1. Requisitos do Sistema

1.1 Objetivos do Sistema

O LinkBox é um sistema de gerenciamento de links que tem como objetivo ajudar seus usuários a organizar seus links, apresentando uma gama de funcionalidades úteis de gerenciamento.

1.2 Requisitos Funcionais

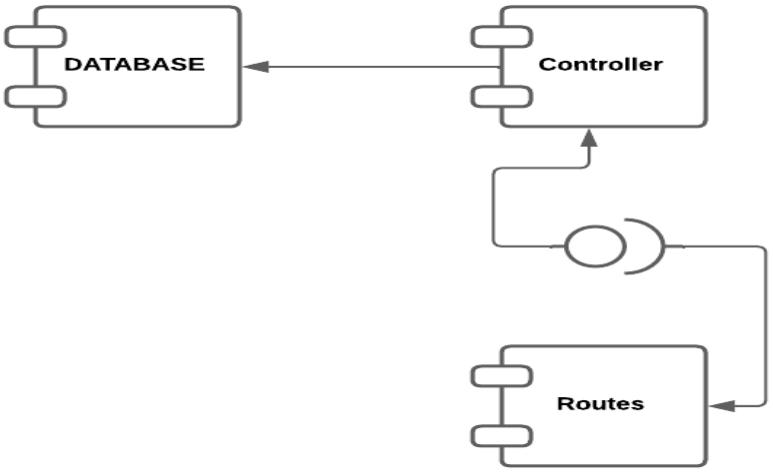
- Salvar links;
- Criar Pastas;
- Copiar e colar links e pastas;
- Cadastrar usuário;
- Pesquisar links ou pastas;

1.3 Requisitos Não Funcionais

- Dados criptografados e autenticação de usuários;
- 99% do tempo 24/7;
- Resposta em até 1,5 segundos;
- O sistema deve ser intuitivo e amigável para o usuário, com uma curva de aprendizado mínima;
- Deve haver medidas para proteger o sistema contra ataques de segurança, como injeção de SQL, cross-site scripting (XSS) e outros;
- O sistema deve funcionar de maneira consistente em diferentes navegadores, como Chrome e Firefox;

2. Arquitetura de Alto Nível

2.1 Componentes Principais



Na parte do backend:
Os principais componentes do sistemas são: Controllers, Routes, Database

- Controllers: são responsáveis por se comunicar com o Database e conter as regras de negócio, foi usado Mongoose para fazer essa comunicação com o Database;
- Routes: são responsáveis por se comunicar com os Controllers e receber as requisições, nesse componente foi usado Express;
- Database: responsável por armazenar os dados foi usado o banco não relacional MongoDB;
- Model: é a camada onde contém os Schemas do banco de dados, por mais que Mongo não possua schemas usamos isso para manter a coesão dos dados;

2.2 Fluxo de Dados

Como mostrado no diagrama anterior:
As Routes recebe a requisição e envia para o Controller o Controller por sua vez aplica às regras de negócio e salva os dados no Banco de Dados e retorna os dados para as Routes.

3. Design Detalhado

3.1 Routes

- Usamos as tecnologias: Express;
- Usamos a estrutura padrão na documentação do Express;
- Tem como responsabilidade receber as requisições e enviar os dados para o Controller que depois de fazer suas operações retorna os dados em formato json para as Routes;

3.2 Controller

- Usamos as tecnologias: O Mongoose tecnologia recomendada pelo próprio MongoDB;
- Usamos a estrutura para criar um classe que contém as funções a serem usadas para cada tipo de requisição;
- Tem como responsabilidade receber os dados das Routes, aplicar as regras de negócio, armazenar os dados no Database e retornar os dados para as Routes;

3.3 Database

- Usamos as tecnologias: MongoDB;
- Usamos a estrutura padrão de um documento onde cada documento de usuário contém subdocumentos com suas dashboards, pastas e links;
- Tem como responsabilidade receber os dados do Controller e armazenar os dados;

4. Escalabilidade

4.1 Escalabilidade Horizontal

O sistema pode escalar horizontalmente usando a funcionalidade de “Sharding” do MongoDB, que permite distribuir os dados de uma coleção em várias máquinas.

4.2 Escalabilidade Vertical

O sistema pode escalar verticalmente com um plano mais elevado do MongoDB Atlas, que fará uso de recursos computacionais melhores para lidar com o banco de dados.

5. Segurança

5.1 Autenticação e Autorização

A autenticação é feita usando JWT (JSON Web Tokens). O servidor gera um token contendo alguns dados do usuário autenticado e envia para o cliente numa requisição HTTP. O cliente armazena o token em um cookie, com o mesmo tempo de expiração do token. Quando o token ou o cookie expira, o usuário tem seu acesso removido e ele precisa entrar novamente no sistema.

