Requisitos Não Funcionais

#### **2.1 Segurança**

* **Autenticação e Autorização**: Utilização de Spring Security com JWT para autenticação de usuários e controle de acesso aos endpoints da API.
* **Proteção de Dados Sensíveis**: Armazenamento seguro de senhas utilizando hash bcrypt.
* **Comunicação Segura**: Implementação de HTTPS para garantir a comunicação segura entre cliente e servidor.
* **Proteção contra Vulnerabilidades**: Implementação de medidas de segurança contra ataques como CSRF, SQL Injection e XSS.

#### **2.2 Performance**

* **Otimização de Consultas**: Consultas SQL e JPA otimizadas para garantir performance de leitura e escrita eficientes, especialmente em sistemas com grandes volumes de dados.
* **Uso de Cache**: Implementação de cache em endpoints de leitura frequente para reduzir o tempo de resposta.
* **Escalabilidade**: Arquitetura modular e escalável utilizando Spring Boot, permitindo o crescimento horizontal e vertical conforme o aumento de usuários e dados.
* **Balanceamento de Carga**: Integração com AWS para garantir balanceamento de carga e alta disponibilidade.

#### **2.3 Usabilidade e Interface do Usuário**

* **Interface Responsiva**: Utilização de Bootstrap para garantir que a interface seja responsiva e compatível com diferentes dispositivos (desktop, tablet e celular).
* **Navegação Intuitiva**: Navegação simples e intuitiva, com menus claros, pesquisa eficiente e links relacionados entre as funcionalidades do sistema.
* **UX/UI Consistente**: Design consistente de interfaces e telas com foco na usabilidade e na experiência do usuário.

#### **2.4 Manutenção e Escalabilidade**

* **Arquitetura Modular**: Utilização de microsserviços para separar funcionalidades de forma modular, facilitando a manutenção e a adição de novas funcionalidades.
* **Monitoramento**: Implementação de ferramentas para monitoramento do sistema em tempo real, como logs centralizados, métricas de performance e monitoramento de uso de API.
* **Tarefas Automatizadas**: Utilização de tarefas automatizadas para processos críticos e agendados no sistema, como envio de e-mails e atualização de dados.

#### **2.5 Confiabilidade**

* **Tratamento de Exceções**: Implementação de tratamento centralizado de exceções na API REST para garantir respostas amigáveis e seguras em casos de erro.
* **Backups**: Implementação de rotinas de backup automáticas para garantir a integridade dos dados armazenados.
* **Disponibilidade**: Utilização da AWS para garantir alta disponibilidade e redundância para os microsserviços, mantendo o sistema disponível mesmo em situações de falhas.

#### **2.6 Testabilidade**

* **Testes Automatizados**: Implementação de testes automatizados para API REST utilizando ferramentas como Postman, Mockito e JUnit para garantir a integridade das funcionalidades.
* **Cobertura de Testes**: Garantia de que as principais funcionalidades e integrações sejam testadas, com alta cobertura de código.
* **Testes de Performance**: Realização de testes de performance para medir a escalabilidade e tempo de resposta sob diferentes condições de carga.

#### **2.7 Integrações**

* **Integração com APIs Externas**: Integração com APIs de transporte, PIX, boleto, cartão de crédito, Nota Fiscal Eletrônica e e-mail marketing para proporcionar funcionalidades como pagamentos, envio de e-mails e geração de documentos fiscais.
* **API RESTful**: Todos os microsserviços devem ser expostos como APIs RESTful com documentação clara e precisa, utilizando Swagger para facilitar a integração com sistemas externos.

#### **2.8 Tecnologias e Arquitetura**

* **Spring Boot**: Uso do Spring Boot para inicialização rápida do projeto, com configuração simplificada e dependências bem definidas.
* **JPA e Spring Data**: Utilização do Spring Data JPA para simplificação do acesso ao banco de dados e criação de entidades com relacionamento adequado.
* **AWS Cloud**: Configuração da AWS para hospedagem de microsserviços, garantindo flexibilidade, escalabilidade e resiliência.

#### **2.9 Banco de Dados**

* **Divisão de Banco por Empresa**: Implementação de banco de dados segregado por empresa, com a possibilidade de múltiplos esquemas para organização dos dados.
* **Consultas Dinâmicas com SQL**: Utilização de consultas SQL dinâmicas para suportar diferentes filtros e condições de busca complexas.
* **Integração com Flyway**: Utilização do Flyway para versionamento e migração do banco de dados, garantindo consistência entre os ambientes de desenvolvimento, teste e produção.

#### **2.10 Monitoramento e Logs**

* **Logs de Aplicação**: Implementação de logging detalhado para facilitar a resolução de problemas e garantir visibilidade sobre a execução das aplicações.
* **Monitoramento de Métricas**: Integração com ferramentas de monitoramento, como AWS CloudWatch ou Prometheus, para coletar métricas de desempenho e utilização.

#### **2.11 Documentação e Manutenção**

* **Documentação de API**: Criação de documentação da API utilizando Swagger, permitindo que as integrações externas e internas sejam feitas de forma fácil e clara.
* **Código Bem Documentado**: Garantia de que o código fonte será bem documentado e organizado, para facilitar a manutenção e evolução do sistema ao longo do tempo.