e-mail para autenticação do serviço de e-mail  
EMAIL\_PASSWORD=  
  
#Link dos termos de serviço  
TERMS\_OF\_SERVICE\_LINK=

**MONGODB\_SERVER\_URL=mongodb://host.docker.internal:27017** - Esta configuração serve para que a aplicação se conecte ao Mongodb localizado no **localhost** da máquina.

**REDIS\_SERVER\_URL=host.docker.internal** - Esta configuração serve para que a aplicação se conecte ao Redis localizado no localhost da máquina.

*Nas configurações de Email, foi adicionado uma pré-configuração para utilizar os serviços do Gmail. Para a Senha do Email, é necessário gerar uma senha para aplicação em "Apps menos seguros" na sua conta do Gmail.*

**Coleta de dados**

Para coleta de dados como região dos IPs, estamos utilizando a API GeoIp <https://www.maxmind.com/en/home>. É necessário realizar o download de 2 bancos de dados disponíveis no site, que se chama **"GeoLite2 ASN"** e **"GeoLite2 City"**, copiar os arquivos **"GeoLite2-ASN.mmdb"** e **"GeoLite2-City.mmdb"** para a pasta **resources/geo/**.

**Página html**

Caso seja necessário que o servidor responda um arquivo **html** na rota principal, crie uma pasta chamada files em resources e insira o seu arquivo com o nome index.html para o servidor mapear e entregar a página.

#### Execute o comando a seguir para construir a imagem Docker:

**docker build -t briefly:latest .**

#### Execute o comando a seguir para executar a imagem criada com o Docker:

**docker run --name briefly -p 9098:9098 briefly:latest**

***Estrutura do Projeto***

***Estrutura do fluxo***

**Rotas e Serviços:** O fluxo do projeto começa com as rotas, que mapeiam as requisições para o serviço correspondente. O serviço, por sua vez, executa os procedimentos necessários e geralmente chama os repositórios para armazenar, consultar ou modificar dados.

**Classes de Serviço e Injeção de Dependência:** As classes de serviço utilizam injeção de dependência para injetar os repositórios, facilitando os testes em diferentes ambientes e com diferentes implementações de repositório. Isso torna o código mais modular e flexível.

**Repositórios e Utilitários:** Os repositórios, por sua vez, utilizam classes e funções contidas na pasta utils. Esses utilitários podem converter, analisar, e gerar dados, entre outras funções. Além disso, os repositórios fazem uso de informações do GlobalContext, como o número máximo de caracteres permitidos, caminhos de pastas, tempo de duração de tokens, etc.

**Conexões com Bancos de Dados e Serviços Externos:** Os repositórios podem também se conectar a bancos de dados ou serviços externos. Por exemplo, o ***DefaultDatabaseRepository*** utiliza o **MongoDB** como seu banco de dados principal para armazenamento e o Redis para dados que exigem acesso rápido.

**Interfaces e Flexibilidade:** Para facilitar a injeção de dependência e permitir mudanças rápidas nas implementações, os repositórios implementam interfaces. Isso garante que a lógica do serviço permaneça desacoplada da implementação específica do repositório.

***Estrutura de pastas***

Para este projeto, na estrutura de pastas temos: **api, config, data, database, errors, plugins, routing, security, services e utils.**