O sistema realiza validação tanto no lado cliente como também no servidor. A validação no lado cliente ocorre em tempo real, em resposta a cada entrada inserida pelo usuário, essa validação faz verificações básicas, que não precisam de acesso ao banco de dados, como se o formato do email está correto ou se a senha a ser cadastrada contém a quantidade mínima de caracteres necessários. No lado do servidor, a validação é mais rigorosa, ela faz todas as verificações que o lado cliente faz, entretanto também possui validações específicas, por exemplo, se o email já está cadastrado no sistema.

5.2 Proteção de Dados

Usamos variáveis de ambiente para esconder dados secretos que precisam ser usados no código. No ambiente de desenvolvimento, essas variáveis são declaradas em um arquivo ignorado pelo VCS e o programador pode definir valores para seu uso local. No ambiente de produção, as variáveis são definidas na própria interface do serviço de hospedagem.

Usamos encriptação baseada no algoritmo bcrypt para a senha do usuário. A senha é encriptada antes de ser salva no banco de dados.

6. Tolerância a Falhas

6.1 Estratégias de Backup e Recuperação

O sistema pode, por meio do MongoDB Atlas, gerar snapshots automatizadas para criar backups do banco de dados periodicamente.

6.2 Planos de Failover

O sistema pode usar a funcionalidade “Replica Set Clusters” do MongoDB Atlas para ter várias instâncias com os mesmos dados, provendo redundância e alta disponibilidade. O Atlas gerencia automaticamente o processo de failover, transferindo as operações para outra instância em caso de falha.

7. Considerações de Desempenho

7.1 Otimização

Os formulários fazem uso de validação no lado cliente para evitar requisições desnecessárias para o servidor quando algum erro pode ser detectado sem acesso ao banco de dados. É importante ressaltar que essa validação no lado cliente é replicada no lado servidor, para evitar falhas de segurança.

A API está estruturada de forma que cada endpoint retorna todos os dados que podem ser úteis para o cliente naquele contexto.

Para lidar com erros de cadastro, por exemplo, a API retorna todos os erros ocorridos (formato inválido de email, senha curta demais, etc) de uma vez, evitando que seja necessário várias requisições sucessivas, isso é tanto uma melhoria de performance, como também de experiência do usuário, pois ele não terá que tentar enviar o formulário várias vezes para saber todos os erros ocorridos.

Outro caso a ser observado é o de pastas e links, pois nas operações de consulta desses itens, a API retorna dados adicionais, como o diretório e a pasta mãe. Isso também previne que o lado cliente tenha que fazer sucessivas requisições.

7.2 Monitoramento

O desempenho do sistema poderá ser monitorado usando ferramentas de testes automatizados de desempenho, esses testes rodarão a cada “push” feito, e impedirá um “merge” caso os testes falhem.

8. Considerações Finais

8.1 Trade-offs e Decisões de Design

Fizemos uso de uma biblioteca de componentes para tornar possível entregar funcionalidades com maior velocidade. Em troca disso, perdemos parte da personalização da interface.

8.2 Melhorias Futuras

- Fazer uso da abordagem de UI otimista. Esse conceito consiste em atualizar a UI imediatamente, assumindo um retorno positivo da requisição feita. Caso a requisição falhe, a UI retorna ao estado anterior. Essa abordagem seria bastante útil para o LinkBox, pois ao interagir com o sistema, o usuário realiza várias operações esperando feedback rápido, como abrir uma pasta, editar uma URL, etc. A desvantagem dessa abordagem é que nos casos de falha ocorre um comportamento contra intuitivo, em que o estado ao que era antes, porém, esses casos seriam raros e não seriam de grande impacto, pois o LinkBox não é um sistema crítico (como um site de compras, por exemplo);
- Múltiplas dashboards. Uma dashboard é uma hierarquia de pastas e links, atualmente o LinkBox só permite uma dashboard. Cada dashboard teria um nome único entre as dashboards do usuário. A vantagem desse conceito em comparação ao de pasta é que ele combina mais para separações de alto nível, então um usuário poderia, por exemplo, ter uma dashboard para cada dispositivo em que ele usa o sistema;
- Importar/exportar favoritos. Quando o usuário for importar favoritos, por padrão, eles serão importados para uma nova dashboard;
- Extensão de navegadores, cujas funcionalidades incluirão, mas não estarão limitadas a:
    - Adicionar página atual no LinkBox;
    - Abrir/editar/remover alguma pasta ou link salvo no LinkBox sem ter que abrir o site;
    - Abrir o site do sistema;
    - Sincronizar favoritos;
- Usar WebSockets para garantir sincronia quando um mesmo usuário usar o sistema a partir de várias fontes ao mesmo tempo;
- Aplicativo nativo para celulares. O sistema atualmente usa a tecnologia PWA, que permite o usuário “instalar o site”, fazendo-o parecer com um aplicativo nativo, porém isso não é intuitivo para a maioria dos usuários, que esperam encontrar o sistema na loja de aplicativos;