**Non-Functional Requirements**

**1. Performance Requirements**

* **Response Time**: The system should respond to user requests within 1-2 seconds for standard operations such as viewing stock data and running predictions. Data-intensive operations like model training should be optimized to run asynchronously and notify users upon completion.
* **Data Processing**: The Python scraping and model training scripts should process data and insert it into the database efficiently, with minimal delay. The scraping process should be capable of handling hundreds of stock items in a batch.
* **Scalability**: The backend should be designed to scale horizontally to handle increasing numbers of concurrent users and requests, particularly when displaying complex data visualizations or handling user predictions.

**2. Reliability Requirements**

* **System Availability**: The system should maintain 99.9% uptime to ensure that it is always accessible for clients and admins. Downtime for maintenance should be minimized and scheduled during non-peak hours.
* **Data Accuracy**: Data scraped from the source website (MSE MK) and stored in the database should be accurate and up-to-date to maintain the reliability of predictions and reports.
* **Error Handling**: The system should handle errors gracefully, with comprehensive logging and error messages provided to both users and system administrators. The Python script should include retry logic for network issues during scraping.

**3. Security Requirements**

* **User Authentication and Authorization**: Implement **Spring Security** with JWT for secure user authentication and role-based authorization. Ensure that only clients and admins have access to their respective functionalities.
* **Data Protection**: All sensitive data, such as passwords, should be stored as secure hashes using industry-standard algorithms like bcrypt. Personal data must comply with data protection regulations (e.g., GDPR).
* **Secure Communication**: Use **HTTPS** to encrypt all data transmitted between the frontend and backend, ensuring that user and system data are secure.
* **Access Control**: Admin-specific functions, such as configuring AI models and viewing logs, should be protected by role-based access controls to prevent unauthorized users from accessing critical operations.
* **SQL Injection Prevention**: Ensure that all database interactions use prepared statements or ORM frameworks (e.g., Spring Data JPA) to avoid SQL injection vulnerabilities.

**4. Usability Requirements**

* **User Interface**: The React frontend with **Material-UI** should provide a responsive, intuitive, and visually appealing interface for clients and admins. The design should be user-friendly and support multiple devices, including desktops, tablets, and smartphones.
* **Ease of Use**: The app should require minimal training for users. Clients should be able to navigate the app easily to view stock data, request predictions, and see overall statistics.
* **Accessibility**: Ensure the application meets basic **WCAG 2.1** standards for accessibility so that it is usable by people with disabilities (e.g., keyboard navigation, screen reader compatibility).

**5. Maintainability Requirements**

* **Code Quality**: The codebase should follow industry best practices for readability, modularity, and documentation. Use tools like **SonarQube** to ensure code quality and maintainability.
* **Documentation**: Comprehensive documentation should be provided for all modules, including backend services, frontend components, and Python scripts. This documentation should detail system architecture, API endpoints, and data structures.
* **Version Control**: All code should be managed using **Git**.
* **Modular Design**: The app should be designed using modular components so that new features can be added or existing ones modified with minimal impact on other parts of the system.

**6. Scalability Requirements**

* **Horizontal Scalability**: The backend should support horizontal scaling, allowing additional instances of the application to be deployed to manage increased user load.
* **Database Scalability**: The PostgreSQL database should be optimized for handling large datasets and allow for sharding or replication if needed.

**7. Interoperability Requirements**

* **Cross-Platform Support**: The frontend should work seamlessly on all major web browsers (e.g., Chrome, Firefox, Safari, Edge) and operating systems (e.g., Windows, macOS, Linux).

**12. Performance Optimization Requirements**

* **Caching**: Use **Redis** or **Memcached** for caching frequent read operations to reduce database load.
* **Asynchronous Processing**: Implement background processing for intensive operations, such as AI training or large data fetches, using tools like **Spring Boot Async**.
* **Pagination**: Implement pagination and data chunking for data-intensive operations in the frontend and backend to reduce the load on the server and improve response times.