**Documento de Requisitos do Projeto: Aplicativo de Lista de Compras com Atualização em Tempo Real**

**1. Visão Geral**

**1.1 Descrição do Projeto**

O **Aplicativo de Lista de Compras** é uma solução colaborativa que permite que grupos de usuários (familiares, amigos) gerenciem listas de compras de forma compartilhada e atualizem os itens em tempo real. O aplicativo oferece funcionalidade de login, permitindo que cada grupo crie e edite listas de compras, marque itens como comprados e veja atualizações de outros usuários em tempo real. A interface é amigável e o sistema é acessível via web.

**1.2 Objetivo**

Criar um sistema de lista de compras que permita interação em tempo real entre múltiplos usuários conectados, com uma arquitetura moderna e escalável, garantindo a segurança dos dados e a usabilidade.

**2. Requisitos Funcionais**

**2.1 Autenticação de Usuários**

* **RF001:** O sistema deve permitir o cadastro de usuários.
* **RF002:** O sistema deve permitir que os usuários façam login utilizando e-mail e senha.
* **RF003:** O sistema deve implementar autenticação JWT para proteger as rotas após o login.
* **RF004:** O sistema deve permitir que os usuários recuperem senhas através de e-mail.

**2.2 Gerenciamento de Listas de Compras**

* **RF005:** O sistema deve permitir que os usuários criem, editem e excluam listas de compras.
* **RF006:** O sistema deve permitir que os usuários adicionem e removam itens dentro de uma lista de compras.
* **RF007:** O sistema deve permitir que os usuários marquem itens como “comprados” ou “não comprados”.
* **RF008:** O sistema deve permitir que os usuários visualizem múltiplas listas de compras.
* **RF009:** O sistema deve permitir que os usuários compartilhem listas com outros usuários.

**2.3 Atualização em Tempo Real**

* **RF010:** O sistema deve permitir que as listas sejam atualizadas em tempo real usando WebSockets.
* **RF011:** As atualizações (adição, remoção, edição de itens) devem ser refletidas imediatamente para todos os usuários conectados à mesma lista.

**2.4 Grupos e Compartilhamento**

* **RF012:** O sistema deve permitir que os usuários criem grupos para compartilhar listas de compras.
* **RF013:** O sistema deve garantir que apenas usuários autorizados dentro do grupo possam editar ou visualizar as listas.

**2.5 Notificações**

* **RF014:** O sistema deve notificar visualmente os usuários sobre as alterações em uma lista enquanto eles estiverem conectados.
* **RF015:** O sistema deve permitir que os usuários configurem alertas (e-mail ou push notification) para quando itens forem marcados como comprados.

**2.6 Busca e Filtros**

* **RF016:** O sistema deve permitir a busca por itens em uma lista.
* **RF017:** O sistema deve permitir filtrar itens por status (comprado/não comprado).

**3. Requisitos Não Funcionais**

**3.1 Performance**

* **RNF001:** O sistema deve ter tempo de resposta de no máximo 2 segundos para operações de CRUD de listas e itens.
* **RNF002:** O sistema deve suportar pelo menos 1000 usuários simultâneos conectados via WebSocket.
* **RNF003:** O sistema deve usar Redis para caching de dados e melhorar a performance das consultas ao banco.

**3.2 Escalabilidade**

* **RNF004:** O sistema deve ser escalável horizontalmente, permitindo adicionar mais servidores para gerenciar cargas maiores.
* **RNF005:** O sistema deve suportar múltiplos grupos e listas sem degradação de performance.

**3.3 Segurança**

* **RNF006:** O sistema deve garantir que apenas usuários autenticados possam acessar suas listas.
* **RNF007:** O sistema deve armazenar senhas usando bcrypt para garantir que estejam protegidas.
* **RNF008:** O sistema deve ser protegido contra ataques comuns como **SQL Injection**, **XSS** e **CSRF**.

**3.4 Usabilidade**

* **RNF009:** A interface do usuário deve ser responsiva e acessível em dispositivos móveis e desktops.
* **RNF010:** A interface deve ser intuitiva e simples de usar, com feedback visual para operações como adicionar/remover itens ou salvar mudanças.

**3.5 Confiabilidade**

* **RNF011:** O sistema deve ter uma taxa de disponibilidade de pelo menos 99,5%.
* **RNF012:** O sistema deve garantir que as listas e itens sejam salvos com integridade em caso de falha de conexão.

**3.6 Manutenção**

* **RNF013:** O código do sistema deve ser modular e seguir padrões de boas práticas de desenvolvimento (SOLID, DRY).
* **RNF014:** O sistema deve ser fácil de atualizar e corrigir, usando uma pipeline de CI/CD para automação de testes e deploy.

**4. Tecnologias**

**4.1 Backend**

* **NestJS**: Framework principal para o backend.
* **NestJS trocado para golang**
* **Prisma**: ORM para interagir com o banco de dados MySQL.
* **JWT**: Para autenticação e segurança.
* **Redis**: Para caching e gerenciamento de sessões.

**4.2 Frontend**

* **React**: Framework principal para a criação da interface.
* **WebSocket**: Para comunicação em tempo real com o backend.
* **Axios**: Para chamadas HTTP RESTful.

**4.3 Infraestrutura**

* **MySQL**: Banco de dados relacional.
* **Docker**: Para facilitar o deploy e a escalabilidade da aplicação.
* **Kubernetes**: Para orquestração e escalabilidade.
* **GitHub Actions**: Para automação de testes e deploy (CI/CD).

**5. Fluxo de Uso do Sistema**

1. **Cadastro/Login de Usuários**
   * O usuário acessa a tela de login e se autentica.
   * O sistema gera um JWT para o usuário autenticado e redireciona para o dashboard.
2. **Criação de Listas de Compras**
   * O usuário cria uma nova lista de compras, atribui um nome e a compartilha com um grupo de usuários.
   * Os membros do grupo podem adicionar e remover itens.
3. **Edição de Itens na Lista**
   * Qualquer usuário autorizado no grupo pode adicionar/remover/editar itens na lista em tempo real.
   * As mudanças são refletidas instantaneamente para todos os membros conectados via WebSocket.
4. **Marcar Itens como Comprados**
   * Os itens podem ser marcados como comprados ou não comprados, com visualização clara de seu status.
   * Notificações podem ser configuradas para alertar sobre compras.
5. **Logout**
   * O usuário pode fazer logout e o token JWT será invalidado no cliente.

**6. Riscos e Desafios**

* **Desempenho de WebSocket**: Garantir que o WebSocket funcione corretamente sob alta carga e múltiplos usuários simultâneos.
* **Segurança**: Proteger as rotas e dados sensíveis contra acessos indevidos e ataques maliciosos.
* **Escalabilidade**: O sistema precisa ser facilmente escalável para suportar o crescimento de usuários sem comprometer o desempenho.