1. Na pasta **api**: Contém duas pastas chamadas **geo** e **ipinfo,** ambas pastas contém objetos que permitem o acesso a dados importantes para o funcionamento do sistema.
   1. Na pasta **geo**, há o objeto ***GeoIpService***, que contém a funcionalidade de pesquisar em um banco de dados fornecido pela **GeoIp** ([*https://dev.maxmind.com/geoip*](https://dev.maxmind.com/geoip)).
   2. Na pasta ipInfo, há o objeto ***IpInfoApi***, e a interface ***IpInfoApiService,*** no momento ambos não estão sendo utilizados, pois há limites de uso.
2. Na pasta **config**: Contém alguns objetos que são utilizados no projeto inteiro, para configurações, valores padrões, e etc.
   1. O objeto **Config**, contém as funcionalidades de inicialização do servidor, inicializando a conexão com o banco de dados, e as rotinas como atualizar os registros de links expirados e a remoção de links inativos.
   2. A classe ***DefaultServerContainer*** que implementa a classe ***ServerContainer***, tem a função de centralizar os 4 repositórios do sistema (Repositórios de banco de dados, sessões, Ip’s e E-mails)
   3. O objeto ***GlobalContext***, tem como função, centralizar todos os dados do servidor, como rotas, versão, os dados do ***DefaultServerContainer***, limites, formatos de dados e etc.
   4. O objeto ***MailContext***, tem como função, centralizar os E-mails que são construídos com HTML DSL, formatando para ***String***, para ser enviado como argumento na funcionalidade que envia e-mails.
   5. O objeto ***WebSockerManager***, tem como função gerenciar as conexões do Socket, e enviar informações às conexões pertinentes
3. Na pasta **data**, temos todas estruturas dos dados do projeto. Temos duas pastas **models** e **repository**
   1. A pasta models, contém todas as estruturas dos dados, como **dataclass**, **enums** e **interfaces**
   2. Na pasta **repository**, contém as classes que implementam interfaces de repositório, contém toda a estrutura de armazenamento de dados, retorno de dados, geralmente o repositório utiliza as informações expostas das classes e objetos da pasta config.
4. Na pasta **database**, encontram-se as classes e objetos necessários para realizar a conexão com os bancos de dados utilizados no projeto, como **MongoDB** e **Redis**. O MongoDB é utilizado para o armazenamento de dados permanentes, como informações de usuários, links, entre outros. Já o Redis é utilizado para o armazenamento de dados que exigem acesso rápido, como informações de sessão, tokens e contagem de cliques do sistema.
5. Na pasta **errors**, contém apenas uma função chamada ***errorResponse***, que centraliza a invocação de todos os erros, e cria uma resposta padrão para cada erro emitido durante o processamento de algum evento no sistema, quando um erro é lançado, é lançado uma **String** do ***Enum Errors***, a função tem como funcionalidade identificar qual erro foi emitido, e gerar uma resposta.
6. Na pasta **plugins**, contém todos os pluguins necessários para o funcionamento do projeto.
   1. ***AccesGuardian***, é um pluguin de segurança, que verifica se o IP requisitante, está bloqueado.
   2. ***ClickerCount***, tem como fucnionalidade armazenar a quantidade de clicks que o sistema recebeu.
   3. ***HTTP***, tem como funcionalidade habilitar o CORS.
   4. O ***RateLimit***, tem como funcionalidade limitar o acesso arbitrário inúmeras vezes a uma rota.
   5. ***Routing***, tem como funcionalidade centralizar e habilitar todas as rotas do projeto.
   6. ***Security***, tem a funcionalidade de verificar o token gerado e repassado via requisições que necessitam do **Bearer Token**.
   7. ***Serialization***, tem a função de transformar objetos json, em objetos que o kotlin consegue construir.
   8. ***Sockets***, habilita e escuta a conexões do Socket.
   9. ***StatusPage***, centraliza e envia os erros gerados da aplicação, ou capturar um erro não tratado.
7. Na pasta ***routing***, contém os códigos responsáveis pelas rotas, informações da rota, como o caminho, e cada caminho com os seus métodos de requisição (POST, GET, DELETE, PUT e etc).
8. Na pasta ***security***, contém o objeto ***AccessGuardianService***, responsável por marcar um IP como bloqueado, e pesquisar se um IP estar bloqueado, geralmente chamado pelo pluguin ***AccesGuardian.***
9. Na pasta ***services***, contém todos os serviços, como os serviços para envio de e-mail, criação de links encurtados, armazenamento de mídia, gerenciamento da conta de moderador e usuário, e ações do servidor. Os serviços utilizam os repositórios através da injeção de dependência para tomar ações.
10. Na pasta utils, contém classes e funções que são utilizados no projeto inteiro, como conversor de data, analisador de informações do .env, geradores de caracteres e etc. Estão divididos em ***analyzers***, ***converters***, ***generators***, ***pagination***, ***schedule***
11. Na pasta resources, contém todos os recursos do projeto.

Na pasta ***files***, contém os arquivos estáticos para a disponibilização da página html. Na pasta ***fonts***, contém as fonts do projeto. Na pasta geo, contém os bancos de dados baixados do GeoIp. Na pasta ***lang***, contém as ***Strings*** que serão utilizados nas notificações.

Temos em **src/test**, todos os códigos de teste da aplicação, para executar os testes.

***Endpoints disponíveis***

Para o conhecimento dos endpoints, foi documentado com o Postman, abaixo segue o link para visualização:

<https://documenter.getpostman.com/view/29175154/2sA3e1BqN4>

***Para Deploy***

Sugere-se para a instalação do projeto em modo de produção instalar e rodar o mongodb e a aplicação dentro de uma rede Docker.

Primeiro crie uma rede com o seguinte comando:  
**docker network create my\_network**

Após, execute o container do mongodb e mapeie-o para a rede criada anteriormente.

**docker run --name mongodb --network my\_network -d mongo**

Após, transfira um arquivo index.html para ***src/main/resources/files/***, para que uma página HTML seja entregue ao requisitar a rota principal.

Construa a imagem do briefly:

**docker build -t briefly:latest .**

E execute a imagem mapeando-a também para a rede.

**docker run --name briefly -p 9098:9098 --network my\_network briefly:latest**