## Primeiros passos

Inicialmente, criei um projeto VueJS, usando o Nuxt. Nele, eu configurei o Jest como framework de testes, além do Axios para o uso de APIs.

Então, dentro da pasta do projeto de Vue, iniciei um projeto de NodeJS, usando express para a criação agilizada do servidor.

Em seguida, criei o banco de dados “encurtador_links”, em mariaDB. Criando duas tabelas: tb_users, que contêm as colunas: id (Int, Chave primária, auto-incremento, not null), e-mail (Varchar 255, not null ) e senha (Varchar 255, not null).

E a tabela: tb_links, que contêm os campos:

- id (Int, Chave primária, auto-incremento, not null);

- título (Varchar 255, not null);

- link_original (Varchar 255, not null);

- link_encurtado (Varchar 255, not null);

- id_usuario (Int, chave estrangeira);

- views (Int, not null).

A relação entre as tabelas seria: Usuário encurta link, eu poderia saber quem é o usuário pelo seu id, passado na chave estrangeira: “id_usuario”; no entanto, fiz questão dela poder ser nula, permitindo que o usuário possa ser anônimo.

## Problemas com a API

No Vue, comecei a enfrentar um problema com a API, pelo fato de eu rodar ambos no mesmo servidor (localhost), meu navegador começou a bloquear a requisição, apontando o erro: “Cross-Origin Request Blocked”, já tinha enfrentado esse erro antes, com minha API em PHP, lá bastou eu definir um Header, para permitir a requisição e no Node, as coisas não foram diferentes, embora tenham sido um pouco mais complicadas, possivelmente pela minha inexperiência com ele. Mas pesquisa vai, pesquisa vem, encontrei os headers que precisava utilizar; assim, finalmente, minha API tinha sido integrada com o meu front.

Com a API configurada, comecei a pensar o design do serviço. Pretendo focar em algo mais simples. Inicialmente, preciso definir as fontes e as cores.

## Definição da identidade visual

As fontes escolhidas foram:

- Open Sans, como fonte padrão, usada em parágrafos, listas, links etc.

- Bebas Neue, como fonte para destaques, usada em título.

Já para as cores, eu utilizei o site: https://coolors.co, que utiliza de uma inteligência artificial para sugerir novas cores, que combinem entre si, agilizando muito o processo de montagem da paleta de cores.

Ao fim da escolha, as cores foram:

- Dark Slate Blue: #4F4789

- Orange Yellow: #EEBA0B

- Eton Blue: #7FC29B

- Cerulean Crayola: #3AAED8

- Sinopia: #CC3F0C

Tendo toda a base inicial definida, comecei o projeto propriamente dito. Criei um componente Header, contendo a logo do serviço, junto de um lema e links para navegação em outras páginas.

## Iniciando nos testes

Antes de prosseguir para a criação do componente Main, decidi me aventurar pelo mundo dos testes. Tentei copiar o teste padrão, modificando algumas coisas com a ajuda da documentação para verificar se o texto do H1, no header era: “Encurtador de URLs”, o título que usei.

Interessantemente, a criação do teste foi mais fácil do que eu esperava, ele é extremamente semântico, então fica fácil de entender o que está acontecendo; além de que, com os testes rodando no terminal, eu não preciso verificar o navegador, mantendo o meu foco no editor de código.

Alguns problemas surgiram, como testar o valor de diversos itens de uma lista. Inicialmente, ele apontava só para o primeiro item, mas com um pouco de pesquisa e tentativa e erro, consegui descobrir um método findAll, que criava um conjunto das ocorrências de um determinado elemento (no caso o <li>). Tendo isso, eu pude testar cada elemento da lista individualmente (eram, dois nesse ponto, dois links); assim, consegui usar a função .at, para selecionar o elemento particular e testá-lo.

## Main

Então passei para criação do componente main, dentro dele criei mais três componentes: CriarLink; LinkGerado e CardErro.

O Componente CriarLink, possuí um formulário com um componente Input personalizado, já estilizado . Ao clicar no botão “Encurtar!”, um método chamado: “gerarLink” é acionado, ele recolhe o título e o link inseridos nos campos de texto e faz algumas verificações de segurança, testando se nenhum deles está vazio e se o link inicia com http(s).

Caso as validações sejam concluídas com sucesso, um *State* chamado de “link_valido”, terá o seu valor mudado para verdadeiro e, então, uma constante: “link_novo” é criada usando de um método próprio que cria um ID único pseudo aleatório, com um número predefinido de caracteres; inicialmente, defini três caracteres, mas dentro desse método há a chamada de outro método: “idExists”, que faz uma busca no banco de dados para verificar se o id gerado já existe; caso já exista, o método generateUNID chama-se a si mesmo novamente, mas com o tamanho do ID incrementado em um, ou seja, agora ele tentará criar um ID único de quatro caracteres; caso não exista, ele irá inserir esse link no banco de dados, já com seu ID único de encurtamento.

Ao final do processo, os *States* título e link_original serão atualizados pelos passados no formulário e o componente LinkGerado será mostrado, com um fundo verde, alertando que o link foi gerado com sucesso e disponibilizando o título e o link encurtado.

Ao clicar no link encurtado, que usa a base da api + id único (P.ex: http://localhost:5000/Xfv), o NodeJS faz uma requisição ao banco de dados para verificar se o link encurtado (Xfv), existe e coletar também o link original; caso sim, ele simplesmente redireciona o usuário para o link original e incrementa a visualização do link; caso não, ele exibe uma mensagem: “Não existe”.

