Non-Functional Requirements

#### **2.1 Security**

* **Authentication and Authorization**: Use of Spring Security with JWT for user authentication and access control to API endpoints.
* **Sensitive Data Protection**: Secure storage of passwords using bcrypt hashing.
* **Secure Communication**: Implementation of HTTPS to ensure secure communication between client and server.
* **Protection Against Vulnerabilities**: Implementation of security measures against attacks like CSRF, SQL Injection, and XSS.

#### **2.2 Performance**

* **Query Optimization**: Optimized SQL and JPA queries to ensure efficient read and write performance, especially in systems with large data volumes.
* **Cache Usage**: Implementation of caching on frequently accessed endpoints to reduce response times.
* **Scalability**: Modular and scalable architecture using Spring Boot, allowing for horizontal and vertical growth as users and data increase.
* **Load Balancing**: Integration with AWS to ensure load balancing and high availability.

#### **2.3 Usability and User Interface**

* **Responsive Interface**: Use of Bootstrap to ensure the interface is responsive and compatible with different devices (desktop, tablet, and mobile).
* **Intuitive Navigation**: Simple and intuitive navigation with clear menus, efficient search, and related links between system functionalities.
* **Consistent UX/UI**: Consistent design of interfaces and screens focused on user experience and usability.

#### **2.4 Maintenance and Scalability**

* **Modular Architecture**: Use of microservices to separate functionalities in a modular way, facilitating maintenance and the addition of new features.
* **Monitoring**: Implementation of real-time system monitoring tools, including centralized logs, performance metrics, and API usage monitoring.
* **Automated Tasks**: Use of automated tasks for critical and scheduled processes in the system, such as sending emails and updating data.

#### **2.5 Reliability**

* **Exception Handling**: Centralized exception handling in the REST API to ensure user-friendly and secure responses in case of errors.
* **Backups**: Implementation of automated backup routines to ensure data integrity.
* **Availability**: Use of AWS to ensure high availability and redundancy for microservices, keeping the system available even in case of failures.

#### **2.6 Testability**

* **Automated Tests**: Implementation of automated tests for the REST API using tools like Postman, Mockito, and JUnit to ensure the integrity of functionalities.
* **Test Coverage**: Ensuring that core functionalities and integrations are tested, with high code coverage.
* **Performance Testing**: Performing performance tests to measure scalability and response times under different load conditions.

#### **2.7 Integrations**

* **External API Integrations**: Integration with external APIs for transportation, PIX, boleto, credit card, electronic invoices, and email marketing to provide functionalities like payments, email sending, and generating fiscal documents.
* **RESTful API**: All microservices should be exposed as RESTful APIs with clear and accurate documentation, using Swagger to facilitate integration with external systems.

#### **2.8 Technologies and Architecture**

* **Spring Boot**: Use of Spring Boot for rapid project setup with simplified configuration and well-defined dependencies.
* **JPA and Spring Data**: Use of Spring Data JPA to simplify database access and the creation of entities with appropriate relationships.
* **AWS Cloud**: Configuration of AWS for hosting microservices, ensuring flexibility, scalability, and resilience.

#### **2.9 Database**

* **Database Segregation by Company**: Implementation of a database segregated by company, with the ability to have multiple schemas for data organization.
* **Dynamic SQL Queries**: Use of dynamic SQL queries to support various filters and complex search conditions.
* **Flyway Integration**: Use of Flyway for database versioning and migration, ensuring consistency across development, testing, and production environments.

#### **2.10 Monitoring and Logs**

* **Application Logs**: Implementation of detailed logging to facilitate troubleshooting and ensure visibility into application execution.
* **Metric Monitoring**: Integration with monitoring tools like AWS CloudWatch or Prometheus to collect performance and usage metrics.

#### **2.11 Documentation and Maintenance**

* **API Documentation**: Creation of API documentation using Swagger, making it easy for both external and internal integrations.
* **Well-Documented Code**: Ensuring that the source code is well-documented and organized to facilitate system maintenance and evolution over time.