**Functional Requirements**

* The system will scrape historical data.

Actors: Scrape Model.

Short Description: The scrape model retrieves data from mse.mk website.

* 1. The scrape model is triggered (manually)
  2. The scrape model sends a request to the stock website mse.mk
  3. The scrape model processes the response and extracts the necessary stock data
  4. The extracted data is stored in a temporary format
* The system will insert the data in a database.

Actors: Scrape Model

Short Description: The scrape model inserts scraped data into the database.

* 1. The scrape model formats the scraped data to match the stock item entity structure
  2. The scrape model connects to the database
  3. The scrape model inserts each stock item into the database
  4. The system logs the success or failure of the insertion process
  5. The connection to the database is closed
* The system will enable training of the AI model.

Actors: AI Model

Short Description: The ai model trains itself using existing data from the database.

* 1. The ai model is triggered by the admin
  2. The ai model queries the database for relevant stock item data
  3. The ai model processes the data and updates its internal model parameters
  4. The system logs the training results for future references
  5. The ai model saves the trained model state for future use in predictions
* The system will allow the admin to configure the AI model

Actors: Admin

Short Description: The admin configures the settings from the ai model.

* 1. The admin navigates to the ai model configuration page
  2. The admin updates parameters such as learning rate, training frequency, data range etc.
  3. The system validates the configuration changes
  4. The ai model saves the new configuration
  5. The system logs the change made by the admin
* The system will offer user registration.

Actors: Client

Short Description: A client registers for an account on the platform.

* 1. The client navigates to the registration page
  2. The client enters their username, email and password
  3. The system validates the input and hashes the password
  4. The system creates a new **user** entity in the database
  5. The system sends a confirmation message to the client
  6. The client is redirected to the login page after successful registration
* The system will offer user login.

Actors: Client, Admin

Short Description: A user logs in to access the platform.

* 1. The user navigates to the login page.
  2. The user inputs their username/email and password.
  3. The system verifies the credentials by checking the hashed password against the stored password\_hash.
  4. If valid, the user is granted access, and a session is created.
  5. The user is redirected to their dashboard.
* The system will offer authentication and session management.

Actors: User

Short Description: Users authenticate to access the platform securely

* 1. The user enters their credentials.
  2. The system verifies the credentials using Spring Security.
  3. If verified, a JWT or session token is generated for the user.
  4. The user session is maintained for further interactions.
* The system will predict the stock market.

Actors: Client, AI Model

Short Description: The client requests the AI Model to predict the best time to buy a specific stock item.

* 1. The client navigates to the prediction page.
  2. The client selects a specific stock item or chooses the overall statistics option.
  3. The system triggers the ai model to use its training results to predict the best time to buy.
  4. The ai model runs the prediction algorithm and returns the result.
  5. The system displays the prediction result to the client.
* The system will log system activities.

Actors: System (Internal Process)

Short Description: The system logs activities related to scraping, training and user actions.

* 1. The system logs activities such as data scraping, data insertion, model training, and user interactions.
  2. Logs are stored in the database for audit purposes.
  3. Admins can access these logs for monitoring.
* The system will offer to update the user information.

Actors: Client

Short Description: A client updates their profile information.

* 1. The client navigates to the profile settings page.
  2. The client updates information such as email or password.
  3. The system verifies and validates the changes.
  4. The system updates the user entity in the database.
  5. The system confirms the update to the client.

**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.