Caso a criação do link encurtado falhe por algum motivo, o componente CardErro será exibido, alertando o usuário que houve um erro ao gerar seu link e pede para que ele verifique os dados que inseriu no formulário.

A página top-cem exibe um componente TopMain, que tem em si um componente TabelaResponsiva, que eu criei para conservar a funcionalidade do serviço mesmo nos dispositivos móveis, dentro da TabelaResponsiva, é feita uma requisição à API, no caminho “get-urls”, que fará uma requisição ao banco de dados, selecionando os 100 links mais visitados.

## Dia 27

A primeira coisa que tentei fazer foi configurar os testes, infelizmente acabei falhando graças a alguns problemas que apareceram, depois de algum tempo mexendo neles, eu decidi ir focar em implementar os outros requisitos, para que depois eu possa focar inteiramente em entendê-los e configurá-los. Sinto que isso será o que mais me dará trabalho, justamente pela falta de experiência com eles.

Assim, criei a página “minha-conta” com os componentes: Header e MainMinhaConta, dentro desse existem os componentes: FazerLogin e TabelaResponsiva.

O componente FazerLogin, apresenta um formulário com componentes de Input personalizados, seus métodos mais importantes são:

**Login** → Coleta o e-mail e senha do formulário, verifica se os campos não são vazios, criptografa a senha com SHA256, usando a biblioteca crypto-js e faz uma requisição via POST para a API, enviando o e-mail e a senha criptografada para o diretório login, da API, esperando sua resposta, um JSON com um booleano de nome logado. Após a resposta da API, o sistema verifica se logado é verdadeiro, caso sim, ele faz um commit para a mutation: “logar” que muda o state: “logged” para true; caso não, ele faz o mesmo processo, mas o state é definido para false.

**Criar** → Similarmente ao método login, o método criar, primeiro recupera o e-mail, senha e nome de usuário do formulário, verifica se existem campos vazios, alertando o usuário caso existam e, então, criptografa a senha; por fim, ele envia tudo isso para o backend, por meio do diretório “cadastro” da API e recupera sua resposta, caso seja true, o usuário recebe um alerta, avisando que sua conta foi criada com sucesso; caso seja false, o usuário também recebe um alerta, mas dessa vez avisando que ocorreu um erro na criação da conta, provavelmente, seu nome de usuário ou e-mail já estão cadastrados no banco de dados.

No backend é criado uma sessão global, com o atributo usuário, que inicia-se como false gerenciador, esse atributo gerencia se o usuário está logado ou não; e o atributo unid, que inicia-se como -1, indicando que não há nenhum usuário logado.

Assim que o usuário loga-se no sistema, esses valores são atualizados para seus devidos dados, dentro da tabela no banco de dados.

Por fim, o componente TabelaResponsiva, verifica se o usuário está logado, fazendo uma requisição para a API, que, por sua vez, verifica se o atributo unid, da sessão, é diferente de 1, em caso afirmativo, o usuário pode recuperar suas próprias URLs e deletá-las. O tipo de tabela a ser exibida, se é o top 100 ou somente as URLs do usuário, é controlado pela prop: “somente_os_meus”, que por padrão é falsa e só se altera nessa página.

Finalizando isso, o sistema estava pronto, então antes de ir para os testes, foquei-me em escrever a documentação do README, explicando como rodar o projeto em outros computadores.

**Dia 28**

## Testes

Penúltimo dia para a entrega do serviço e apenas três coisas me restam: a criação dos testes, a criação de um vídeo de demonstração do sistema e a explicação de partes do código que poderiam ser melhoradas se eu tivesse mais tempo e porque.

Certamente, essas partes estão principalmente nos testes, os quais eu ainda sou muito inexperiente em fazer.

Levei horas de pesquisa para conseguir resolver o problema que tive ontem, mas consegui e como sempre a solução era mais simples do que parecia. O problema consistia em acessar o $store, com os states da aplicação pelo Jest, eu tentei muitas coisas, pesquisei muito e, no fim, descobri que bastava a criação de um objeto constânte (ou variável), com um objeto dos getters dentro. A dor de aprender foi grande, mas a felicidade de conseguir ter feito foi maior. Essa é uma das grandes máximas que aprendi: a aprender dói e você deve gostar disso. Aristóteles disse: “A educação tem raízes amargas, mas seus frutos são doces” e é verdade. Todo o processo para a criação dos testes se baseou nisso, em muita dor para entender o que devia fazer, o terceiro dia todo se consistiu nisso, das 8 da manhã às 19:00 da noite, mas ao final, lá estava, os testes estavam funcionando, talvez não da melhor for possível; na verdade, provavelmente de uma forma horrenda, mas estavam funcionando.

Feito os testes, decidi ir gravar o vídeo e finalizar os relatórios, coisa que estou fazendo nesse exato momento. Após isso, escreverei um pouco sobre partes do código que poderia ser melhoradas e enviarei um e-mail com o acesso ao repositório do GitHub.

Como eu imaginava, fazer os testes foi a coisa mais díficil, mas aprendi muito graças a eles, a partir de hoje vou tentar integrar mais testes em meus projetos para melhorar. No mais é isso, agradeço pela leitura, caso tenha lido. Um abraço, fique com Deus!